Summer Research Internship
Report

Etienne Collin
20237904
Université de Montréal, CIELA, MILA,
contact@etiennecollin.com

Under the supervision of:
Yashar Hezaveh
Laurence Perreault Levasseur

Projet Informatique Honor - IFT4055

Section A
Professor Louis Salvail

Université de Montréal
Summer 2024
September 1, 2024

Abstract In various scientific fields, the exponential growth in data volume and resolution, driven by technological advancements, presents a formidable challenge for probabilistic data analysis. Traditional computational methods become impractical as some simulations, for example in cosmology, require an exorbitant 100 million CPU hours to evaluate a single likelihood or posterior. Because of this, scientists turn to modeling accurate approximations of their problems. But this creates a "chicken or egg" dilemma: training models to approximate these computationally intensive oracles necessitates prior evaluations of the oracle to generate training samples. There is a need for efficient and intelligent sampling.

Addressing this problem is hard; we propose an innovative approach utilizing a Gaussian Process (GP) as a surrogate model, which provides an uncertainty on approximations. To efficiently train the GP, we employ a Generative Flow Network (GFN) for acquiring training samples. The inputs to the GFN are preprocessed using a Set Transformer, enhancing the model's capacity to identify and leverage patterns within the data. By training the GFN on inexpensive-to-sample problems of the same class, it learns underlying patterns, structures and symmetries in the data, thereby optimizing the sampling process for the GP. This methodology significantly reduces the computational burden, enabling more feasible and efficient analysis of high-resolution and high-dimensional data across various scientific disciplines.

Contents

1. Terminology

Oracle The oracle is the object (function or distribution) that we are trying to emulate/ approximate. In our case, it is a distribution that is difficult to sample from due to its computational complexity.

Surrogate network The surrogate network is a model that approximates the oracle. In our case, it is immensly easier to sample-from than the oracle. Through training, the approximation should become more and more accurate.

Acquisition model The acquisition model is a model that decides where to sample next. It uses the surrogate network to estimate the uncertainty in the model and decides where to sample the oracle next to reduce this uncertainty. It is akin to the concept of acquisition functions in Bayesian Optimization.

2. Surrogate Network

When trying to approximate a complex problem, the first step should be to determine which surrogate network to use. In our case, a Gaussian Process (GP) was chosen as a surrogate network. The GP is a powerful tool for approximating complex functions and distributions. It provides an uncertainty estimate on its predictions, which is crucial for applications where accuracy is important.

In general, GPs are used in the context of Bayesian Optimization (BO) to model an unknown function. The goal of BO itself is to optimize that unknown function. In other words, BO is regularly used to optimize parameters of a process in order to maximize its output. In BO, along with the GP, is an acquisition function used to decide where the unknown function should be sampled next to provide new training data for the GP. The acquisition function uses the uncertainty estimate provided by the GP to decide where to sample next. The GP is then fitted with the new point and the loop repeats until the problem is optimized (hoping for a global solution).

Hence, GPs take a dataset as an input and output a distribution over the function that generated the dataset. This distribution is represented by a mean and a variance, which can be used to make predictions and provide uncertainty.

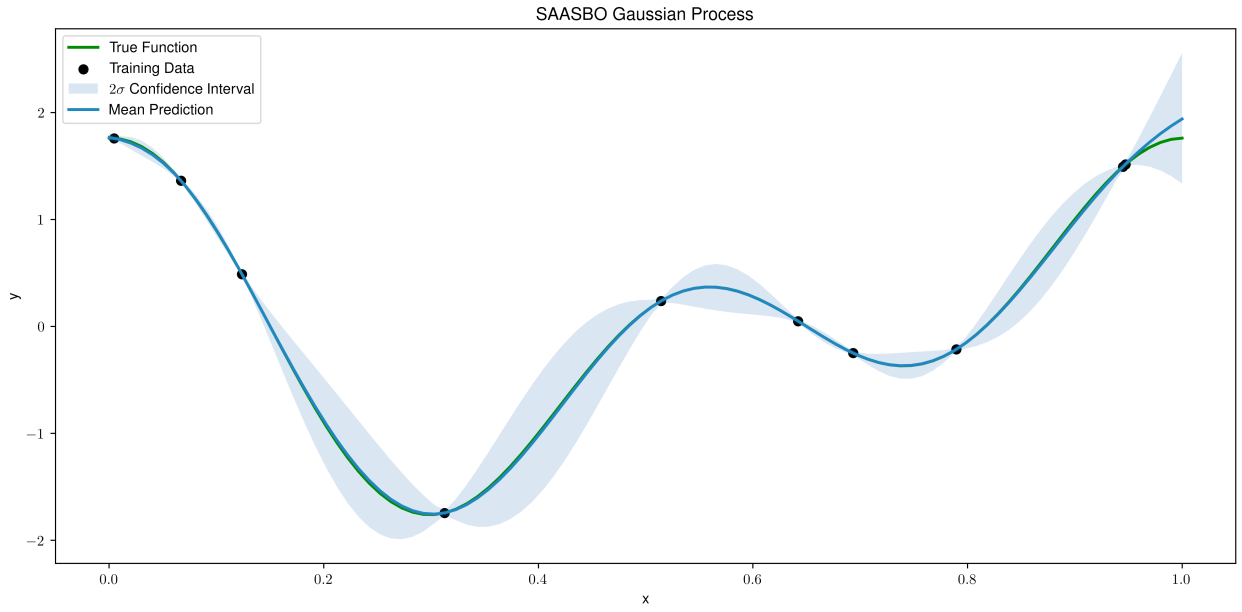

Figure 1: A simple example of a Gaussian Process trained on $y = \sin(-2\pi(x - 0.15)) + \sin(-4\pi(x - 0.15))$. The shaded area representing the 2σ uncertainty of the model and the blue line the mean of the model. Although the mean of the model often corresponds to the true function, as expected, its uncertainty grows when training samples are further away from each other.

This is somewhat similar to our problem, where we are trying to approximate a complex distribution. However, our work aims at training the GP to model the entire distribution and not only finding its minimum/maximum.

Moreover, this training process is incredibly important, as in order to sample efficiently, we need to first see how the surrogate network is doing before proposing new samples that will actually improve its performance. This is like a student taking a test, seeing the results, and then studying the material they did poorly on. A GFlowNet will fulfill this role.

Many other candidates were considered and researched to fill the surrogate network role, such as Probabilistic Neural Networks, Variational Inference networks and GFlowNets. However, GPs were chosen for their simplicity, relatively inexpensive-to-sample and train nature and their ability to provide uncertainty estimates. This last property proves to be crucial for our problem.

In particular, the GP used was based on the "High-Dimensional Bayesian Optimization with Sparse Axis-Aligned Subspaces" (SAASBO) paper [1] and implemented using [2]. This paper introduces a novel method to train GPs that is more efficient on high-dimensional problems where training samples are limited and expensive to obtain. This makes a lot of sense for our problem, as we aim to minimize the number of samples required to train the GP because of the cost of evaluating the oracle. As for BOTorch, it is an efficient and modular framework built on top of PyTorch for Bayesian Optimization.

3. Acquisition Model

Similar to Bayesian Optimization (BO), we require an acquisition model to determine the next location for sampling the oracle, which will then provide the training point for the Gaussian Process (GP). Typical BO uses predetermined acquisition functions such as Expected Improvement (EI), Probability of Improvement (PI) or Upper Confidence Bound (UCB). However, these functions are not well suited for our problem, as they are designed to optimize a function and not to sample from a distribution. Moreover, such acquisition functions would not allow leveraging the structure and patterns of the data to optimize the sampling process.

Let's imagine a simple example: we have class of problems in two dimensions where each instance of the problem consists of a gaussian mixture of two gaussians that are always symmetrically placed around the origin. The goal is to sample from the oracle to train the GP. The acquisition model should be able to leverage the fact that the problem is symmetric and that the oracle is a gaussian mixture to propose samples that are likely to be informative. In other words, when the acquisition model finds the first gaussian, it should not get stuck in that mode, be able to infer the presence of the second gaussian and propose samples that are likely to be close to it. All of this because the model has learned the properties of that class of problems. This is meta-learning [3]–[5].

Just like the surrogate network, many model types were considered for the acquisition model. Reinforcement Learning (RL) was chosen as the best approach because RL allows the model to learn the best strategy for the sampling process by interacting with the environment (the oracle) over the course of many iterations. This enables our training strategy presented further down.

RL learns thanks to a reward function that tells the model how well it is doing. Contrary to other kinds of networks which use a loss function, rewards do not need to be differentiable which allows us to use statistical analysis tools such as PQMass [6] in their design.

RL also allows for fast inference as, given a state, the network can propose a sample without having to compute a reward. This is interesting as when training with inexpensive-to-sample problems, we can design a powerful reward that would be too expensive to compute in inference on expensive problems [7].

3.1. GFlowNet

RL is a vast domain which covers multiple learning strategies. Once again, many were considered such as Deep Q-Learning, PPO, actor-critic, etc. One of the challenges was making RL work in a continuous action space, where actions are not constrained to a limited set of possibilities (such as the set of valid moves in a game). In our situation, the points sampled from the oracle can be anywhere in the space. Although classical RL has ways to make this work, we settled on a different solution.

GFlowNets were chosen for their ability to sample proportionally to the reward. In other words, GFNs do not simply maximize the reward; they are samplers that will sample better points (according to the reward) more often, and points that are equally good have the same probability of being sampled [8]. This is a great property for our problem as it will make sure that the model samples the most informative points first. Moreover, here are a few advantages of GFlowNets and their sampling nature [9]:

- By sampling proportionally to the reward, GFlowNets explore a wider variety of high-reward solutions rather than prematurely converging on suboptimal solutions and getting stuck in a local maximum. This helps in understanding the entire landscape of potential solutions, not just the highest peak.
- Solutions found by considering a broader range of high-reward paths are often more robust and generalize better to new, unseen data or scenarios.
- By sampling according to the reward distribution, the GFlowNet can focus computational resources on more promising areas of the solution space, leading to more efficient learning.
- In large or continuous action spaces, directly finding the maximum reward can be computationally infeasible. Proportional sampling provides a tractable way to explore these spaces effectively.
- In applications where outcomes are stochastic, the optimal strategy often involves probabilistically favouring high-reward actions rather than deterministically choosing a single action.

GFlowNets are based on flow networks and are comprised of a forward model and a backward model. The forward model, or forward policy, is a policy that decides the probability of transitioning from one state to another. It is denoted by $P_F(s' \mid s)$ where s is the current state and s' is the next state after taking an action. Conversely, the backward model, or backward policy, is a policy that determines the probability of reversing the transition, i.e., moving from state s' back to state s. The backward policy is denoted by $P_B(s \mid s')$. To learn, these models require a training objective.

3.1.a. Training Objectives

In recent years, several training objectives were devised to train GFlowNets. These are, in a way, a set of rules or constraints that the model follows such that sampling of the model is proportional to the reward. The most popular ones are [9]:

- Flow-matching objective
- Detailed balance objective
- Trajectory balance objective
- Subtrajectory balance objective

The **flow-matching objective** is the simplest, as it ensures that the total flow into each non-terminal state equals the total flow out of it. In equation form, this gives

$$\sum_{s'} F(s' \to s) = \sum_{s''} F(s \to s'')$$

which can be converted into a loss function to train the model. This enforces consistency in the way states are visited during the generation process.

The **detailed balance objective** indirectly uses flow matching by making the output of the model a softmax of the possible actions in each state. This is enforced by the following constraint:

$$F(s)P_F(s'|s) = P_B(s|s')F(s')$$

which can be useful in cases where the number parents for each state is large, as there is no sum to compute for each state compared to flow matching.

The **trajectory balance objective** is a more complex objective that ensures that the model samples trajectories that are consistent with the reward. A trajectory is a sequence of states that the model visits that starts at s_0 and ends at a terminal state s_n. This is enforced by the following equation:

$$F(s_0)\prod_{t=1}^{n} P_F(s_t|s_{t-1}) = R(s_n)\prod_{t=1}^{n} P_B(s_{t-1}|s_t)$$

$$R(s_n) = \frac{F(s_0)\prod_{t=1}^{n} P_F(s_t|s_{t-1})}{\prod_{t=1}^{n} P_B(s_{t-1}|s_t)}$$

According to [10], this training objective can lead to faster convergence of the model.

Finally, the **subtrajectory balance objective** is a variant of trajectory balance that, as the name implies, balance the flow over subtrajectories instead of complete trajectories. Accorvding to [11], this can accelerate convergence and provide better performance in sparse reward spaces.

In practice, our GFlowNet implementation is inspired by the examples presented in the TorchGFN library [12] and is based on the trajectory balance training objective. This training objective well known, relatively easy to implement and produces good results.

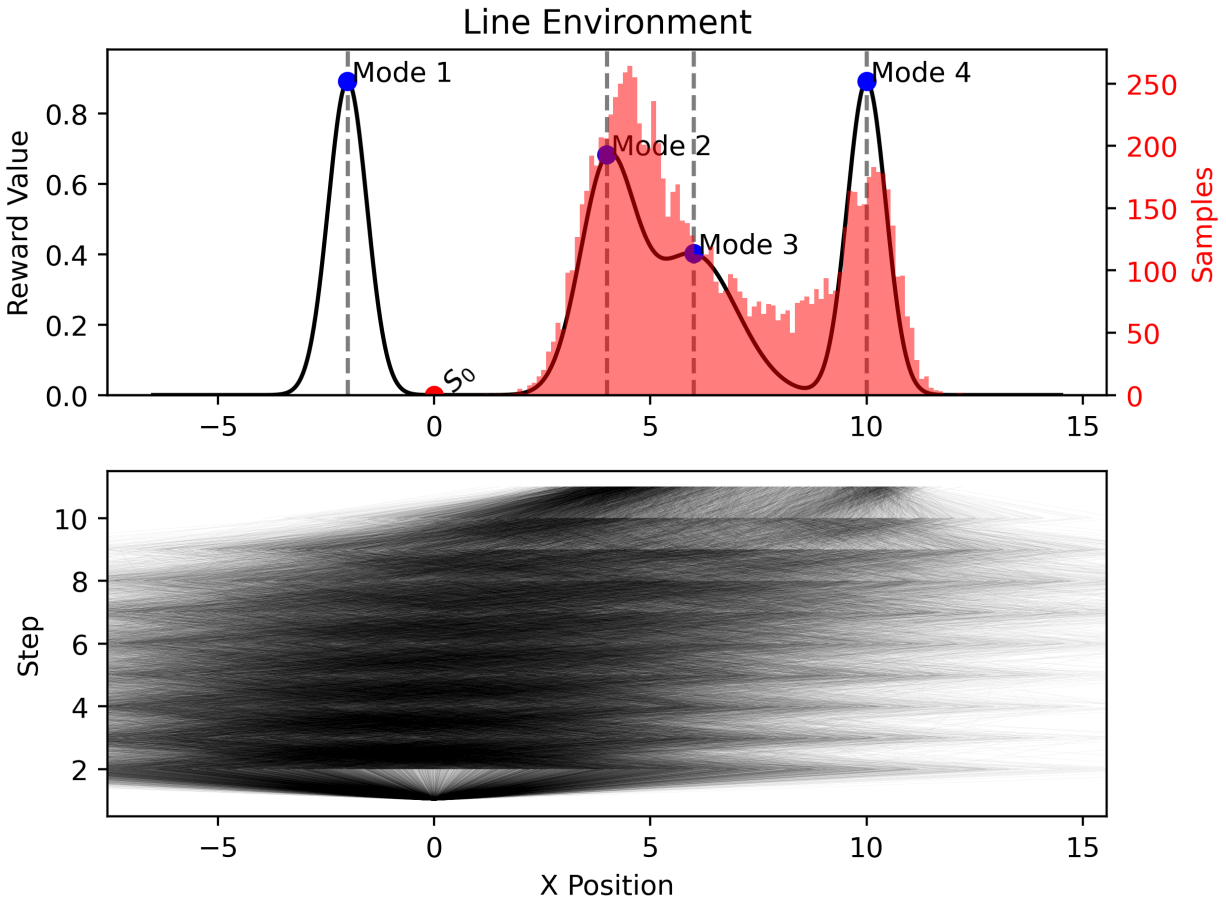

Figure 2: A simple example of a GFlowNet trained on a gaussian mixture reward using a trajectory balance training objective [12]. Each action taken by the model represents a relative move on the x axis (position) and s_0 is the initial x position. The upper section of the figure shows in red what the network learned to sample and, in black, the gaussian mixture reward. The lower section of the figure shows the exploration/exploitation process of the network at each step. In this case, the GFlowNet isn't able to learn the leftmost mode of the gaussian mixture.

3.2. Designing the Reward Function

The reward function is one of the most important parts of the acquisition model. In essence, it is dictating what the network learns when training. In our case, there are a few things that must be conveyed to the model:

- We want to prevent sampling the same point twice
  - This is important because the oracle is expensive to sample from. Sampling the same point twice is a waste of resources.
- We want to minimize the number of steps taken for training
  - This is important because the oracle is expensive to sample from. At each training step, we sample the oracle to get new training points for the surrogate network. The

fewer steps taken, the cheaper the training process. This will also teach the network that it is important to sample the most informative points first.

- We want to make sure the surrogate network improves thanks to the new training point we feed it each step.
  - This is important because the goal of the acquisition model is to provide the surrogate network with the most informative points to improve its approximation of the oracle. If the points provided do not improve the "score" of the surrogate, then the acquisition model is not doing its job.
- We want to make sure that the surrogate network learns to approximate the oracle accurately.
  - This is important because the goal of the surrogate network is to approximate the oracle. If the surrogate network is not accurate, then the acquisition model is not proposing points that are significant to the oracle; it is not helping the surrogate learn the "signature" of the oracle.

Conveying these three points to the model mathematically is not an easy task. Furthermore, each term in the reward function has a different scale, we therefore need normalize them. Using the trajectory balance training objective, the reward is usually the log probability of the action taken by the model. We will therefore rescale the terms of our reward such that it is between 0 and 1 before taking its log. Here is what we came up with:

First, to minimize the number of steps taken for training, we use include the following term in the reward function: $1 - \left(\frac{\text{n\_samples}}{\text{max\_samples}}\right)$ Where "n_samples" is the total number of samples taken during training. This will make sure that each step taken by the model is penalized. The model will want to take the fewest samples possible to maximize its reward. We use a hyperparameter representing the maximal number of samples allowed in training to normalize "n_samples". We believe this is reasonable as it allows the user to control the maximal cost of their training.

To make sure the surrogate network improves thanks to the new training point we feed it each step, we first need a way to compute how well the model is doing. An intuitive way to do that is by computing the entropy of the surrogate network before and after the new training point is fed to it. If the entropy decreases, then the model has improved. If the entropy increases, then the model has not improved. In practice, we use the Kullback-Leibler divergence (KL divergence) in our reward term to quantify the entropy of the surrogate network. The KL divergence is a measure of how much a probability distribution is different from another reference probability distribution. In our case, the first probability distribution is composed of the posterior samples of a surrogate network, and the second is the dataset after the new training point is added to it. This means we are computing how different the predictions of the surrogate network are from what it is trying to learn.

Hence, we include the following term C in the reward function:

$$A = D_{KL}(\text{gaussian\_process}_{t-1}, \text{new\_dataset})$$
$$B = D_{KL}(\text{gaussian\_process}_{t}, \text{new\_dataset})$$
$$C = \text{sigmoid}(A - B) = \frac{1}{1 + \exp(-(A - B))}$$

Where t represents a specific training step. We normalize each KL divergence using the sigmoid function element-wise which maps the difference $A - B$ to a codomain of $]0, 1[$. The idea is that if the GP is improving, then $A > B$ because at step $t - 1$ (in A), the GP had not been trained on the new data point and the KL divergence is bigger.

Finally, to make sure that the surrogate network learns to approximate the oracle accurately, we use PQMass [6], which is non-differentiable, to quantify the probability that the predictions of the GP and the test dataset come from the same distribution. We use the same tool to quantify how well the train and test dataset distributions match.

During training, inexpensively sampled problems of the same class are used. Hence, although the train dataset needs to be kept minimal such that the network learns to sample efficiently, it is possible to use a large test dataset. This means that, given a tightly uniformly sampled test dataset, we can accurately compute how the predictions of the GP differ from the oracle with PQMass.

The PQMass package features two metrics, the χ^2 and the p-value. The former has a few interesting properties:

> For the χ^2 metric, given your two sets of samples, if they come from the same distribution, the histogram of your χ^2 values should follow the χ^2 distribution. The peak of this distribution will be at $\text{DoF} - 2$, and the standard deviation will be $\sqrt{2 * \text{DoF}}$. If your histogram shifts to the right of the expected χ^2 distribution, it suggests that the samples are out of distribution. Conversely, if the histogram shifts to the left, it indicates potential duplication or memorization (particularly relevant for generative models).
>
> — From the PQMass GitHub repository [13]

Note. In this last quote, DoF stands for the degrees of freedom of the χ^2 distribution.

We compute the PQMass χ^2 values x. We then use the mean and variance of x in our reward once they are normalized. According to the mentioned properties, we expect the mean of x to be close to $\text{DoF} - 2$ and the standard deviation to be close to $\sqrt{2 * \text{DoF}}$. Hence, we use the following to normalize the mean of x:

$$\frac{1}{0.5 * |\text{mean}(x) - (\text{DoF} - 2)| + 1}$$

and the following to normalize the standard deviation of x:

$$\frac{1}{0.5 * \left|\text{std}(x) - \sqrt{2 * \text{DoF}}\right| + 1}$$

These two equations have a maximum of 1 when the mean and variance of x are equal to their respective expected values, and a minimum of 0 as they get infinitely far from that expected value. They are then multiplied by a hyperparameter to change the rate at which the function gets from 1 to 0.

We take the average of these two terms to get the final reward term. This final reward term is computed between the GP's posterior samples and the test dataset, and between the train dataset and the test dataset.

With all these terms combined, we have a reward function that should allow the GFlowNet to provide meaningful points to the surrogate network. All of these terms have a codomain between 0 and 1 and are averaged together. To prevent sampling the same point twice, if the new point was present in the training dataset, we divide that averaged reward by some constant. This will make sure that the model is penalized for sampling a point that has already been sampled.

Finally, we take the log of the reward and, therefore, respect the constraints of the trajectory balance training objective. In other words, our reward function some kind of "log probability".

Overall, the reward function allows the model quantify how well it is doing and to learn to sample the most informative points first, efficiently and accurately.

3.3. Designing the State and Action Space

Having a good reward function is not enough. The model must also be able to understand the state of the problem it is in. It can be challenging to determine what the state should be and how it should be encoded into the network.

Here, we use state in the broad sense of the term. There exist a "state" variable in the GFlowNet that is updated at each step of the trajectory. But here, we mean all the information that is given to the model at each step and that is not necessarily part of the trajectory state.

Ideally, to help with training, we want the state to be as informative as possible and to contain all the information the model needs to make a decision. In our case, we determined that the state should contain the following information.

To compute the reward:

- The list of points used to train the GP
- The list of oracle evaluations corresponding to the sampled points
- The posterior samples of the GP, at step t and $t-1$, which represent a distribution over the approximations of the oracle
  - This $m \times n$ tensor is obtained by sampling the GP on a large range of points $[x_0, ..., x_n]$. This is extremely fast to obtain and the output is represented by a two-dimensional tensor, viewed as a matrix, with each column $\{C_i \mid i \in [0, n]\}$ containing m realizations of the GP on input x_i. This means we have discretized the GP's output distribution into n points which have their mean and variance computed from the m realizations.

In the trajectory state:

- The new point that will be evaluated by the oracle and added to the training dataset
- t a time step
  - One of the important restrictions of a GFlowNet is that there must not be cycles in the state space. Adding an entry t to the state prevents loops because the network cannot go back in time [12].

The action space, being continuous, will need to be represented by a probability distribution, or policy distribution, obtained from the forward model of our GFlowNet. At each step, the action can then be sampled from that distribution and applied to update the state. In our case, the action is a new point to sample from the oracle.

We use a modified version of that policy distribution called the exploration distribution. This distribution is obtained by adding a small amount of noise, determined by a schedule, to the variance of the policy distribution. This is done to make sure the model does not get stuck in a local minimum and to increase exploration at the beginning of training. This is a common practice in Reinforcement Learning.

One problem remains: how can we encode the list of points and the output of the GP into the state? The solution is to use a Set Transformer.

4. Set Transformers

Because we cannot directly feed the list of points and the output of the GP into the state of the GFlowNet, we need to use a Set Transformer to process the input and use its output embeddings as the state. Set Transformers are a type of transformer that is particularly useful to process inputs when the order of the data does not matter, as is the case with our list of points and the output of the GP. Using a set transformer, it is possible to make sure that every permutation of a list of numbers is treated the same way.

We use a model architecture based on transformers which use an encoder and a decoder relying on attention mechanisms. This approach allows the model to be more efficient by reducing "the computation time of self-attention from quadratic to linear in the number of elements in the set" [14]. According to testing by [14], this method achieves state-of-the-art performance when compared to other alternatives in 2019.

4.1. Semi-Supervised Training

We pre-train the set transformer on a dataset of inexpensive-to-sample problems and use it to process the input to the GFlowNet. This allows the GFN model to learn the underlying patterns, structures and symmetries in the data without being hindered by the order of the points.

In practice, we wish to avoid labelling the data ourselves as this can be achieved by using a semi-supervised learning approach. We use a function that generates lists of points; one generates pairs of lists that are permutations of one another and the other generates pairs of lists that are completely different and not permutations of one another. The model is then trained to differentiate between these two by using the embeddings of the set transformer. That is, at each training step, we compute the output of the model on these 4 lists. This output is the embedding that we wish represents the lists. We then compute the L2 norm between the embeddings. The goal is to have the distance between the embeddings of the permutations be 0, and infinity for the distance between the embeddings of the non-permutations.

This kind of contrastive learning is what we mean by semi-supervised learning. The model is not given the labels of the data, but it is given a way to differentiate between the two classes of data.

To do so, given x_1 and y_1 the pair of embeddings that are permutations of one another and x_2 and y_2 the pair of embeddings that are not, we compute the following distances:

$$\begin{aligned} a &= \text{dist}(x_1, y_1), \quad b = \text{dist}(x_2, y_2) \\ c &= \text{dist}(x_1, y_2), \quad d = \text{dist}(x_1, x_2) \\ e &= \text{dist}(x_2, y_1), \quad f = \text{dist}(y_1, y_2) \end{aligned}$$

where:

$$\text{dist}(x, y) = \|x - y\|_2 = \sqrt{\sum_{i=1}^{n} |x_i - y_i|^2}$$

Here, a and b are to be minimized, whereas c, d, e and f are to be maximized. We use these to create a loss function that is separated into two parts:

$$l_a = a + b$$
$$l_b = \text{scaling}(c + d + e + f)$$

because dist has a codomain restricted to $[0, \infty[$, we do not rescale l_a since we try to minimize it and that minimum is bounded to 0. However, we need to rescale l_b to make sure that l_b does not completely overshadow l_a. We use the function scaling for that:

$$\text{scaling}(x) = \frac{-x}{1 + |x|} + 1$$

This function scales the L2Norm between 0 and 1: 1 when the distance is 0 and 0 as the distance approaches infinity. We then use the sum of l_a and l_b as the loss.

Overall, when used in our set transformer, this loss allows us to generate embeddings that are able to differentiate between permutations and non-permutations of the input lists. This is exactly what is needed to train a model that does not care about the order of the points.

5. Training Architecture and Process

Following is a short overview of how the models are used together.

First, we train the set transformer on a dataset of inexpensive-to-sample as explained above. Then, the GFlowNet may be trained. The surrogate model is trained per-problem. Hence, it will be trained and used multiple times when training the GFN as the latter uses it in computing its reward.

When learning the hard-to-sample oracle, we use the already trained GFN and set transformer to train the GP.

Following is a diagram of the training process:

Figure 3: Architecture of the project. The dotted lines and the bloc inside the GFlowNet represent a path only taken when training it. When using the trained GFN on the hard-to-sample problem, the solid line path is used

Although extremely high-level, Figure 3 gives a good idea of how the models interact with one another. The process outlined in the diagram is repeated until the gaussian process produces an accurate approximation of the oracle or until the maximum number of samples allowed is reached. This loop, when training the GFN, is repeated on many problems of the same class to make sure the GFN learns the patterns of the oracle(s).

6. Results

Now that the architecture of the project was presented, we can discuss the implementation and results. The project is still in its early stages, and challenges remain to be addressed. However, the initial results are promising.

6.1. Set Transformer

First, we present the results of training the set transformer:

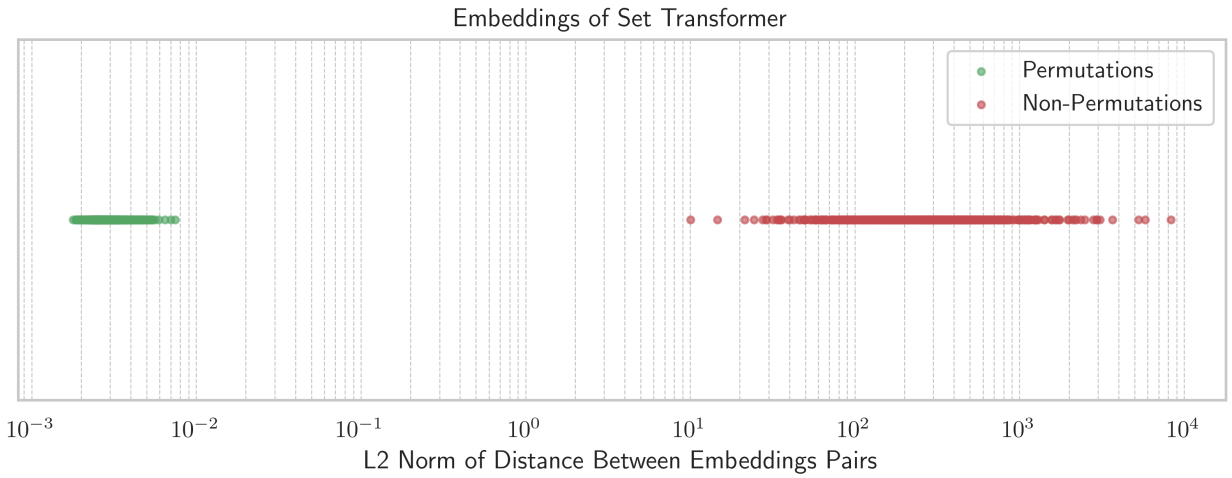

Figure 4: This figure shows the distance between the embeddings of the set transformer. The set transformer was trained to generate embeddings of size 128 on lists of between 2^2 and 2^{12} points, each in 10 dimensions. 1000 samples were used for both the permutations and the non-permutations to generate this graph. The distance between the embeddings of permutations has a mean of 3.026×10^{-3} and a variance of 5.225×10^{-7}, whereas the distance between the embeddings of non-permutations has a mean of 3.783×10^{2} and a variance of 2.373×10^{5}.

Clearly, in Figure 4, it is possible to see that the set transformer is able to differentiate between permutations and non-permutations. The means show that a safe threshold could be set at around 1 to differentiate between embeddings of permutations and non-permutations. Furthermore, another good sign is that the variance of the permuted embeddings is much smaller than the variance of the non-permuted. This makes sense as the non-permutation pairs were generated randomly and therefore could have been more or less similar to one another.

During the final steps of the implementation, the reward of the acquisition GFlowNet was updated such that it does no require the embeddings of the set transformer. In fact, statistical tools such as the KL divergence and PQMass were used in the reward, and such tools do not require a set representation. Set Transformers were therefore not used in the final implementation of the project, but are still included in this report as they were part of the initial design and could be reintroduced in the future.

6.2. Surrogate Gaussian Process

Then, we present the results of the surrogate Gaussian Process (GP):

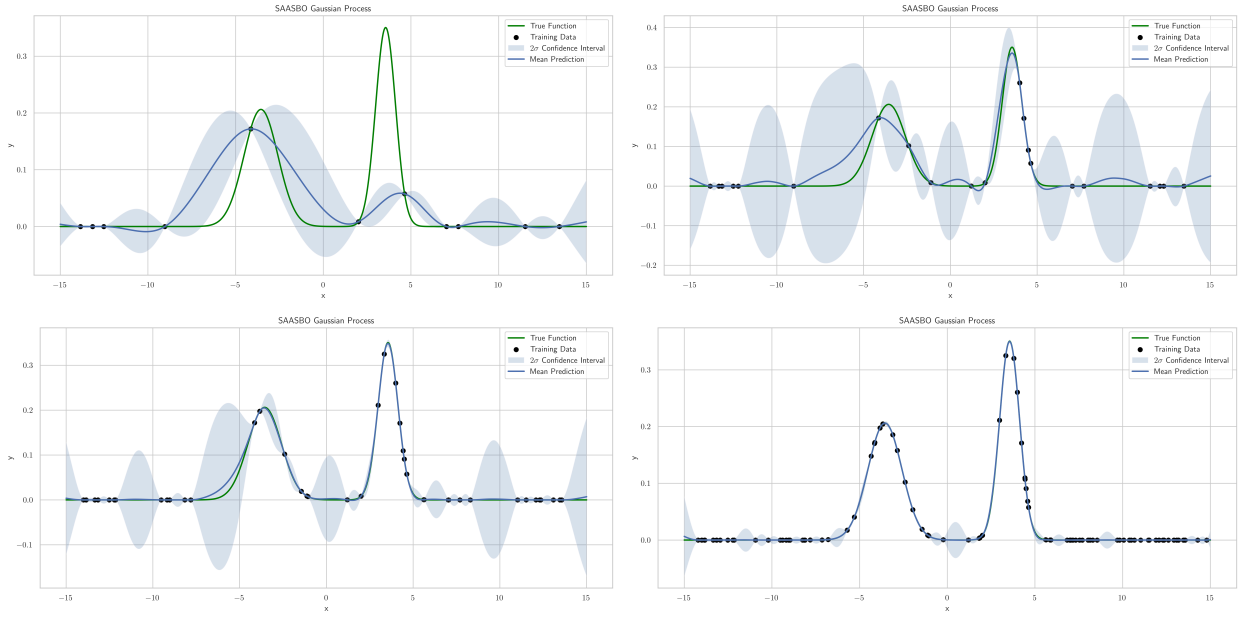

Figure 5: Iterative training of a Gaussian Process. Each step, a training point is randomly selected on the domain $[-15, 15]$ of the function and added to the training dataset along with its corresponding oracle evaluation.

As points are added to the training dataset (as seen in Figure 5), the GP is able to approximate the function more accurately. The uncertainty of the GP decreases as more points are added to the training dataset, but even where the uncertainty is high, the true function is almost always within the 2σ uncertainty range and close to the mean of the predictions. This shows how the GP is able to accurately approximate an oracle function.

6.3. Acquisition GFlowNet

Finally, we present the results of the acquisition GFlowNet (GFN).

First we show a visualization of the training process of the GFN on a simple gaussian mixture problem:

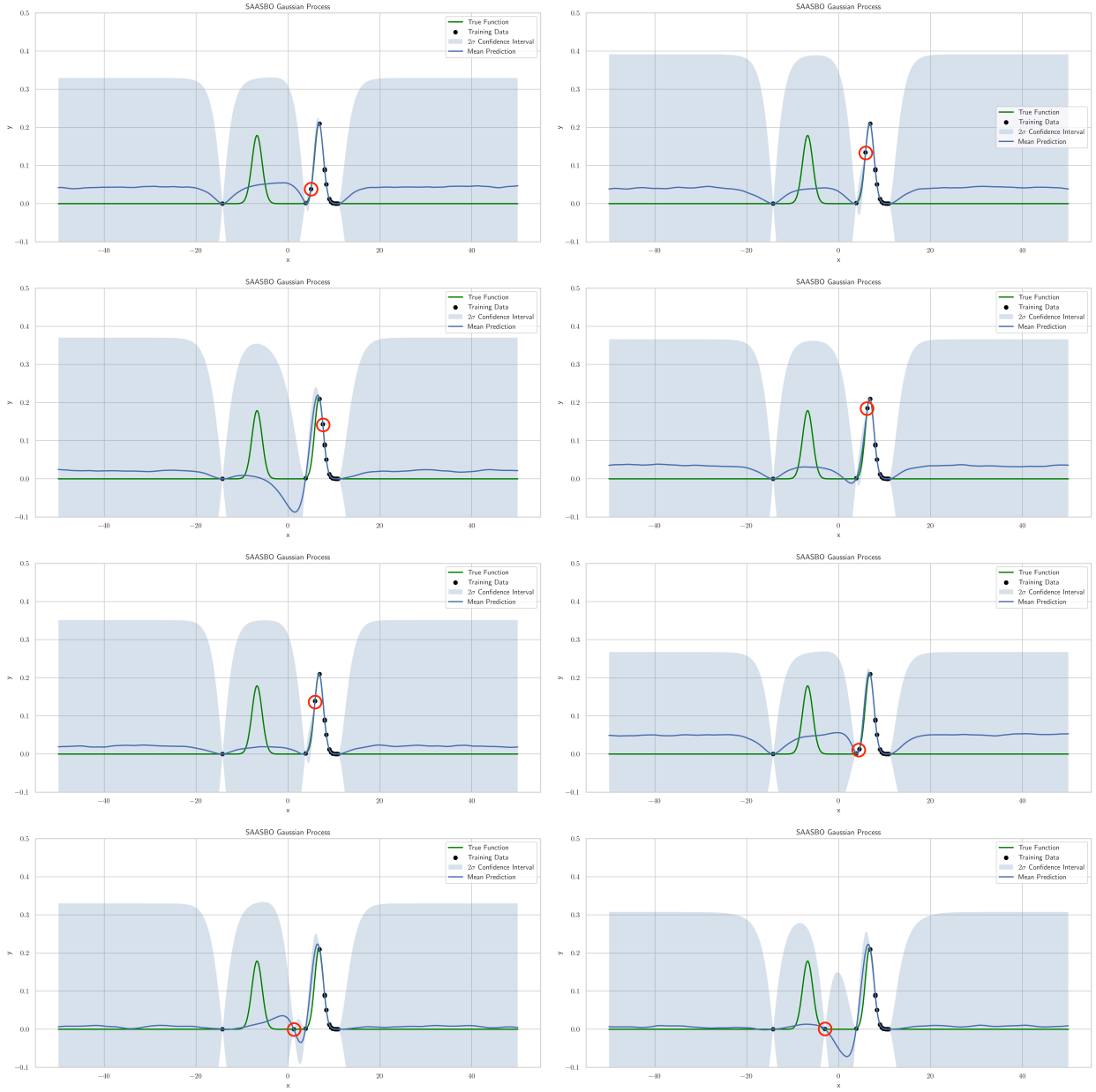

Figure 6: Visualization of the acquisition GFlowNet's (GFN's) training. Each plot represents the Gaussian Process distribution computed with the new point tested by the GFN at a specific training step. Each plot is taken at an interval of 25 steps. At each step, the GFN tries to add a new point and computes its reward before removing the point. This allows the network to learn which points are the most informative and how to sample them. Circled in red is the point tested by the GFN.

Figure 6 also helps visualize the training of the GFN. As the reward got smaller, the actions taken by the model got bigger. This is logical as the model is trying to find the most informative points first. If it cannot find such a point, it will try to explore more of the space. This is visible in the last plot where the model starts to explore the second mode of the gaussian mixture.

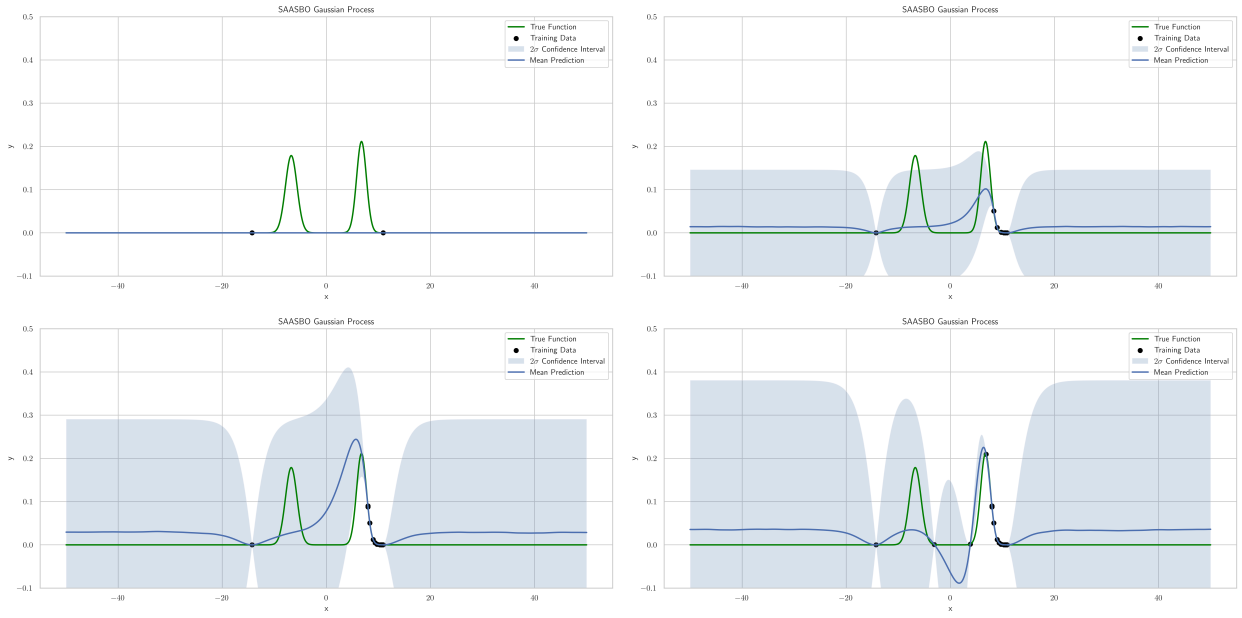

Figure 7: Usage of the acquisition GFlowNet (GFN) to train a Gaussian Process surrogate on a gaussian mixture distribution. This GFN was trained on 5 gaussian mixture distributions featuring each two gaussians equaly separated from the origin. Each plot shows different training steps of the GP.

Figure 7 shows the usage of the GFN to train a Gaussian Process. The most interesting and important fact in this result is that, even with a really short training on 5 prior examples of gaussian mixtures, the acquisition model is able to learn to provide informative points and does not simply converge to 0 or diverge to infinity. This is crucial as it means that with more work, time and training, such a model architecture could be used to train on a hard-to-sample problems.

7. Discussion

The proposed approach addresses the computational challenges posed by the rapid increase in data volume and complexity across various scientific fields. Traditional methods for evaluating complex oracles, such as the ones used in cosmology, are often impractical due to the high computational costs involved. By employing a Gaussian Process (GP) as a surrogate model, we gain the ability to approximate these computationally intensive oracles while keeping the computational costs manageable. The GP's inherent ability to provide uncertainty estimates is especially valuable in scientific research, where simply achieving "good results" is not sufficient in search of the truth.

A key component of our approach is the use of a Generative Flow Network (GFN) as the acquisition model. Unlike traditional acquisition functions used in Bayesian Optimization, which are designed for function optimization rather than distribution sampling, the GFN uses reinforcement learning to learn the optimal sampling strategy. This is essential for efficiently training the GP, as it enables the model to more effectively explore the input space and identify informative samples that enhance the surrogate's accuracy. In other words, when trained on a class of problems, the GFN can learn, for example, symmetries in the data and use that information to immediately start sampling from a second mode once the first has been found. Incorporating Set Transformers to pre-process inputs could, in the future, boost the GFN's ability to detect and exploit patterns within the data, further optimizing the sampling process.

Our choice of training objectives for the GFN, specifically the trajectory balance objective, makes for a robust sampling process. By ensuring that the samples are proportional to the reward, and using an exploration policy, the model is pushed to explore a broader range of high-reward solutions rather than prematurely converging on suboptimal local maxima. This is particularly advantageous in high-dimensional or continuous action spaces, where directly finding the global maximum can be very challenging. Moreover, the sampling strategy ensures that computational resources are focused on the most promising areas of the solution space, leading to more efficient learning and better generalization to new data.

The design of the reward function is critical to the success of our approach. It is specifically thought-out to prevent redundant sampling, minimize the number of training steps, and ensure that each new sample improves the surrogate model's performance. By incorporating metrics such as the KL divergence and PQMass, we provide a quantitative measure of how well the GP approximates the oracle. These metrics help guide the acquisition model towards proposing samples that are not only new and diverse, but also highly informative. This approach ensures that the surrogate network continually refines its approximation of the oracle, ultimately leading to a more accurate model with fewer computational resources.

One of the limitations of this project is the computational expense associated with training the GFN. The training process involves repeatedly training the GP model at each

step of the GFN training, which itself is nested within a loop. In each iteration of this outer loop, we must train the GFN, evaluate it, obtain a new data point and train the GP with that new point. This iterative procedure is computationally intensive. However, it is worth noting that while this approach requires significant initial investment in terms of computational resources, it has the potential to reduce costs in the long term by spreading these expenses across future evaluations, making it a more efficient alternative over time. Future works should focus on optimizing the training process to reduce computational overhead and determine a threshold at which our proposed approach becomes more efficient than traditional methods. Given more time, one of the first things to do would be to tweak the GFN and let it train on a few hundred examples of the same class of problems.

Moreover, GFNs are still in their infancy and not as well understood as other types of models. As time passes and they become more common, researched and documented, it is possible that new training objectives, or entirely new training techniques, will be developed that could improve the performance of the model. This is an exciting prospect, as it could lead to even more efficient and effective models.

Overall, our methodology demonstrates significant potential in the field of probabilistic data analysis by combining a GP surrogate model with a novel GFN-based acquisition strategy. The ability to efficiently train the surrogate network using inexpensive-to-sample problems of the same class ensures that the computational costs remain manageable, even as the complexity of the problems increases. This innovative approach opens up new possibilities for high-dimensional and high-resolution data analysis across a wide range of scientific disciplines, offering a scalable solution to a problem that has long impeded progress in fields reliant on large-scale simulations.

8. Conclusion

In summary, this study introduces a new framework for approximating complex oracles by integrating Gaussian Processes (GPs) and Generative Flow Networks (GFNs). Our approach addresses the limitations of traditional sampling methods, enabling efficient and accurate surrogate modelling for high-dimensional, computationally demanding problems. The use of GFNs as an acquisition model provides a versatile and robust tool for optimizing the sampling process, guided by a custom-designed reward function that balances exploration with exploitation.

The results of this work suggest that our methodology significantly reduces the computational burden associated with training surrogate models, making it feasible to analyze complex datasets that were previously impractical to handle. The approach also demonstrates potential for generalizing to new data, which is crucial for advancing scientific research in data-intensive fields.

Looking ahead, several directions could be explored to enhance the performance of the GFN and the overall methodology. One promising approach is to switch from the Trajectory Balance to the Subtrajectory Balance training objective, which may lead to more effective training according to [9]–[11]. Additionally, tweaking the reward function and fine-tuning hyperparameters could further improve model accuracy, training time and efficiency. Another interesting avenue is to find a way to incorporate the embeddings from the Set Transformer into the training of the GFN, potentially adding valuable contextual information to the output. Furthermore, the GFN could be modified to output a list of points where it can accurately predict the oracle's output; these points could be directly used to train the GP without needing additional evaluations of the expensive oracle. These enhancements could collectively contribute to a more efficient and effective optimization framework.

We are excited to see where this research leads and how this project will be used. We are confident that this project could lead to significant advancements in the field of AI.

9. Implementation

All the code for this project is available on the following GitHub repository:

- https://github.com/etiennecollin/ift4055

10. Author's Note

This project has been both a valuable and intense introduction to the fields of AI and research. It has deepened my understanding of the areas I am passionate about, as well as those I may choose not to pursue in my future career. Throughout this experience, I faced many challenges, particularly in adapting to the unique concepts of productivity and progress in research, which are quite different from other fields. Overcoming these challenges taught me to adjust my expectations and develop resilience, ultimately contributing to my growth as a researcher.

11. Contributions

Thank you to Yashar Hezaveh for his guidance and support throughout this project. His experience, insights and encouragement have been invaluable in helping me overcome challenges.

Thank you to Laurence Perreault Levasseur for taking the time to suggest this project to me and for the idea of using the acquisition GFlowNet to suggest points that may be used directly to train the GP, therefore bypassing the need to sample from the oracle.

Thank you to Nicolas Payot for the help debugging and for the extremely interesting papers he sent me over the project. He took the time to enlighten me on the weird kwirks and features of pytorch and it is truly appreciated.

Thank you to Clement Gehring for his time and the help he provided. He helped me understand some of the papers I read and gave me a lot of ideas as to how the models would interact with one another.

Thank you to Félix Guilbert-Savary for being an awesome rubber duck; you helped me a lot in organising my thoughts.

References

[1] D. Eriksson and M. Jankowiak, "High-Dimensional Bayesian Optimization with Sparse Axis-Aligned Subspaces." Accessed: Jul. 09, 2024. [Online]. Available: http://arxiv.org/abs/2103.00349

[2] M. Balandat *et al.*, "BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization." Accessed: Aug. 05, 2024. [Online]. Available: http://arxiv.org/abs/1910.06403

[3] M. Volpp *et al.*, "Meta-Learning Acquisition Functions for Transfer Learning in Bayesian Optimization." Accessed: Jun. 14, 2024. [Online]. Available: http://arxiv.org/abs/1904.02642

[4] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey, "Meta-Learning in Neural Networks: A Survey." Accessed: Jul. 10, 2024. [Online]. Available: http://arxiv.org/abs/2004.05439

[5] A. Maraval, M. Zimmer, A. Grosnit, and H. B. Ammar, "End-to-End Meta-Bayesian Optimisation with Transformer Neural Processes." Accessed: Jun. 11, 2024. [Online]. Available: http://arxiv.org/abs/2305.15930

[6] P. Lemos, S. Sharief, N. Malkin, L. Perreault-Levasseur, and Y. Hezaveh, "PQMass: Probabilistic Assessment of the Quality of Generative Models Using Probability Mass Estimation." Accessed: May 13, 2024. [Online]. Available: http://arxiv.org/abs/2402.04355

[7] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, Second edition, in progress. London, England: The MIT Pess, 2015.

[8] "Are GFlowNets the Future of AI?." Accessed: Aug. 06, 2024. [Online]. Available: https://www.youtube.com/watch?v=o0Ju9NQa5Ko

[9] Y. Bengio, K. Malkin, and M. Jain, "The GFlowNet Tutorial." Accessed: May 20, 2024. [Online]. Available: https://yoshuabengio.org/gflownet-tutorial

[10] N. Malkin, M. Jain, E. Bengio, C. Sun, and Y. Bengio, "Trajectory Balance: Improved Credit Assignment in GFlowNets." Accessed: Aug. 11, 2024. [Online]. Available: http://arxiv.org/abs/2201.13259

[11] K. Madan *et al.*, "Learning GFlowNets from Partial Episodes for Improved Convergence and Stability." Accessed: Aug. 11, 2024. [Online]. Available: http://arxiv.org/abs/2209.12782

[12] S. Lahlou, J. D. Viviano, V. Schmidt, and Y. Bengio, "Torchgfn: A PyTorch GFlowNet Library." Accessed: Aug. 07, 2024. [Online]. Available: http://arxiv.org/abs/2305.14594

[13] Ciela-Institute, "PQMass: Probabilistic Assessment of the Quality of Generative Models Using Probability Mass Estimation." Accessed: Aug. 24, 2024. [Online]. Available: https://github.com/Ciela-Institute/PQM

[14] J. Lee, Y. Lee, J. Kim, A. R. Kosiorek, S. Choi, and Y. W. Teh, "Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks." Accessed: Aug. 12, 2024. [Online]. Available: http://arxiv.org/abs/1810.00825