

Bayesian Posterior-Guided Domain Shrinking for Efficient Gaussian Process Optimization

Ayush Saun

MT24024

Bayesian Machine Learning Project

Guide: Dr. Ranjitha Prasad

October 27, 2025

1 Introduction: What's the Problem?

1.1 The Core Challenge

Imagine you're trying to find the best setting for a complex system—maybe tuning hyperparameters for a machine learning model or optimizing a chemical process—but each evaluation is expensive (takes hours or costs money).

Why is this hard?

Traditional optimization methods would test thousands of settings randomly or in a grid. But when each test costs \$100 or takes an hour, you need to be smart about which settings to try next.

Bayesian Optimization solves this by:

- **Building a surrogate model** - Uses Gaussian Processes (GP) to model the unknown function based on past observations
- **Balancing exploration vs. exploitation** - Decides whether to test settings that look promising (exploitation) or explore uncertain regions (exploration)
- **Sequential decision-making** - Each new evaluation updates the GP model and informs the next decision

1.2 The Computational Bottleneck

Classical Bayesian Optimization methods like **GP-UCB (Upper Confidence Bound)** face a critical problem:

- At each iteration, they must optimize an acquisition function over the entire search space
- This acquisition optimization becomes expensive as the space grows

- For high-dimensional problems, this can take longer than the actual function evaluation!
- The computational cost scales poorly: $O(T^3)$ for GP updates where T is the number of evaluations

1.3 The REDS Algorithm Solution

The **REDS (Random Exploration with Domain Shrinking)** algorithm addressed computational efficiency through a clever idea:

1. **Random exploration instead of acquisition optimization** - Pick points randomly instead of solving an expensive optimization problem
2. **Domain shrinking** - Progressively eliminate regions of the search space that are unlikely to contain the optimum
3. **Confidence-based pruning** - Use UCB/LCB (Lower Confidence Bound) thresholds to decide which regions to keep

Key achievement: REDS achieves order-optimal regret $\tilde{O}(\sqrt{T\gamma_T})$ while being computationally efficient—no expensive acquisition optimization needed!

1.4 But There's a Gap...

Here's the problem: **REDS uses deterministic confidence bounds for shrinking decisions, which don't fully leverage the probabilistic structure of Gaussian Process posteriors.**

What does this mean?

When REDS decides whether to keep or discard a region, it uses a hard threshold:

- Keep region x if: $\mu(x) + \beta\sigma(x) \geq l_t^*$ (where l_t^* is best LCB so far)
- This is a **binary decision** based on point estimates
- It ignores the **full posterior distribution**—the entire landscape of plausible functions

The gap in existing methods:

- Classical methods (GP-UCB, Thompson Sampling) are fully Bayesian but computationally expensive
- Efficient methods (REDS) use deterministic shrinking rules that ignore posterior distributions
- **Nobody has combined computational efficiency with full Bayesian posterior sampling for domain shrinking**

2 My Original Proposal

2.1 The Core Idea

According to my BML project proposal titled "*Bayesian Posterior-Guided Domain Shrinking for Efficient Gaussian Process Optimization*," **I want to replace REDS' deterministic pruning rule with a Bayesian approach that samples from the GP posterior.**

The key insight: Instead of using a single confidence bound to make shrinking decisions, sample multiple plausible functions from the GP posterior and make probabilistic decisions based on how many samples suggest a region is promising.

2.2 What I Propose: BPG-DS (Bayesian Posterior-Guided Domain Shrinking)

Main innovation:

Replace deterministic thresholds with **posterior sampling**:

1. Sample K functions from GP posterior:

- Draw $f^{(1)}, f^{(2)}, \dots, f^{(K)} \sim \text{GP}(\mu_r, k_r)$
- Each sample represents a plausible objective function given observations so far
- Use Cholesky decomposition for efficient sampling

2. Compute posterior probability:

- For each region x , count how many samples have $f^{(k)}(x)$ near-optimal
- $p_{\text{posterior}}(x) = \frac{1}{K} \sum_{k=1}^K 1[f^{(k)}(x) \geq f_{\max}^{(k)} - \epsilon]$
- This is the empirical probability that x contains the optimum

3. Probabilistic pruning:

- Keep region x if $p_{\text{posterior}}(x) \geq p_{\text{thresh}}$
- This is a **soft decision** based on posterior uncertainty
- Leverages the full distribution, not just point estimates

2.3 Why This is Different from REDS

Aspect	REDS (Deterministic)	BPG-DS (Bayesian)
Pruning criterion	$\mu(x) + \beta\sigma(x) \geq l_t^*$	$p_{\text{posterior}}(x) \geq p_{\text{thresh}}$
Decision type	Binary (keep or discard)	Probabilistic (soft threshold)
Information used	Mean μ and variance σ^2	Full posterior distribution
Sampling	No function sampling	Samples K functions from GP
Uncertainty modeling	Confidence intervals	Posterior probability
Computational cost	Lower (just μ, σ)	Higher (requires sampling)

Table 1: REDS vs. BPG-DS: Key Differences

The conceptual difference:

- **REDS asks:** "Is this region's best-case scenario good enough?"
- **BPG-DS asks:** "How likely is this region to contain the optimum across all plausible functions?"

3 What I Learned from Reading Papers

I read 7 key papers to understand the landscape and identify gaps. Let me walk through each one.

3.1 Paper 1: REDS Algorithm (Salgia et al., ICML 2024)

What they did:

This is my baseline—the paper I'm building on. REDS introduced random exploration with domain shrinking for efficient Bayesian optimization.

Key contributions:

1. **Random exploration:** No expensive acquisition optimization—just sample uniformly from current domain
2. **Domain shrinking:** Progressively eliminate regions using UCB/LCB bounds
 - Compute UCB: $u_t(x) = \mu_t(x) + \beta_t\sigma_t(x)$
 - Compute LCB: $l_t(x) = \mu_t(x) - \beta_t\sigma_t(x)$
 - Best LCB so far: $l_t^* = \max_{s \leq t} l_s(x_s)$
 - Prune x if: $u_t(x) < l_t^*$ (upper bound worse than best lower bound)
3. **Theoretical guarantee:** Achieves $\tilde{O}(\sqrt{T\gamma_T})$ regret where γ_T is maximum information gain

4. **Computational efficiency:** $O(T^2)$ instead of $O(T^3)$ for standard GP methods

What I learned:

Domain shrinking is a powerful idea for efficiency. But the pruning rule is purely deterministic—it doesn't sample from the posterior or use probabilistic reasoning about function uncertainty.

The gap: Uses only point estimates (μ, σ) for shrinking. Ignores the rich information in the full GP posterior distribution.

3.2 Paper 2: GP-UCB (Srinivas et al., 2010)

What they did:

This is the classical benchmark for GP-based Bayesian optimization.

Algorithm:

1. At iteration t , select: $x_t = \arg \max_{x \in \mathcal{D}} \mu_{t-1}(x) + \beta_t \sigma_{t-1}(x)$
2. Evaluate: $y_t = f(x_t) + \epsilon_t$ (noisy observation)
3. Update GP: Condition on (x_t, y_t) to get new posterior
4. Repeat

Key results:

- Sublinear regret: $O(\sqrt{T \gamma_T \log T})$
- β_t grows as $O(\log T)$ for proper exploration
- Works well in practice for low-dimensional problems

Why it's expensive:

- Must solve $\arg \max$ over entire domain at each iteration
- For complex landscapes, this optimization is expensive
- Scales poorly to high dimensions or large domains

What I learned: UCB-based acquisition is theoretically sound but computationally expensive. REDS addressed efficiency; my work addresses the deterministic nature of pruning.

3.3 Paper 3: Thompson Sampling for GPs (Russo & Van Roy, 2018)

What they did:

Thompson Sampling (TS) is a Bayesian approach to exploration:

1. Sample a function from posterior: $f^{(t)} \sim \text{GP}(\mu_{t-1}, k_{t-1})$
2. Optimize the sample: $x_t = \arg \max_x f^{(t)}(x)$
3. Evaluate and update

Why it's Bayesian:

- Samples capture the full posterior uncertainty
- Naturally balances exploration (high uncertainty \rightarrow diverse samples) and exploitation (high mean \rightarrow consistently good)
- Bayesian regret bounds: $\tilde{O}(\sqrt{T\gamma_T})$

Computational cost:

- Sampling from GP: $O(T^3)$ for Cholesky decomposition
- Optimizing each sample: Expensive for complex functions
- Total: More expensive than GP-UCB

What I learned: Posterior sampling is powerful and fully Bayesian, but expensive. **This is exactly what I want to use, but only for pruning decisions, not for selecting evaluation points.**

3.4 Paper 4: Posterior Sampling-Based BO (Takeno et al., 2023)

What they did:

Recent work combining posterior sampling with acquisition functions:

- Sample functions from GP posterior
- Use samples to estimate acquisition function (e.g., Expected Improvement)
- Achieved tighter regret bounds than classical TS

Key insight:

Posterior samples can be used not just for Thompson Sampling, but for various decision-making strategies in Bayesian optimization.

What I learned: **Posterior sampling is a flexible tool—it's not limited to TS. I can use it for domain shrinking decisions!**

3.5 Paper 5: GP Thompson Sampling via Rootfinding (2024)

What they did:

Addressed the computational cost of GP posterior sampling through efficient algorithms:

- Traditional sampling: Generate n samples by drawing $z \sim \mathcal{N}(0, I)$ and computing $f = \mu + Lz$ where $LL^T = K$ (Cholesky)
- Cost: $O(n^3)$ for Cholesky, $O(n^2)$ per sample
- Their method: Use pathwise conditioning and rootfinding for faster sampling

What I learned: Efficient posterior sampling is an active research area. Standard Cholesky-based sampling is feasible for moderate n (hundreds to thousands), which is sufficient for my domain shrinking application.

3.6 Paper 6: Optimizing Posterior Samples for BO (2024)

What they did:

Most recent work on using posterior samples for optimization:

- Instead of optimizing samples independently, optimize over the space of samples
- Can find better points faster by leveraging sample structure
- Practical code available on GitHub

What I learned: There's growing interest in posterior sampling methods. [Implementation references are available, which will help me validate my code using GenAI tools.](#)

3.7 Paper 7: Efficient GP Posterior Sampling (GPflux)

What they provided:

Practical implementation guidance for GP posterior sampling:

- Use decoupled sampling for efficiency
- Batch sampling strategies
- Memory-efficient implementations in TensorFlow/PyTorch

What I learned: [There are well-established libraries and code examples for posterior sampling. This makes implementation feasible.](#)

4 The Gap: What's Missing in All These Approaches

4.1 The Core Problem Nobody Addressed

After reading all these papers, here's what I found:

Method	Computationally Efficient?	Fully Bayesian?
GP-UCB	No (expensive acquisition opt)	No (deterministic UCB)
Thompson Sampling	No (expensive sampling + opt)	Yes (posterior sampling)
REDS	Yes (random + pruning)	No (deterministic bounds)
BPG-DS (My approach)	Yes (random + Bayesian pruning)	Yes (posterior sampling)

Table 2: The Gap in Existing Methods

Nobody has combined random exploration efficiency with full Bayesian posterior sampling for domain shrinking.

4.2 Why This Matters - A Detailed Example

Let me show you exactly why deterministic pruning is limiting.

Scenario: Optimizing a 2D function. Current domain has 4 regions to consider.

What REDS does (deterministic):

At iteration t :

- Best LCB so far: $l_t^* = 8.5$
- Region A: $u_t(A) = \mu(A) + \beta\sigma(A) = 7.0 + 2.0 = 9.0$ Keep ($9.0 \geq 8.5$)
- Region B: $u_t(B) = 7.5 + 1.0 = 8.5$ Keep ($8.5 = 8.5$, borderline)
- Region C: $u_t(C) = 6.5 + 1.5 = 8.0 < 8.5$ Discard ($8.0 < 8.5$)
- Region D: $u_t(D) = 7.2 + 0.8 = 8.0 < 8.5$ Discard ($8.0 < 8.5$)

Problem: Regions C and D are discarded based on a single deterministic bound. But what if:

- Region C has high variance (uncertain)—many posterior samples might have $f(C) > 8.5$
- Region D has *very low* variance (certain)—almost no posterior samples have $f(D) > 8.5$

REDS treats them the same! Both are discarded because both fail the UCB threshold.

What BPG-DS does (Bayesian):

Sample $K = 100$ functions from GP posterior:

Region	μ	σ	$p_{\text{posterior}}$	Decision
A	7.0	2.0	0.72 (72/100 samples near-optimal)	Keep
B	7.5	1.0	0.65 (65/100 samples near-optimal)	Keep
C	6.5	1.5	0.42 (42/100 samples near-optimal)	Keep!
D	7.2	0.8	0.08 (8/100 samples near-optimal)	Discard

Table 3: BPG-DS Posterior Probability Analysis ($p_{\text{thresh}} = 0.3$)

Key difference: Region C is *retained* by BPG-DS because 42% of posterior samples suggest it could be optimal. This is a more nuanced decision that respects uncertainty! Region D is correctly discarded—very few samples support it.

5 My Proposed Solution: BPG-DS

5.1 The Complete Algorithm

Input:

- Search domain $\mathcal{D} \subset \mathbb{R}^d$
- Number of samples K (e.g., 50-100)

- Probability threshold p_{thresh} (e.g., 0.3)
- Exploration parameter $\beta_t = 2 \log(t)$
- Tolerance ϵ for near-optimality

Algorithm:

1. Initialize:

- $t = 1, \mathcal{D}_1 = \mathcal{D}$
- Observe initial points

2. At iteration t :

Step 1: Random Exploration

- Sample x_t uniformly from current domain \mathcal{D}_t
- Evaluate: $y_t = f(x_t) + \epsilon_t$
- Update GP posterior: μ_t, k_t

Step 2: Posterior Sampling (NEW)

- Discretize \mathcal{D}_t into grid of n points: $\{x_1, \dots, x_n\}$
- Compute mean vector: $\boldsymbol{\mu} = [\mu_t(x_1), \dots, \mu_t(x_n)]^T$
- Compute covariance matrix: $K_{ij} = k_t(x_i, x_j)$
- Cholesky decomposition: $K = LL^T$
- For $k = 1, \dots, K$:
 - Draw $\mathbf{z}^{(k)} \sim \mathcal{N}(0, I_n)$
 - Compute sample: $\mathbf{f}^{(k)} = \boldsymbol{\mu} + L\mathbf{z}^{(k)}$

Step 3: Compute Posterior Probabilities

- For each sample k , find: $f_{\max}^{(k)} = \max_i f_i^{(k)}$
- For each point x_i , compute:

$$p_{\text{posterior}}(x_i) = \frac{1}{K} \sum_{k=1}^K 1[f_i^{(k)} \geq f_{\max}^{(k)} - \epsilon]$$

Step 4: Domain Shrinking (Bayesian)

- Shrink domain: $\mathcal{D}_{t+1} = \{x \in \mathcal{D}_t : p_{\text{posterior}}(x) \geq p_{\text{thresh}}\}$

3. Repeat until: $t = T$ (budget exhausted) or $|\mathcal{D}_t|$ sufficiently small

4. Return: $x^* = \arg \max_{x \in \mathcal{D}_T} \mu_T(x)$

5.2 Implementation Details

How to implement posterior sampling efficiently:

```
import numpy as np
from scipy.linalg import cholesky

def sample_gp_posterior(mu, K, num_samples=100):
    """
    Sample functions from GP posterior.

    Args:
        mu: Mean vector (n,)
        K: Covariance matrix (n, n)
        num_samples: Number of function samples

    Returns:
        samples: (num_samples, n) array of function samples
    """
    n = len(mu)
    # Cholesky decomposition
    L = cholesky(K + 1e-6 * np.eye(n), lower=True)

    # Generate samples
    samples = []
    for _ in range(num_samples):
        z = np.random.randn(n)
        f = mu + L @ z
        samples.append(f)

    return np.array(samples)

def compute_posterior_probs(samples, epsilon=0.1):
    """
    Compute posterior probabilities from samples.

    Args:
        samples: (K, n) array of function samples
        epsilon: Near-optimality tolerance

    Returns:
        probs: (n,) array of posterior probabilities
    """
    K, n = samples.shape
    probs = np.zeros(n)

    for k in range(K):
        f_max = samples[k].max()
        near_optimal = samples[k] >= (f_max - epsilon)
```

```

    probs += near_optimal

probs /= K
return probs

```

Libraries I'll use:

- **GPy** or **scikit-learn**: For GP regression
- **NumPy/SciPy**: For matrix operations and Cholesky decomposition
- **Matplotlib**: For visualization

6 Why This is Novel and Important

6.1 What Makes It Different

Aspect	REDS	BPG-DS (My Approach)
Shrinking criterion	Deterministic UCB/LCB	Bayesian posterior probability
Uncertainty treatment	Confidence intervals	Full posterior distribution
Decision type	Binary threshold	Probabilistic threshold
Information used	(μ, σ^2) only	Entire GP posterior
Sampling	No sampling	K posterior samples
Theoretical guarantee	$\tilde{O}(\sqrt{T\gamma_T})$	To be analyzed

Table 4: Novelty Comparison

6.2 My Specific Contributions

1. **First application of posterior sampling to domain shrinking**
 - Thompson Sampling uses posterior samples for *point selection*
 - REDS uses deterministic bounds for *domain shrinking*
 - I combine: random exploration + posterior sampling for shrinking
 - This is a novel combination
2. **Fully Bayesian treatment of uncertainty in efficient optimization**
 - Maintains computational efficiency of random exploration
 - Adds Bayesian rigor to pruning decisions
 - Bridges the gap between efficiency and full Bayesian methodology
3. **Probabilistic framework for domain reduction**

- $p_{\text{posterior}}(x)$ provides interpretable measure of region promise
- Can adjust p_{thresh} to control exploration-exploitation trade-off
- More flexible than hard thresholds

4. Practical implementation with reference code

- Can be implemented with standard libraries (GPy, NumPy)
- Reference implementations for GP posterior sampling exist
- GenAI can help with coding using these references

7 Implementation Plan

7.1 Benchmark Functions for Testing

I'll evaluate on standard Bayesian optimization benchmarks:

Function	Dimension	Domain	Characteristics
Branin	2D	$[-5, 10] \times [0, 15]$	Multimodal, 3 global minima
Hartmann-3D	3D	$[0, 1]^3$	Multimodal, smooth
Hartmann-6D	6D	$[0, 1]^6$	High-dimensional, challenging
Rosenbrock	2D-10D	$[-5, 5]^d$	Narrow valley, difficult
Ackley	2D-10D	$[-5, 5]^d$	Many local minima

Table 5: Benchmark Functions

7.2 Evaluation Metrics

1. **Simple Regret:** $r_T = f(x^*) - f(x_T)$ where x^* is true optimum, x_T is best found
2. **Cumulative Regret:** $R_T = \sum_{t=1}^T [f(x^*) - f(x_t)]$
3. **Computational Time:** Wall-clock time per iteration
4. **Domain Size Evolution:** Track $|\mathcal{D}_t|$ over time
5. **Comparison with baselines:** GP-UCB, Thompson Sampling, REDS

8 Expected Results

8.1 Performance Hypothesis

I expect BPG-DS to:

1. **Maintain computational efficiency similar to REDS**
 - Random exploration: No acquisition optimization
 - Posterior sampling cost: $O(n^3)$ for Cholesky + $O(Kn^2)$ for K samples
 - For moderate n (hundreds), this is feasible

- Should be faster than Thompson Sampling (which optimizes samples)

2. Achieve better regret than REDS

- More nuanced pruning decisions
- Retains promising regions with high uncertainty
- Better exploration-exploitation balance
- Expected improvement: 10-20% better simple regret

3. Be competitive with Thompson Sampling

- Thompson Sampling is fully Bayesian but expensive
- BPG-DS uses Bayesian sampling but only for pruning (cheaper)
- Might have slightly higher regret but much lower computational cost

8.2 Planned Ablation Studies

To prove the value of Bayesian sampling, I'll test:

Configuration	What It Tests
REDS (deterministic baseline)	Original algorithm
BPG-DS with $K = 10$	Low sample count
BPG-DS with $K = 50$	Medium sample count
BPG-DS with $K = 100$	High sample count
BPG-DS with $p_{\text{thresh}} = 0.1$	Aggressive pruning
BPG-DS with $p_{\text{thresh}} = 0.5$	Conservative pruning
BPG-DS without shrinking	Posterior sampling for ranking only

Table 6: Planned Ablation Experiments

9 Challenge Compliance and Feasibility

9.1 Is It Really Novel?

Yes, for these reasons:

1. **Domain shrinking exists** - REDS does it deterministically
2. **Posterior sampling exists** - Thompson Sampling uses it for point selection
3. **Combining them is novel** - Nobody has used posterior sampling specifically for domain shrinking in the REDS framework

This is the first method to replace deterministic pruning rules with Bayesian posterior sampling-based decisions in efficient Bayesian optimization.

9.2 Is It Truly Bayesian?

Yes, completely:

1. **GP prior:** Gaussian Process models the unknown function
2. **Posterior updating:** GP conditioned on observations
3. **Posterior sampling:** Drawing function samples from $\text{GP}(\mu_t, k_t)$
4. **Probabilistic decisions:** Using $p_{\text{posterior}}(x)$ for shrinking

Every step uses the posterior distribution—this is fully Bayesian.

9.3 Is Implementation Feasible with GenAI?

Yes, very feasible:

1. Standard libraries available:

- GPy for GP regression: <https://gpy.readthedocs.io/>
- scikit-learn GP module: Well-documented
- NumPy/SciPy: Standard tools

2. Reference code exists:

- BayesianOptimization library: <https://github.com/bayesian-optimization/BayesianOptimization>
- Thompson Sampling examples: https://num.pyro.ai/en/0.7.1/examples/thompson_sampling.html
- GP posterior sampling tutorials: Multiple sources

3. GenAI can help with:

- Implementing Cholesky-based sampling
- Debugging GP conditioning
- Optimizing NumPy operations
- Validating against reference implementations

I can validate GenAI output against existing TS and REDS implementations to ensure correctness.

10 Conclusion

10.1 Summary

This project addresses a gap in Bayesian optimization: **existing efficient methods (REDS) use deterministic shrinking rules that ignore the full posterior distribution.**

My solution (BPG-DS):

- Replace deterministic UCB/LCB bounds with posterior sampling
- Compute $p_{\text{posterior}}(x)$ from K function samples
- Make probabilistic pruning decisions based on these probabilities
- Maintain computational efficiency through random exploration

10.2 Key Strengths

1. **Novel:** First to combine posterior sampling with domain shrinking in REDS framework
2. **Fully Bayesian:** Uses GP priors, posterior updates, and posterior sampling for decisions
3. **Feasible:** Can be implemented with standard libraries (GPpy, NumPy); reference code exists; GenAI can assist
4. **Challenging:** Requires understanding GPs, Bayesian inference, optimization theory, and efficient sampling
5. **Practical:** Addresses real computational bottleneck in Bayesian optimization

10.3 The Core Insight

Deterministic confidence bounds make binary decisions. Bayesian posterior sampling reveals the spectrum of plausible outcomes. By using this richer information for domain shrinking, we can make more informed pruning decisions that better respect uncertainty.

This is especially important when:

- Function evaluations are expensive (can't afford to discard promising regions)
- Uncertainty varies across the domain (some regions are more certain than others)
- Exploration-exploitation balance is critical

References

- [1] S. Salgia, V. Vakili, and Q. Zhao, *Random Exploration in Bayesian Optimization: Order-Optimal Regret and Computational Efficiency*, ICML 2024. Available: <https://arxiv.org/abs/2310.15351>
- [2] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger, *Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design*, IEEE Transactions on Information Theory, 2012.
- [3] D. Russo and B. Van Roy, *A Tutorial on Thompson Sampling*, Foundations and Trends in Machine Learning, 2018. Available: <https://arxiv.org/abs/1707.02038>
- [4] S. Takeno et al., *Posterior Sampling-Based Bayesian Optimization with Tighter Bayesian Regret Bounds*, ICML 2024. Available: <https://arxiv.org/abs/2311.03760>
- [5] J. Maraval et al., *Gaussian Process Thompson Sampling via Rootfinding*, arXiv preprint, 2024. Available: <https://arxiv.org/abs/2410.08071>
- [6] T. Takahashi et al., *Optimizing Posterior Samples for Bayesian Optimization via Rootfinding*, arXiv preprint, 2024. Available: <https://arxiv.org/abs/2410.22322>
- [7] P. I. Frazier, *A Tutorial on Bayesian Optimization*, arXiv preprint, 2018. Available: <https://arxiv.org/pdf/1807.02811.pdf>
- [8] J. Wang, *An Intuitive Tutorial to Gaussian Process Regression*, arXiv preprint, 2024. Available: <https://arxiv.org/abs/2009.10862>
- [9] GPflux Documentation, *Efficient Posterior Gaussian Process Sampling*, Available: https://secondmind-labs.github.io/GPflux/notebooks/efficient_posterior_sampling.html