Advanced Machine Unlearning: Programming Assignment 4

Tejas Sharma, Muskaan Jain, Ojas Maheshwari

Spring 2025

1 Task 0: Environment Setup and Result Reproduction

```
Using device: mps

--- Model Architecture ---
EnergyRegressor(
    (net): Sequential(
    (0): Linear(in_features=784, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Linear(in_features=4096, out_features=2048, bias=True)
    (3): ReLU(inplace=True)
    (4): Linear(in_features=2048, out_features=1024, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=1024, out_features=512, bias=True)
    (7): ReLU(inplace=True)
    (8): Linear(in_features=512, out_features=256, bias=True)
    (9): ReLU(inplace=True)
    (10): Linear(in_features=256, out_features=128, bias=True)
    (11): ReLU(inplace=True)
    (12): Linear(in_features=128, out_features=64, bias=True)
    (13): ReLU(inplace=True)
    (14): Linear(in_features=64, out_features=32, bias=True)
    (15): ReLU(inplace=True)
    (16): Linear(in_features=32, out_features=16, bias=True)
    (17): ReLU(inplace=True)
    (18): Linear(in_features=16, out_features=8, bias=True)
    (19): ReLU(inplace=True)
    (20): Linear(in_features=8, out_features=4, bias=True)
    (21): ReLU(inplace=True)
    (22): Linear(in_features=4, out_features=2, bias=True)
    (23): ReLU(inplace=True)
    (24): Linear(in_features=2, out_features=1, bias=True)
    )
)
------------------------

Loading dataset from ./A4_test_data.pt...
Dataset loaded in 0.18s. Shape: x=torch.Size([100000, 784]), energy=torch.Size([100000, 1])
Using device: mps
Using device: mps

--- Test Results ---
Loss: 288.1554
--- Script Finished ---
```

We set `DATASET_PATH='./A4_test_data.pt'` and loaded the state dictionary of the `EnergyRegressor` model from `MODEL_WEIGHTS_PATH='./trained_model_weights.pth'`. Above is the output on running `get_results.py`.

The files `A4_test_data.pt` and `trained_model_weights.pth` are downloaded from the assignment drive link, and pasted into the directory containing the python script `get_results.py`.

2 Task 1: MCMC Sampling Implementation

We imported from the task 0 file `get_results.py`, the `EnergyRegressor` class. We also used the `TSNE` class from `sklearn.manifold` to visualize the samples generated by the MCMC sampling algorithm that we implemented. Lastly, we imported the `clock_gettime` function from the `time` library to measure the time taken by our sampling algorithms. The file for task 1 is `sampling_algos.py`.

We assume that this file is in the same directory as the task 0 file and also has the trained model weights file. On executing this file, we got the following output:

```
Using device: mps
--- Sampling Times ---
Algo-1 Burn-in Time: 26.6275 seconds
Algo-1 Completion Time: 48.6032 seconds
Algo-2 Burn-in Time: 10.4676 seconds
Algo-2 Completion Time: 21.3794 seconds
```

We found that the best results were obtained on setting the value of burn-in to 1000 (1000 samples pre-generated before actual sampling), and 1000 samples were generated after that, which were plotted. The value of τ or the learning rate is set to 0.01.

The above results were generated using these hyperparameters. It is evident that the second algorithm is more than twice as fast as the first, which is understandable considering that we do not evaluate any function (probabilistic) that decides whether we reject or accept the new sample.

2.1 2D and 3D visualization of generated samples

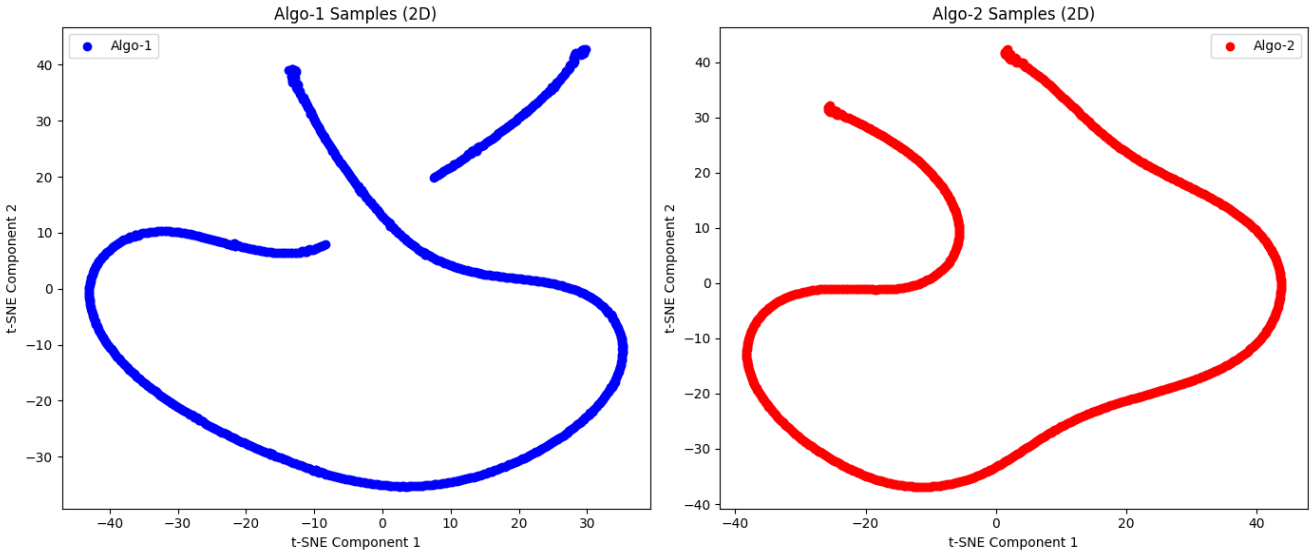


Figure 1: 2D visualization of generated samples

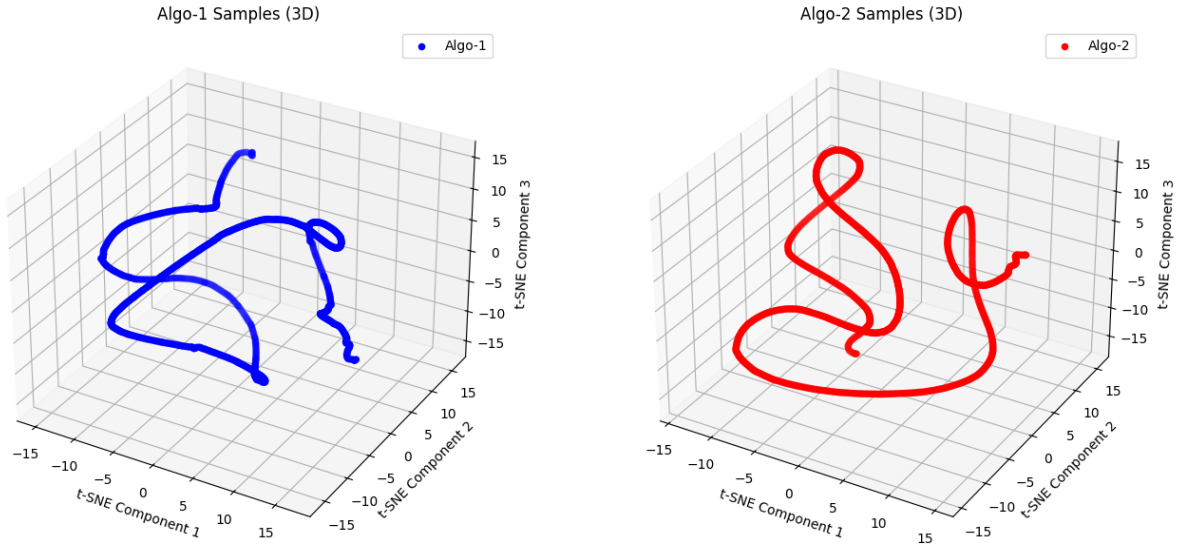
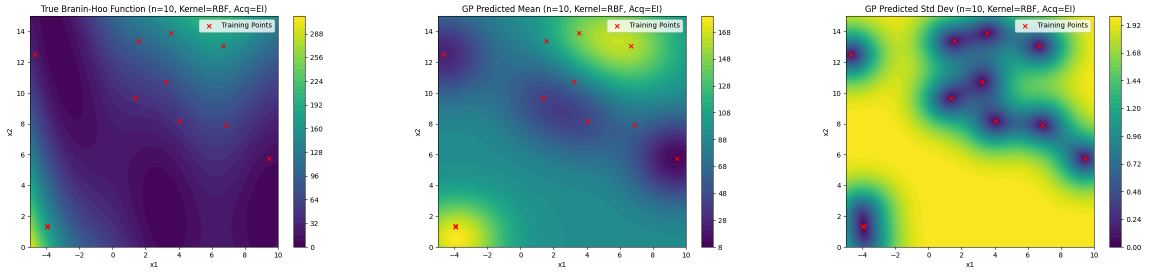


Figure 2: 3D visualization of generated samples

3 Task 2: Approximating Branin-Hoo Using Gaussian Processes

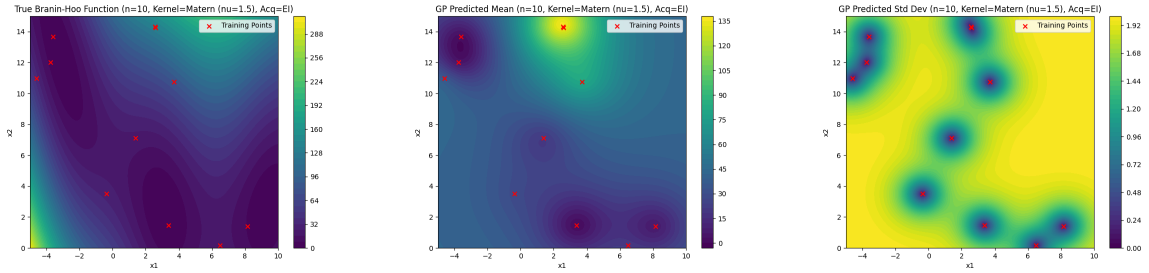
3.1 Plots:

3.1.1 EI:



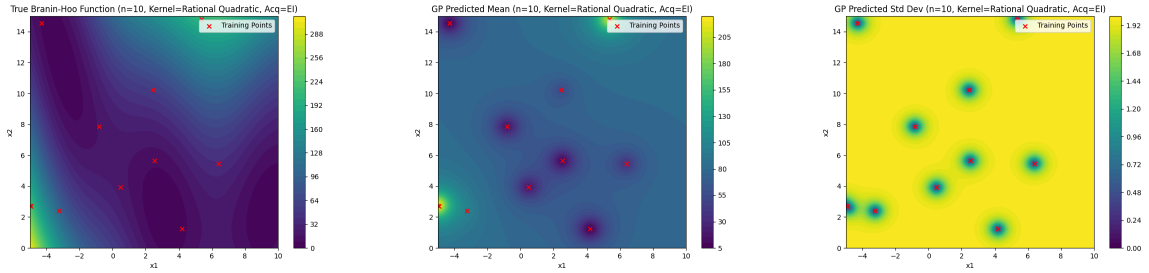
(a) True Function (b) Predicted Mean (c) Predicted Std Dev

Figure 3: Kernel: rbf, Acquisition: EI, Samples: 10



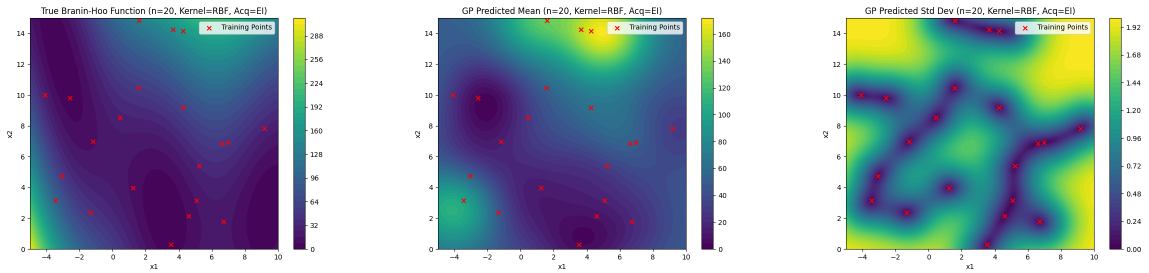
(a) True Function (b) Predicted Mean (c) Predicted Std Dev

Figure 4: Kernel: matern, Acquisition: EI, Samples: 10



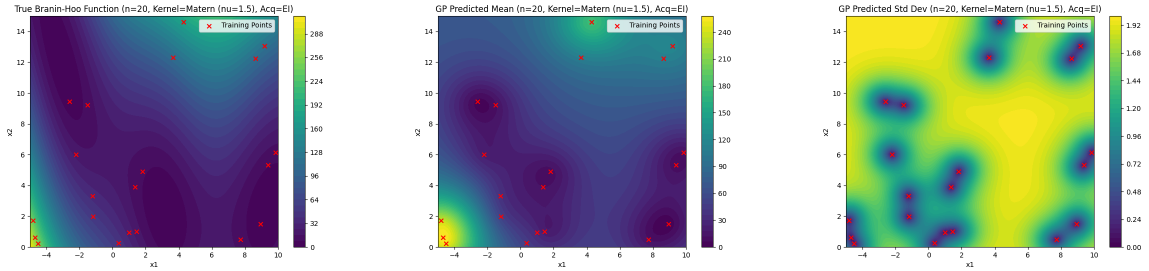
(a) True Function (b) Predicted Mean (c) Predicted Std Dev

Figure 5: Kernel: rational_quadratic, Acquisition: EI, Samples: 10



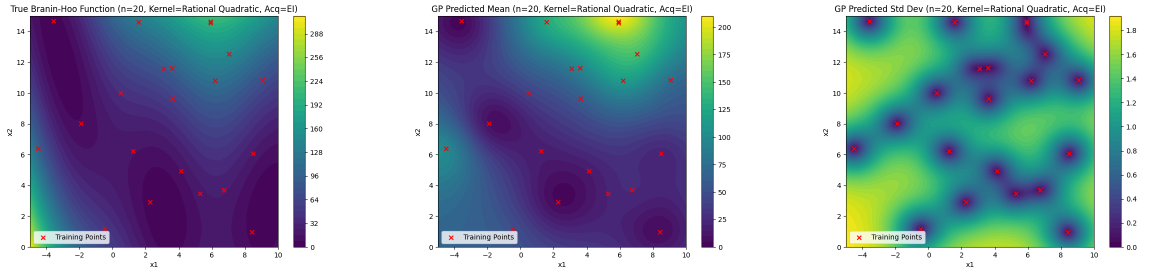
(a) True Function (b) Predicted Mean (c) Predicted Std Dev

Figure 6: Kernel: rbf, Acquisition: EI, Samples: 20



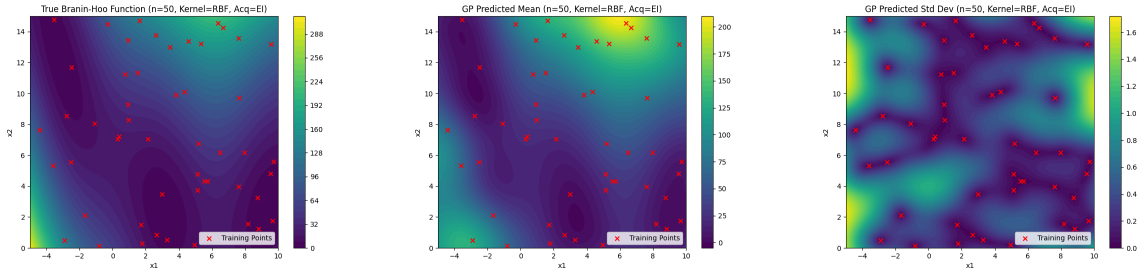
(a) True Function (b) Predicted Mean (c) Predicted Std Dev

Figure 7: Kernel: matern, Acquisition: EI, Samples: 20



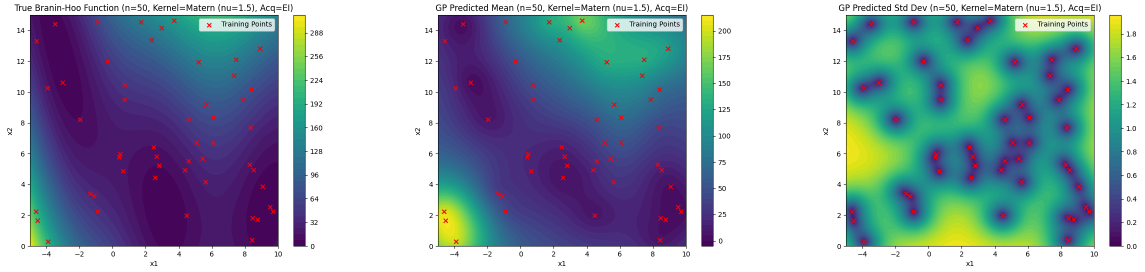
(a) True Function (b) Predicted Mean (c) Predicted Std Dev

Figure 8: Kernel: rational_quadratic, Acquisition: EI, Samples: 20



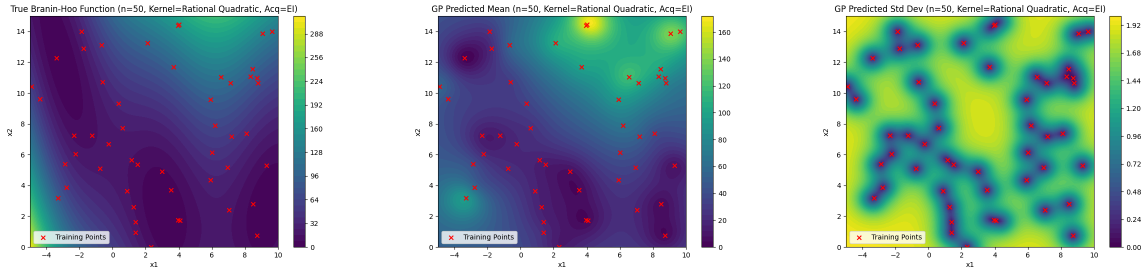
(a) True Function (b) Predicted Mean (c) Predicted Std Dev

Figure 9: Kernel: rbf, Acquisition: EI, Samples: 50



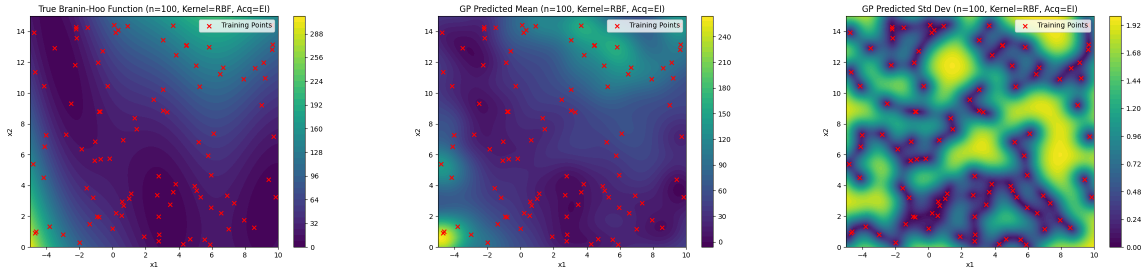
(a) True Function (b) Predicted Mean (c) Predicted Std Dev

Figure 10: Kernel: matern, Acquisition: EI, Samples: 50



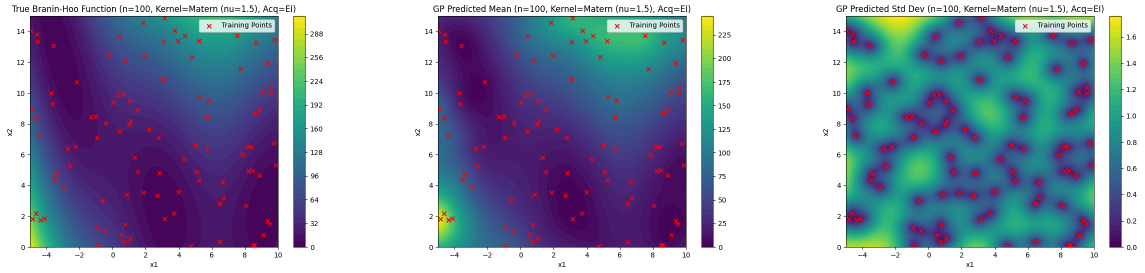
(a) True Function (b) Predicted Mean (c) Predicted Std Dev

Figure 11: Kernel: rational_quadratic, Acquisition: EI, Samples: 50



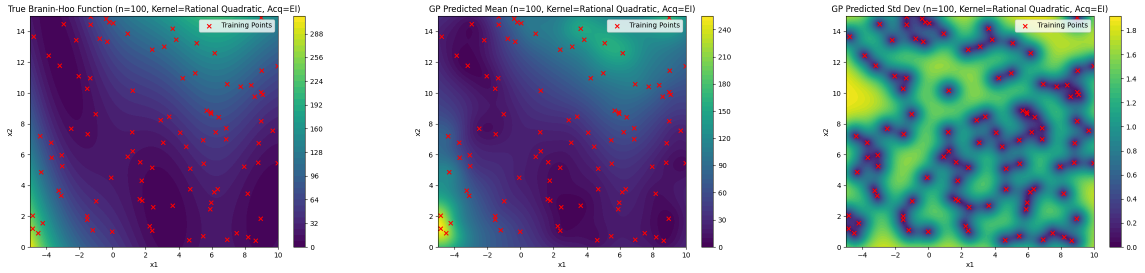
(a) True Function (b) Predicted Mean (c) Predicted Std Dev

Figure 12: Kernel: rbf, Acquisition: EI, Samples: 100



(a) True Function (b) Predicted Mean (c) Predicted Std Dev

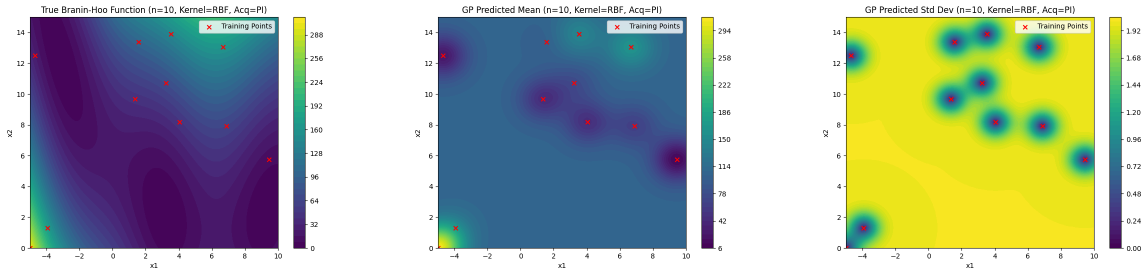
Figure 13: Kernel: matern, Acquisition: EI, Samples: 100



(a) True Function (b) Predicted Mean (c) Predicted Std Dev

Figure 14: Kernel: rational_quadratic, Acquisition: EI, Samples: 100

3.1.2 PI:



(a) True Function (b) Predicted Mean (c) Predicted Std Dev

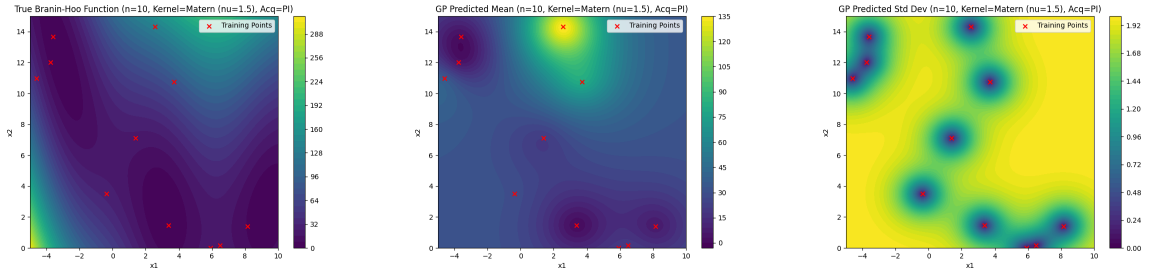
Figure 15: Kernel: rbf, Acquisition: PI, Samples: 10

3.1.3 Random:

3.2 Analysis and Comparison:

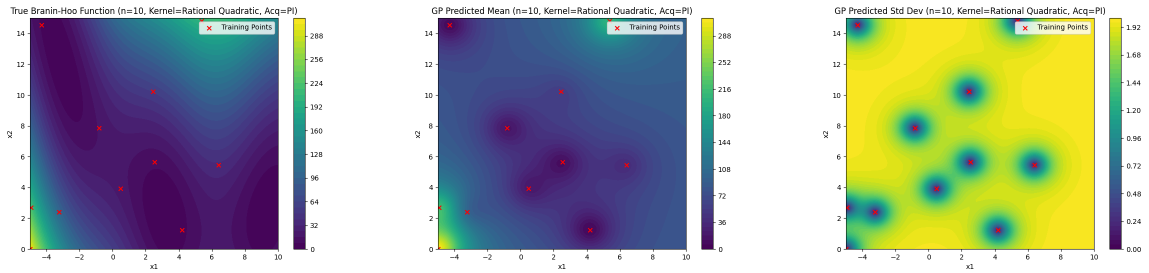
3.2.1 Impact of Sample Size

The performance of Gaussian Process (GP) models is directly proportional to the amount of available training data. As the sample size increases, the model gains more information about the black-box function, which reduces its predictive uncertainty. This is evident in the decreasing standard deviation across the input space. With more data, the GP has better coverage of the domain, allowing it to interpolate with greater confidence and capture complex function behavior more accurately.



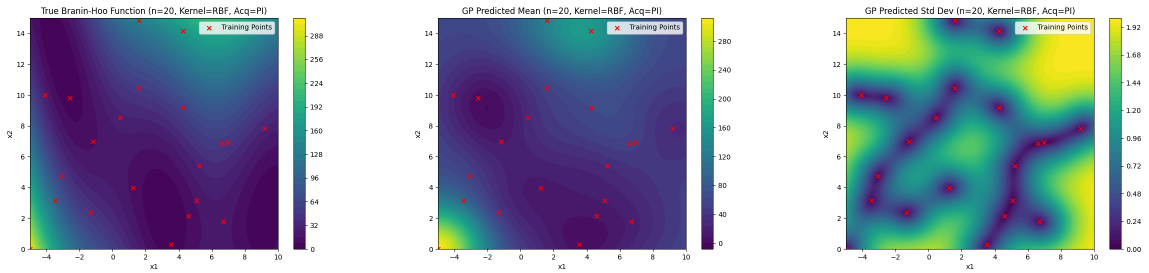
(a) True Function (b) Predicted Mean (c) Predicted Std Dev

Figure 16: Kernel: matern, Acquisition: PI, Samples: 10



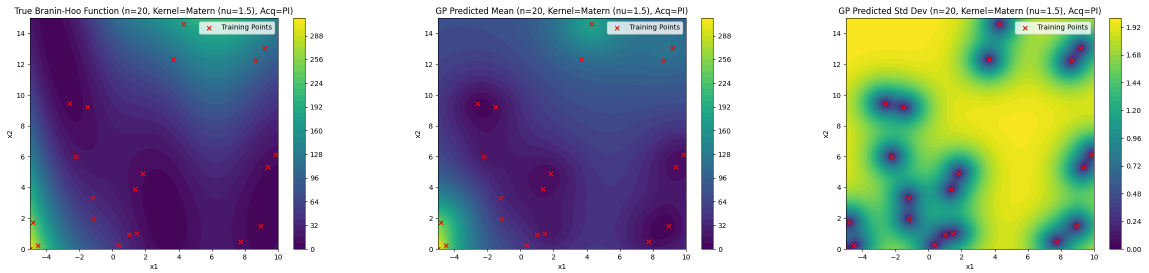
(a) True Function (b) Predicted Mean (c) Predicted Std Dev

Figure 17: Kernel: rational_quadratic, Acquisition: PI, Samples: 10



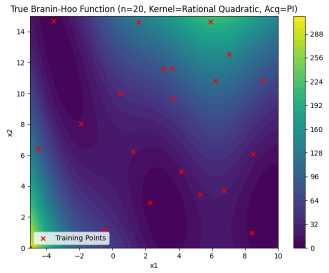
(a) True Function (b) Predicted Mean (c) Predicted Std Dev

Figure 18: Kernel: rbf, Acquisition: PI, Samples: 20

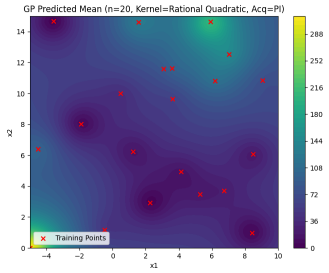


(a) True Function (b) Predicted Mean (c) Predicted Std Dev

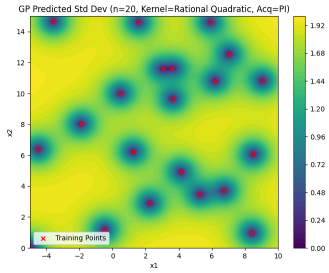
Figure 19: Kernel: matern, Acquisition: PI, Samples: 20



(a) True Function

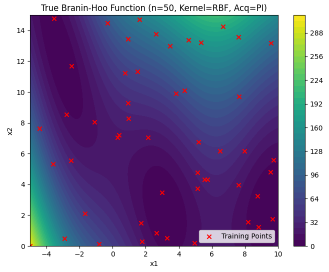


(b) Predicted Mean

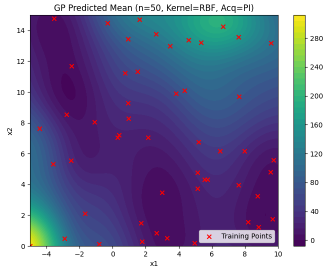


(c) Predicted Std Dev

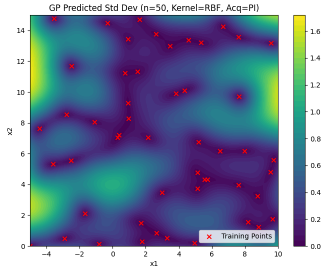
Figure 20: Kernel: rational_quadratic, Acquisition: PI, Samples: 20



(a) True Function

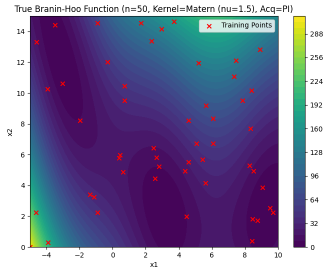


(b) Predicted Mean

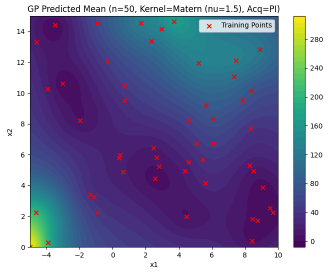


(c) Predicted Std Dev

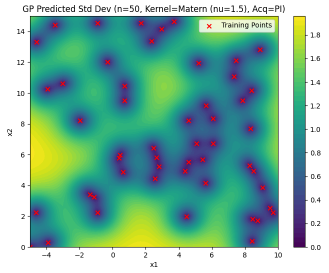
Figure 21: Kernel: rbf, Acquisition: PI, Samples: 50



(a) True Function

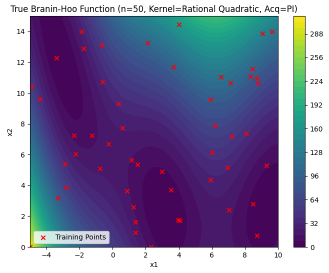


(b) Predicted Mean

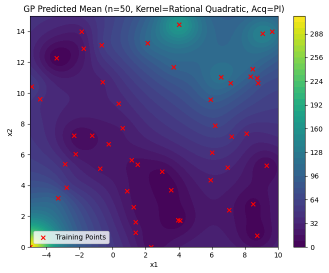


(c) Predicted Std Dev

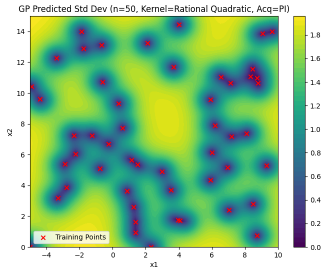
Figure 22: Kernel: matern, Acquisition: PI, Samples: 50



(a) True Function

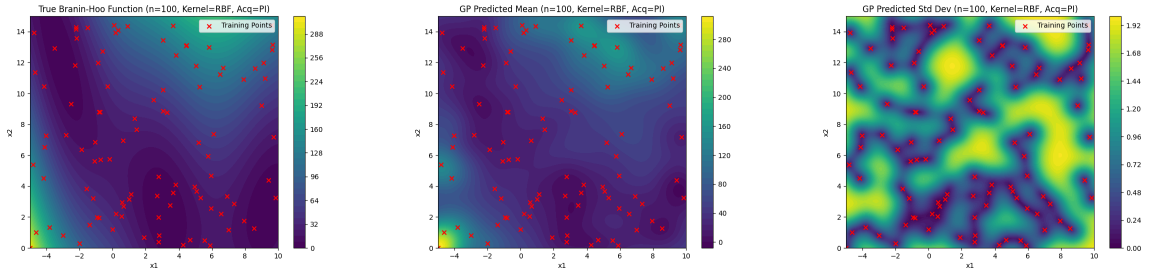


(b) Predicted Mean



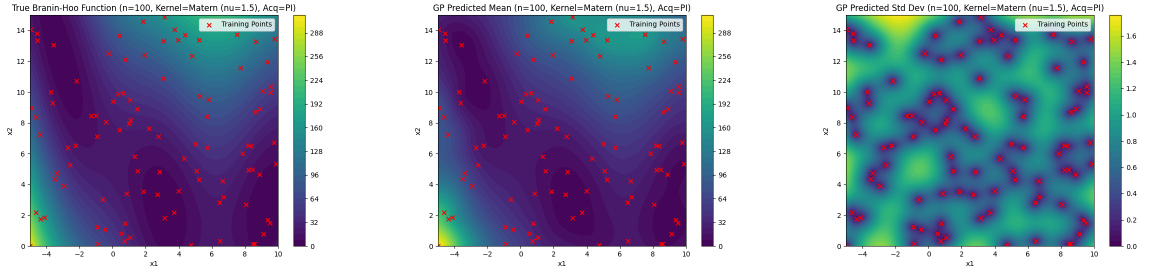
(c) Predicted Std Dev

Figure 23: Kernel: rational_quadratic, Acquisition: PI, Samples: 50



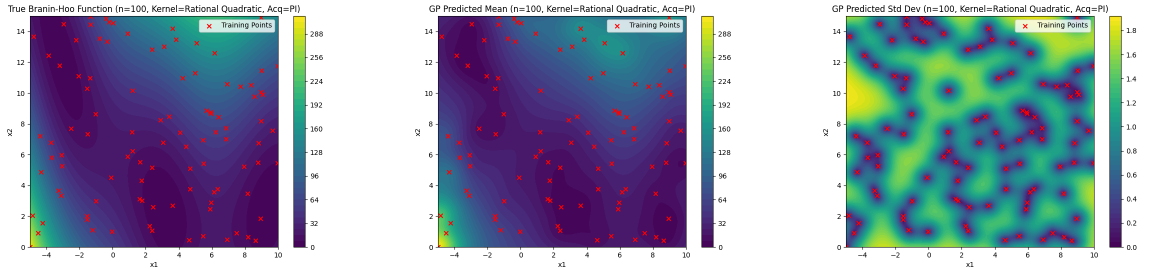
(a) True Function (b) Predicted Mean (c) Predicted Std Dev

Figure 24: Kernel: rbf, Acquisition: PI, Samples: 100



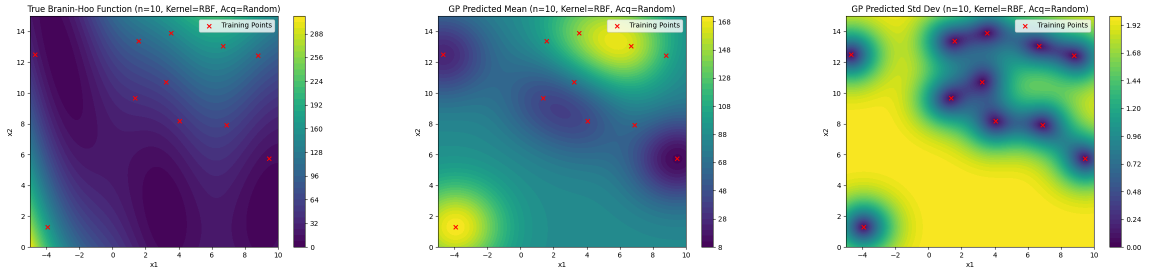
(a) True Function (b) Predicted Mean (c) Predicted Std Dev

Figure 25: Kernel: matern, Acquisition: PI, Samples: 100



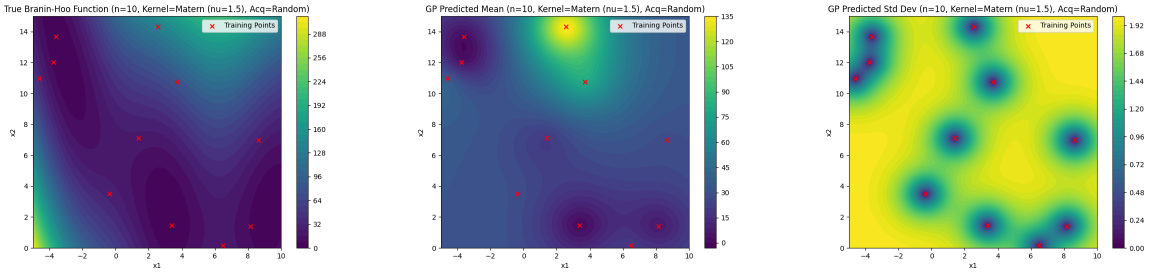
(a) True Function (b) Predicted Mean (c) Predicted Std Dev

Figure 26: Kernel: rational_quadratic, Acquisition: PI, Samples: 100



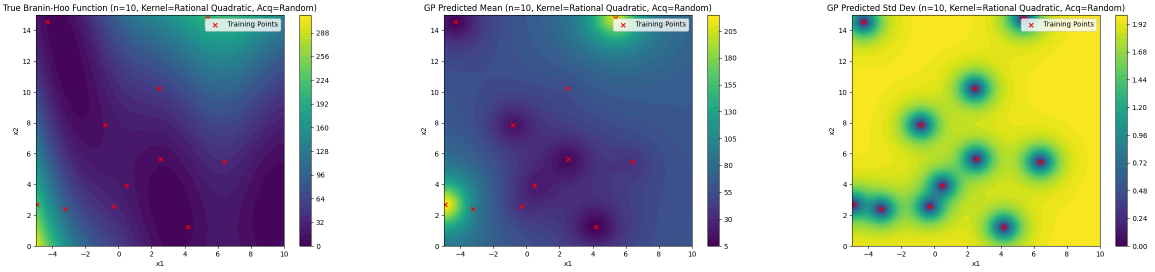
(a) True Function (b) Predicted Mean (c) Predicted Std Dev

Figure 27: Kernel: rbf, Acquisition: Random, Samples: 10



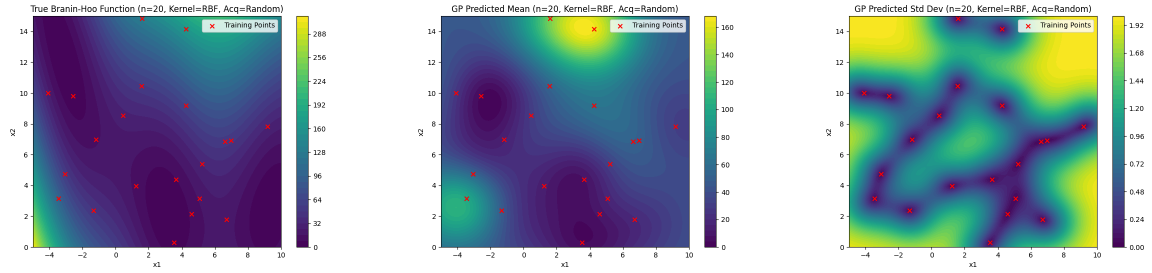
(a) True Function (b) Predicted Mean (c) Predicted Std Dev

Figure 28: Kernel: matern, Acquisition: Random, Samples: 10



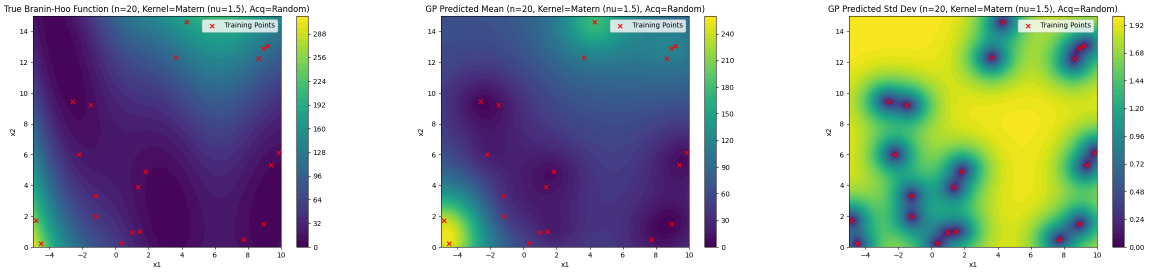
(a) True Function (b) Predicted Mean (c) Predicted Std Dev

Figure 29: Kernel: rational_quadratic, Acquisition: Random, Samples: 10



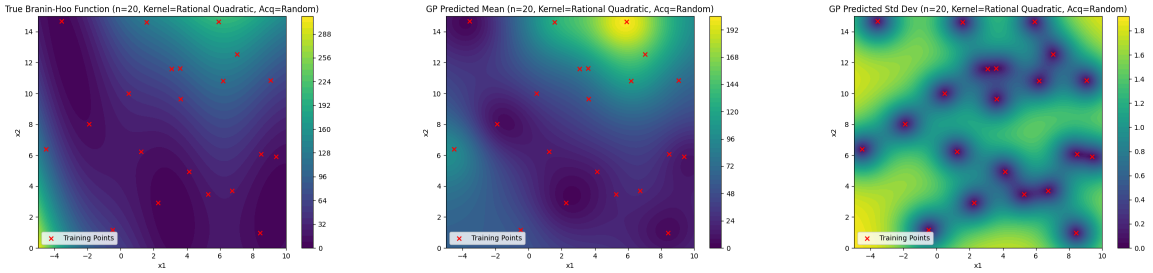
(a) True Function (b) Predicted Mean (c) Predicted Std Dev

Figure 30: Kernel: rbf, Acquisition: Random, Samples: 20



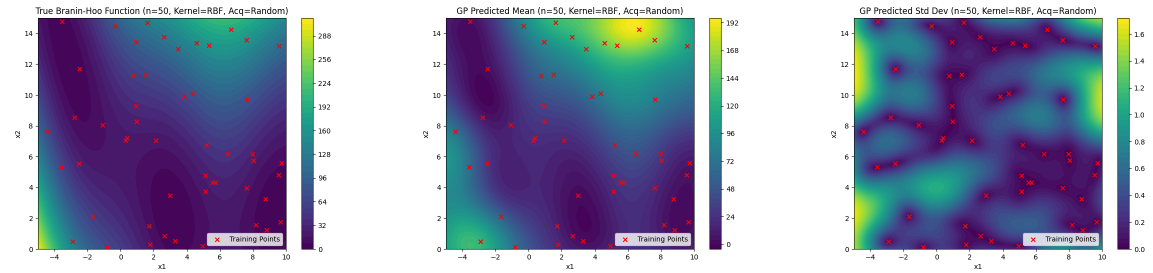
(a) True Function (b) Predicted Mean (c) Predicted Std Dev

Figure 31: Kernel: matern, Acquisition: Random, Samples: 20



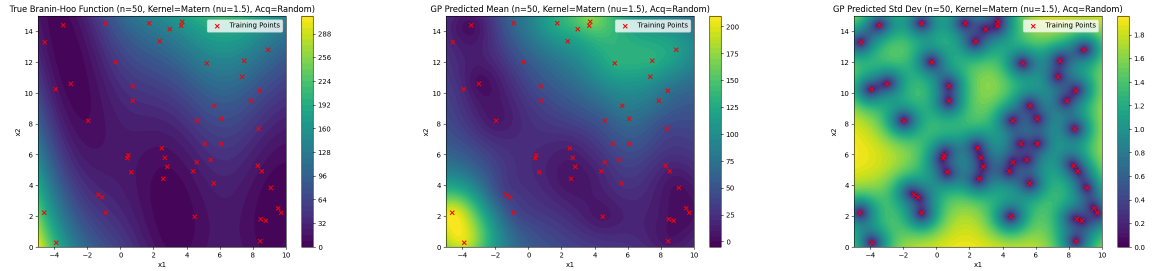
(a) True Function (b) Predicted Mean (c) Predicted Std Dev

Figure 32: Kernel: rational_quadratic, Acquisition: Random, Samples: 20



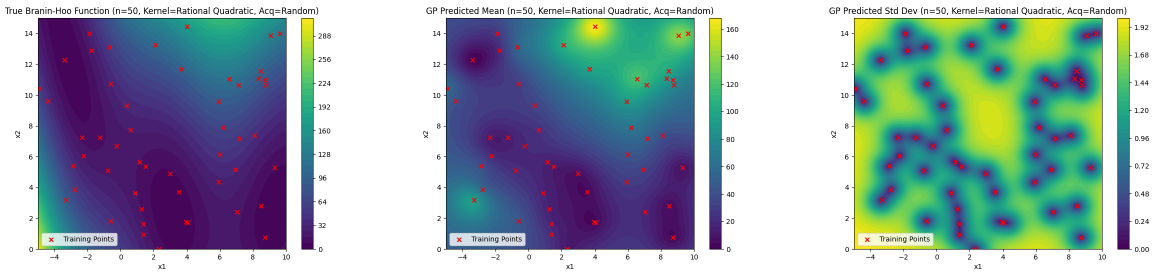
(a) True Function (b) Predicted Mean (c) Predicted Std Dev

Figure 33: Kernel: rbf, Acquisition: Random, Samples: 50



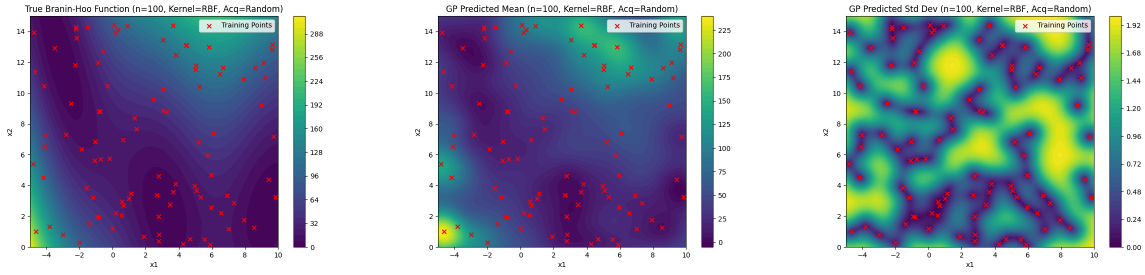
(a) True Function (b) Predicted Mean (c) Predicted Std Dev

Figure 34: Kernel: matern, Acquisition: Random, Samples: 50



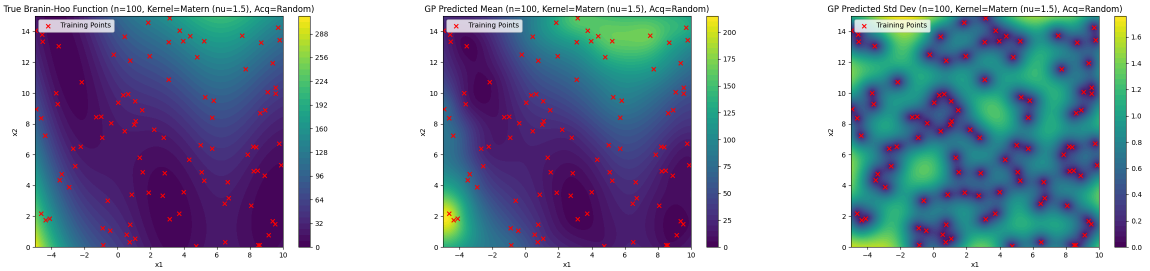
(a) True Function (b) Predicted Mean (c) Predicted Std Dev

Figure 35: Kernel: rational_quadratic, Acquisition: Random, Samples: 50



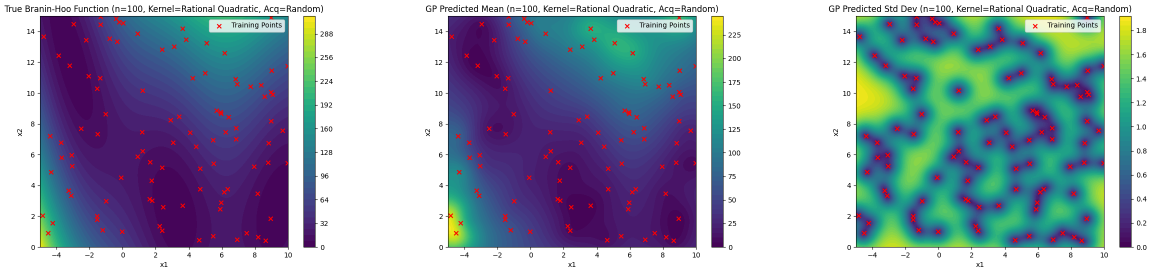
(a) True Function (b) Predicted Mean (c) Predicted Std Dev

Figure 36: Kernel: rbf, Acquisition: Random, Samples: 100



(a) True Function (b) Predicted Mean (c) Predicted Std Dev

Figure 37: Kernel: matern, Acquisition: Random, Samples: 100



(a) True Function (b) Predicted Mean (c) Predicted Std Dev

Figure 38: Kernel: rational_quadratic, Acquisition: Random, Samples: 100

At 100 samples, the predicted mean closely resembles the structure of the true Branin-Hoo function and exhibits smoother transitions and finer detail, especially near function minima. The reduction in uncertainty is due to having more data and also reflects the GP's probabilistic nature, where confidence grows as it observes more consistent patterns.

3.2.2 Influence of Kernel Choice

The kernel (or covariance function) defines the assumptions a GP makes about the smoothness and structure of the function being modeled.

The RBF kernel leads to very smooth and confident predictions with large sample sizes. Its infinite differentiability assumption makes it useful for modeling functions that are known to be smooth, like Branin-Hoo. This is why at 100 samples, RBF shows very low uncertainty across the domain.

RBF's smoothness assumption can be a problem when data is scarce. It tends to overgeneralize, leading to overconfident predictions in regions far from observed data. On the other hand, Matérn and Rational Quadratic kernels allow for more flexibility and less smoothness and are therefore better for scenarios with fewer samples. They preserve local uncertainty, which can prevent misleadingly confident predictions in undersampled areas.

3.2.3 Understanding Uncertainty and Model Confidence

One of the advantages of GPs is their ability to quantify uncertainty. This feature is important in active learning and optimization settings where decisions about where to sample next depend on the predicted values and on the model's confidence.

At low sample sizes, uncertainty is high across most of the input domain. This is expected given the limited observations. The only exceptions are near training points, where the model has direct evidence and can make accurate predictions. As more data is added, this uncertainty reduces.

Note: If the kernel has a strong bias (RBF has a tendency to oversmooth), not all low-uncertainty regions imply correctness.

3.2.4 Performance of Acquisition Strategies

Acquisition functions in Bayesian optimization provide a non-deterministic method for selecting the next best sample point. They attempt to balance uncertainty reduction and refining promising areas.

EI generally proves more effective because it considers both the mean and variance of the GP prediction, guiding the search toward regions that are uncertain and potentially optimal. This balance is important when the global optimum lies in an undersampled area.

PI focuses on improving the best current prediction. While this can lead to faster improvements initially, it may ignore unexplored regions, increasing the chance of getting stuck in local optima, especially when the GP's uncertainty is underestimated.

These strategies work best when the sample size is small, as they make intelligent use of limited data. As the model becomes more confident with larger datasets, the relative impact of the acquisition function reduces, though EI still performs better in convergence efficiency.

3.2.5 Conclusion

With a large enough dataset, rigid kernels like RBF perform well, but under limited data, more adaptable kernels and intelligent acquisition functions make a big difference. The takeaway is that Bayesian optimization is very powerful when uncertainty is correctly estimated and intelligently used.

4 Individual Contributions:

All 3 team members made significant contributions to the assignment.

- Tejas Sharma: Task 0,1
- Muskaan Jain: Task 2 Code and Analysis
- Ojas Maheshwari: Task 2 Analysis

5 Sources:

- Task 0,1: ChatGPT for plotting and understanding TSNE from SkLearn.
- Task 2: ChatGPT for Python array dimensions and small fixes. The algorithm was coded from scratch, referring to the class slides. AI assistance for understanding the exact use and implications of different kernels and acquisition strategies.