CSCI 699 - ProbGen

Probabilistic and Generative Models

Willie Neiswanger

Lecture 12 - Predictive UQ, Active Learning, and Bayesian Optimization

Today

Today

Lecture: Using predictive UQ models for active learning and sequential decision making, including Bayesian optimization and optimal experimental design.

Today

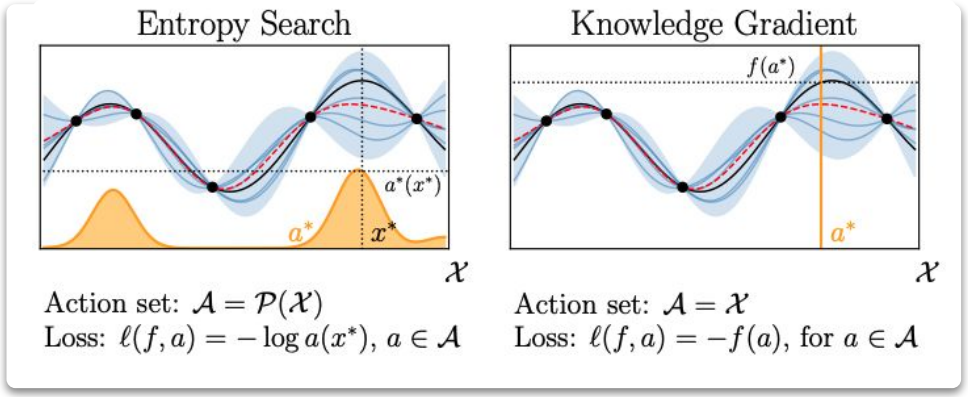
Lecture: Using predictive UQ models for active learning and sequential decision making, including Bayesian optimization and optimal experimental design.

- Finish: deep uncertainty quantification.

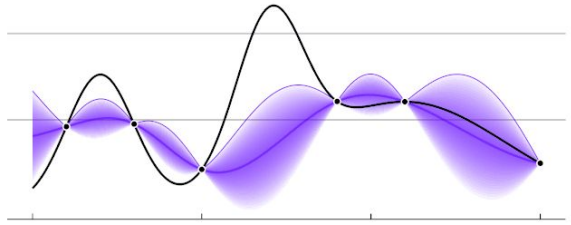
Today

Lecture: Using predictive UQ models for active learning and sequential decision making, including Bayesian optimization and optimal experimental design.

- Finish: deep uncertainty quantification.
- Decision making under uncertainty.
- Active learning.
- Bayesian optimization.
- *E.g.*, UCB, PI, EI, KG, ES, etc.



"Generalizing Bayesian Optimization with Decision-theoretic Entropies", 2022

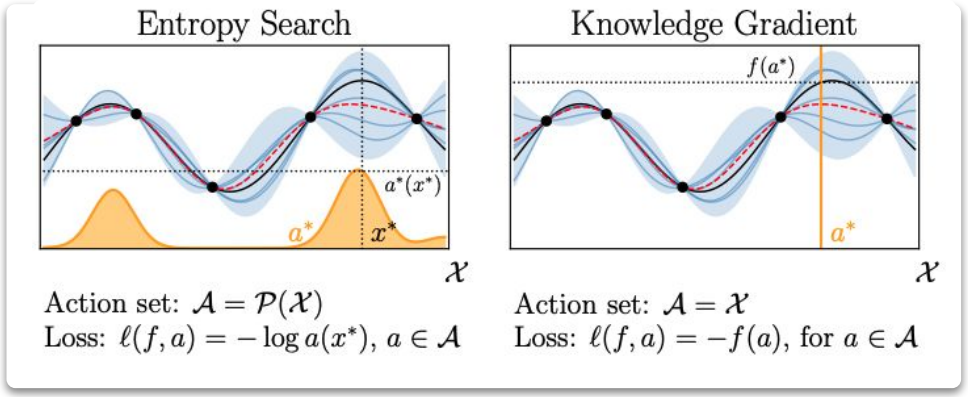


Bayesian Optimization, Wikipedia

Today

Lecture: Using predictive UQ models for active learning and sequential decision making, including Bayesian optimization and optimal experimental design.

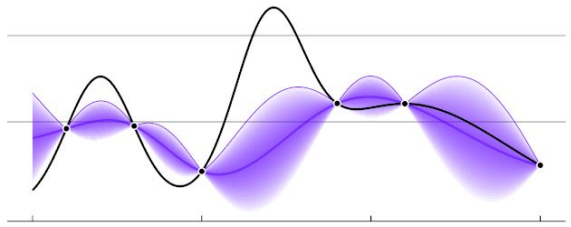
- Finish: deep uncertainty quantification.
- Decision making under uncertainty.
- Active learning.
- Bayesian optimization.
- *E.g.*, UCB, PI, EI, KG, ES, etc.



"Generalizing Bayesian Optimization with Decision-theoretic Entropies", 2022

After:

- Paper presentations from students.



Bayesian Optimization, Wikipedia

Today

After: Next batch of four **paper presentations**.

TextGrad: Automatic "Differentiation" via Text

Mert Yuksekgonul[1*] MERTY@STANFORD.EDU
Federico Bianchi[1*] FEDE@STANFORD.EDU
Joseph Boen[2*] TBOEN@STANFORD.EDU
Sheng Liu[2*] SHENGL@STANFORD.EDU
Zhi Huang[2*] ZHIHUANG@STANFORD.EDU
Carlos Guestrin[1,3] GUESTRIN@STANFORD.EDU
James Zou[1,2,3] JAMESZ@STANFORD.EDU
[1]Department of Computer Science, Stanford University
[2]Department of Biomedical Data Science, Stanford University
[3]Chan Zuckerberg Biohub
Correspondence: merty@stanford.edu and jamesz@stanford.edu

Repository and Tutorials

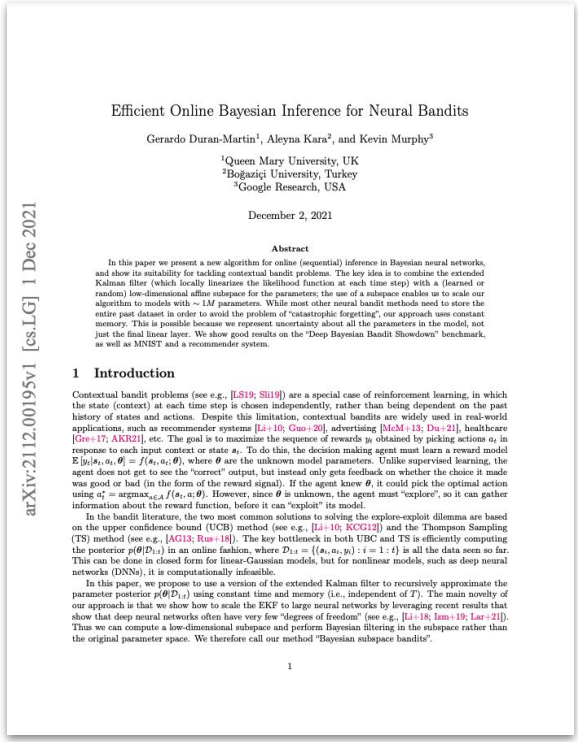
Abstract

AI is undergoing a paradigm shift, with breakthroughs achieved by systems orchestrating multiple large language models (LLMs) and other complex components. As a result, developing principled and automated optimization methods for compound AI systems is one of the most important new challenges. Neural networks faced a similar challenge in its early days until backpropagation and automatic differentiation transformed the field by making optimization turn-key. Inspired by this, we introduce TextGrad, a powerful framework performing automatic "differentiation" via text. TextGrad backpropagates textual feedback provided by LLMs to improve individual components of a compound AI system. In our framework, LLMs provide rich, general, natural language suggestions to optimize variables in computation graphs, ranging from code snippets to molecular structures. TextGrad follows PyTorch's syntax and abstraction and is flexible and easy-to-use. It works out-of-the-box for a variety of tasks, where the users only provide the objective function without tuning components or prompts of the framework. We showcase TextGrad's effectiveness and generality across a diverse range of applications, from question answering and molecule optimization to radiotherapy treatment planning. Without modifying the framework, TextGrad improves the zero-shot accuracy of GPT-4o in Google-Proof Question Answering from 51% to 55%, yields 20% relative performance gain in optimizing LeetCode-Hard coding problem solutions, improves prompts for reasoning, designs new druglike small molecules with desirable *in silico* binding, and designs radiation oncology treatment plans with high specificity. TextGrad lays a foundation to accelerate the development of the next-generation of AI systems.

1 Introduction

There is an emerging paradigm shift in how AI systems are built, owing to the breakthroughs of Large Language Models (LLMs) [1–6]. The new generation of AI applications are increasingly compound systems involving multiple sophisticated components, where each component could be an LLM-based agent, a tool such as a simulator, or web search. For instance, a system of LLMs communicating with symbolic solvers can solve olympiad-level math problems [7]; a system of LLMs using search engines and code interpreter tools performs comparably to human competitive programmers [8] and are solving real-world

*Co-first authors.



Efficient Online Bayesian Inference for Neural Bandits

Gerardo Duran-Martin[1], Aleyna Kara[2], and Kevin Murphy[3]
[1]Queen Mary University, UK
[2]Boğaziçi University, Turkey
[3]Google Research, USA

December 2, 2021

Abstract

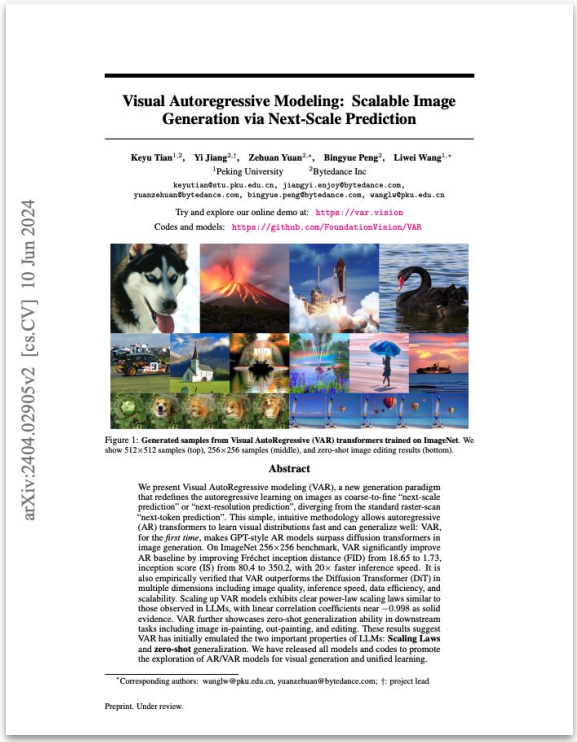
In this paper we present a new algorithm for online (sequential) inference in Bayesian neural networks, and show its suitability for tackling contextual bandit problems. The key idea is to combine the extended Kalman filter (which locally linearizes the likelihood function at each time step) with a (learned or random) low-dimensional affine subspace for the parameters; the use of a subspace enables us to scale our algorithm to models with $\sim 1M$ parameters. While most other neural bandit methods need to store the entire past dataset in order to avoid the problem of "catastrophic forgetting", our approach uses constant memory. This is possible because we represent uncertainty about all the parameters in the model, not just the final linear layer. We show good results on the "Deep Bayesian Bandit Showdown" benchmark, as well as MNIST and a recommender system.

1 Introduction

Contextual bandit problems (see e.g., [LS19; Sli19]) are a special case of reinforcement learning, in which the state (context) at each time step is chosen independently, rather than being dependent on the past history of states and actions. Despite this limitation, contextual bandits are widely used in real-world applications, such as recommender systems [Li+10; Guo+20], advertising [McM+13; Du+21], healthcare [Gre+17; AKR21], etc. The goal is to maximize the sequence of rewards y_t obtained by picking actions a_t in response to each input context or state $\boldsymbol{s}_t$. To do this, the decision making agent must learn a reward model $\mathbb{E}\left[y_t|\boldsymbol{s}_t, a_t, \boldsymbol{\theta}\right] = f(\boldsymbol{s}_t, a_t; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ are the unknown model parameters. Unlike supervised learning, the agent does not get to see the "correct" output, but instead only gets feedback on whether the choice it made was good or bad (in the form of the reward signal). If the agent knew $\boldsymbol{\theta}$, it could pick the optimal action using $a_t^* = \text{argmax}_{a \in \mathcal{A}} f(\boldsymbol{s}_t, a; \boldsymbol{\theta})$. However, since $\boldsymbol{\theta}$ is unknown, the agent must "explore", so it can gather information about the reward function, before it can "exploit" its model.

In the bandit literature, the two most common solutions to solving the explore-exploit dilemma are based on the upper confidence bound (UCB) method (see e.g., [Li+10; KCG12]) and the Thompson Sampling (TS) method (see e.g., [AG13; Rus+18]). The key bottleneck in both UBC and TS is efficiently computing the posterior $p(\boldsymbol{\theta}|\mathcal{D}_{1:t})$ in an online fashion, where $\mathcal{D}_{1:t} = \{(\boldsymbol{s}_i, a_i, y_i) : i = 1 : t\}$ is all the data seen so far. This can be done in closed form for linear-Gaussian models, but for nonlinear models, such as deep neural networks (DNNs), it is computationally infeasible.

In this paper, we propose to use a version of the extended Kalman filter to recursively approximate the parameter posterior $p(\boldsymbol{\theta}|\mathcal{D}_{1:t})$ using constant time and memory (i.e., independent of T). The main novelty of our approach is that we show how to scale the EKF to large neural networks by leveraging recent results that show that deep neural networks often have very few "degrees of freedom" (see e.g., [Li+18; Izm+19; Lar+21]). Thus we can compute a low-dimensional subspace and perform Bayesian filtering in the subspace rather than the original parameter space. We therefore call our method "Bayesian subspace bandits".



Visual Autoregressive Modeling: Scalable Image Generation via Next-Scale Prediction

Keyu Tian[1,2], Yi Jiang[2,†], Zehuan Yuan[2,*], Bingyue Peng[2], Liwei Wang[1,*]
[1]Peking University [2]Bytedance Inc
keyutian@stu.pku.edu.cn, jiangyi.enjoy@bytedance.com,
yuanzehuan@bytedance.com, bingyue.peng@bytedance.com, wanglw@pku.edu.cn

Try and explore our online demo at: https://var.vision
Codes and models: https://github.com/FoundationVision/VAR

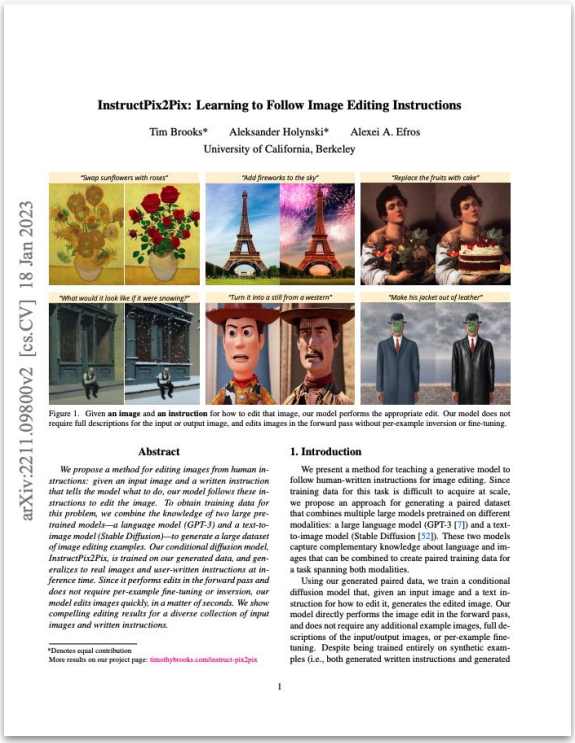
Figure 1: **Generated samples from Visual AutoRegressive (VAR) transformers trained on ImageNet**. We show 512×512 samples (top), 256×256 samples (middle), and zero-shot image editing results (bottom).

Abstract

We present Visual AutoRegressive modeling (VAR), a new generation paradigm that redefines the autoregressive learning on images as coarse-to-fine "next-scale prediction" or "next-resolution prediction", diverging from the standard raster-scan "next-token prediction". This simple, intuitive methodology allows autoregressive (AR) transformers to learn visual distributions fast and can generalize well: VAR, for the *first time*, makes GPT-style AR models surpass diffusion transformers in image generation. On ImageNet 256×256 benchmark, VAR significantly improve AR baseline by improving Fréchet inception distance (FID) from 18.65 to 1.73, inception score (IS) from 80.4 to 350.2, with 20× faster inference speed. It is also empirically verified that VAR outperforms the Diffusion Transformer (DiT) in multiple dimensions including image quality, inference speed, data efficiency, and scalability. Scaling up VAR models exhibits clear power-law scaling laws similar to those observed in LLMs, with linear correlation coefficients near −0.998 as solid evidence. VAR further showcases zero-shot generalization ability in downstream tasks including image in-painting, out-painting, and editing. These results suggest VAR has initially emulated the two important properties of LLMs: **Scaling Laws** and **zero-shot** generalization. We have released all models and codes to promote the exploration of AR/VAR models for visual generation and unified learning.

*Corresponding authors: wanglw@pku.edu.cn, yuanzehuan@bytedance.com; †: project lead

Preprint. Under review.



InstructPix2Pix: Learning to Follow Image Editing Instructions

Tim Brooks* Aleksander Holynski* Alexei A. Efros
University of California, Berkeley

Figure 1. Given **an image** and **an instruction** for how to edit that image, our model performs the appropriate edit. Our model does not require full descriptions for the input or output image, and edits images in the forward pass without per-example inversion or fine-tuning.

Abstract

We propose a method for editing images from human instructions: given an input image and a written instruction that tells the model what to do, our model follows these instructions to edit the image. To obtain training data for this problem, we combine the knowledge of two large pre-trained models—a language model (GPT-3) and a text-to-image model (Stable Diffusion)—to generate a large dataset of image editing examples. Our conditional diffusion model, InstructPix2Pix, is trained on our generated data, and generalizes to real images and user-written instructions at inference time. Since it performs edits in the forward pass and does not require per-example fine-tuning or inversion, our model edits images quickly, in a matter of seconds. We show compelling editing results for a diverse collection of input images and written instructions.

*Denotes equal contribution
More results on our project page: timothybrooks.com/instruct-pix2pix

I. Introduction

We present a method for teaching a generative model to follow human-written instructions for image editing. Since training data for this task is difficult to acquire at scale, we propose an approach for generating a paired dataset that combines multiple large models pretrained on different modalities: a large language model (GPT-3 [7]) and a text-to-image model (Stable Diffusion [52]). These two models capture complementary knowledge about language and images that can be combined to create paired training data for a task spanning both modalities.

Using our generated paired data, we train a conditional diffusion model that, given an input image and a text instruction for how to edit it, generates the edited image. Our model directly performs the image edit in the forward pass, and does not require any additional example images, full descriptions of the input/output images, or per-example fine-tuning. Despite being trained entirely on synthetic examples (i.e., both generated written instructions and generated

Today

After: Next batch of four **paper presentations**.

arXiv:2406.07496v1 [cs.CL] 11 Jun 2024

TextGrad: Automatic "Differentiation" via Text

Mert Yuksekgonul[1*] MERTY@STANFORD.EDU
Federico Bianchi[1*] FEDE@STANFORD.EDU
Joseph Boen[2*] TBOEN@STANFORD.EDU
Sheng Liu[2*] SHENGL@STANFORD.EDU
Zhi Huang[2*] ZHIHUANG@STANFORD.EDU
Carlos Guestrin[1,3] GUESTRIN@STANFORD.EDU
James Zou[1,2,3] JAMESZ@STANFORD.EDU
[1]Department of Computer Science, Stanford University
[2]Department of Biomedical Data Science, Stanford University
[3]Chan Zuckerberg Biohub
Correspondence: merty@stanford.edu and jamesz@stanford.edu

Repository and Tutorials

Abstract

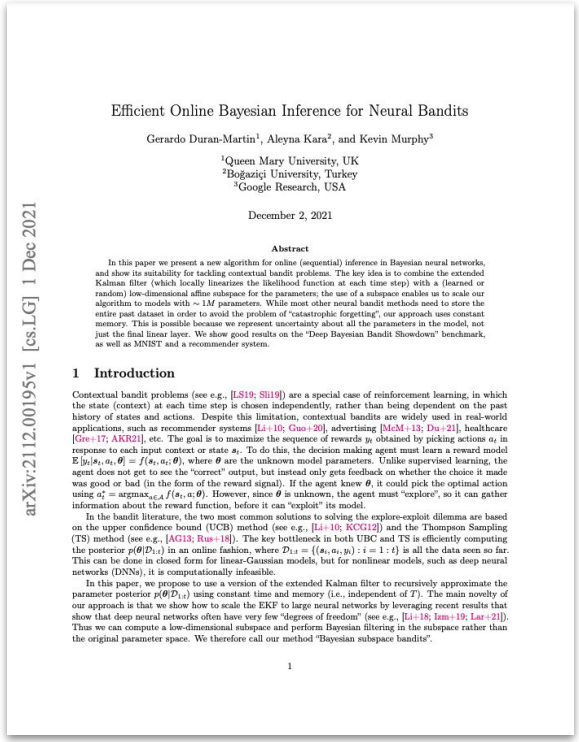
AI is undergoing a paradigm shift, with breakthroughs achieved by systems orchestrating multiple large language models (LLMs) and other complex components. As a result, developing principled and automated optimization methods for compound AI systems is one of the most important new challenges. Neural networks faced a similar challenge in its early days until backpropagation and automatic differentiation transformed the field by making optimization turn-key. Inspired by this, we introduce TextGrad, a powerful framework performing automatic "differentiation" via text. TextGrad backpropagates textual feedback provided by LLMs to improve individual components of a compound AI system. In our framework, LLMs provide rich, general, natural language suggestions to optimize variables in computation graphs, ranging from code snippets to molecular structures. TextGrad follows PyTorch's syntax and abstraction and is flexible and easy-to-use. It works out-of-the-box for a variety of tasks, where the users only provide the objective function without tuning components or prompts of the framework. We showcase TextGrad's effectiveness and generality across a diverse range of applications, from question answering and molecule optimization to radiotherapy treatment planning. Without modifying the framework, TextGrad improves the zero-shot accuracy of GPT-4o in Google-Proof Question Answering from 51% to 55%, yields 20% relative performance gain in optimizing LeetCode-Hard coding problem solutions, improves prompts for reasoning, designs new druglike small molecules with desirable *in silico* binding, and designs radiation oncology treatment plans with high specificity. TextGrad lays a foundation to accelerate the development of the next-generation of AI systems.

1 Introduction

There is an emerging paradigm shift in how AI systems are built, owing to the breakthroughs of Large Language Models (LLMs) [1–6]. The new generation of AI applications are increasingly compound systems involving multiple sophisticated components, where each component could be an LLM-based agent, a tool such as a simulator, or web search. For instance, a system of LLMs communicating with symbolic solvers can solve olympiad-level math problems [7]; a system of LLMs using search engines and code interpreter tools performs comparably to human competitive programmers [8] and are solving real-world

*Co-first authors.

1



arXiv:2112.00195v1 [cs.LG] 1 Dec 2021

Efficient Online Bayesian Inference for Neural Bandits

Gerardo Duran-Martin[1], Aleyna Kara[2], and Kevin Murphy[3]
[1]Queen Mary University, UK
[2]Boğaziçi University, Turkey
[3]Google Research, USA

December 2, 2021

Abstract

In this paper we present a new algorithm for online (sequential) inference in Bayesian neural networks, and show its suitability for tackling contextual bandit problems. The key idea is to combine the extended Kalman filter (which locally linearizes the likelihood function at each time step) with a (learned or random) low-dimensional affine subspace for the parameters; the use of a subspace enables us to scale our algorithm to models with $\sim 1M$ parameters. While most other neural bandit methods need to store the entire past dataset in order to avoid the problem of "catastrophic forgetting", our approach uses constant memory. This is possible because we represent uncertainty about all the parameters in the model, not just the final linear layer. We show good results on the "Deep Bayesian Bandit Showdown" benchmark, as well as MNIST and a recommender system.

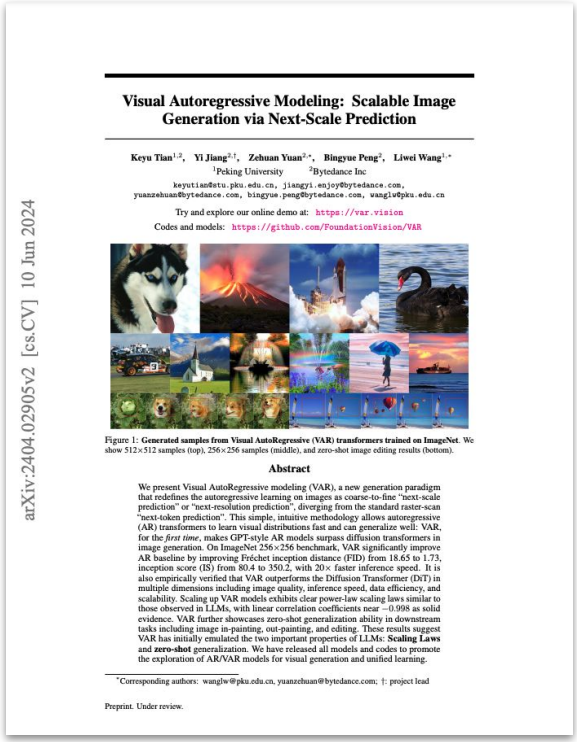
1 Introduction

Contextual bandit problems (see e.g., [LS19; Sli19]) are a special case of reinforcement learning, in which the state (context) at each time step is chosen independently, rather than being dependent on the past history of states and actions. Despite this limitation, contextual bandits are widely used in real-world applications, such as recommender systems [Li+10; Guo+20], advertising [McM+13; Du+21], healthcare [Gre+17; AKR21], etc. The goal is to maximize the sequence of rewards y_t obtained by picking actions a_t in response to each input context or state $\boldsymbol{s}_t$. To do this, the decision making agent must learn a reward model $\mathbb{E}\left[y_t|\boldsymbol{s}_t, a_t, \boldsymbol{\theta}\right] = f(\boldsymbol{s}_t, a_t; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ are the unknown model parameters. Unlike supervised learning, the agent does not get to see the "correct" output, but instead only gets feedback on whether the choice it made was good or bad (in the form of the reward signal). If the agent knew $\boldsymbol{\theta}$, it could pick the optimal action using $a_t^* = \text{argmax}_{a \in \mathcal{A}} f(\boldsymbol{s}_t, a; \boldsymbol{\theta})$. However, since $\boldsymbol{\theta}$ is unknown, the agent must "explore", so it can gather information about the reward function, before it can "exploit" its model.

In the bandit literature, the two most common solutions to solving the explore-exploit dilemma are based on the upper confidence bound (UCB) method (see e.g., [Li+10; KCG12]) and the Thompson Sampling (TS) method (see e.g., [AG13; Rus+18]). The key bottleneck in both UBC and TS is efficiently computing the posterior $p(\boldsymbol{\theta}|\mathcal{D}_{1:t})$ in an online fashion, where $\mathcal{D}_{1:t} = \{(\boldsymbol{s}_i, a_i, y_i) : i = 1 : t\}$ is all the data seen so far. This can be done in closed form for linear-Gaussian models, but for nonlinear models, such as deep neural networks (DNNs), it is computationally infeasible.

In this paper, we propose to use a version of the extended Kalman filter to recursively approximate the parameter posterior $p(\boldsymbol{\theta}|\mathcal{D}_{1:t})$ using constant time and memory (i.e., independent of T). The main novelty of our approach is that we show how to scale the EKF to large neural networks by leveraging recent results that show that deep neural networks often have very few "degrees of freedom" (see e.g., [Li+18; Izm+19; Lar+21]). Thus we can compute a low-dimensional subspace and perform Bayesian filtering in the subspace rather than the original parameter space. We therefore call our method "Bayesian subspace bandits".

1



arXiv:2404.02905v2 [cs.CV] 10 Jun 2024

Visual Autoregressive Modeling: Scalable Image Generation via Next-Scale Prediction

Keyu Tian[1,2], Yi Jiang[2,†], Zehuan Yuan[2,*], Bingyue Peng[2], Liwei Wang[1,*]
[1]Peking University [2]Bytedance Inc
keyutian@stu.pku.edu.cn, jiangyi.enjoy@bytedance.com,
yuanzehuan@bytedance.com, bingyue.peng@bytedance.com, wanglw@pku.edu.cn

Try and explore our online demo at: https://var.vision

Codes and models: https://github.com/FoundationVision/VAR

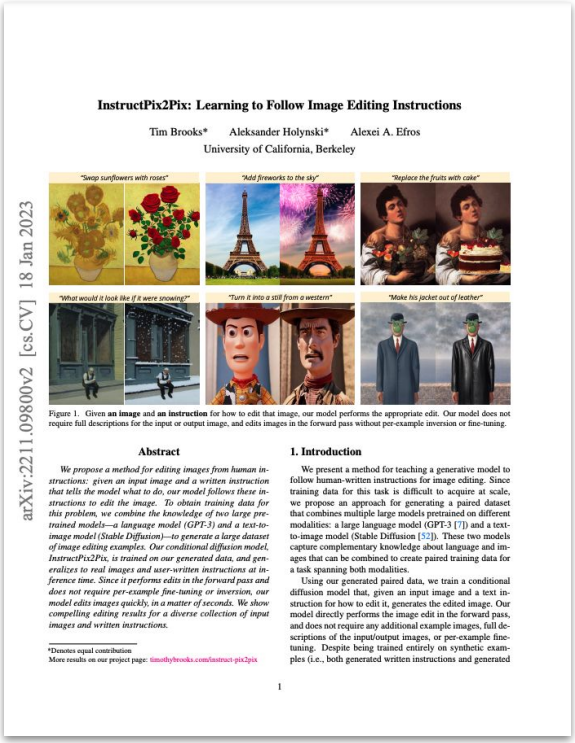
Figure 1: **Generated samples from Visual AutoRegressive (VAR) transformers trained on ImageNet**. We show 512×512 samples (top), 256×256 samples (middle), and zero-shot image editing results (bottom).

Abstract

We present Visual AutoRegressive modeling (VAR), a new generation paradigm that redefines the autoregressive learning on images as coarse-to-fine "next-scale prediction" or "next-resolution prediction", diverging from the standard raster-scan "next-token prediction". This simple, intuitive methodology allows autoregressive (AR) transformers to learn visual distributions fast and can generalize well: VAR, for the *first time*, makes GPT-style AR models surpass diffusion transformers in image generation. On ImageNet 256×256 benchmark, VAR significantly improve AR baseline by improving Fréchet inception distance (FID) from 18.65 to 1.73, inception score (IS) from 80.4 to 350.2, with 20× faster inference speed. It is also empirically verified that VAR outperforms the Diffusion Transformer (DiT) in multiple dimensions including image quality, inference speed, data efficiency, and scalability. Scaling up VAR models exhibits clear power-law scaling laws similar to those observed in LLMs, with linear correlation coefficients near −0.998 as solid evidence. VAR further showcases zero-shot generalization ability in downstream tasks including image in-painting, out-painting, and editing. These results suggest VAR has initially emulated the two important properties of LLMs: **Scaling Laws** and **zero-shot** generalization. We have released all models and codes to promote the exploration of AR/VAR models for visual generation and unified learning.

*Corresponding authors: wanglw@pku.edu.cn, yuanzehuan@bytedance.com; †: project lead

Preprint. Under review.



arXiv:2211.09800v2 [cs.CV] 18 Jan 2023

InstructPix2Pix: Learning to Follow Image Editing Instructions

Tim Brooks* Aleksander Holynski* Alexei A. Efros
University of California, Berkeley

Figure 1. Given **an image** and **an instruction** for how to edit that image, our model performs the appropriate edit. Our model does not require full descriptions for the input or output image, and edits images in the forward pass without per-example inversion or fine-tuning.

Abstract

We propose a method for editing images from human instructions: given an input image and a written instruction that tells the model what to do, our model follows these instructions to edit the image. To obtain training data for this problem, we combine the knowledge of two large pre-trained models—a language model (GPT-3) and a text-to-image model (Stable Diffusion)—to generate a large dataset of image editing examples. Our conditional diffusion model, InstructPix2Pix, is trained on our generated data, and generalizes to real images and user-written instructions at inference time. Since it performs edits in the forward pass and does not require per-example fine-tuning or inversion, our model edits images quickly, in a matter of seconds. We show compelling editing results for a diverse collection of input images and written instructions.

*Denotes equal contribution
More results on our project page: timothybrooks.com/instruct-pix2pix

I. Introduction

We present a method for teaching a generative model to follow human-written instructions for image editing. Since training data for this task is difficult to acquire at scale, we propose an approach for generating a paired dataset that combines multiple large models pretrained on different modalities: a large language model (GPT-3 [7]) and a text-to-image model (Stable Diffusion [52]). These two models capture complementary knowledge about language and images that can be combined to create paired training data for a task spanning both modalities.

Using our generated paired data, we train a conditional diffusion model that, given an input image and a text instruction for how to edit it, generates the edited image. Our model directly performs the image edit in the forward pass, and does not require any additional example images, full descriptions of the input/output images, or per-example fine-tuning. Despite being trained entirely on synthetic examples (i.e., both generated written instructions and generated

1

TextGrad — backprop-like procedures via LLMs/text

Today

After: Next batch of four **paper presentations**.

arXiv:2406.07496v1 [cs.CL] 11 Jun 2024

TextGrad: Automatic "Differentiation" via Text

Mert Yuksekgonul[1*] MERTY@STANFORD.EDU
Federico Bianchi[1*] FEDE@STANFORD.EDU
Joseph Boen[2*] TBOEN@STANFORD.EDU
Sheng Liu[2*] SHENGL@STANFORD.EDU
Zhi Huang[2*] ZHIHUANG@STANFORD.EDU
Carlos Guestrin[1,3] GUESTRIN@STANFORD.EDU
James Zou[1,2,3] JAMESZ@STANFORD.EDU
[1]Department of Computer Science, Stanford University
[2]Department of Biomedical Data Science, Stanford University
[3]Chan Zuckerberg Biohub
Correspondence: merty@stanford.edu and jamesz@stanford.edu

Repository and Tutorials

Abstract

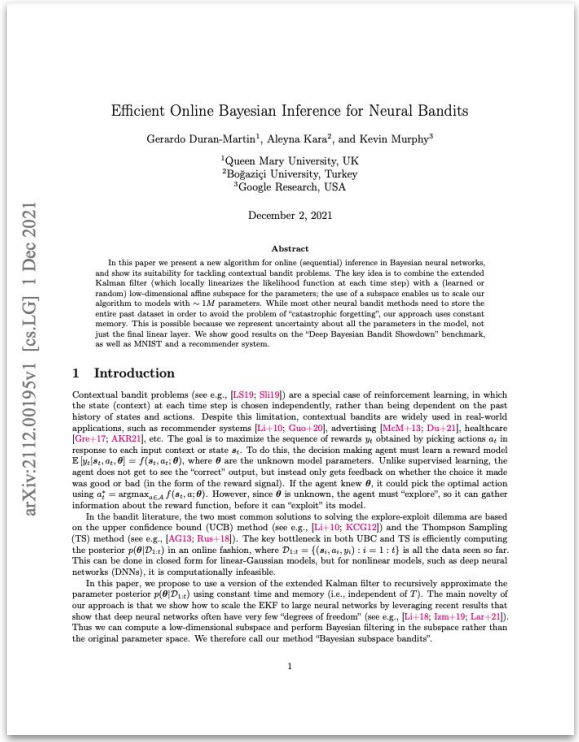
AI is undergoing a paradigm shift, with breakthroughs achieved by systems orchestrating multiple large language models (LLMs) and other complex components. As a result, developing principled and automated optimization methods for compound AI systems is one of the most important new challenges. Neural networks faced a similar challenge in its early days until backpropagation and automatic differentiation transformed the field by making optimization turn-key. Inspired by this, we introduce TextGrad, a powerful framework performing automatic "differentiation" via text. TextGrad backpropagates textual feedback provided by LLMs to improve individual components of a compound AI system. In our framework, LLMs provide rich, general, natural language suggestions to optimize variables in computation graphs, ranging from code snippets to molecular structures. TextGrad follows PyTorch's syntax and abstraction and is flexible and easy-to-use. It works out-of-the-box for a variety of tasks, where the users only provide the objective function without tuning components or prompts of the framework. We showcase TextGrad's effectiveness and generality across a diverse range of applications, from question answering and molecule optimization to radiotherapy treatment planning. Without modifying the framework, TextGrad improves the zero-shot accuracy of GPT-4o in Google-Proof Question Answering from 51% to 55%, yields 20% relative performance gain in optimizing LeetCode-Hard coding problem solutions, improves prompts for reasoning, designs new druglike small molecules with desirable *in silico* binding, and designs radiation oncology treatment plans with high specificity. TextGrad lays a foundation to accelerate the development of the next-generation of AI systems.

1 Introduction

There is an emerging paradigm shift in how AI systems are built, owing to the breakthroughs of Large Language Models (LLMs) [1–6]. The new generation of AI applications are increasingly compound systems involving multiple sophisticated components, where each component could be an LLM-based agent, a tool such as a simulator, or web search. For instance, a system of LLMs communicating with symbolic solvers can solve olympiad-level math problems [7]; a system of LLMs using search engines and code interpreter tools performs comparably to human competitive programmers [8] and are solving real-world

*Co-first authors.

1



arXiv:2112.00195v1 [cs.LG] 1 Dec 2021

Efficient Online Bayesian Inference for Neural Bandits

Gerardo Duran-Martin[1], Aleyna Kara[2], and Kevin Murphy[3]
[1]Queen Mary University, UK
[2]Boğaziçi University, Turkey
[3]Google Research, USA

December 2, 2021

Abstract

In this paper we present a new algorithm for online (sequential) inference in Bayesian neural networks, and show its suitability for tackling contextual bandit problems. The key idea is to combine the extended Kalman filter (which locally linearizes the likelihood function at each time step) with a (learned or random) low-dimensional affine subspace for the parameters; the use of a subspace enables us to scale our algorithm to models with $\sim 1M$ parameters. While most other neural bandit methods need to store the entire past dataset in order to avoid the problem of "catastrophic forgetting", our approach uses constant memory. This is possible because we represent uncertainty about all the parameters in the model, not just the final linear layer. We show good results on the "Deep Bayesian Bandit Showdown" benchmark, as well as MNIST and a recommender system.

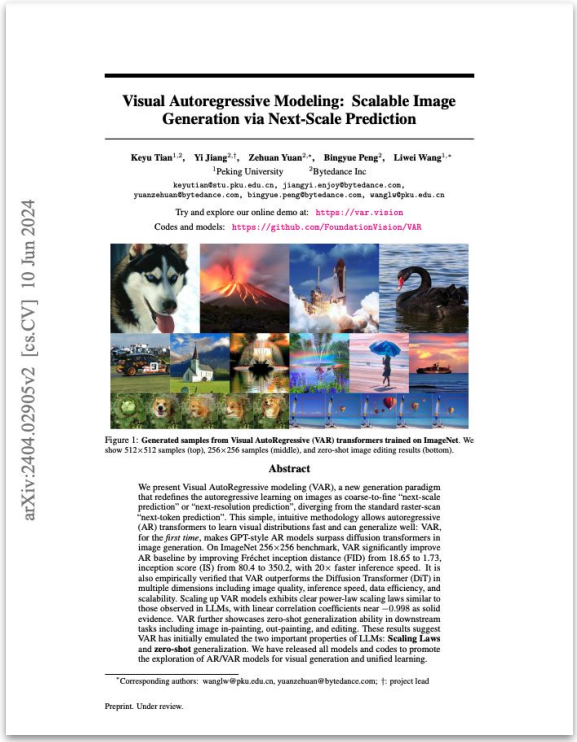
1 Introduction

Contextual bandit problems (see e.g., [LS19; Sli19]) are a special case of reinforcement learning, in which the state (context) at each time step is chosen independently, rather than being dependent on the past history of states and actions. Despite this limitation, contextual bandits are widely used in real-world applications, such as recommender systems [Li+10; Guo+20], advertising [McM+13; Du+21], healthcare [Gre+17; AKR21], etc. The goal is to maximize the sequence of rewards y_t obtained by picking actions a_t in response to each input context or state $\boldsymbol{s}_t$. To do this, the decision making agent must learn a reward model $\mathbb{E}\left[y_t|\boldsymbol{s}_t, a_t, \boldsymbol{\theta}\right] = f(\boldsymbol{s}_t, a_t; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ are the unknown model parameters. Unlike supervised learning, the agent does not get to see the "correct" output, but instead only gets feedback on whether the choice it made was good or bad (in the form of the reward signal). If the agent knew $\boldsymbol{\theta}$, it could pick the optimal action using $a_t^* = \text{argmax}_{a \in \mathcal{A}} f(\boldsymbol{s}_t, a; \boldsymbol{\theta})$. However, since $\boldsymbol{\theta}$ is unknown, the agent must "explore", so it can gather information about the reward function, before it can "exploit" its model.

In the bandit literature, the two most common solutions to solving the explore-exploit dilemma are based on the upper confidence bound (UCB) method (see e.g., [Li+10; KCG12]) and the Thompson Sampling (TS) method (see e.g., [AG13; Rus+18]). The key bottleneck in both UBC and TS is efficiently computing the posterior $p(\boldsymbol{\theta}|\mathcal{D}_{1:t})$ in an online fashion, where $\mathcal{D}_{1:t} = \{(\boldsymbol{s}_i, a_i, y_i) : i = 1 : t\}$ is all the data seen so far. This can be done in closed form for linear-Gaussian models, but for nonlinear models, such as deep neural networks (DNNs), it is computationally infeasible.

In this paper, we propose to use a version of the extended Kalman filter to recursively approximate the parameter posterior $p(\boldsymbol{\theta}|\mathcal{D}_{1:t})$ using constant time and memory (i.e., independent of T). The main novelty of our approach is that we show how to scale the EKF to large neural networks by leveraging recent results that show that deep neural networks often have very few "degrees of freedom" (see e.g., [Li+18; Izm+19; Lar+21]). Thus we can compute a low-dimensional subspace and perform Bayesian filtering in the subspace rather than the original parameter space. We therefore call our method "Bayesian subspace bandits".

1



arXiv:2404.02905v2 [cs.CV] 10 Jun 2024

Visual Autoregressive Modeling: Scalable Image Generation via Next-Scale Prediction

Keyu Tian[1,2], Yi Jiang[2,†], Zehuan Yuan[2,*], Bingyue Peng[2], Liwei Wang[1,*]
[1]Peking University [2]Bytedance Inc
keyutian@stu.pku.edu.cn, jiangyi.enjoy@bytedance.com,
yuanzehuan@bytedance.com, bingyue.peng@bytedance.com, wanglw@pku.edu.cn

Try and explore our online demo at: https://var.vision

Codes and models: https://github.com/FoundationVision/VAR

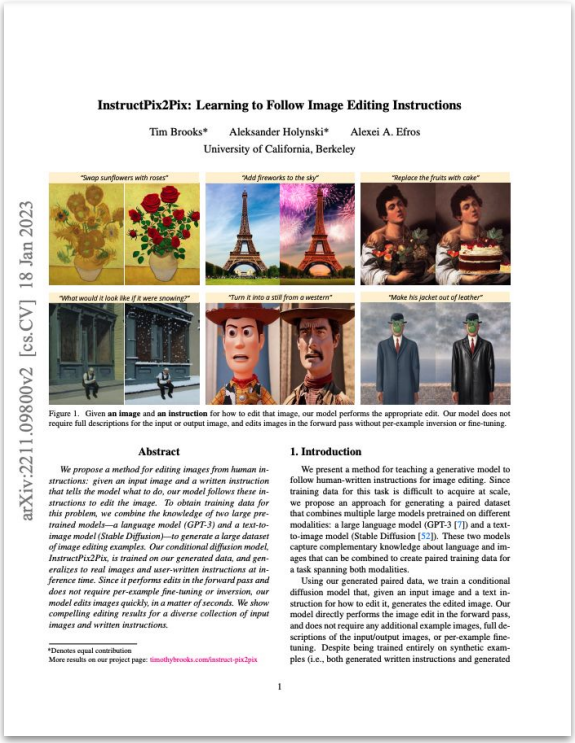
Figure 1: **Generated samples from Visual AutoRegressive (VAR) transformers trained on ImageNet**. We show 512×512 samples (top), 256×256 samples (middle), and zero-shot image editing results (bottom).

Abstract

We present Visual AutoRegressive modeling (VAR), a new generation paradigm that redefines the autoregressive learning on images as coarse-to-fine "next-scale prediction" or "next-resolution prediction", diverging from the standard raster-scan "next-token prediction". This simple, intuitive methodology allows autoregressive (AR) transformers to learn visual distributions fast and can generalize well: VAR, for the *first time*, makes GPT-style AR models surpass diffusion transformers in image generation. On ImageNet 256×256 benchmark, VAR significantly improve AR baseline by improving Fréchet inception distance (FID) from 18.65 to 1.73, inception score (IS) from 80.4 to 350.2, with 20× faster inference speed. It is also empirically verified that VAR outperforms the Diffusion Transformer (DiT) in multiple dimensions including image quality, inference speed, data efficiency, and scalability. Scaling up VAR models exhibits clear power-law scaling laws similar to those observed in LLMs, with linear correlation coefficients near −0.998 as solid evidence. VAR further showcases zero-shot generalization ability in downstream tasks including image in-painting, out-painting, and editing. These results suggest VAR has initially emulated the two important properties of LLMs: **Scaling Laws** and **zero-shot** generalization. We have released all models and codes to promote the exploration of AR/VAR models for visual generation and unified learning.

*Corresponding authors: wanglw@pku.edu.cn, yuanzehuan@bytedance.com; †: project lead

Preprint. Under review.



arXiv:2211.09800v2 [cs.CV] 18 Jan 2023

InstructPix2Pix: Learning to Follow Image Editing Instructions

Tim Brooks* Aleksander Holynski* Alexei A. Efros
University of California, Berkeley

Figure 1. Given **an image** and **an instruction** for how to edit that image, our model performs the appropriate edit. Our model does not require full descriptions for the input or output image, and edits images in the forward pass without per-example inversion or fine-tuning.

Abstract

We propose a method for editing images from human instructions: given an input image and a written instruction that tells the model what to do, our model follows these instructions to edit the image. To obtain training data for this problem, we combine the knowledge of two large pre-trained models—a language model (GPT-3) and a text-to-image model (Stable Diffusion)—to generate a large dataset of image editing examples. Our conditional diffusion model, InstructPix2Pix, is trained on our generated data, and generalizes to real images and user-written instructions at inference time. Since it performs edits in the forward pass and does not require per-example fine-tuning or inversion, our model edits images quickly, in a matter of seconds. We show compelling editing results for a diverse collection of input images and written instructions.

*Denotes equal contribution
More results on our project page: timothybrooks.com/instruct-pix2pix

I. Introduction

We present a method for teaching a generative model to follow human-written instructions for image editing. Since training data for this task is difficult to acquire at scale, we propose an approach for generating a paired dataset that combines multiple large models pretrained on different modalities: a large language model (GPT-3 [7]) and a text-to-image model (Stable Diffusion [52]). These two models capture complementary knowledge about language and images that can be combined to create paired training data for a task spanning both modalities.

Using our generated paired data, we train a conditional diffusion model that, given an input image and a text instruction for how to edit it, generates the edited image. Our model directly performs the image edit in the forward pass, and does not require any additional example images, full descriptions of the input/output images, or per-example fine-tuning. Despite being trained entirely on synthetic examples (i.e., both generated written instructions and generated

1

Efficient online inference & bandits with BNNs

Today

After: Next batch of four **paper presentations**.

TextGrad: Automatic "Differentiation" via Text

Mert Yuksekgonul[1*] MERTY@STANFORD.EDU
Federico Bianchi[1*] FEDE@STANFORD.EDU
Joseph Boen[2*] TBOEN@STANFORD.EDU
Sheng Liu[2*] SHENGL@STANFORD.EDU
Zhi Huang[2*] ZHIHUANG@STANFORD.EDU
Carlos Guestrin[1,3] GUESTRIN@STANFORD.EDU
James Zou[1,2,3] JAMESZ@STANFORD.EDU
[1]Department of Computer Science, Stanford University
[2]Department of Biomedical Data Science, Stanford University
[3]Chan Zuckerberg Biohub
Correspondence: merty@stanford.edu and jamesz@stanford.edu

Repository and Tutorials

Abstract

AI is undergoing a paradigm shift, with breakthroughs achieved by systems orchestrating multiple large language models (LLMs) and other complex components. As a result, developing principled and automated optimization methods for compound AI systems is one of the most important new challenges. Neural networks faced a similar challenge in its early days until backpropagation and automatic differentiation transformed the field by making optimization turn-key. Inspired by this, we introduce TextGrad, a powerful framework performing automatic "differentiation" via text. TextGrad backpropagates textual feedback provided by LLMs to improve individual components of a compound AI system. In our framework, LLMs provide rich, general, natural language suggestions to optimize variables in computation graphs, ranging from code snippets to molecular structures. TextGrad follows PyTorch's syntax and abstraction and is flexible and easy-to-use. It works out-of-the-box for a variety of tasks, where the users only provide the objective function without tuning components or prompts of the framework. We showcase TextGrad's effectiveness and generality across a diverse range of applications, from question answering and molecule optimization to radiotherapy treatment planning. Without modifying the framework, TextGrad improves the zero-shot accuracy of GPT-4o in Google-Proof Question Answering from 51% to 55%, yields 20% relative performance gain in optimizing LeetCode-Hard coding problem solutions, improves prompts for reasoning, designs new druglike small molecules with desirable *in silico* binding, and designs radiation oncology treatment plans with high specificity. TextGrad lays a foundation to accelerate the development of the next-generation of AI systems.

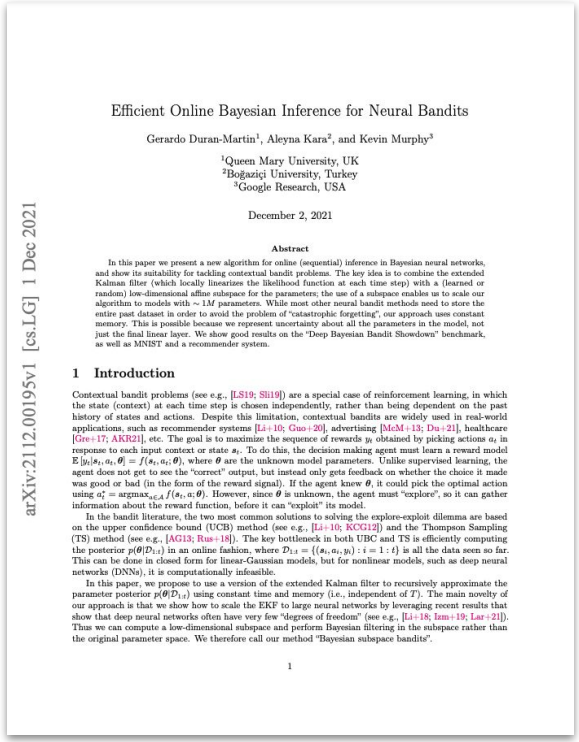
1 Introduction

There is an emerging paradigm shift in how AI systems are built, owing to the breakthroughs of Large Language Models (LLMs) [1–6]. The new generation of AI applications are increasingly compound systems involving multiple sophisticated components, where each component could be an LLM-based agent, a tool such as a simulator, or web search. For instance, a system of LLMs communicating with symbolic solvers can solve olympiad-level math problems [7]; a system of LLMs using search engines and code interpreter tools performs comparably to human competitive programmers [8] and are solving real-world

*Co-first authors.

1

arXiv:2406.07496v1 [cs.CL] 11 Jun 2024



Efficient Online Bayesian Inference for Neural Bandits

Gerardo Duran-Martin[1], Aleyna Kara[2], and Kevin Murphy[3]

[1]Queen Mary University, UK
[2]Boğaziçi University, Turkey
[3]Google Research, USA

December 2, 2021

Abstract

In this paper we present a new algorithm for online (sequential) inference in Bayesian neural networks, and show its suitability for tackling contextual bandit problems. The key idea is to combine the extended Kalman filter (which locally linearizes the likelihood function at each time step) with a (learned or random) low-dimensional affine subspace for the parameters; the use of a subspace enables us to scale our algorithm to models with $\sim 1M$ parameters. While most other neural bandit methods need to store the entire past dataset in order to avoid the problem of "catastrophic forgetting", our approach uses constant memory. This is possible because we represent uncertainty about all the parameters in the model, not just the final linear layer. We show good results on the "Deep Bayesian Bandit Showdown" benchmark, as well as MNIST and a recommender system.

1 Introduction

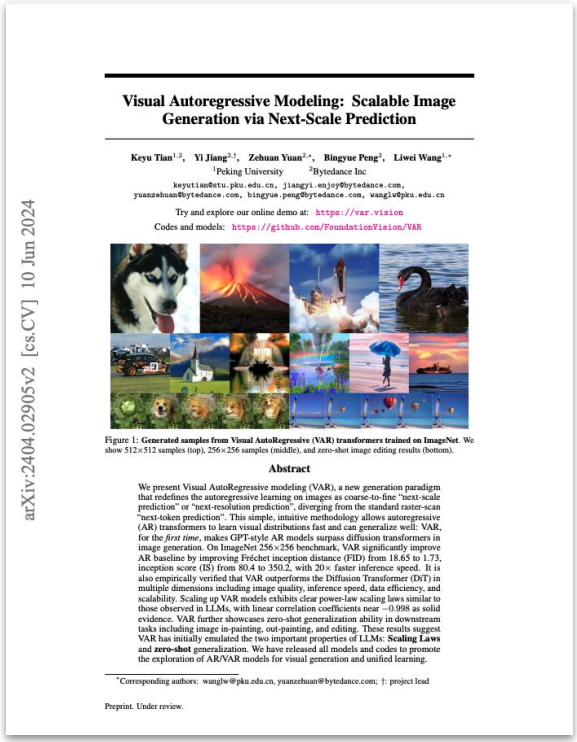
Contextual bandit problems (see e.g., [LS19; Sli19]) are a special case of reinforcement learning, in which the state (context) at each time step is chosen independently, rather than being dependent on the past history of states and actions. Despite this limitation, contextual bandits are widely used in real-world applications, such as recommender systems [Li+10; Guo+20], advertising [McM+13; Du+21], healthcare [Gre+17; AKR21], etc. The goal is to maximize the sequence of rewards y_t obtained by picking actions a_t in response to each input context or state $\boldsymbol{s}_t$. To do this, the decision making agent must learn a reward model $\mathbb{E}\left[y_t|\boldsymbol{s}_t, a_t, \boldsymbol{\theta}\right] = f(\boldsymbol{s}_t, a_t; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ are the unknown model parameters. Unlike supervised learning, the agent does not get to see the "correct" output, but instead only gets feedback on whether the choice it made was good or bad (in the form of the reward signal). If the agent knew $\boldsymbol{\theta}$, it could pick the optimal action using $a_t^* = \text{argmax}_{a \in \mathcal{A}} f(\boldsymbol{s}_t, a; \boldsymbol{\theta})$. However, since $\boldsymbol{\theta}$ is unknown, the agent must "explore", so it can gather information about the reward function, before it can "exploit" its model.

In the bandit literature, the two most common solutions to solving the explore-exploit dilemma are based on the upper confidence bound (UCB) method (see e.g., [Li+10; KCG12]) and the Thompson Sampling (TS) method (see e.g., [AG13; Rus+18]). The key bottleneck in both UBC and TS is efficiently computing the posterior $p(\boldsymbol{\theta}|\mathcal{D}_{1:t})$ in an online fashion, where $\mathcal{D}_{1:t} = \{(\boldsymbol{s}_i, a_i, y_i) : i = 1 : t\}$ is all the data seen so far. This can be done in closed form for linear-Gaussian models, but for nonlinear models, such as deep neural networks (DNNs), it is computationally infeasible.

In this paper, we propose to use a version of the extended Kalman filter to recursively approximate the parameter posterior $p(\boldsymbol{\theta}|\mathcal{D}_{1:t})$ using constant time and memory (i.e., independent of T). The main novelty of our approach is that we show how to scale the EKF to large neural networks by leveraging recent results that show that deep neural networks often have very few "degrees of freedom" (see e.g., [Li+18; Izm+19; Lar+21]). Thus we can compute a low-dimensional subspace and perform Bayesian filtering in the subspace rather than the original parameter space. We therefore call our method "Bayesian subspace bandits".

1

arXiv:2112.00195v1 [cs.LG] 1 Dec 2021



Visual Autoregressive Modeling: Scalable Image Generation via Next-Scale Prediction

Keyu Tian[1,2], Yi Jiang[2,†], Zehuan Yuan[2,*], Bingyue Peng[2], Liwei Wang[1,*]
[1]Peking University [2]Bytedance Inc
keyutian@stu.pku.edu.cn, jiangyi.enjoy@bytedance.com, yuanzehuan@bytedance.com, bingyue.peng@bytedance.com, wanglw@pku.edu.cn

Try and explore our online demo at: https://var.vision
Codes and models: https://github.com/FoundationVision/VAR

Figure 1: **Generated samples from Visual AutoRegressive (VAR) transformers trained on ImageNet**. We show 512×512 samples (top), 256×256 samples (middle), and zero-shot image editing results (bottom).

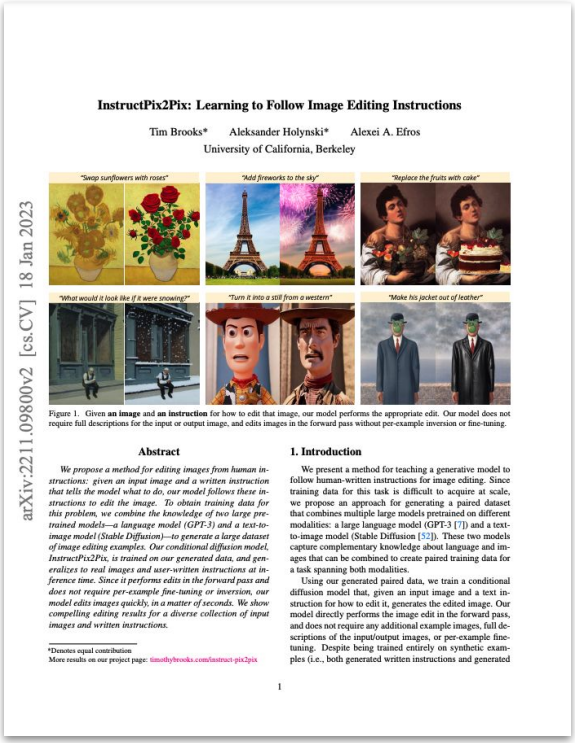
Abstract

We present Visual AutoRegressive modeling (VAR), a new generation paradigm that redefines the autoregressive learning on images as coarse-to-fine "next-scale prediction" or "next-resolution prediction", diverging from the standard raster-scan "next-token prediction". This simple, intuitive methodology allows autoregressive (AR) transformers to learn visual distributions fast and can generalize well: VAR, for the *first time*, makes GPT-style AR models surpass diffusion transformers in image generation. On ImageNet 256×256 benchmark, VAR significantly improve AR baseline by improving Fréchet inception distance (FID) from 18.65 to 1.73, inception score (IS) from 80.4 to 350.2, with 20× faster inference speed. It is also empirically verified that VAR outperforms the Diffusion Transformer (DiT) in multiple dimensions including image quality, inference speed, data efficiency, and scalability. Scaling up VAR models exhibits clear power-law scaling laws similar to those observed in LLMs, with linear correlation coefficients near −0.998 as solid evidence. VAR further showcases zero-shot generalization ability in downstream tasks including image in-painting, out-painting, and editing. These results suggest VAR has initially emulated the two important properties of LLMs: **Scaling Laws** and **zero-shot** generalization. We have released all models and codes to promote the exploration of AR/VAR models for visual generation and unified learning.

*Corresponding authors: wanglw@pku.edu.cn, yuanzehuan@bytedance.com; †: project lead

Preprint. Under review.

arXiv:2404.02905v2 [cs.CV] 10 Jun 2024



InstructPix2Pix: Learning to Follow Image Editing Instructions

Tim Brooks* Aleksander Holynski* Alexei A. Efros
University of California, Berkeley

"Swap sunflowers with roses" "Add fireworks to the sky" "Replace the fruits with cake"
"What would it look like if it were snowing?" "Turn it into a still from a western" "Make his jacket out of leather"

Figure 1. Given **an image** and **an instruction** for how to edit that image, our model performs the appropriate edit. Our model does not require full descriptions for the input or output image, and edits images in the forward pass without per-example inversion or fine-tuning.

Abstract

We propose a method for editing images from human instructions: given an input image and a written instruction that tells the model what to do, our model follows these instructions to edit the image. To obtain training data for this problem, we combine the knowledge of two large pre-trained models—a language model (GPT-3) and a text-to-image model (Stable Diffusion)—to generate a large dataset of image editing examples. Our conditional diffusion model, InstructPix2Pix, is trained on our generated data, and generalizes to real images and user-written instructions at inference time. Since it performs edits in the forward pass and does not require per-example fine-tuning or inversion, our model edits images quickly, in a matter of seconds. We show compelling editing results for a diverse collection of input images and written instructions.

*Denotes equal contribution
More results on our project page: timothybrooks.com/instruct-pix2pix

I. Introduction

We present a method for teaching a generative model to follow human-written instructions for image editing. Since training data for this task is difficult to acquire at scale, we propose an approach for generating a paired dataset that combines multiple large models pretrained on different modalities: a large language model (GPT-3 [7]) and a text-to-image model (Stable Diffusion [52]). These two models capture complementary knowledge about language and images that can be combined to create paired training data for a task spanning both modalities.

Using our generated paired data, we train a conditional diffusion model that, given an input image and a text instruction for how to edit it, generates the edited image. Our model directly performs the image edit in the forward pass, and does not require any additional example images, full descriptions of the input/output images, or per-example fine-tuning. Despite being trained entirely on synthetic examples (i.e., both generated written instructions and generated

1

arXiv:2211.09800v2 [cs.CV] 18 Jan 2023

VAR: image generation via autoregressive models

Today

After: Next batch of four **paper presentations**.

TextGrad: Automatic "Differentiation" via Text

Mert Yuksekgonul[1*] MERTY@STANFORD.EDU
Federico Bianchi[1*] FEDE@STANFORD.EDU
Joseph Boen[2*] TBOEN@STANFORD.EDU
Sheng Liu[2*] SHENGL@STANFORD.EDU
Zhi Huang[2*] ZHIHUANG@STANFORD.EDU
Carlos Guestrin[1,3] GUESTRIN@STANFORD.EDU
James Zou[1,2,3] JAMESZ@STANFORD.EDU
[1]Department of Computer Science, Stanford University
[2]Department of Biomedical Data Science, Stanford University
[3]Chan Zuckerberg Biohub
Correspondence: merty@stanford.edu and jamesz@stanford.edu

Repository and Tutorials

Abstract

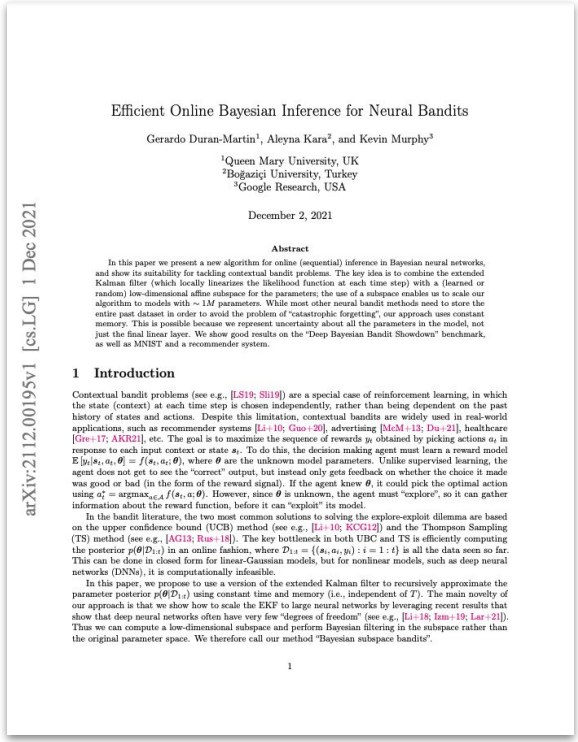
AI is undergoing a paradigm shift, with breakthroughs achieved by systems orchestrating multiple large language models (LLMs) and other complex components. As a result, developing principled and automated optimization methods for compound AI systems is one of the most important new challenges. Neural networks faced a similar challenge in its early days until backpropagation and automatic differentiation transformed the field by making optimization turn-key. Inspired by this, we introduce TextGrad, a powerful framework performing automatic "differentiation" via text. TextGrad backpropagates textual feedback provided by LLMs to improve individual components of a compound AI system. In our framework, LLMs provide rich, general, natural language suggestions to optimize variables in computation graphs, ranging from code snippets to molecular structures. TextGrad follows PyTorch's syntax and abstraction and is flexible and easy-to-use. It works out-of-the-box for a variety of tasks, where the users only provide the objective function without tuning components or prompts of the framework. We showcase TextGrad's effectiveness and generality across a diverse range of applications, from question answering and molecule optimization to radiotherapy treatment planning. Without modifying the framework, TextGrad improves the zero-shot accuracy of GPT-4o in Google-Proof Question Answering from 51% to 55%, yields 20% relative performance gain in optimizing LeetCode-Hard coding problem solutions, improves prompts for reasoning, designs new druglike small molecules with desirable *in silico* binding, and designs radiation oncology treatment plans with high specificity. TextGrad lays a foundation to accelerate the development of the next-generation of AI systems.

1 Introduction

There is an emerging paradigm shift in how AI systems are built, owing to the breakthroughs of Large Language Models (LLMs) [1–6]. The new generation of AI applications are increasingly compound systems involving multiple sophisticated components, where each component could be an LLM-based agent, a tool such as a simulator, or web search. For instance, a system of LLMs communicating with symbolic solvers can solve olympiad-level math problems [7]; a system of LLMs using search engines and code interpreter tools performs comparably to human competitive programmers [8] and are solving real-world

*Co-first authors.

1



Efficient Online Bayesian Inference for Neural Bandits

Gerardo Duran-Martin[1], Aleyna Kara[2], and Kevin Murphy[3]
[1]Queen Mary University, UK
[2]Boğaziçi University, Turkey
[3]Google Research, USA

December 2, 2021

Abstract

In this paper we present a new algorithm for online (sequential) inference in Bayesian neural networks, and show its suitability for tackling contextual bandit problems. The key idea is to combine the extended Kalman filter (which locally linearizes the likelihood function at each time step) with a (learned or random) low-dimensional affine subspace for the parameters; the use of a subspace enables us to scale our algorithm to models with $\sim 1M$ parameters. While most other neural bandit methods need to store the entire past dataset in order to avoid the problem of "catastrophic forgetting", our approach uses constant memory. This is possible because we represent uncertainty about all the parameters in the model, not just the final linear layer. We show good results on the "Deep Bayesian Bandit Showdown" benchmark, as well as MNIST and a recommender system.

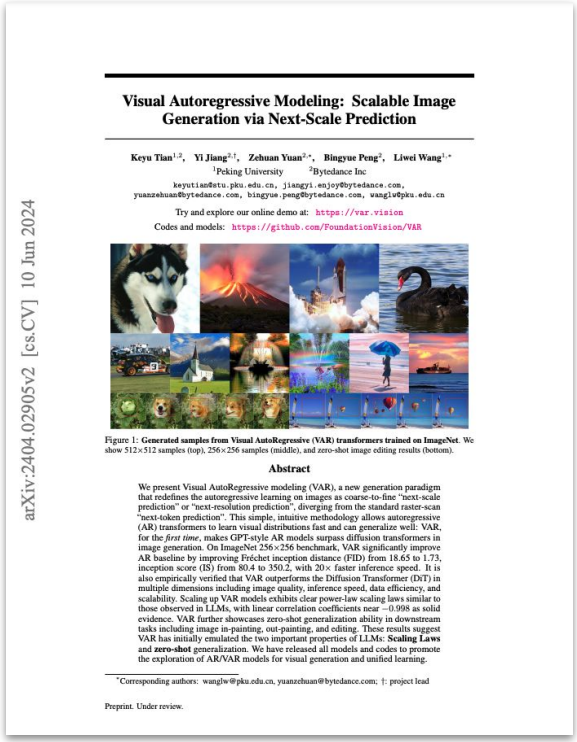
1 Introduction

Contextual bandit problems (see e.g., [LS19; Sli19]) are a special case of reinforcement learning, in which the state (context) at each time step is chosen independently, rather than being dependent on the past history of states and actions. Despite this limitation, contextual bandits are widely used in real-world applications, such as recommender systems [Li+10; Guo+20], advertising [McM+13; Du+21], healthcare [Gre+17; AKR21], etc. The goal is to maximize the sequence of rewards y_t obtained by picking actions a_t in response to each input context or state $\boldsymbol{s}_t$. To do this, the decision making agent must learn a reward model $\mathbb{E}\left[y_t|\boldsymbol{s}_t, a_t, \boldsymbol{\theta}\right] = f(\boldsymbol{s}_t, a_t; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ are the unknown model parameters. Unlike supervised learning, the agent does not get to see the "correct" output, but instead only gets feedback on whether the choice it made was good or bad (in the form of the reward signal). If the agent knew $\boldsymbol{\theta}$, it could pick the optimal action using $a_t^* = \text{argmax}_{a \in \mathcal{A}} f(\boldsymbol{s}_t, a; \boldsymbol{\theta})$. However, since $\boldsymbol{\theta}$ is unknown, the agent must "explore", so it can gather information about the reward function, before it can "exploit" its model.

In the bandit literature, the two most common solutions to solving the explore-exploit dilemma are based on the upper confidence bound (UCB) method (see e.g., [Li+10; KCG12]) and the Thompson Sampling (TS) method (see e.g., [AG13; Rus+18]). The key bottleneck in both UBC and TS is efficiently computing the posterior $p(\boldsymbol{\theta}|\mathcal{D}_{1:t})$ in an online fashion, where $\mathcal{D}_{1:t} = \{(\boldsymbol{s}_i, a_i, y_i) : i = 1 : t\}$ is all the data seen so far. This can be done in closed form for linear-Gaussian models, but for nonlinear models, such as deep neural networks (DNNs), it is computationally infeasible.

In this paper, we propose to use a version of the extended Kalman filter to recursively approximate the parameter posterior $p(\boldsymbol{\theta}|\mathcal{D}_{1:t})$ using constant time and memory (i.e., independent of T). The main novelty of our approach is that we show how to scale the EKF to large neural networks by leveraging recent results that show that deep neural networks often have very few "degrees of freedom" (see e.g., [Li+18; Izm+19; Lar+21]). Thus we can compute a low-dimensional subspace and perform Bayesian filtering in the subspace rather than the original parameter space. We therefore call our method "Bayesian subspace bandits".

1



Visual Autoregressive Modeling: Scalable Image Generation via Next-Scale Prediction

Keyu Tian[1,2], Yi Jiang[2,†], Zehuan Yuan[2,*], Bingyue Peng[2], Liwei Wang[1,*]
[1]Peking University [2]Bytedance Inc
keyutian@stu.pku.edu.cn, jiangyi.enjoy@bytedance.com,
yuanzehuan@bytedance.com, bingyue.peng@bytedance.com, wanglw@pku.edu.cn

Try and explore our online demo at: https://var.vision
Codes and models: https://github.com/FoundationVision/VAR

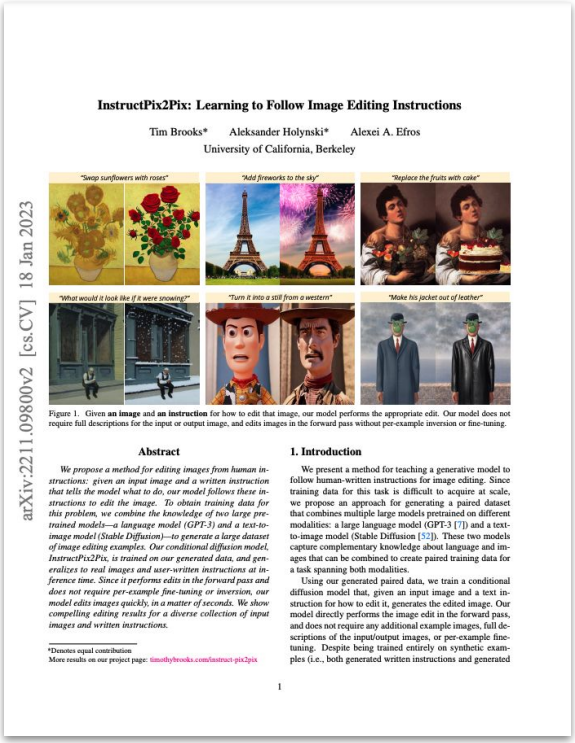
Figure 1: **Generated samples from Visual AutoRegressive (VAR) transformers trained on ImageNet**. We show 512×512 samples (top), 256×256 samples (middle), and zero-shot image editing results (bottom).

Abstract

We present Visual AutoRegressive modeling (VAR), a new generation paradigm that redefines the autoregressive learning on images as coarse-to-fine "next-scale prediction" or "next-resolution prediction", diverging from the standard raster-scan "next-token prediction". This simple, intuitive methodology allows autoregressive (AR) transformers to learn visual distributions fast and can generalize well: VAR, for the *first time*, makes GPT-style AR models surpass diffusion transformers in image generation. On ImageNet 256×256 benchmark, VAR significantly improve AR baseline by improving Fréchet inception distance (FID) from 18.65 to 1.73, inception score (IS) from 80.4 to 350.2, with 20× faster inference speed. It is also empirically verified that VAR outperforms the Diffusion Transformer (DiT) in multiple dimensions including image quality, inference speed, data efficiency, and scalability. Scaling up VAR models exhibits clear power-law scaling laws similar to those observed in LLMs, with linear correlation coefficients near −0.998 as solid evidence. VAR further showcases zero-shot generalization ability in downstream tasks including image in-painting, out-painting, and editing. These results suggest VAR has initially emulated the two important properties of LLMs: **Scaling Laws** and **zero-shot** generalization. We have released all models and codes to promote the exploration of AR/VAR models for visual generation and unified learning.

*Corresponding authors: wanglw@pku.edu.cn, yuanzehuan@bytedance.com; †: project lead

Preprint. Under review.



InstructPix2Pix: Learning to Follow Image Editing Instructions

Tim Brooks* Aleksander Holynski* Alexei A. Efros
University of California, Berkeley

Figure 1. Given **an image** and **an instruction** for how to edit that image, our model performs the appropriate edit. Our model does not require full descriptions for the input or output image, and edits images in the forward pass without per-example inversion or fine-tuning.

Abstract

We propose a method for editing images from human instructions: given an input image and a written instruction that tells the model what to do, our model follows these instructions to edit the image. To obtain training data for this problem, we combine the knowledge of two large pre-trained models—a language model (GPT-3) and a text-to-image model (Stable Diffusion)—to generate a large dataset of image editing examples. Our conditional diffusion model, InstructPix2Pix, is trained on our generated data, and generalizes to real images and user-written instructions at inference time. Since it performs edits in the forward pass and does not require per-example fine-tuning or inversion, our model edits images quickly, in a matter of seconds. We show compelling editing results for a diverse collection of input images and written instructions.

*Denotes equal contribution
More results on our project page: timothybrooks.com/instruct-pix2pix

I. Introduction

We present a method for teaching a generative model to follow human-written instructions for image editing. Since training data for this task is difficult to acquire at scale, we propose an approach for generating a paired dataset that combines multiple large models pretrained on different modalities: a large language model (GPT-3 [7]) and a text-to-image model (Stable Diffusion [52]). These two models capture complementary knowledge about language and images that can be combined to create paired training data for a task spanning both modalities.

Using our generated paired data, we train a conditional diffusion model that, given an input image and a text instruction for how to edit it, generates the edited image. Our model directly performs the image edit in the forward pass, and does not require any additional example images, full descriptions of the input/output images, or per-example fine-tuning. Despite being trained entirely on synthetic examples (i.e., both generated written instructions and generated

1

InstructPix2Pix: model to perform edits on images

Quick Aside

Midway Reports and Grading:

- We (Oliver and I) finished reviewing through and grading all midway reports.
  - Everyone did a nice job :D!
- We have a few light comments on each report — which could be useful for final report.
  - We will likely send these via our group message threads on slack.
- And will also upload grades on Brightspace.

Quick Aside

Reminder — Final Course Project Timeline

- Final Presentations (30 mins long)
  - **April 25** - Groups 1-5.
  - **May 02** - Groups 6-9.

Quick Aside

Reminder — Final Course Project Timeline

- Final Presentations (30 mins long)
  - **April 25** - Groups 1-5.
  - **May 02** - Groups 6-9.

- Final Report (8-10 pages, extended version of midway report)
  - (LaTeX template will be provided soon).
  - **Due May 9, EOD**

Review: Predictive UQ and Gaussian Processes

Review — Gaussian Processes

Review — Gaussian Processes

Intuitively, a *GP* is a probabilistic model, which can be viewed as a distribution over functions.

Review — Gaussian Processes

Intuitively, a *GP* is a probabilistic model, which can be viewed as a distribution over functions.

But it uses a (multivariate) **Gaussian distribution** to model these functions 🤯!

Review — Gaussian Processes

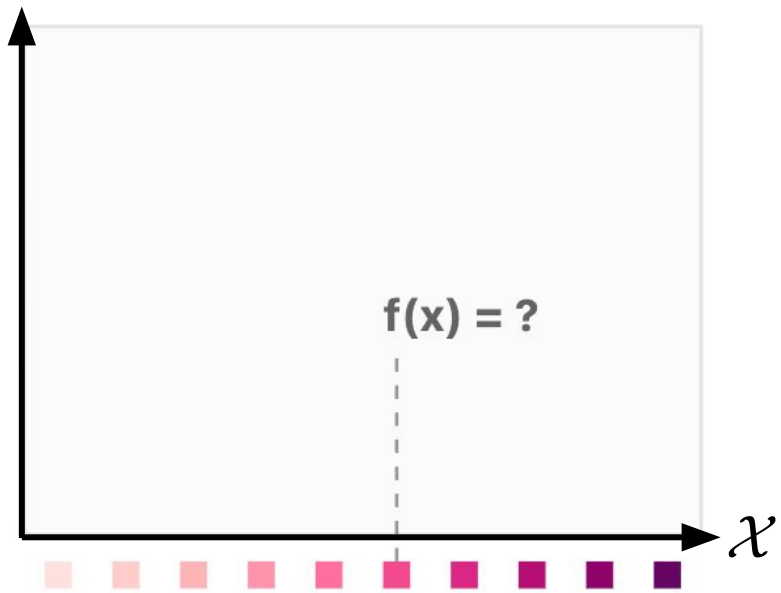
Intuitively, a *GP* is a probabilistic model, which can be viewed as a distribution over functions.

But it uses a (multivariate) **Gaussian distribution** to model these functions 🤯!

Let’s visualize this…

Review — Gaussian Processes

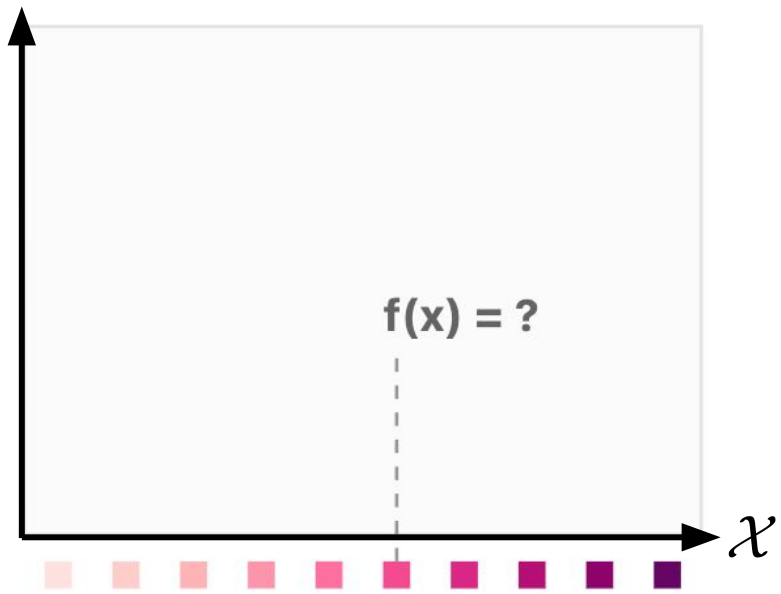
Suppose we pick a set of *d=10* points on the *x*-axis, and want to define a **function** (or set of functions) over these points:



Source: A Visual Exploration of Gaussian Processes

Review — Gaussian Processes

Suppose we pick a set of *d=10* points on the *x*-axis, and want to define a **function** (or set of functions) over these points:

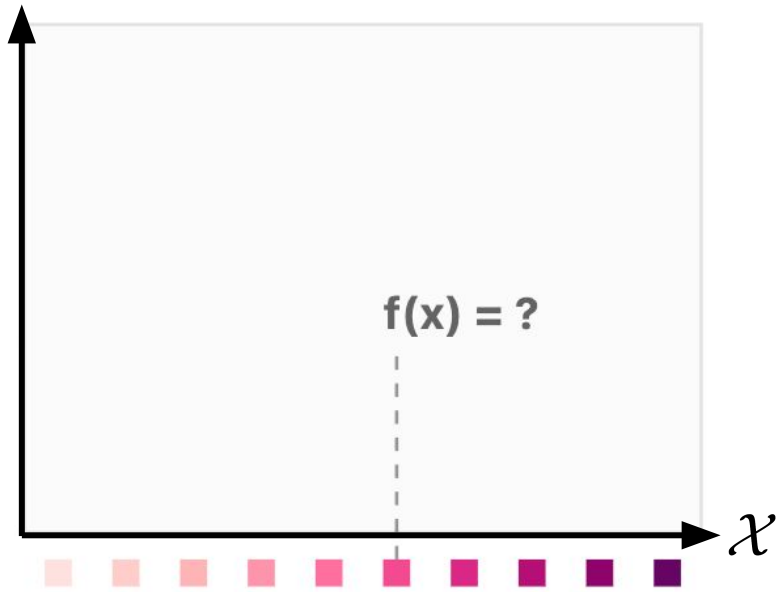


10 points are denoted by different colors

Source: A Visual Exploration of Gaussian Processes

Review — Gaussian Processes

Suppose we pick a set of *d=10* points on the *x*-axis, and want to define a **function** (or set of functions) over these points:



10 points are denoted by different colors

We can define a *distribution of functions*, defined on these 10 points, in the following way…

Source: A Visual Exploration of Gaussian Processes

Review — Gaussian Processes

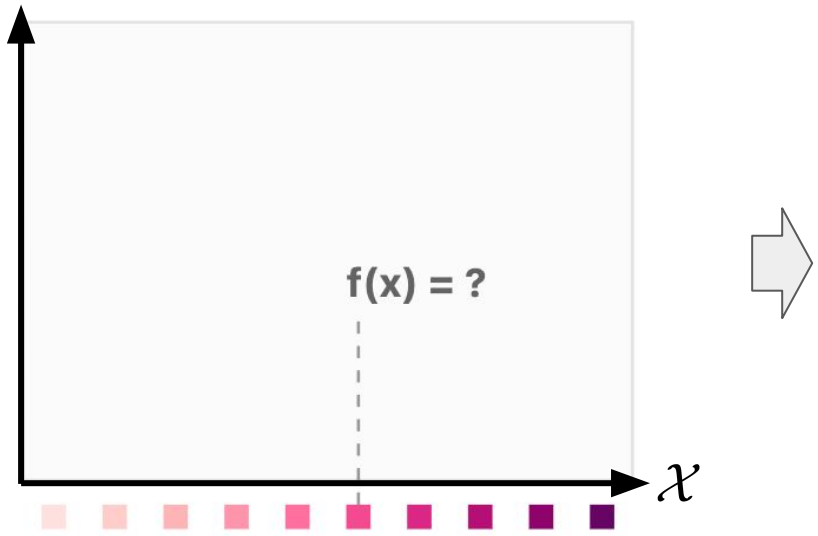
We define a MVN of ***d* dimensions**, with a mean and covariance matrix parameter.

Mean parameter $\mu \in \mathbb{R}^d$ — assume we set this to ***0*** for now.

Covariance matrix $\sum \in \mathbb{R}^{d \times d}$ — define using a *kernel function* based on distance between input points (next slide).

Review — Gaussian Processes

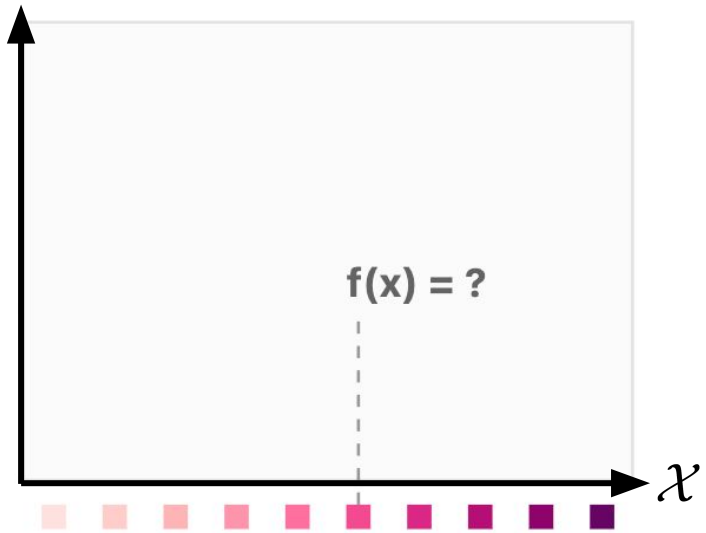
For these 10 points:



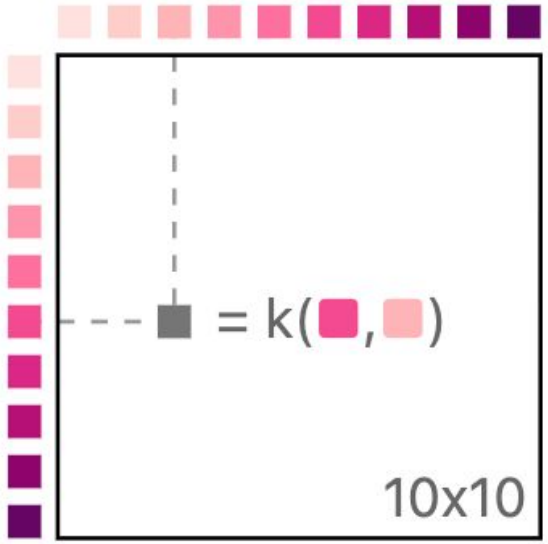
Source: A Visual Exploration of Gaussian Processes

Review — Gaussian Processes

For these 10 points:



Covariance matrix:



Define covariance between any pair of inputs using kernel function $k(\cdot, \cdot)$.

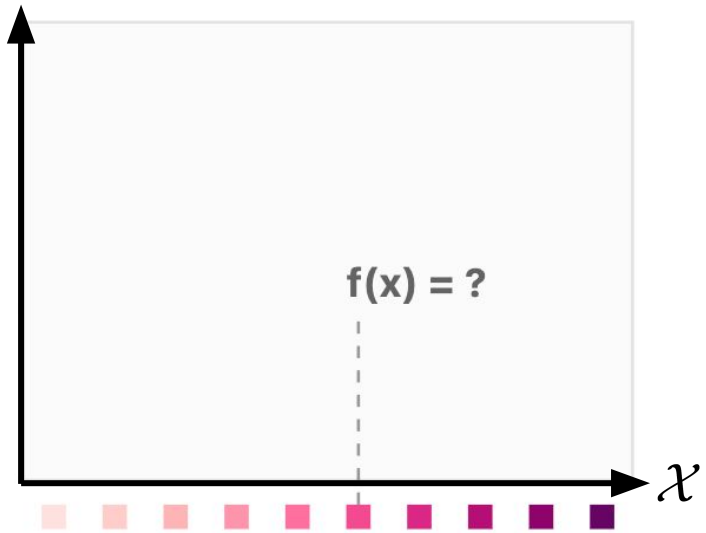
One example: *RBF kernel*,

$$k(x_1, x_2) = \sigma^2 \left(-\frac{\|x_1 - x_2\|^2}{2\ell^2} \right)$$

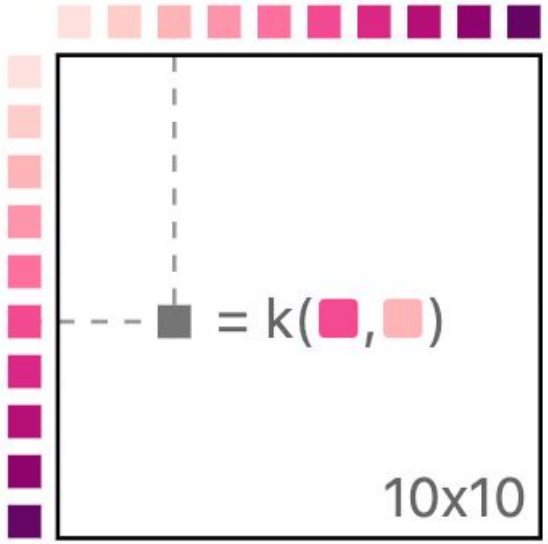
Source: A Visual Exploration of Gaussian Processes

Review — Gaussian Processes

For these 10 points:



Covariance matrix:



Define covariance between any pair of inputs using kernel function $k(\cdot, \cdot)$.

One example: *RBF kernel*,

$$k(x_1, x_2) = \sigma^2 \left(-\frac{\|x_1 - x_2\|^2}{2\ell^2} \right)$$

Highest when $x_1 = x_2$, lowest when x_2 very different from x_2.

Source: A Visual Exploration of Gaussian Processes

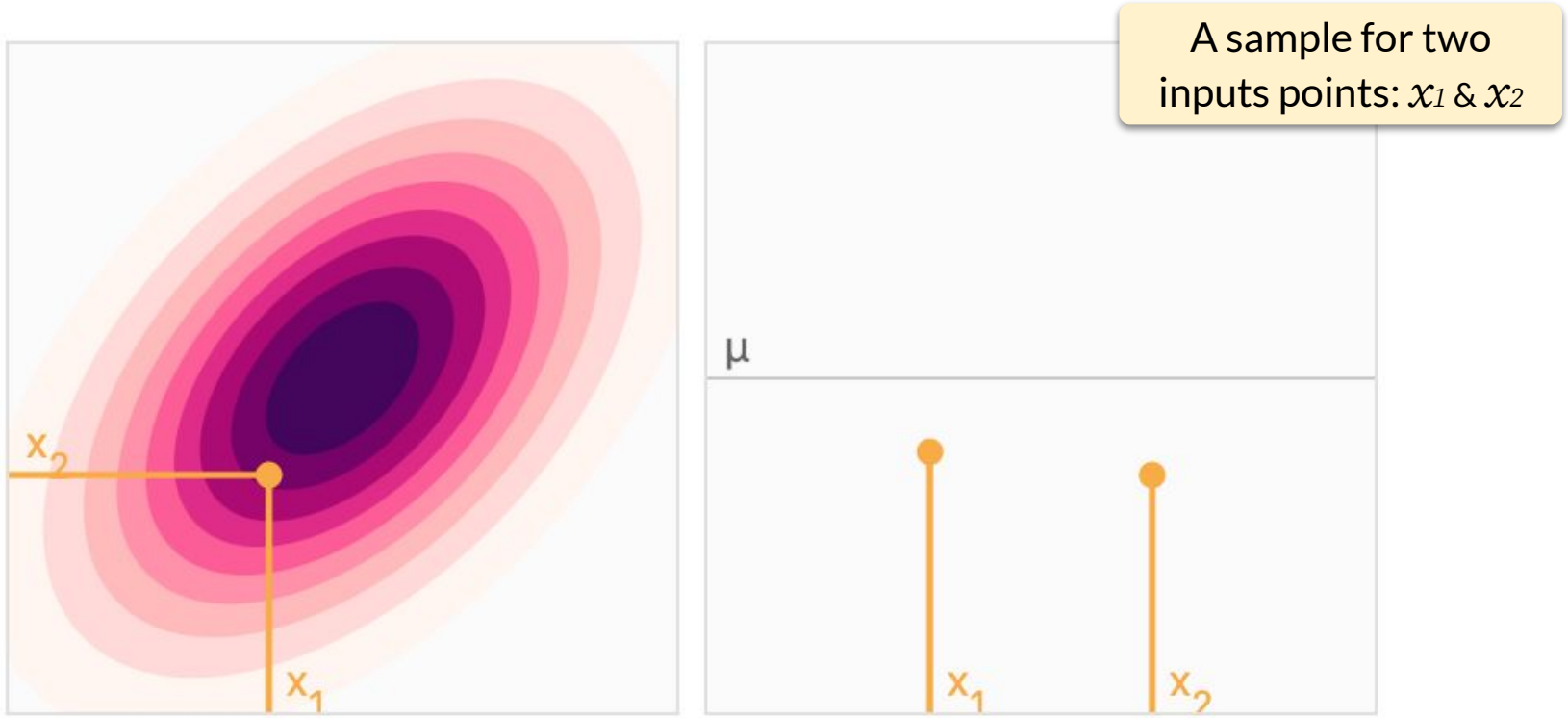
Review — Gaussian Processes

This yields a MVN, where samples give function values for the *d* input points:

Source: A Visual Exploration of Gaussian Processes

Review — Gaussian Processes

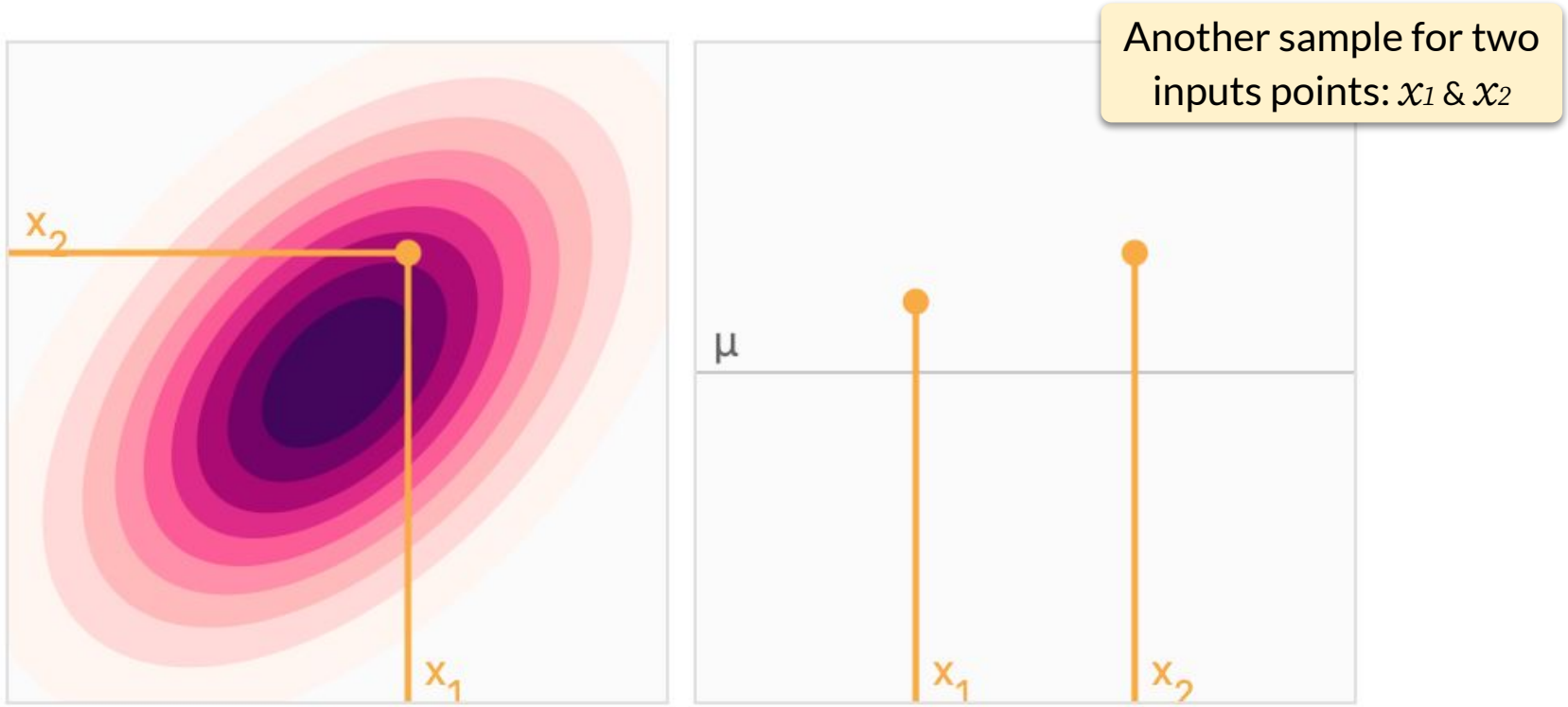
This yields a MVN, where samples give function values for the *d* input points:



Source: A Visual Exploration of Gaussian Processes

Review — Gaussian Processes

This yields a MVN, where samples give function values for the *d* input points:

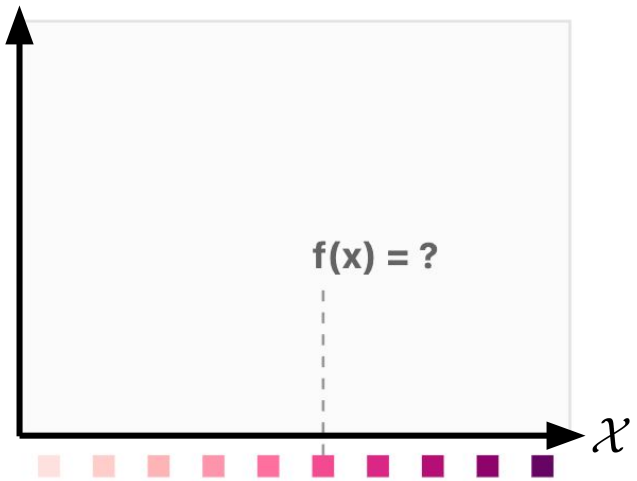


Source: A Visual Exploration of Gaussian Processes

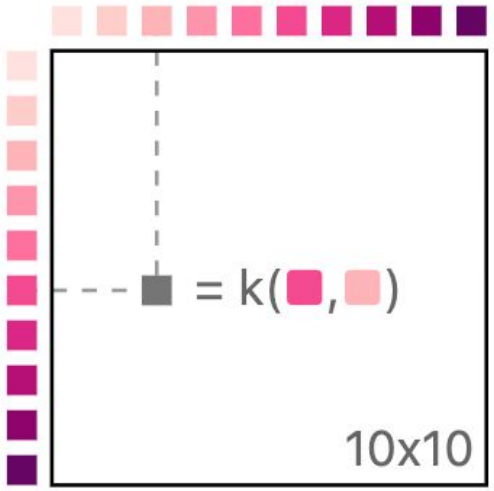
Review — Gaussian Processes

This yields a MVN, where samples give function values for the *d* input points:

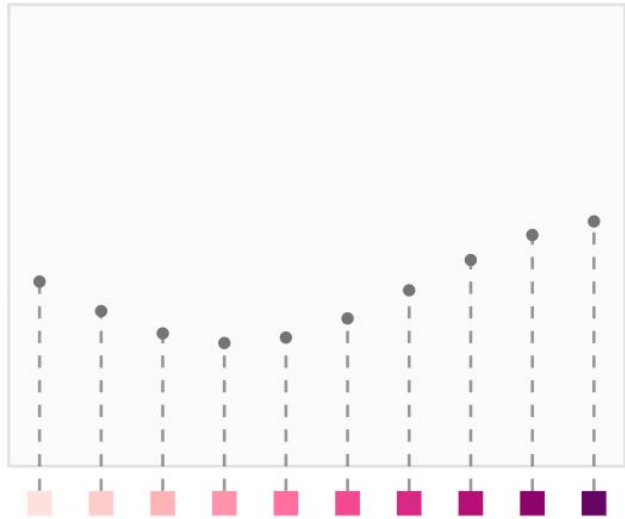
For these 10 points:



Covariance matrix:



Sampled function values



Source: A Visual Exploration of Gaussian Processes

Review — Gaussian Processes

We define a **Gaussian Process (GP)** as a distribution over functions, parameterized by a *mean function* $m(\cdot)$ and *covariance kernel function* $k(\cdot, \cdot)$, written

$$f \sim \mathrm{GP}(m(\cdot), k(\cdot, \cdot))$$

Review — Gaussian Processes

We define a **Gaussian Process (GP)** as a distribution over functions, parameterized by a *mean function* $m(\cdot)$ and *covariance kernel function* $k(\cdot, \cdot)$, written

$$f \sim \mathrm{GP}(m(\cdot), k(\cdot, \cdot))$$

Where, for any finite collection of *d* input points, we can:

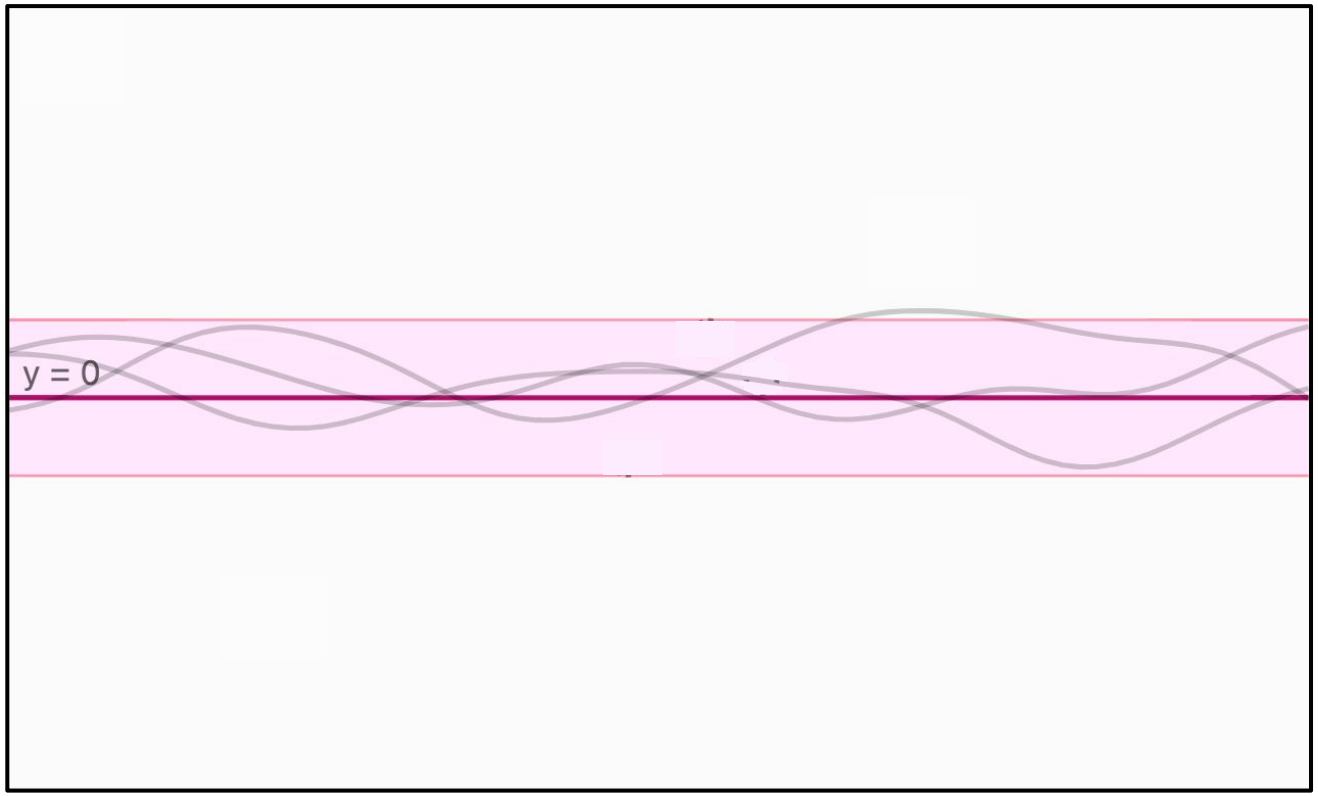
- Define a MVN *on these d points*, using the mean/kernel functions to define a mean parameter and covariance matrix.
- And this MVN yields a distribution over function values.

Review — Gaussian Processes

What does this distribution look like?

Review — Gaussian Processes

What does this distribution look like? → **often drawn like this:**

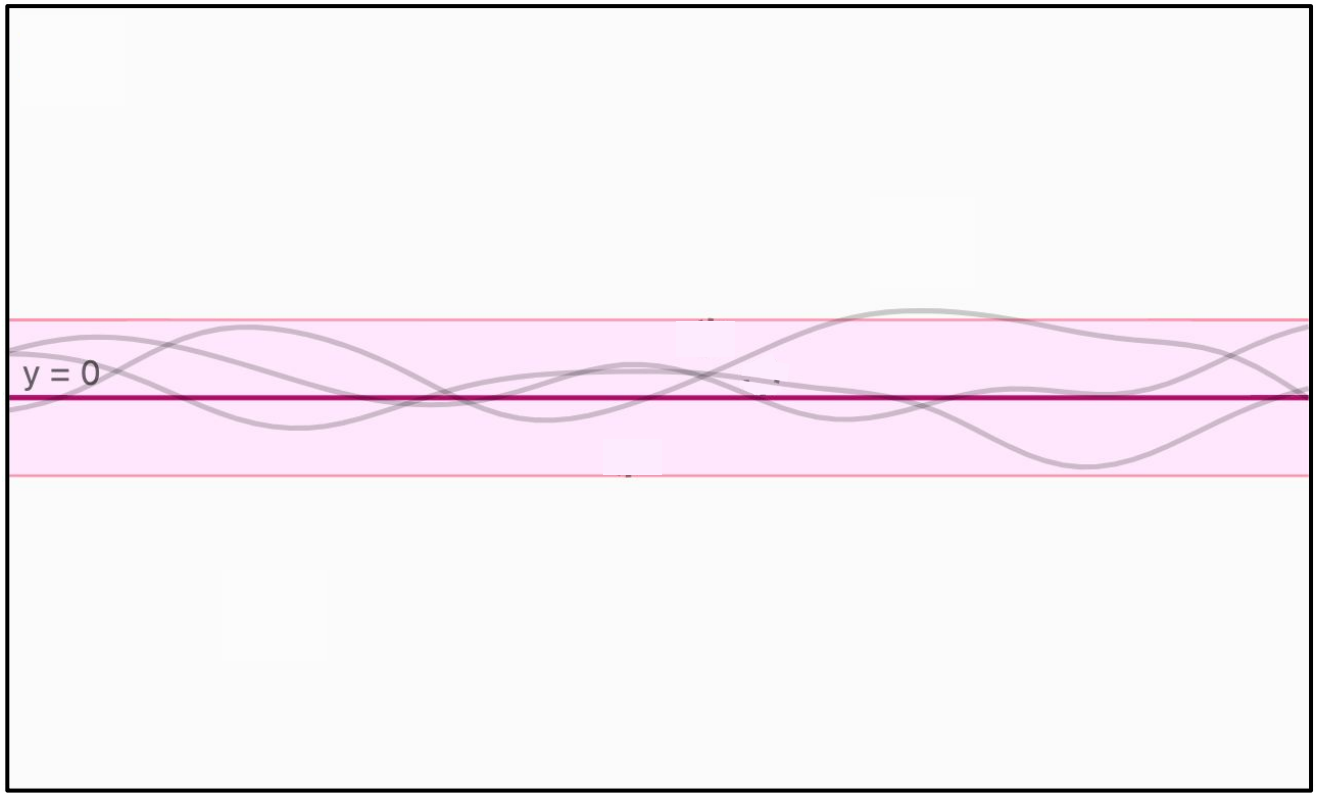


Source: A Visual Exploration of Gaussian Processes

Review — Gaussian Processes

Called a *Sausage Plot* because it looks like a sausage

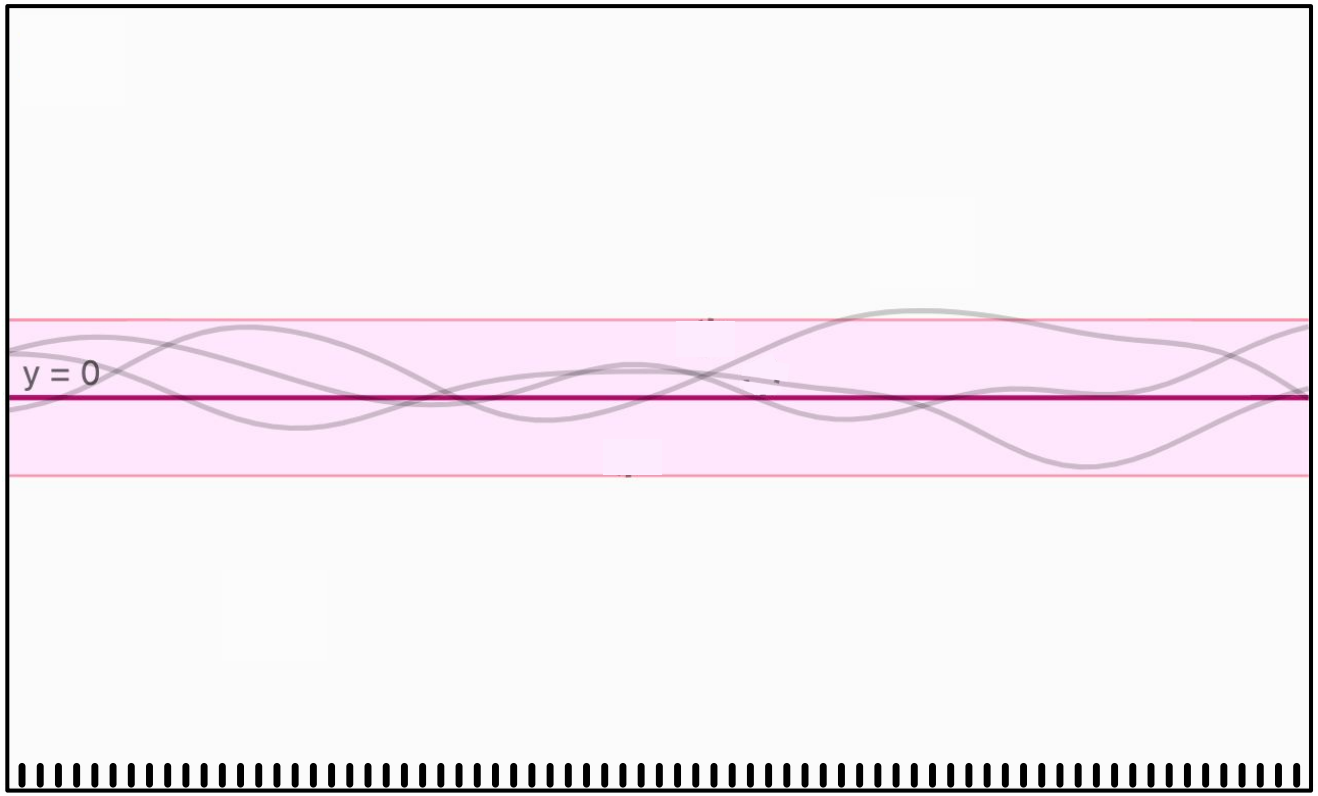
What does this distribution look like? → **often drawn like this:**



Source: A Visual Exploration of Gaussian Processes

Review — Gaussian Processes

What does this distribution look like? → **often drawn like this:**

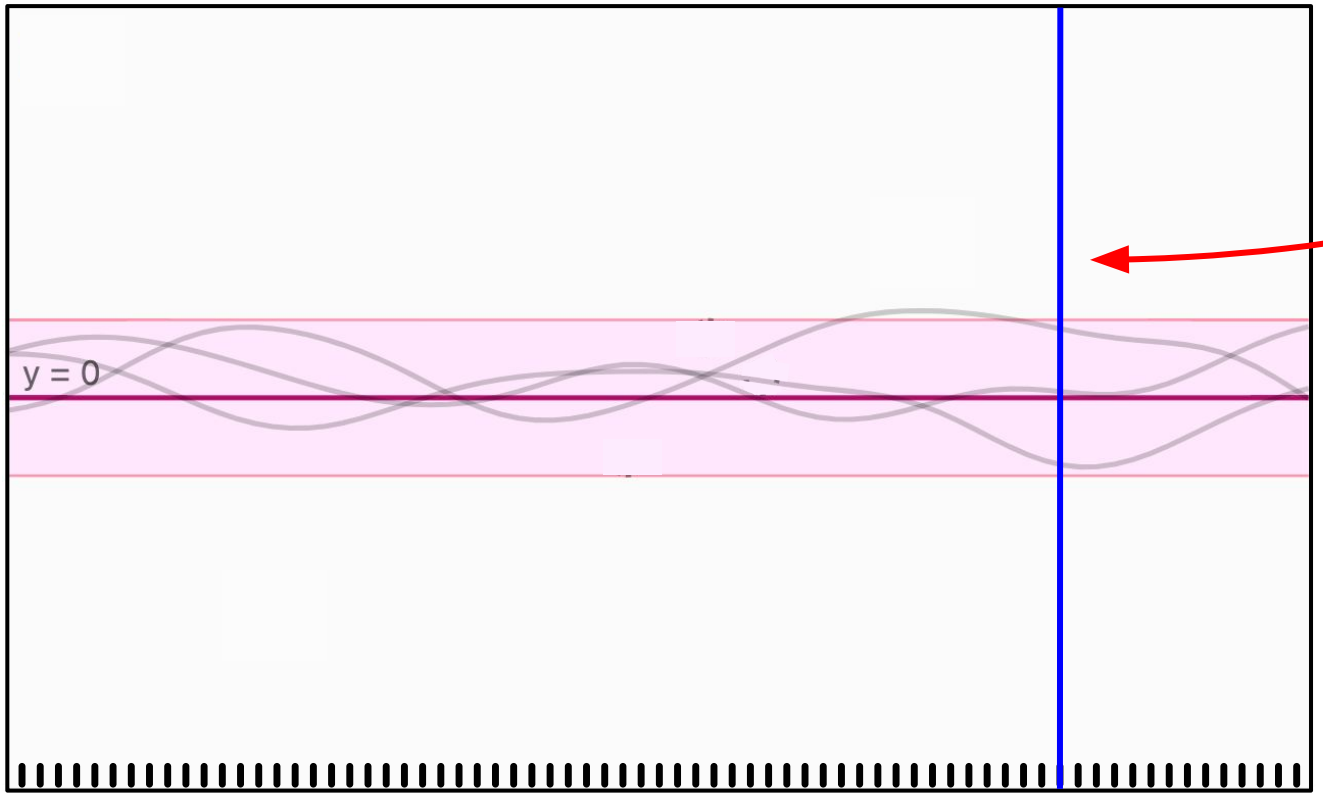


This plot depicts a MVN defined on a large set of grid points

Source: A Visual Exploration of Gaussian Processes

Review — Gaussian Processes

What does this distribution look like? → **often drawn like this:**



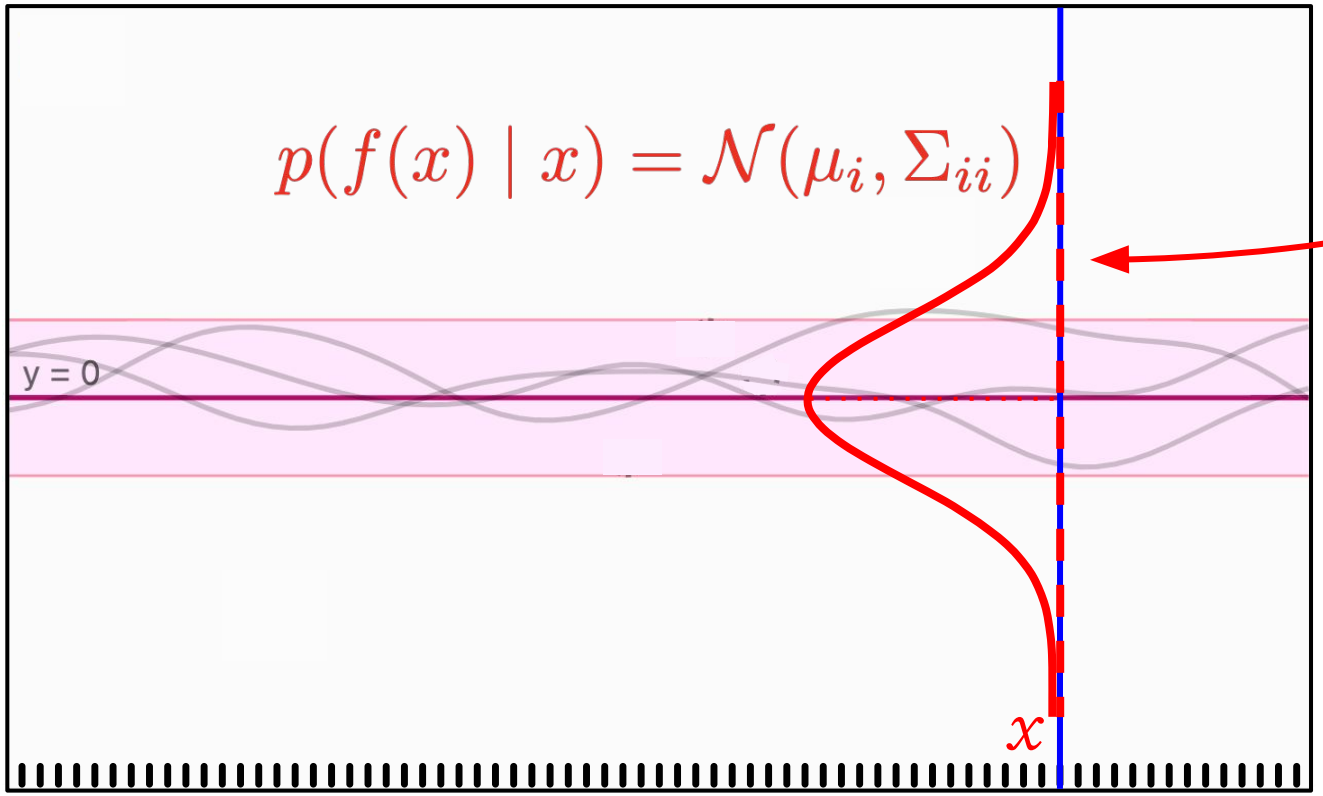
For any grid point, can compute a marginal distribution of MVN

This plot depicts a MVN defined on a large set of grid points

Source: A Visual Exploration of Gaussian Processes

Review — Gaussian Processes

What does this distribution look like? → **often drawn like this:**



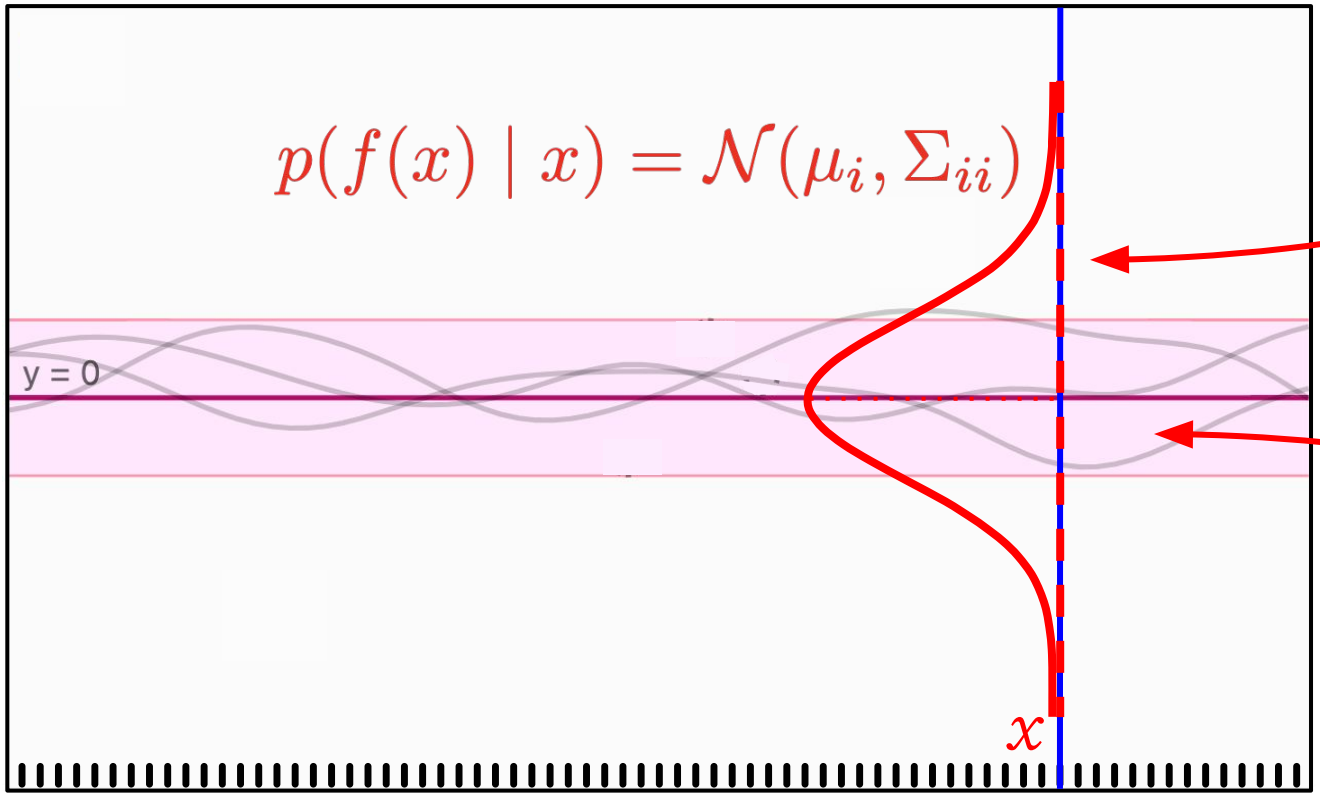
For any grid point, can compute a marginal distribution of MVN

This plot depicts a MVN defined on a large set of grid points

Source: A Visual Exploration of Gaussian Processes

Review — Gaussian Processes

What does this distribution look like? → **often drawn like this:**



For any grid point, can compute a marginal distribution of MVN

This plot depicts a MVN defined on a large set of grid points

Samples from the MVN are shown in grey (appear as functions!)

Source: A Visual Exploration of Gaussian Processes

Review — Gaussian Process Posterior

How do we use a GP as a predictive uncertainty model?

Review — Gaussian Process Posterior

How do we use a GP as a predictive uncertainty model?

⇒ Use it as a prior distribution and do Bayesian inference!

Review — Gaussian Process Posterior

How do we use a GP as a predictive uncertainty model?

⇒ Use it as a prior distribution and do Bayesian inference!

In particular, we will:

- Use a GP to define a prior over functions.
- Observe a set of (x, y) pairs.
- Compute a posterior GP over functions.

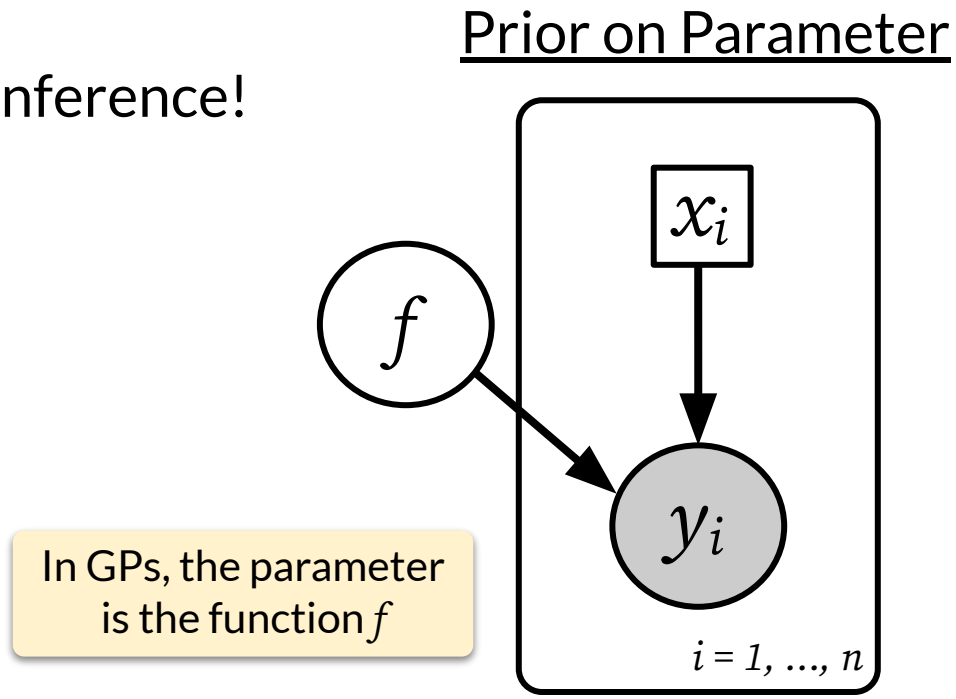
Review — Gaussian Process Posterior

How do we use a GP as a predictive uncertainty model?

⇒ Use it as a prior distribution and do Bayesian inference!

In particular, we will:

- Use a GP to define a prior over functions.
- Observe a set of (x, y) pairs.
- Compute a posterior GP over functions.



Review — Gaussian Process Posterior

Computing the posterior:

Review — Gaussian Process Posterior

Computing the posterior:

- We have a GP prior *(essentially a multivariate normal)* : $f \sim \mathrm{GP}(m(\cdot), k(\cdot, \cdot))$

Review — Gaussian Process Posterior

Computing the posterior:

- We have a GP prior *(essentially a multivariate normal)* : $f \sim \mathrm{GP}(m(\cdot), k(\cdot, \cdot))$
- We have a Gaussian likelihood for observations: $y \sim p(y \mid f\ ;\ x) = \mathcal{N}(f, \sigma^2)$

Review — Gaussian Process Posterior

Computing the posterior:

- We have a GP prior *(essentially a multivariate normal)* : $f \sim \mathrm{GP}(m(\cdot), k(\cdot, \cdot))$
- We have a Gaussian likelihood for observations: $y \sim p(y \mid f\ ;\ x) = \mathcal{N}(f, \sigma^2)$
- ⇒ Posterior is *(essentially)* also a multivariate normal distribution.

Review — Gaussian Process Posterior

Computing the posterior:

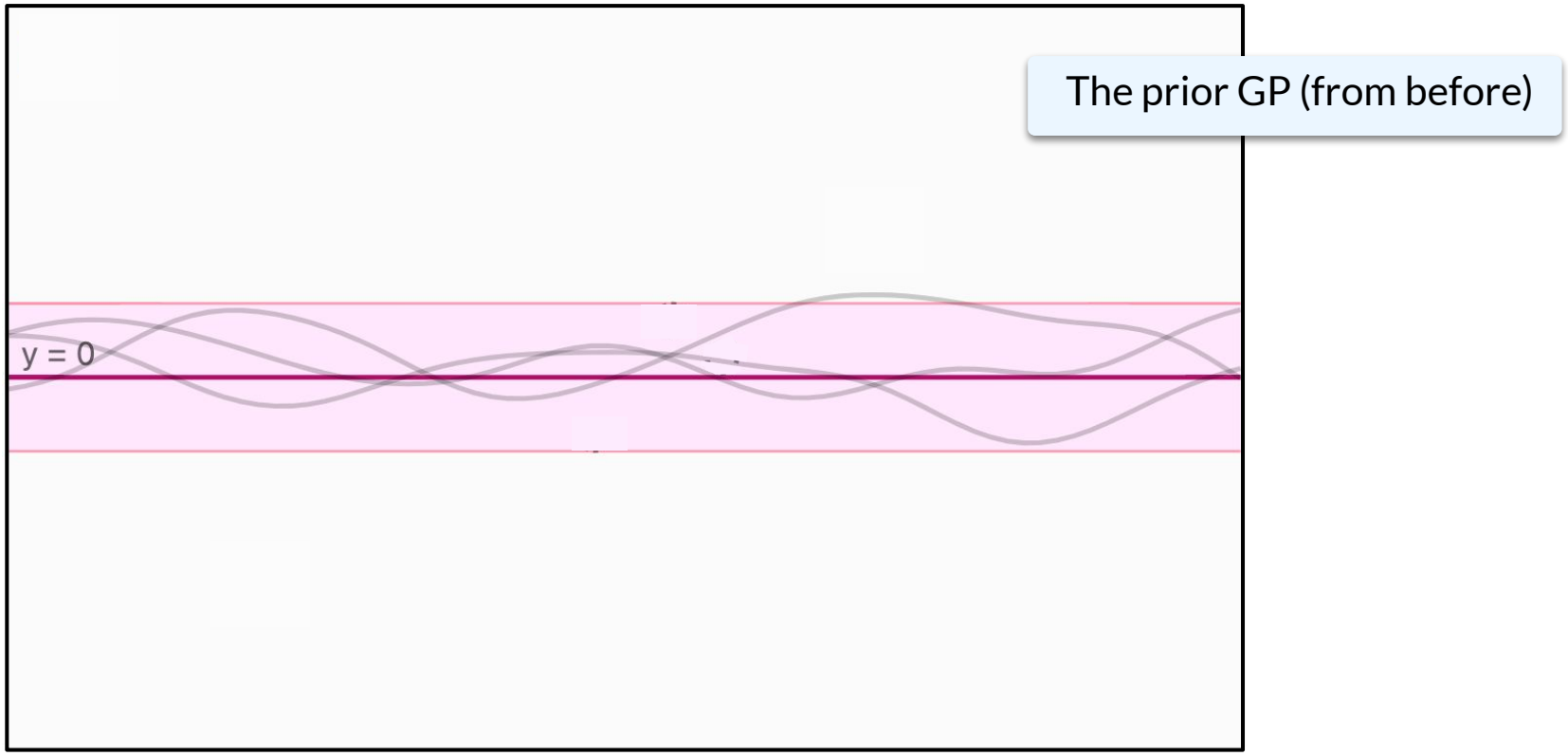
- We have a GP prior *(essentially a multivariate normal)* : $f \sim \mathrm{GP}(m(\cdot), k(\cdot, \cdot))$
- We have a Gaussian likelihood for observations: $y \sim p(y \mid f\ ;\ x) = \mathcal{N}(f, \sigma^2)$
- ⇒ Posterior is *(essentially)* also a multivariate normal distribution.

Posterior is technically a **posterior Gaussian process**, so it's another GP with a mean function and covariance kernel parameters

⇒ which can be computed in closed form.

Review — Gaussian Process Posterior

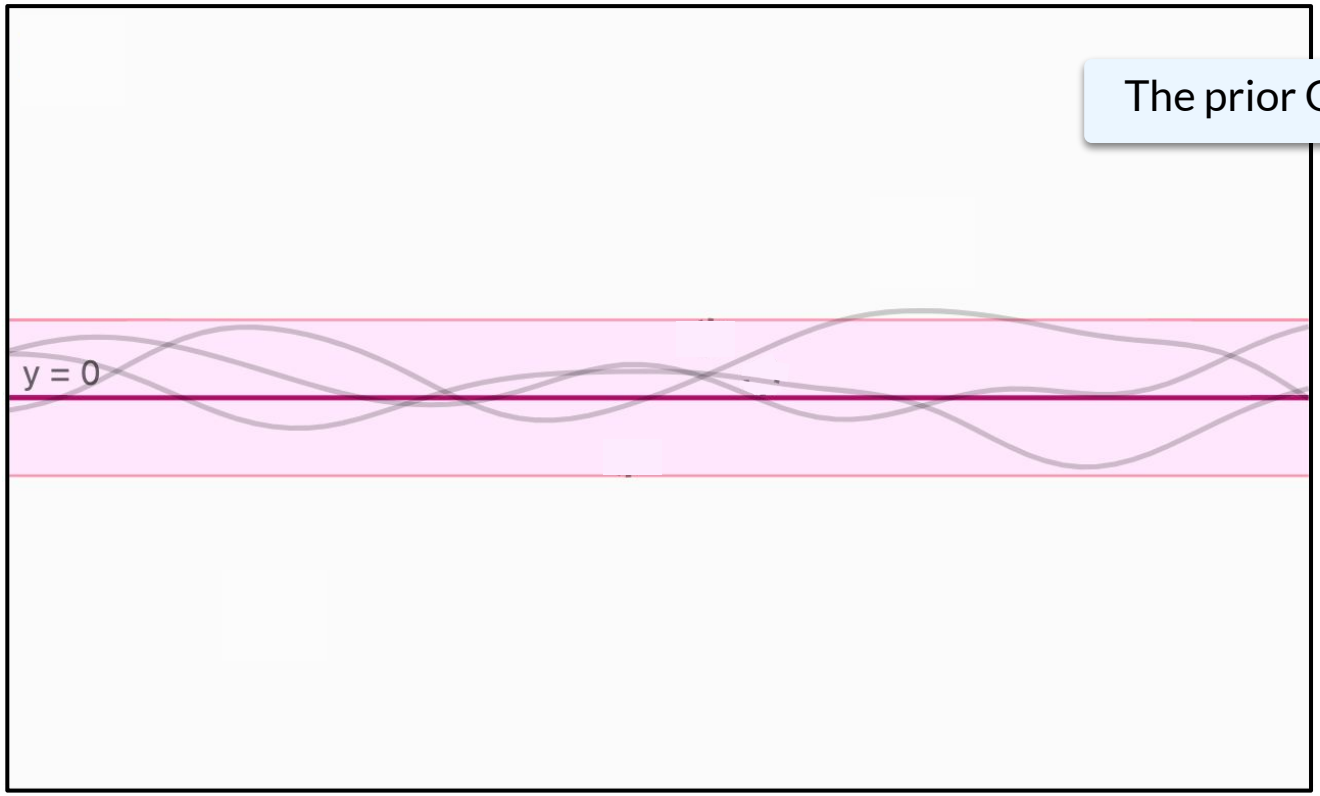
Visualizing this:



Source: A Visual Exploration of Gaussian Processes

Review — Gaussian Process Posterior

Visualizing this:



The prior GP (from before)

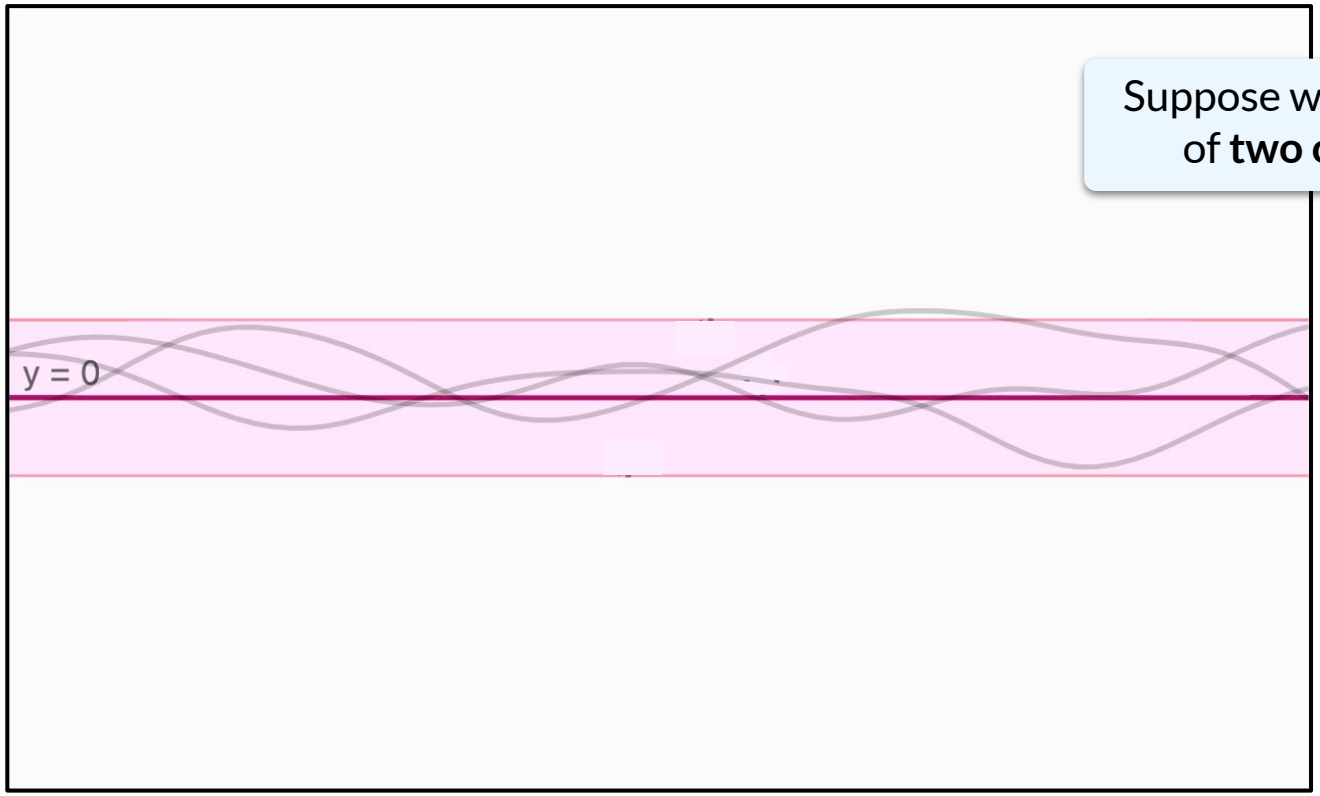
Mean function is constant **0** function

Source: A Visual Exploration of Gaussian Processes

Review — Gaussian Process Posterior

Visualizing this:

Mean function is constant **0** function

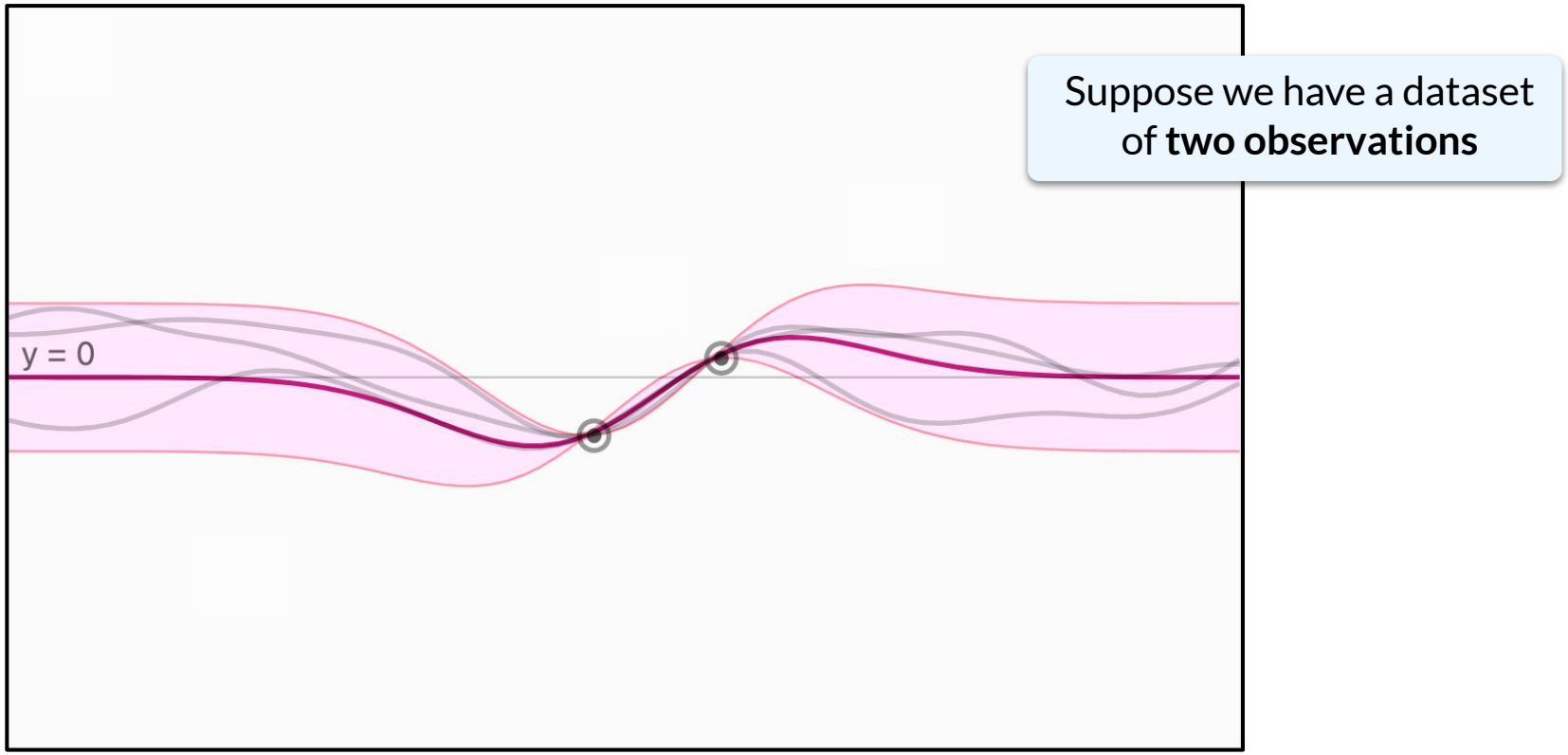


Suppose we have a dataset of **two observations**

Source: A Visual Exploration of Gaussian Processes

Review — Gaussian Process Posterior

Visualizing this:

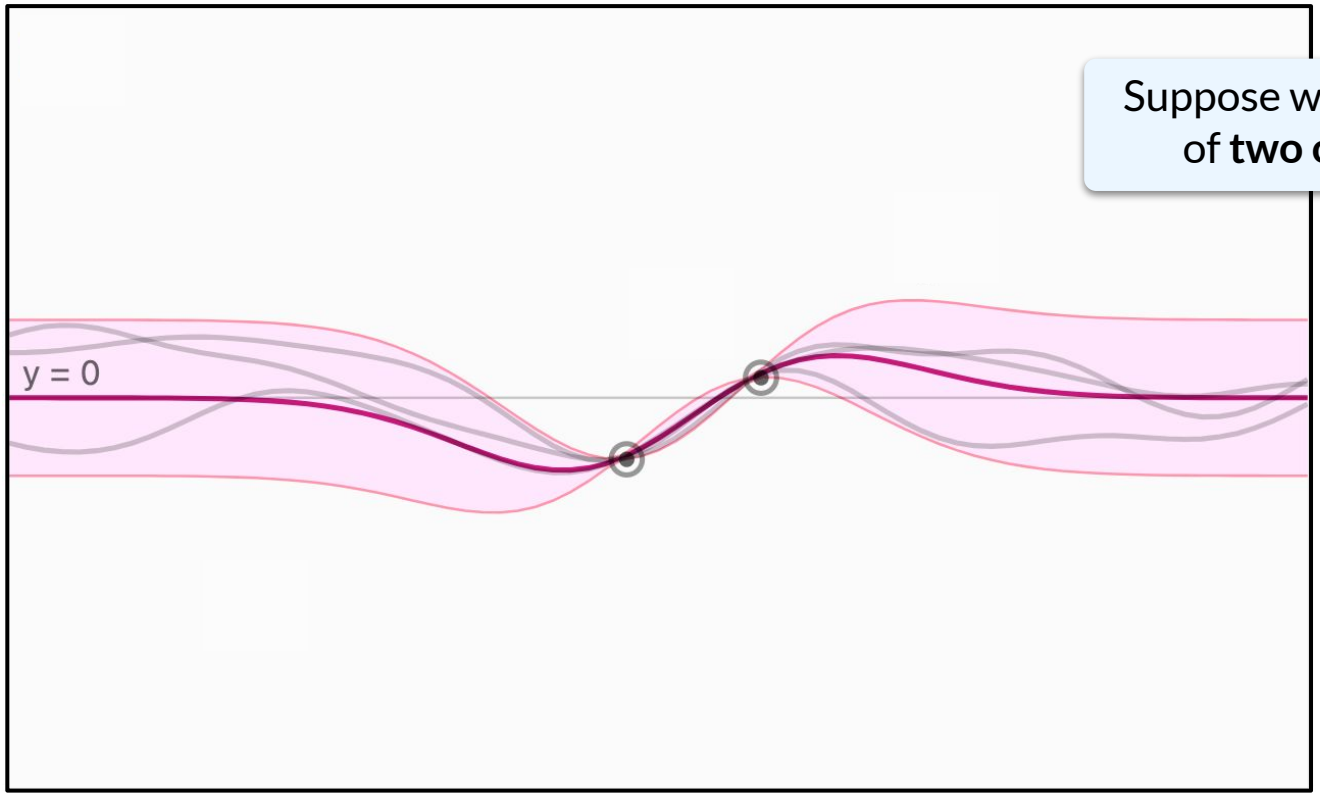


Source: A Visual Exploration of Gaussian Processes

Review — Gaussian Process Posterior

Visualizing this:

Mean function is now **not constant**

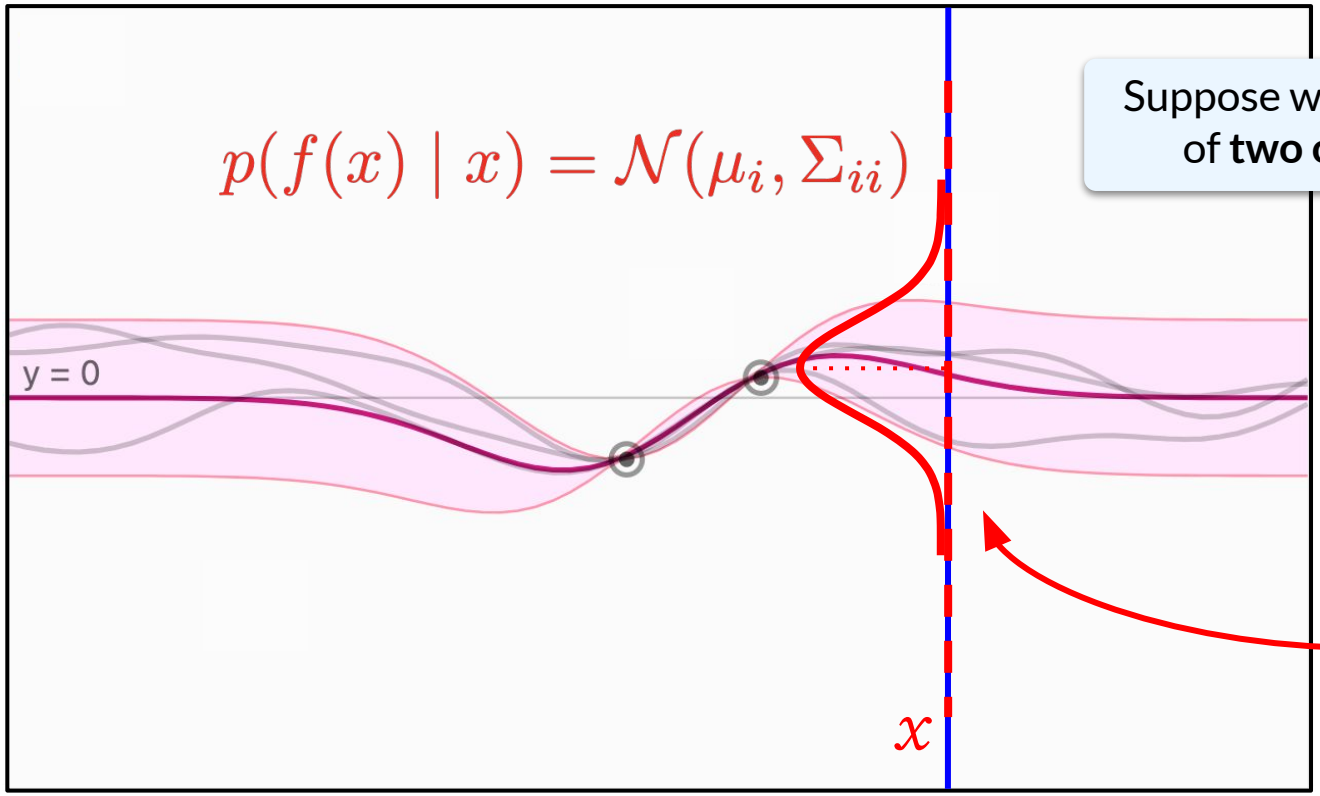


Suppose we have a dataset of **two observations**

Source: A Visual Exploration of Gaussian Processes

Review — Gaussian Process Posterior

Visualizing this:



Suppose we have a dataset of **two observations**

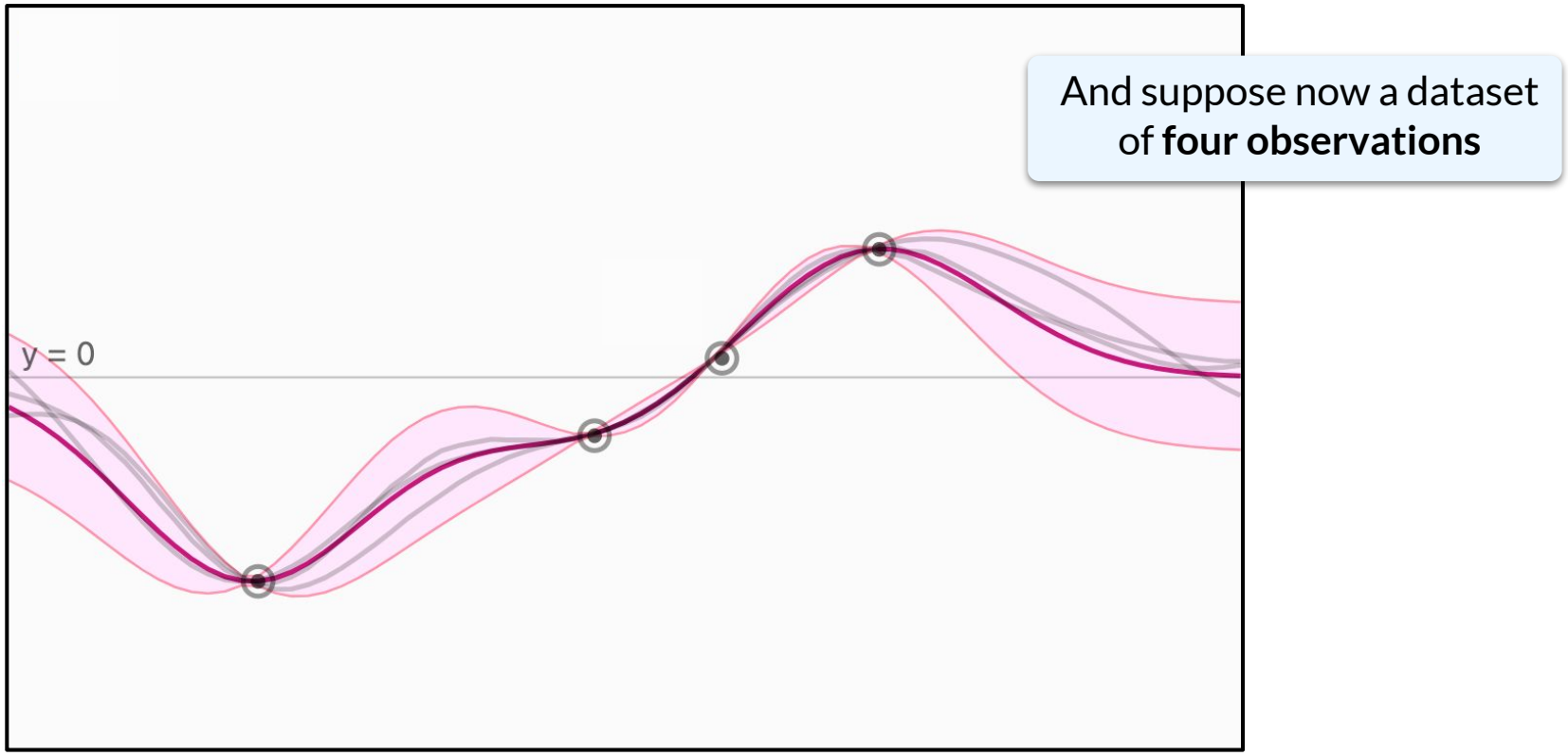
Mean function is now **not constant**

To make predictions for a given x, just compute marginal of posterior GP

Source: A Visual Exploration of Gaussian Processes

Review — Gaussian Process Posterior

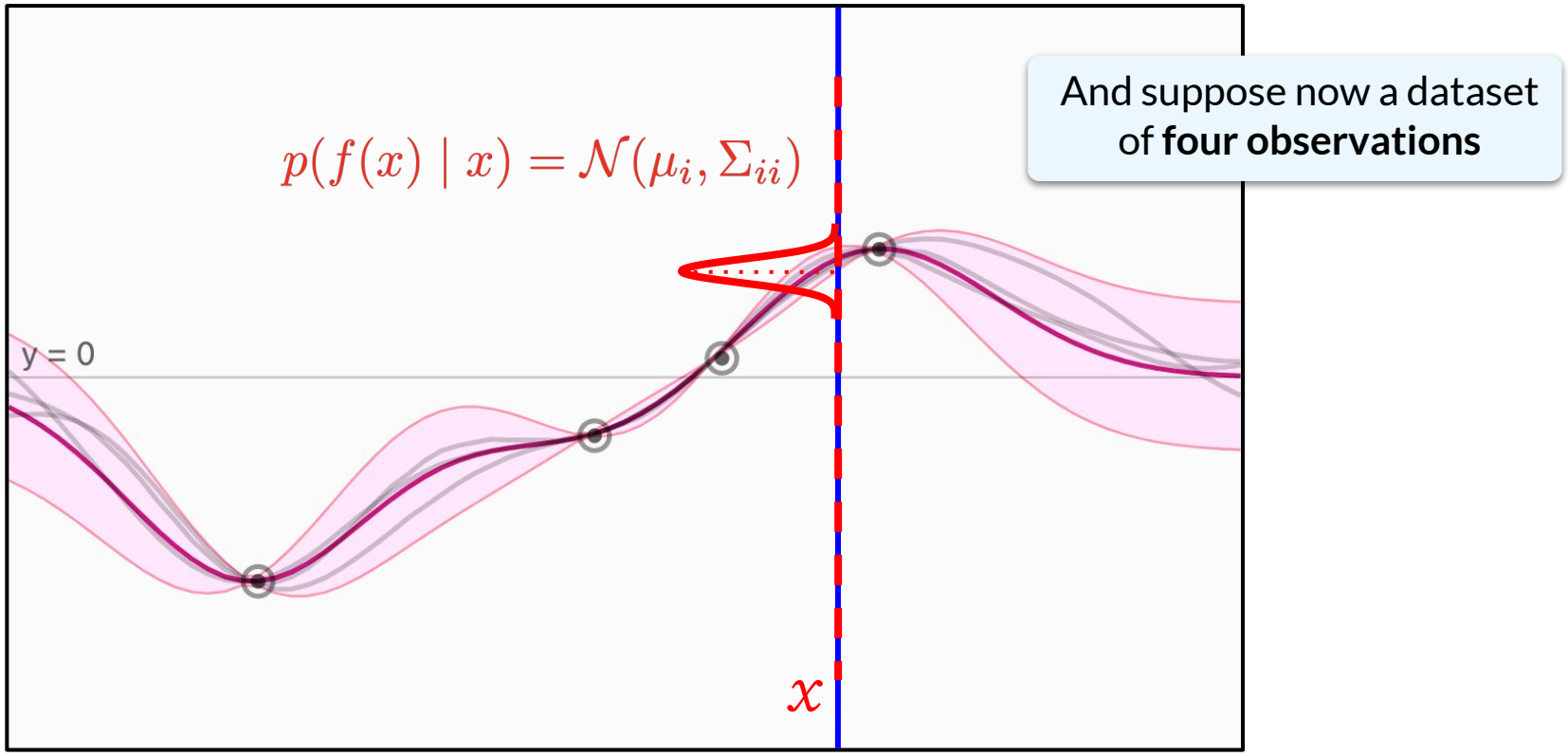
Visualizing this:



Source: A Visual Exploration of Gaussian Processes

Review — Gaussian Process Posterior

Visualizing this:



Source: A Visual Exploration of Gaussian Processes

Gaussian Processes — Summary

GP Summary:

Gaussian Processes — Summary

GP Summary:

- GP is parameterized (defined) by a *mean function* and *covariance kernel function*.

Gaussian Processes — Summary

GP Summary:

- GP is parameterized (defined) by a *mean function* and *covariance kernel function*.
- For any set of input points x, these parameters yield a MVN distribution

Gaussian Processes — Summary

GP Summary:

- GP is parameterized (defined) by a *mean function* and *covariance kernel function*.
- For any set of input points x, these parameters yield a MVN distribution
- And this MVN can be viewed as a **distribution over function values** (for the input points).

Gaussian Processes — Summary

GP Summary:

- GP is parameterized (defined) by a *mean function* and *covariance kernel function*.
- For any set of input points x, these parameters yield a MVN distribution
- And this MVN can be viewed as a **distribution over function values** (for the input points).
- We can then do Bayesian inference:

Gaussian Processes — Summary

GP Summary:

- GP is parameterized (defined) by a *mean function* and *covariance kernel function*.
- For any set of input points x, these parameters yield a MVN distribution
- And this MVN can be viewed as a **distribution over function values** (for the input points).
- We can then do Bayesian inference:
  - Use GP as a prior distribution.
  - Define a Gaussian likelihood for observations.
  - Then compute a GP posterior, given a set of data.

Gaussian Processes — Summary

GP Summary:

- GP is parameterized (defined) by a *mean function* and *covariance kernel function*.
- For any set of input points x, these parameters yield a MVN distribution
- And this MVN can be viewed as a **distribution over function values** (for the input points).
- We can then do Bayesian inference:
  - Use GP as a prior distribution.
  - Define a Gaussian likelihood for observations.
  - Then compute a GP posterior, given a set of data.
- ⇒ This yields a predictive UQ model!

Deep Uncertainty Models

Deep Uncertainty Models

We can also define predictive UQ models using neural networks.

Deep Uncertainty Models

We can also define predictive UQ models using neural networks.

I will go through a few examples of these.

Deep Uncertainty Models

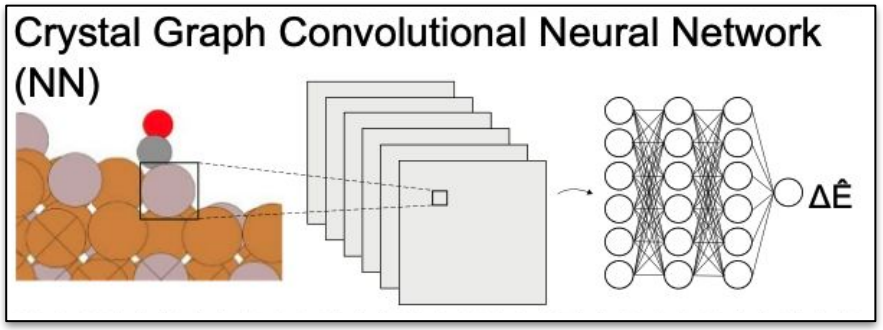
As a running example, we'll look at methods for incorporating predictive uncertainty into the following neural network:

Deep Uncertainty Models —Example: Computational Catalyst Design

As a running example, we'll look at methods for incorporating predictive uncertainty into the following neural network:

Deep Uncertainty Models —Example: Computational Catalyst Design

As a running example, we'll look at methods for incorporating predictive uncertainty into the following neural network:



*Xie, Tian, and Jeffrey C. Grossman. "**Crystal graph convolutional neural networks for an accurate and interpretable prediction of material properties."** Physical review letters, 2018.*

CGCNN Model *(a graph convolutional neural network model)*

- *Inputs*: three-dimensional atomic structure (a graph).
- *Outputs*: DFT-calculated site adsorption energies ΔE.

  (⇒ *Regression*).

"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

Deep Uncertainty Models —Example: Computational Catalyst Design

Suppose we want to *incorporate uncertainty into this neural network*.

(*I.e.*, use neural network for both point predictions and uncertainties).

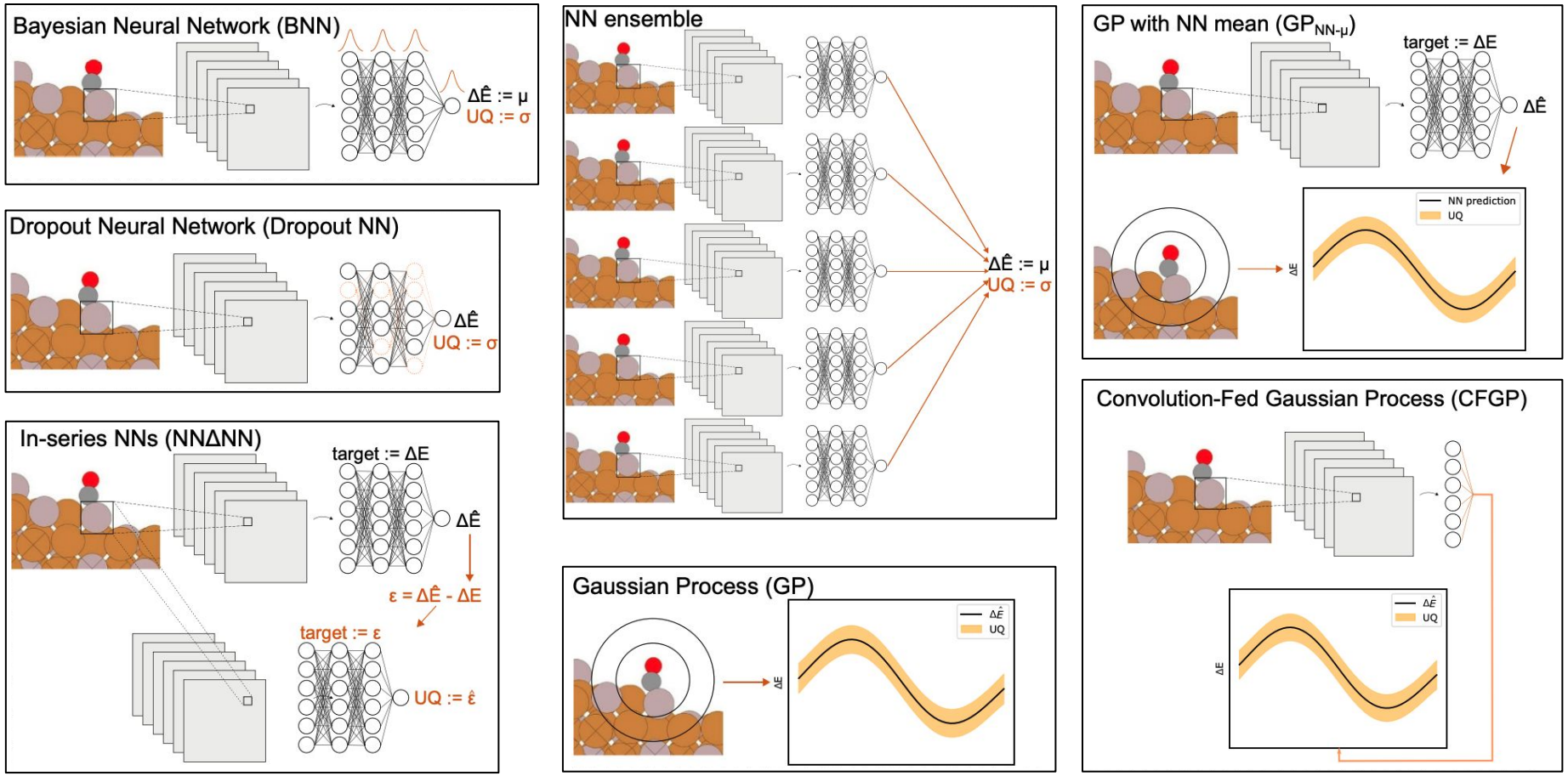
Deep Uncertainty Models —Example: Computational Catalyst Design

Suppose we want to *incorporate uncertainty into this neural network*.

(*I.e.*, use neural network for both point predictions and uncertainties).

⇒ there are many ways this can be done, and it's an active area of research!

Deep Uncertainty Models —Example: Computational Catalyst Design

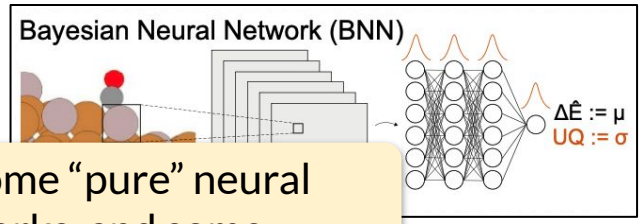


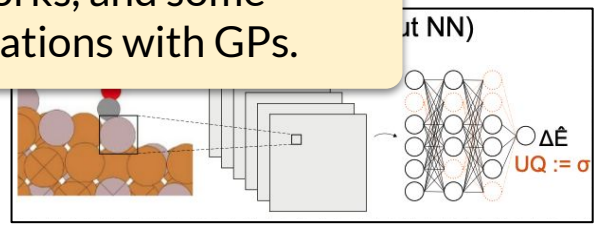
"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

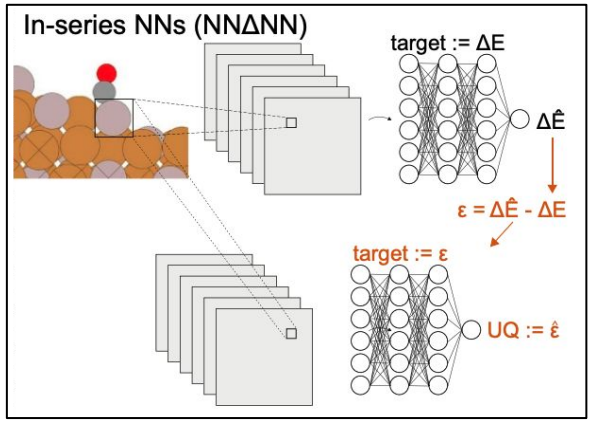
"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

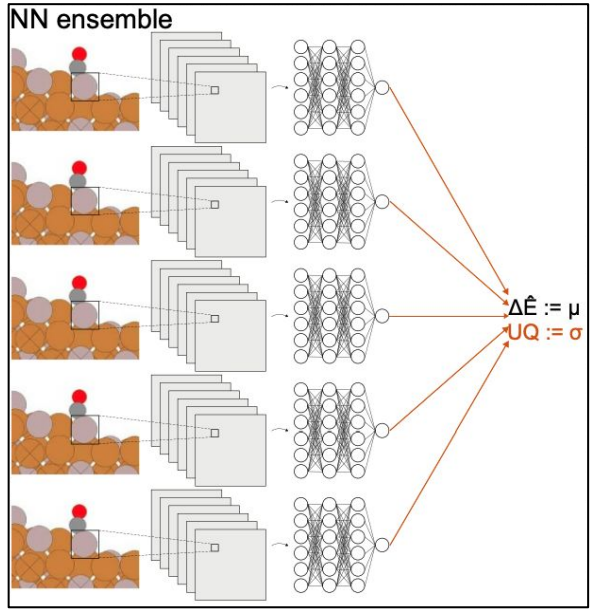
Deep Uncertainty Models —Example: Computational Catalyst Design

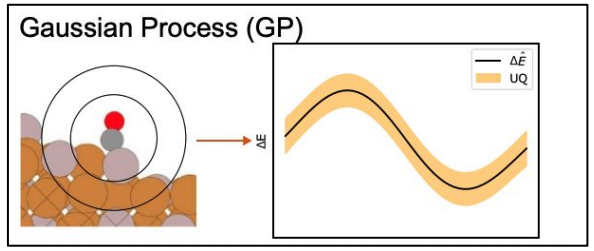
Note: some “pure” neural networks, and some combinations with GPs.

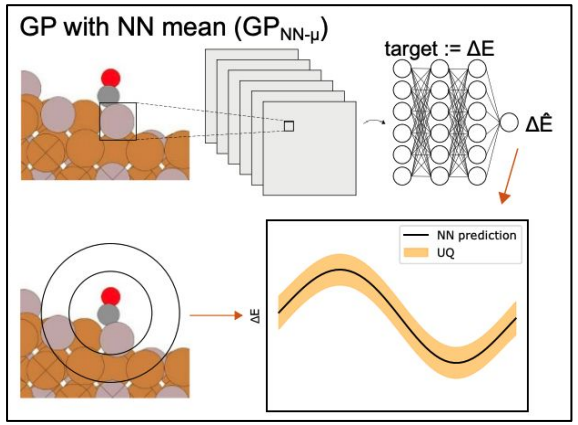


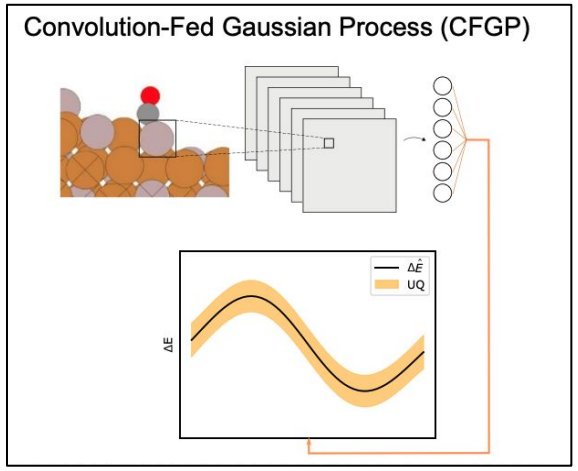








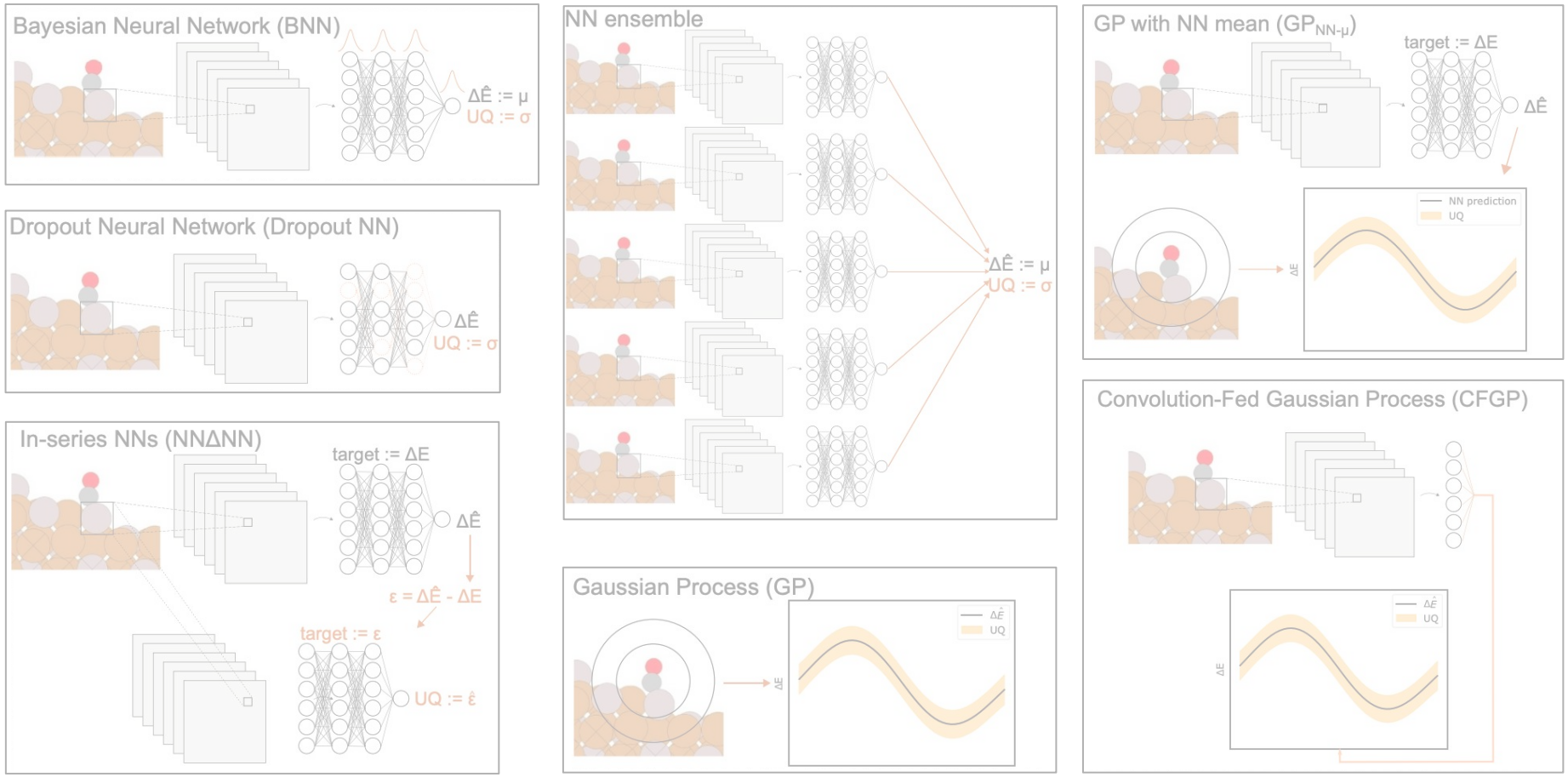




“Methods for comparing uncertainty quantifications for material property predictions”, *Tran, *Neiswanger, et al., MLST. 2020

“Computational catalyst discovery: Active classification through myopic multiscale sampling”, *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

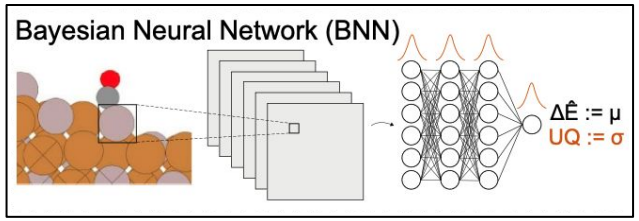
Deep Uncertainty Models —Example: Computational Catalyst Design

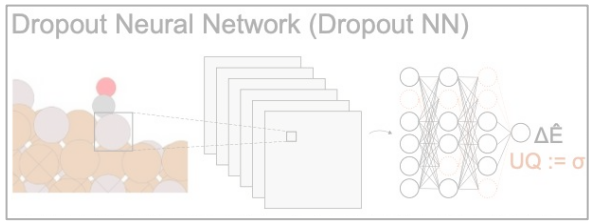


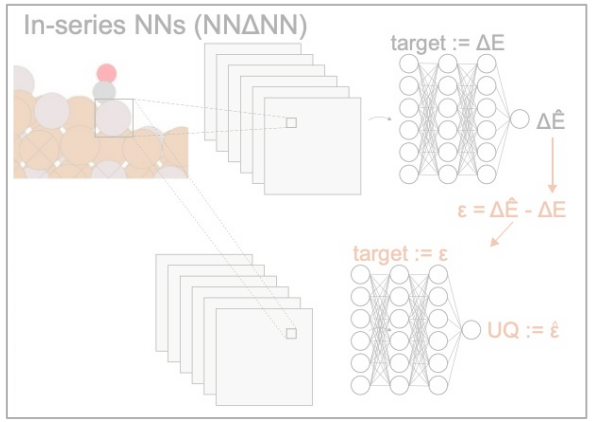
"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

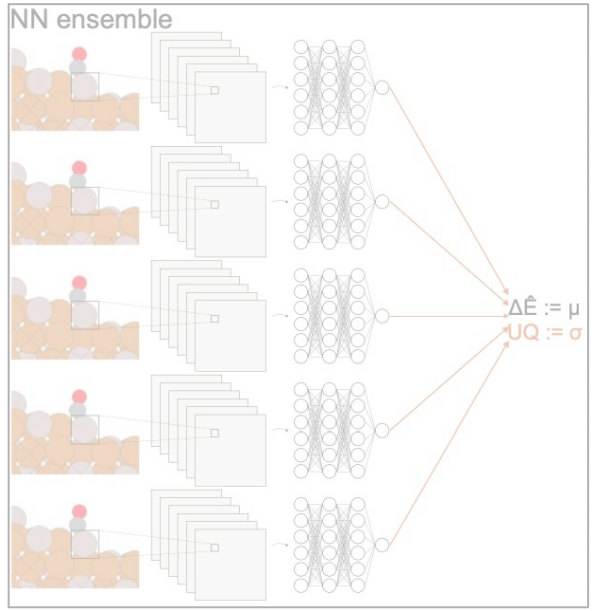
"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

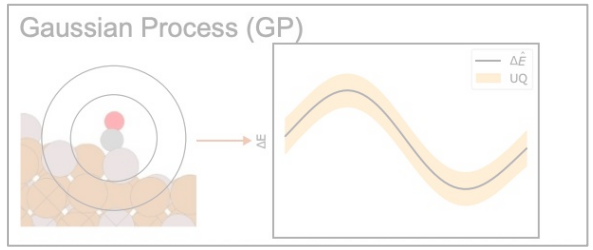
Deep Uncertainty Models —Example: Computational Catalyst Design

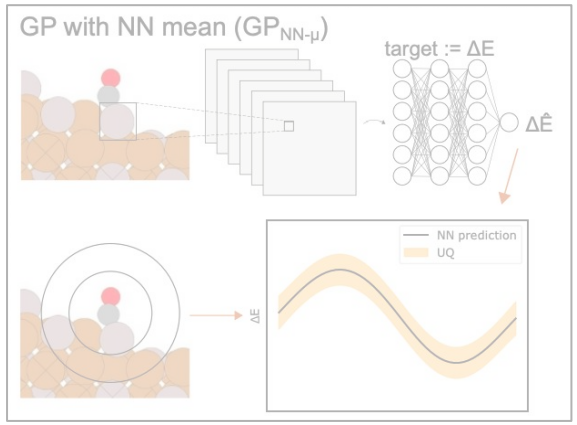


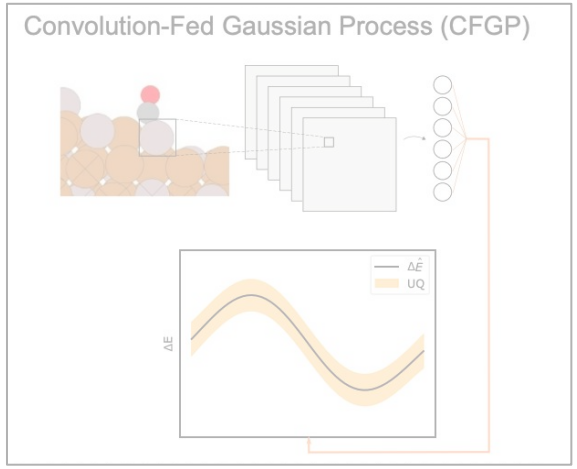








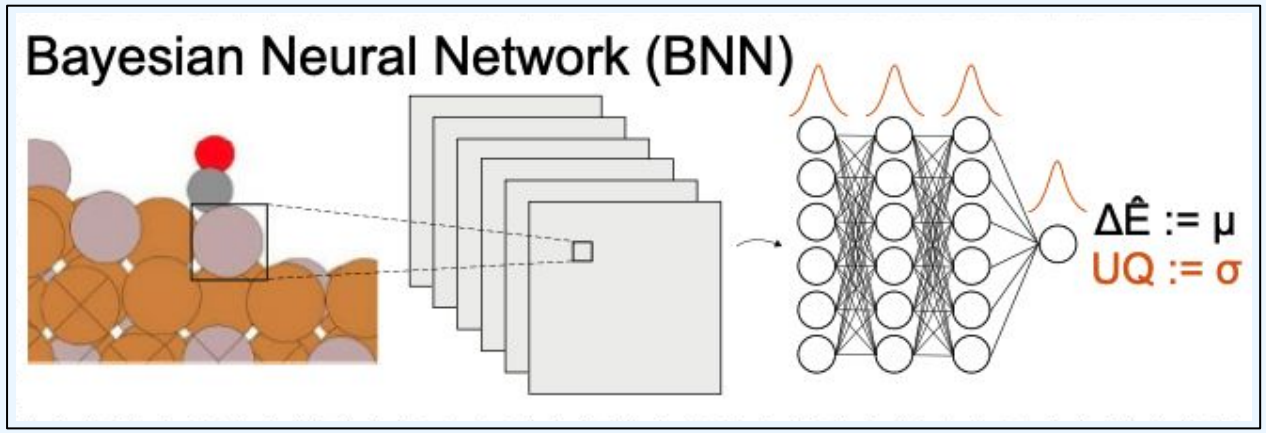




"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

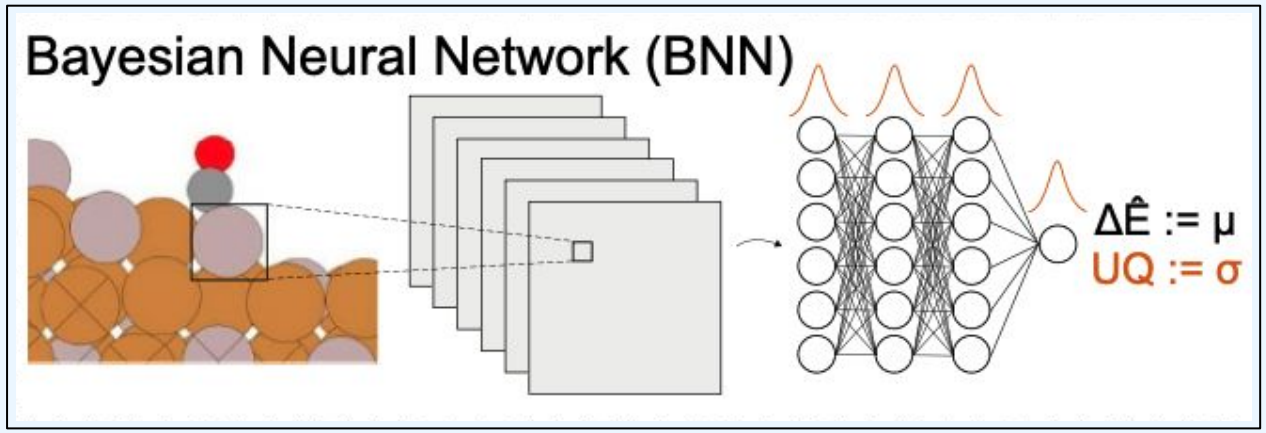
Deep Uncertainty Models —Example: Computational Catalyst Design



"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

Deep Uncertainty Models —Example: Computational Catalyst Design

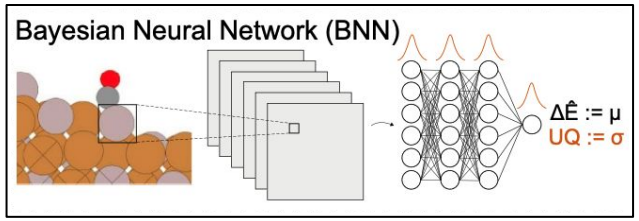


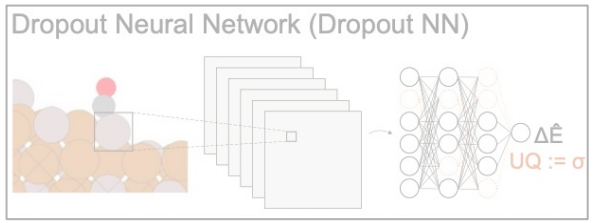
- Define a prior on neural network weights.
- In training, compute posterior over neural network weights (rather than a point estimate).
- Often using some type of MCMC (*e.g.*, LMC) or variational inference.

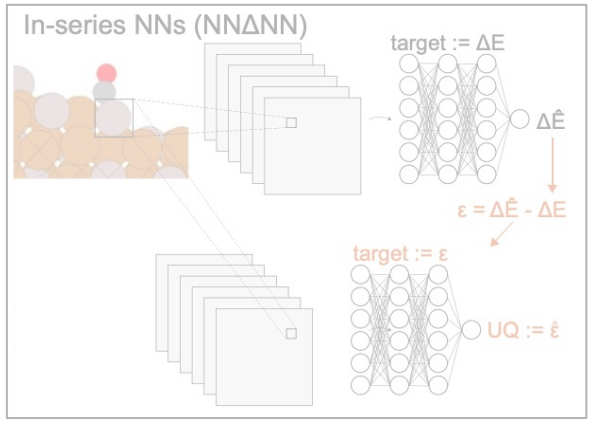
"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

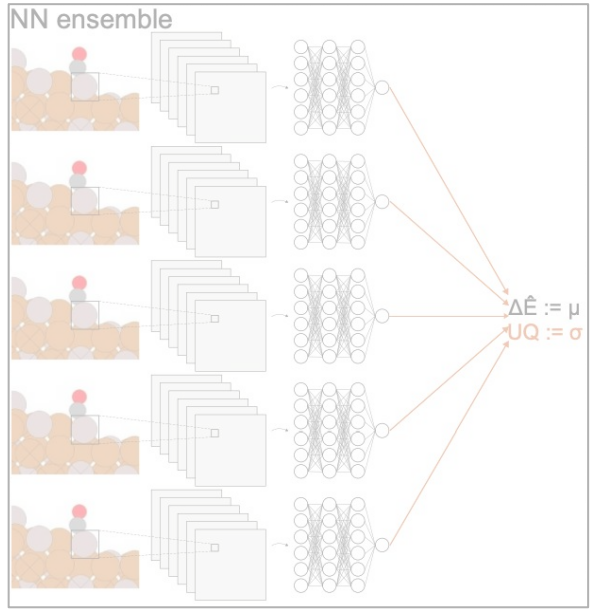
"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

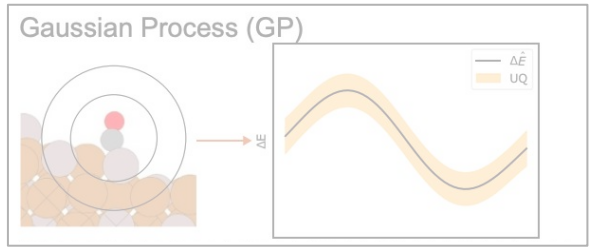
Deep Uncertainty Models —Example: Computational Catalyst Design

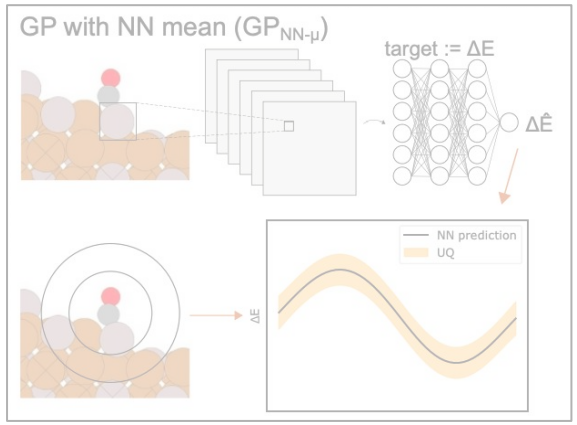


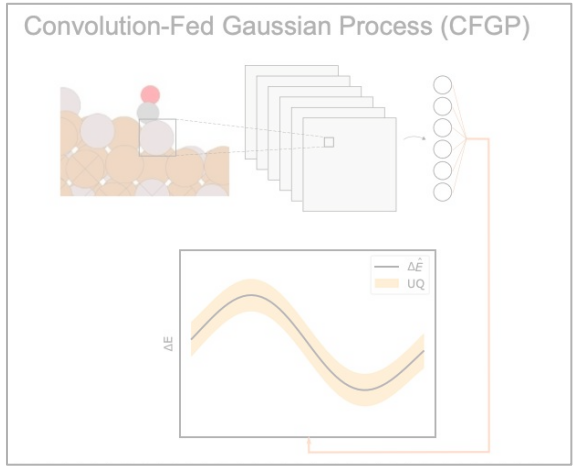








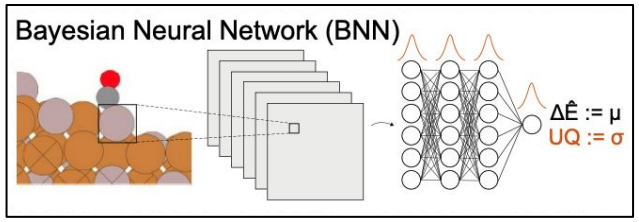


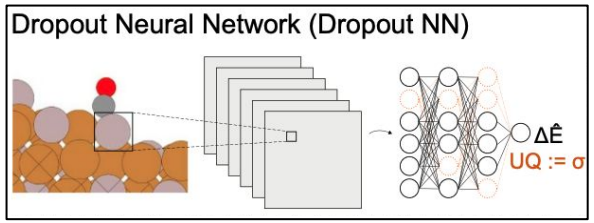


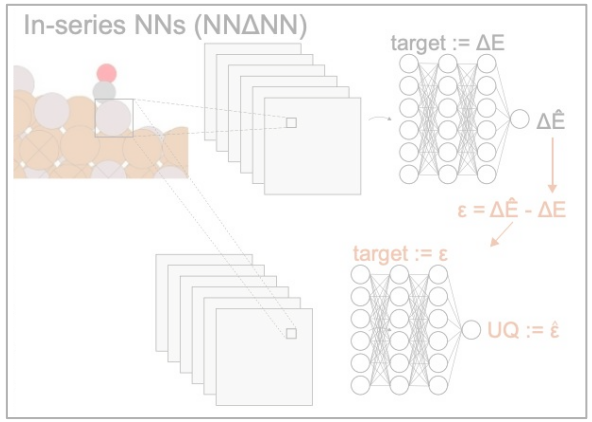
"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

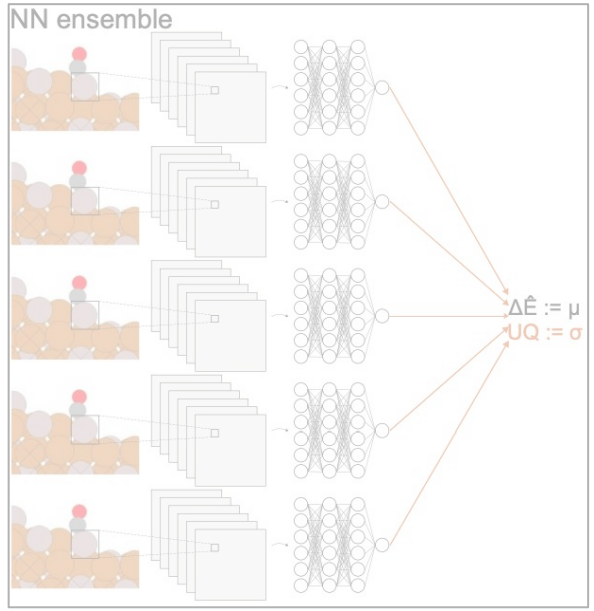
"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

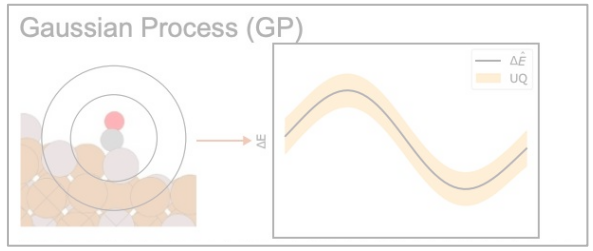
Deep Uncertainty Models —Example: Computational Catalyst Design

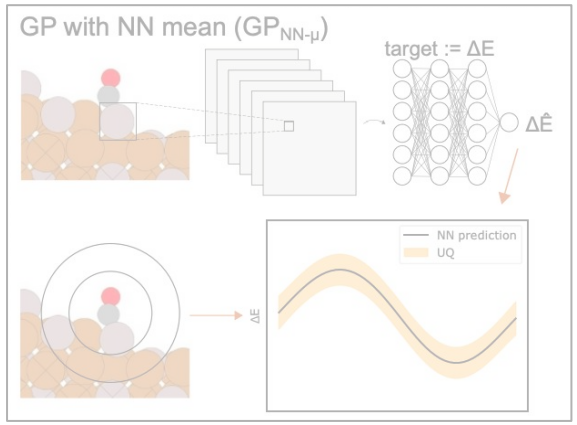


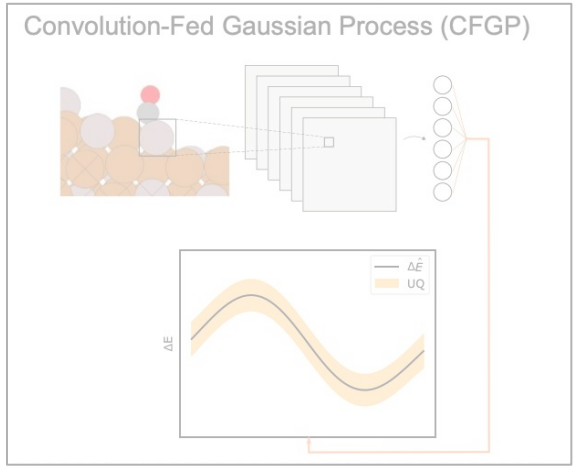








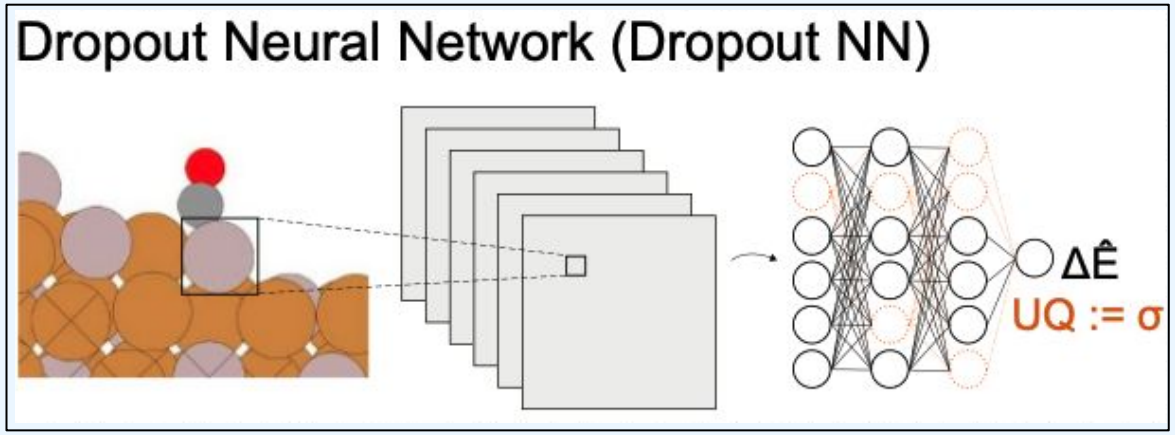




"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

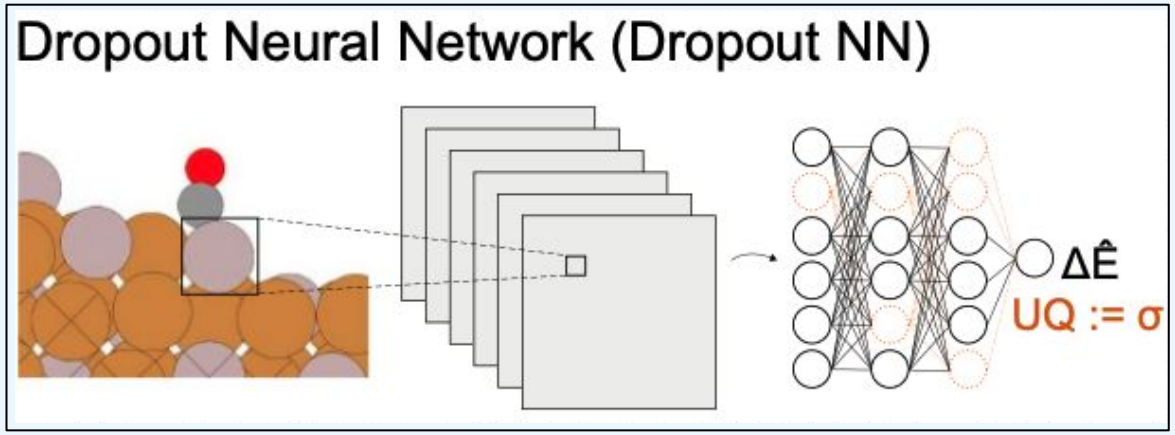
Deep Uncertainty Models —Example: Computational Catalyst Design



“Methods for comparing uncertainty quantifications for material property predictions”, *Tran, *Neiswanger, et al., MLST. 2020

“Computational catalyst discovery: Active classification through myopic multiscale sampling”, *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

Deep Uncertainty Models —Example: Computational Catalyst Design

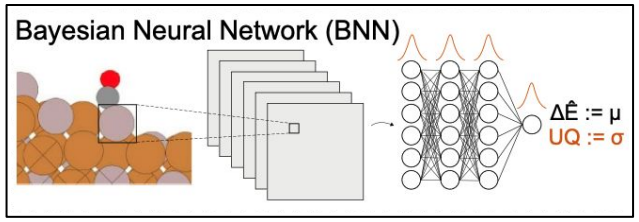


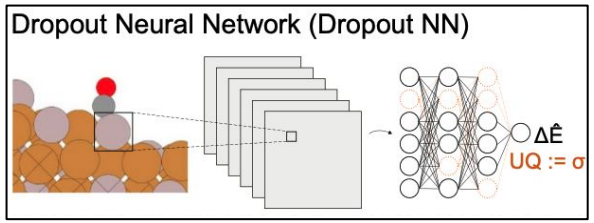
- During train/inference-time, perform *Monte Carlo Dropout*.
- (*I.e.*, set a random fraction of the neuron's outputs to zero during forward pass of the model).
- ⇒ A set of forward passes yields a set of samples from an output distribution.

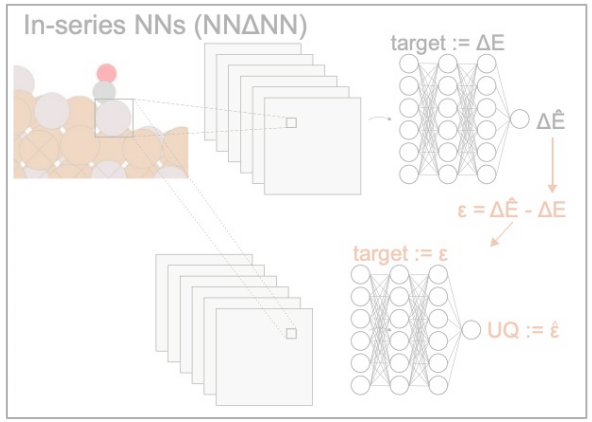
"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

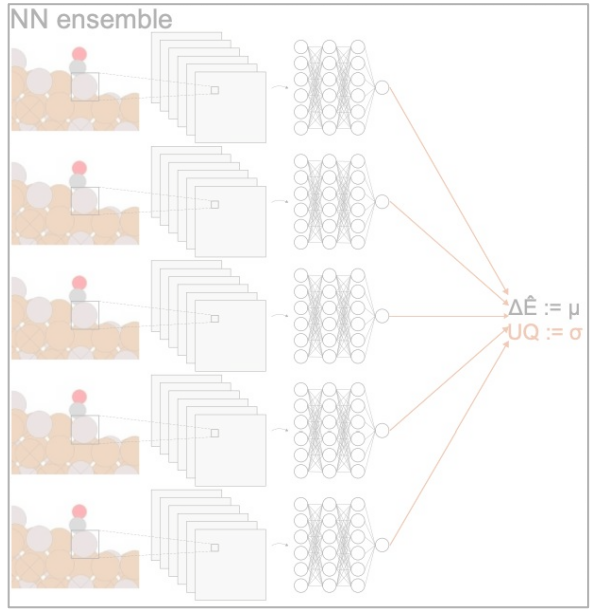
"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

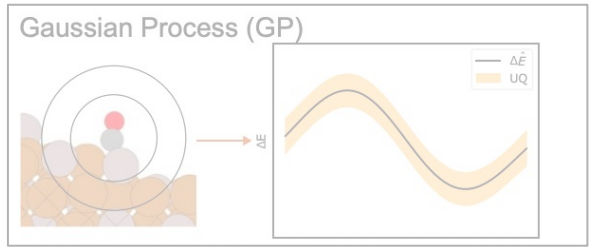
Deep Uncertainty Models —Example: Computational Catalyst Design

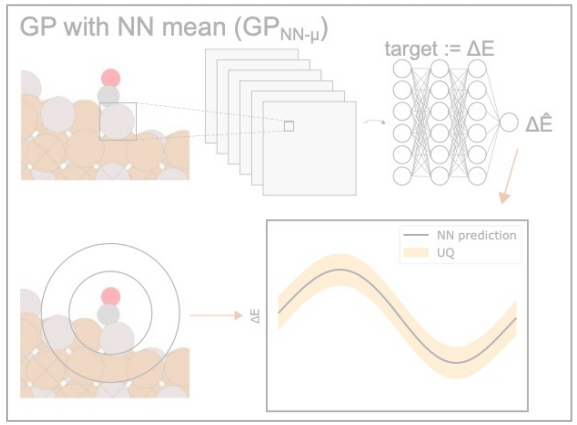


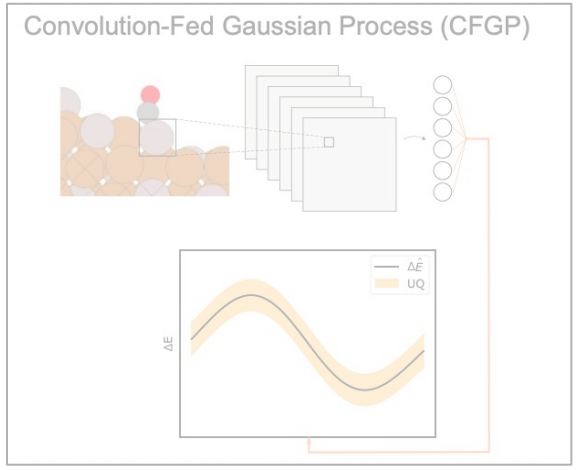








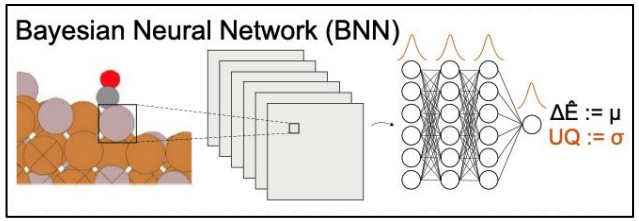


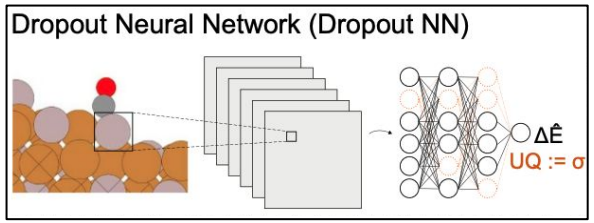


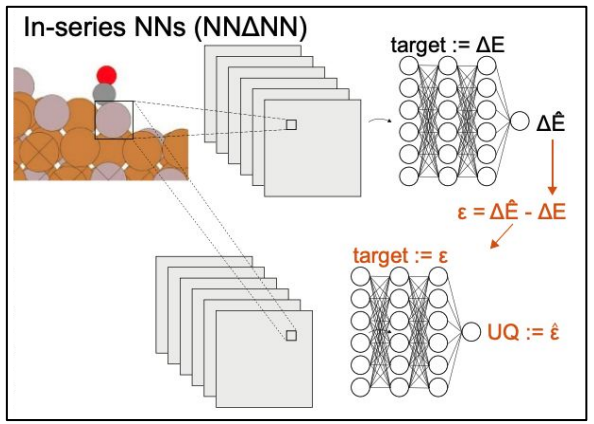
"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

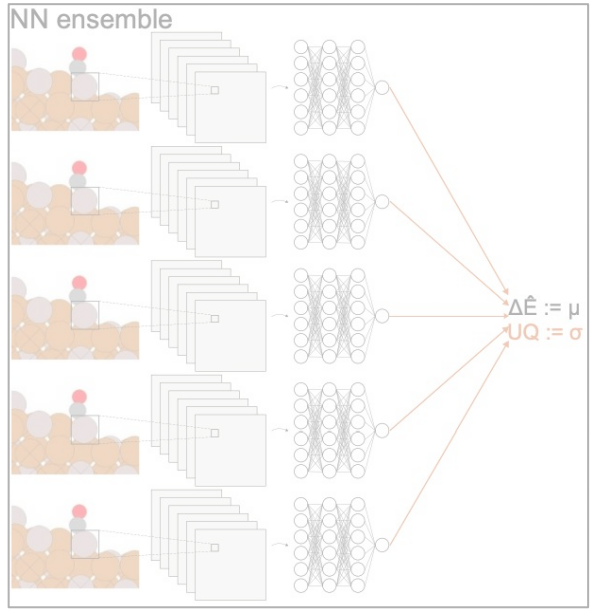
"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

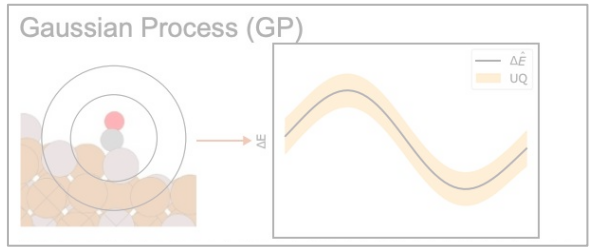
Deep Uncertainty Models —Example: Computational Catalyst Design

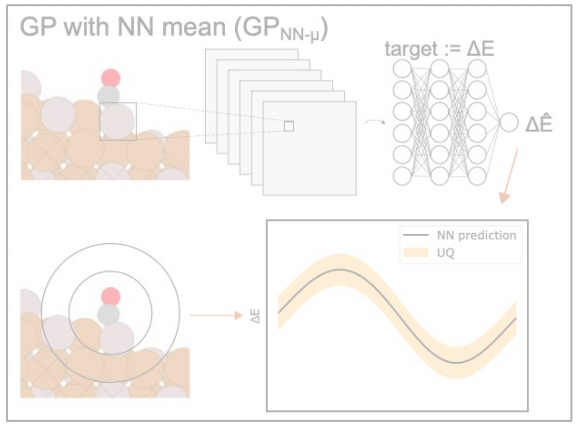


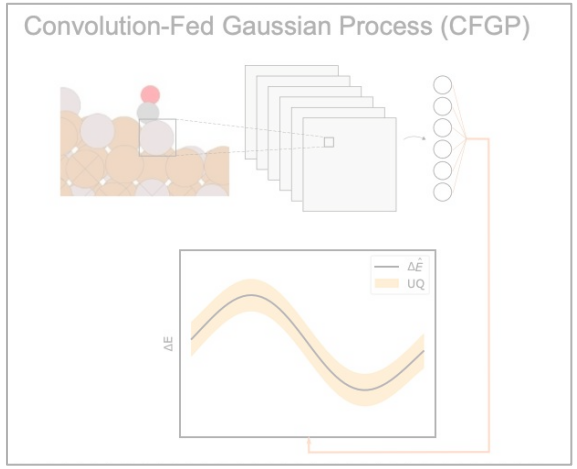








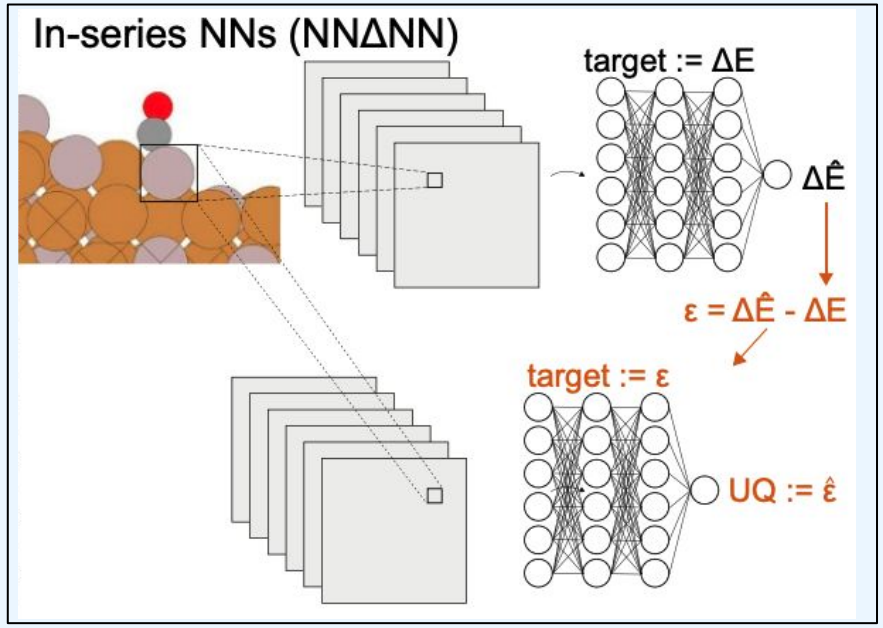




"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

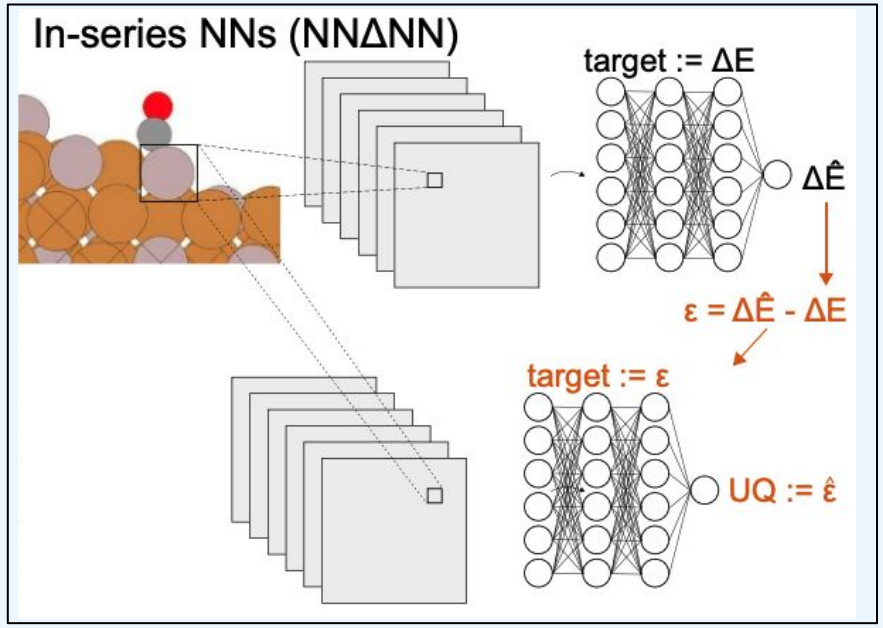
Deep Uncertainty Models —Example: Computational Catalyst Design



"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

Deep Uncertainty Models —Example: Computational Catalyst Design



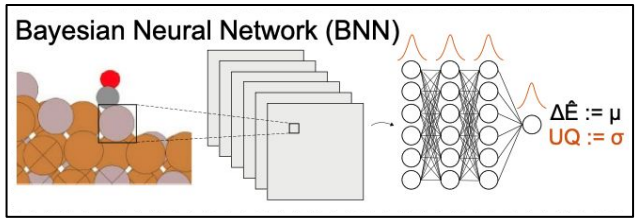
- First train a neural network for point prediction.
- Then compute residuals on a held-out set.
- Then use another neural network to try and predict residuals.

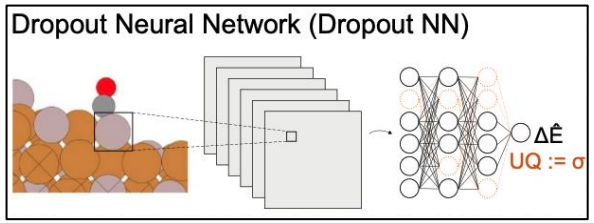
- ⇒ Second network predicts error, which can be used as UQ model.

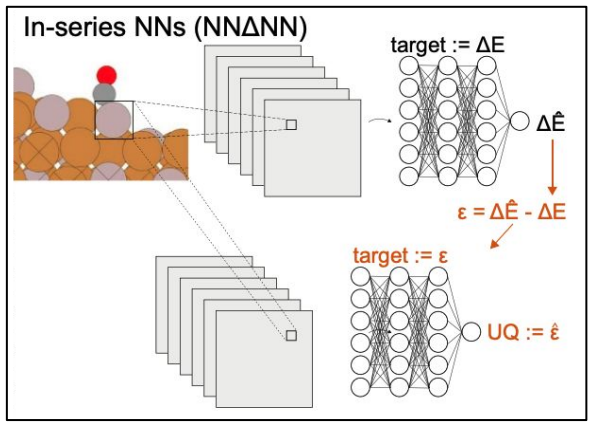
"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

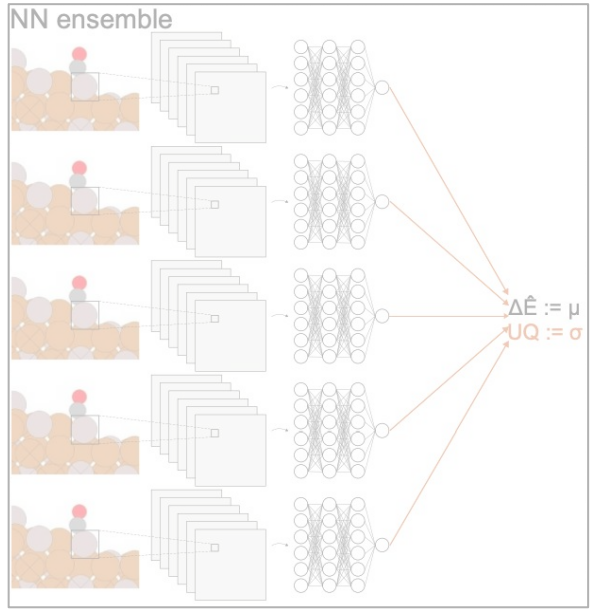
"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

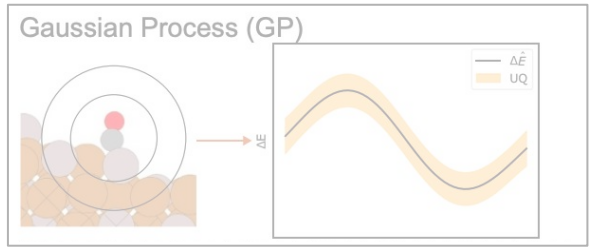
Deep Uncertainty Models —Example: Computational Catalyst Design

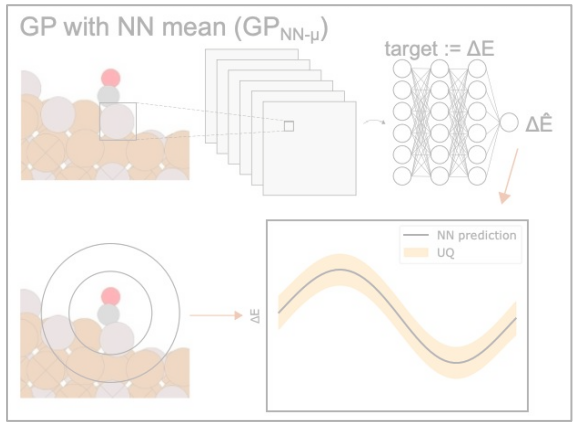


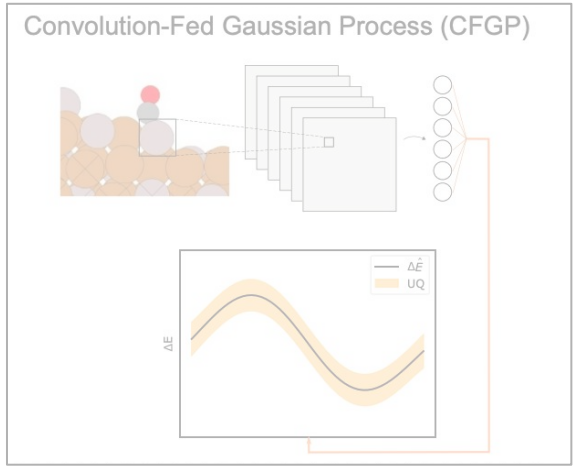








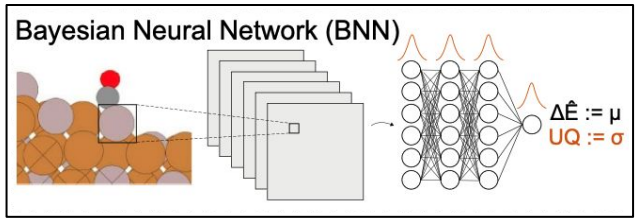


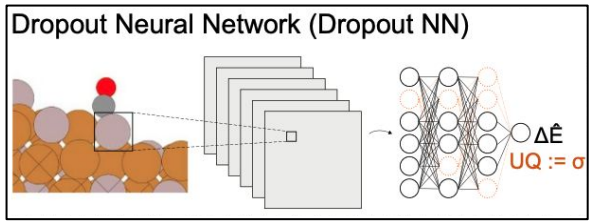


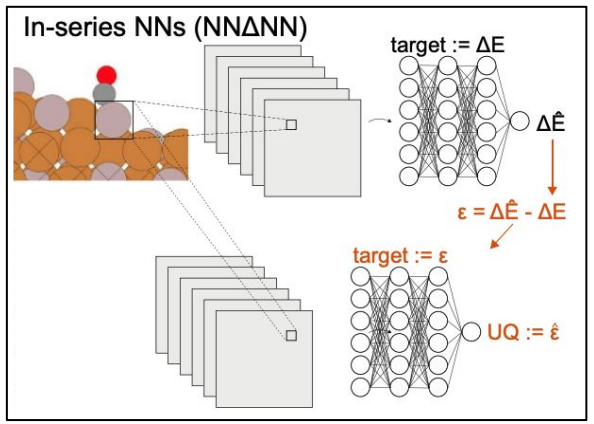
"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

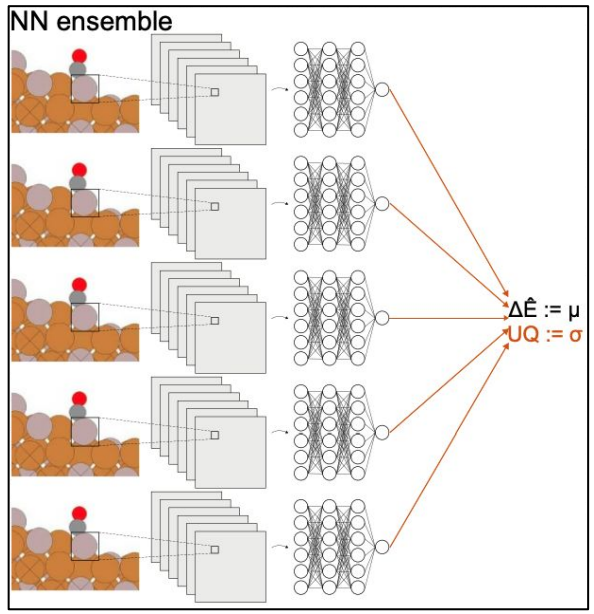
"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

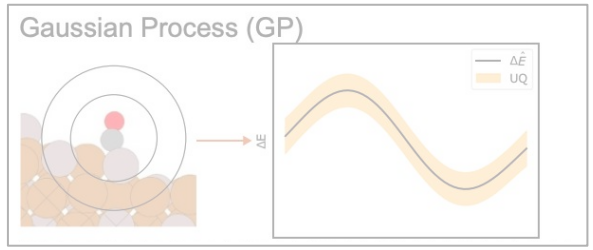
Deep Uncertainty Models —Example: Computational Catalyst Design

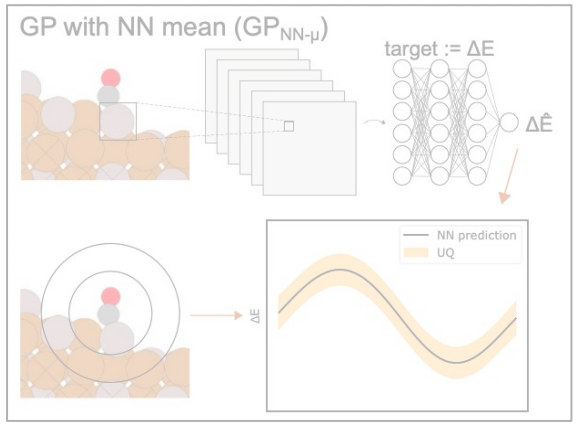


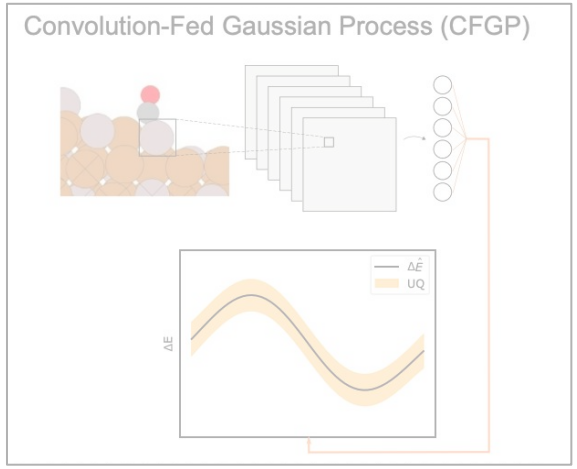








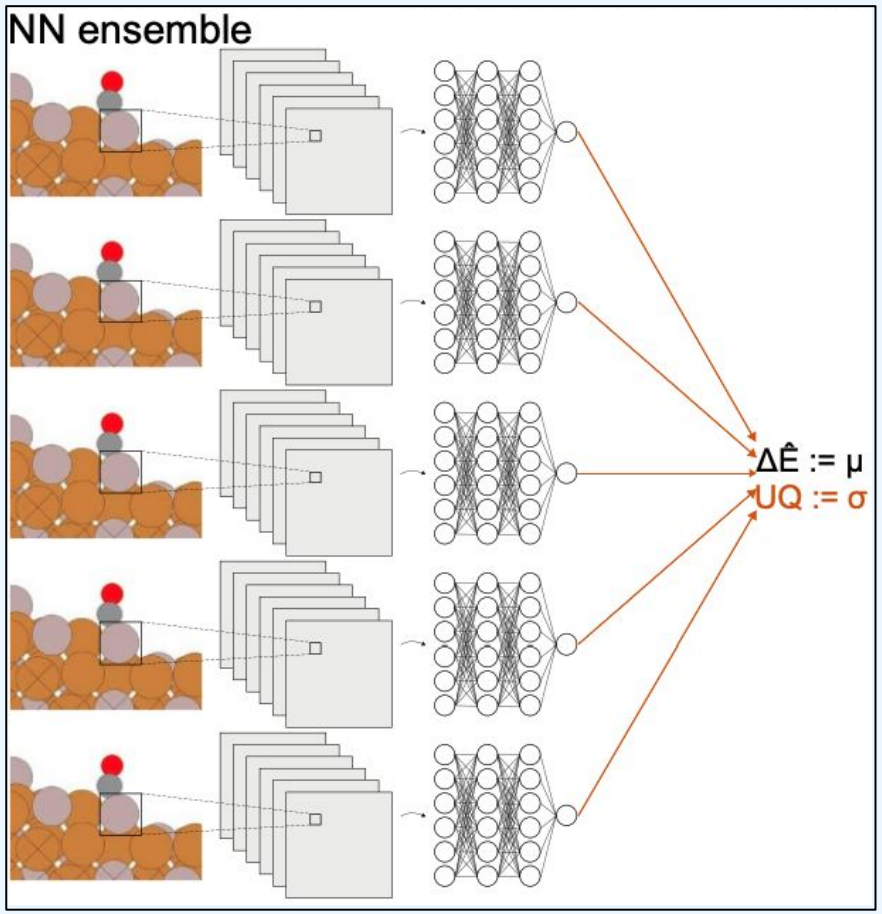




"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

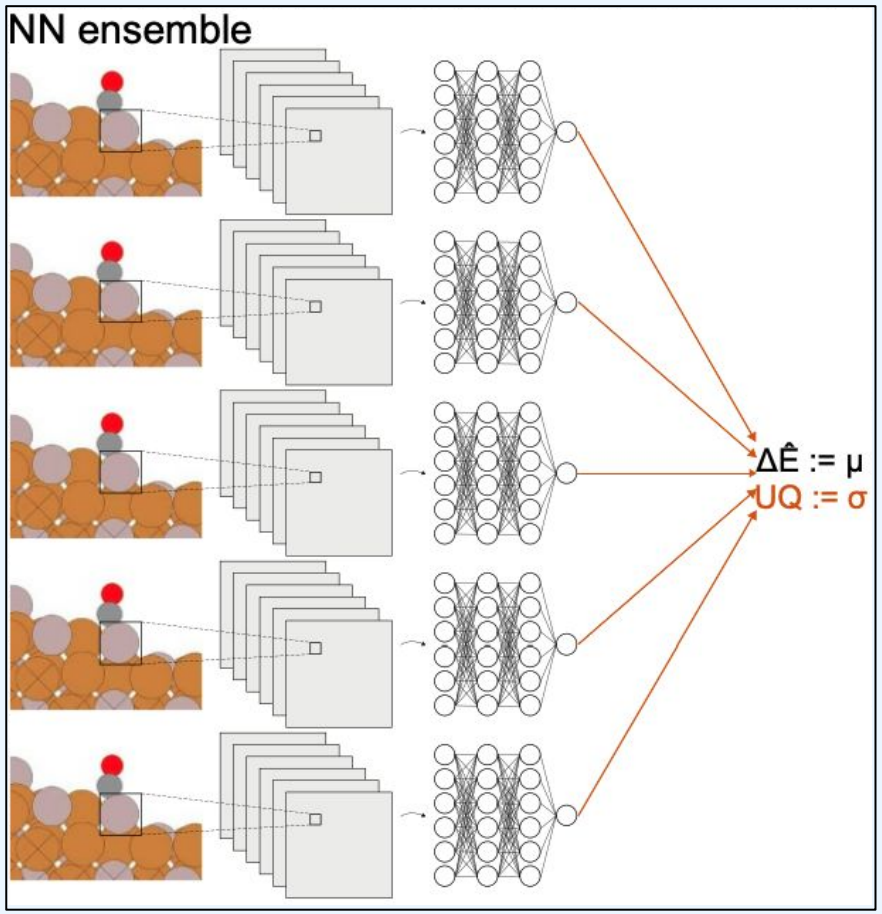
Deep Uncertainty Models —Example: Computational Catalyst Design



“Methods for comparing uncertainty quantifications for material property predictions”, *Tran, *Neiswanger, et al., MLST. 2020

“Computational catalyst discovery: Active classification through myopic multiscale sampling”, *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

Deep Uncertainty Models —Example: Computational Catalyst Design

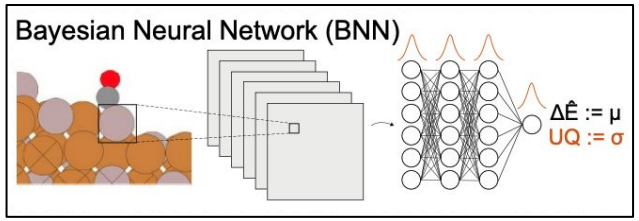


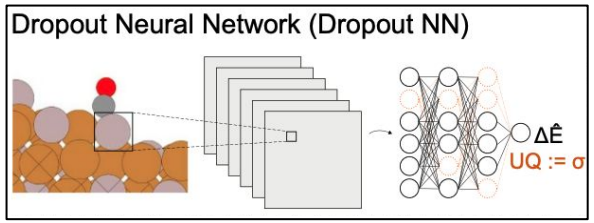
- Train a set of neural networks.
- Often each neural network has a different initialization, subset of data, or hyperparameter setting.
- Use diversity of predictions (agreement) as a measure of uncertainty in predictions.

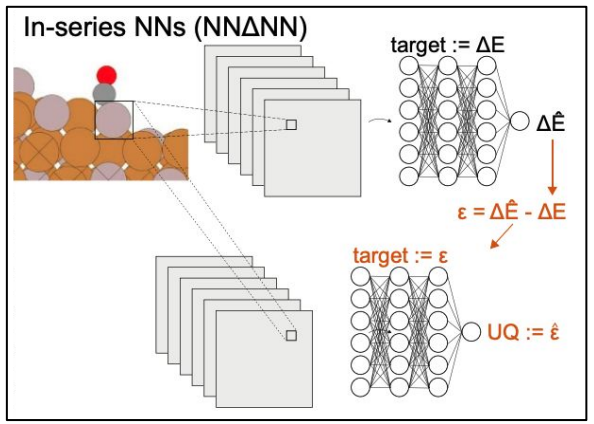
"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

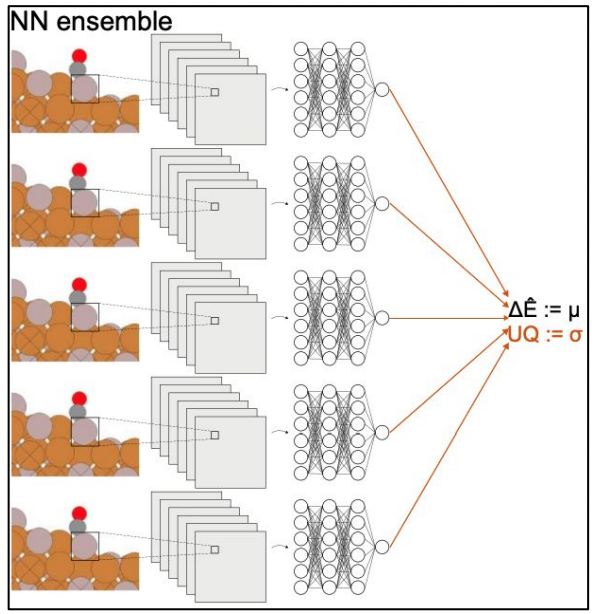
"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

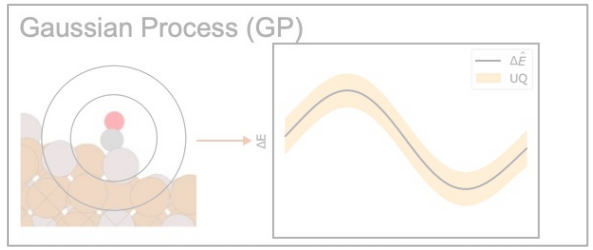
Deep Uncertainty Models —Example: Computational Catalyst Design

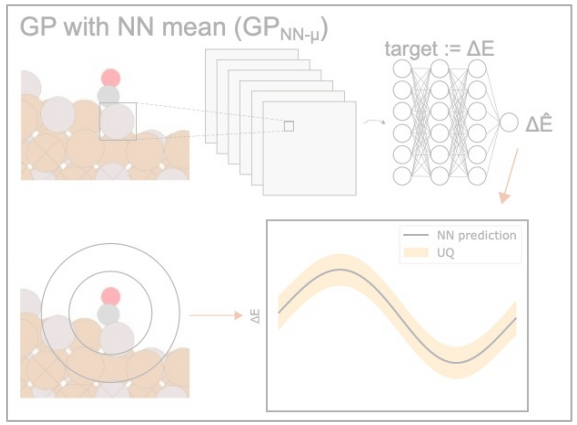


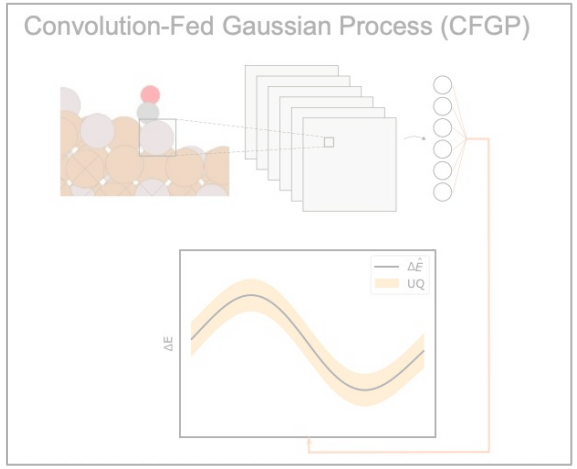








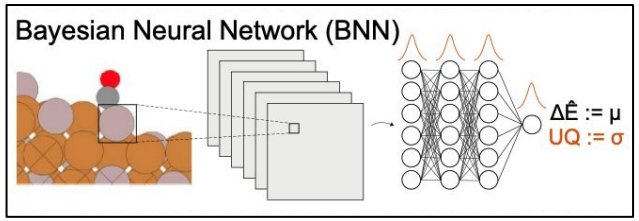


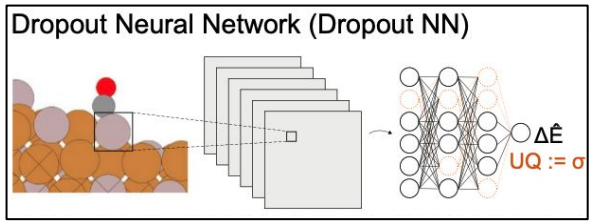


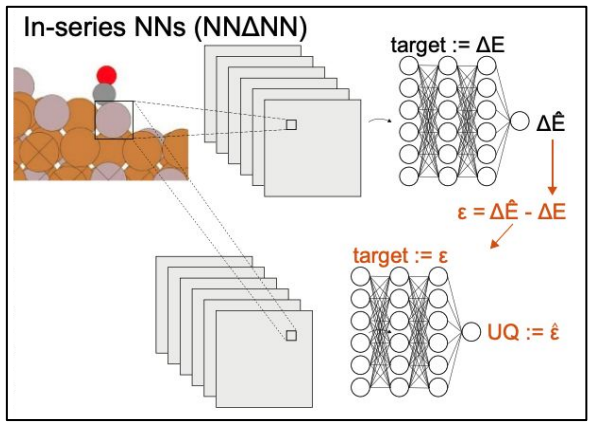
"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

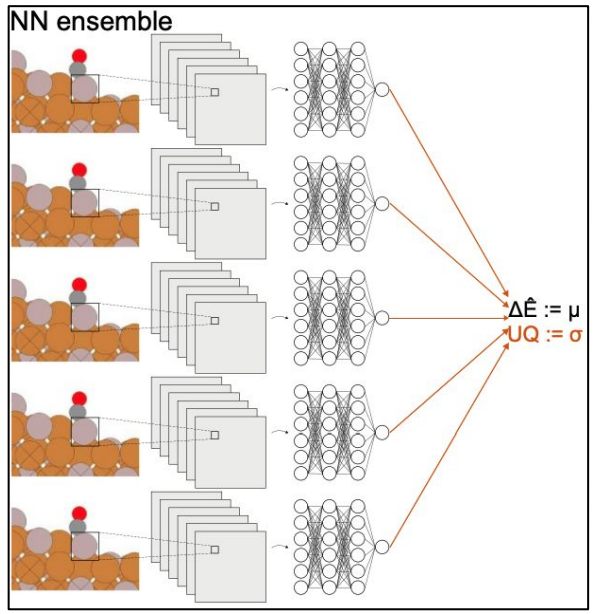
"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

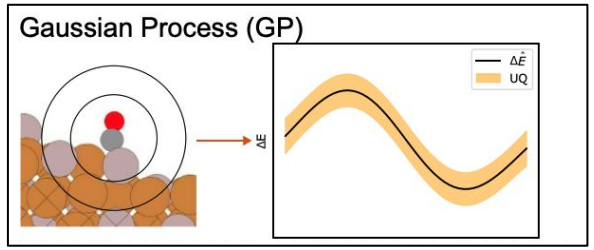
Deep Uncertainty Models —Example: Computational Catalyst Design

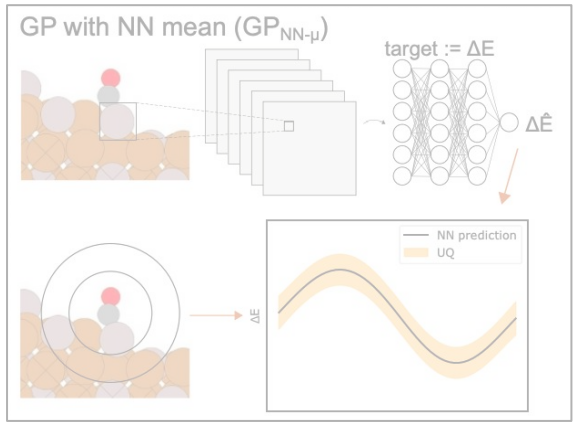


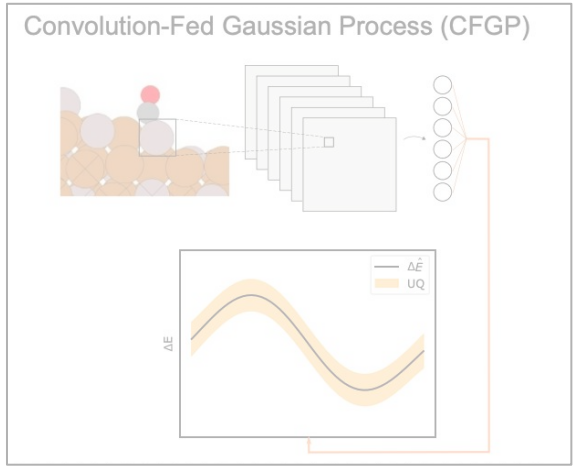








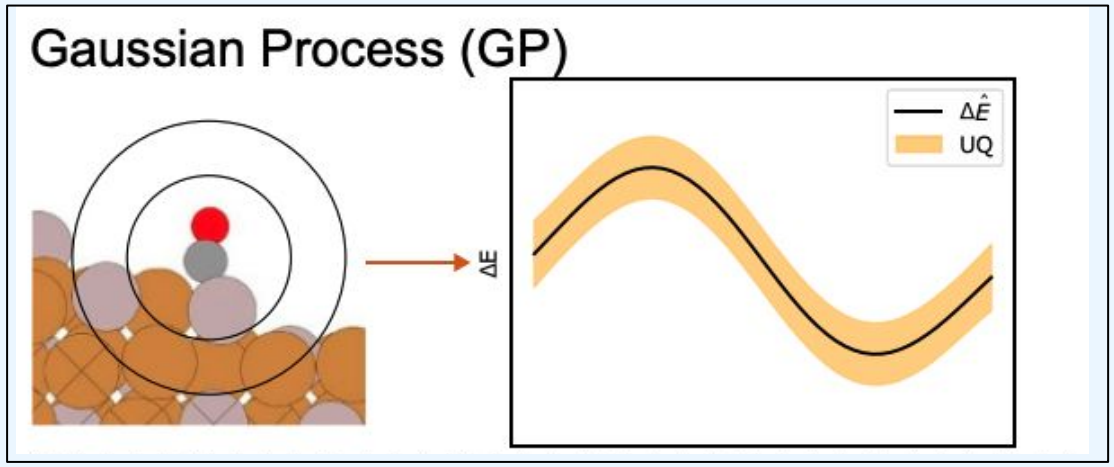




"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

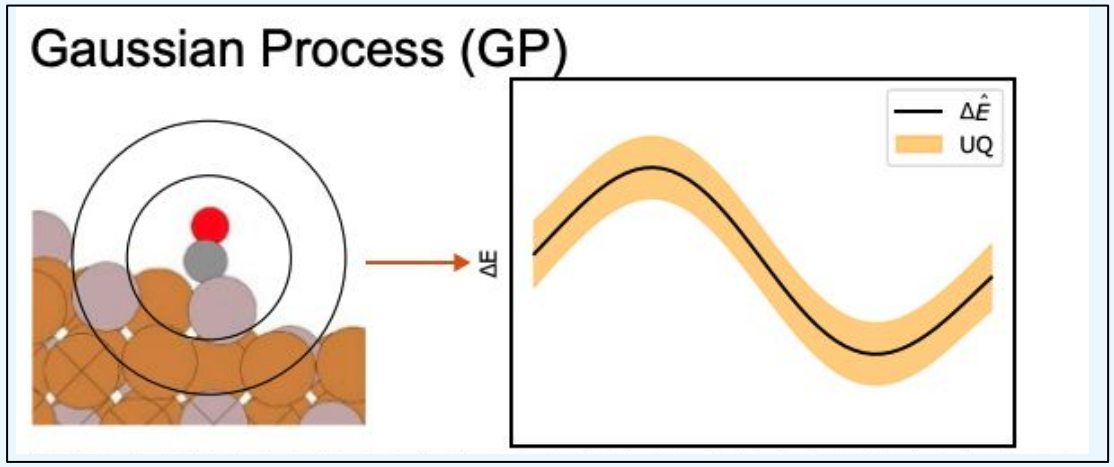
Deep Uncertainty Models —Example: Computational Catalyst Design



"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

Deep Uncertainty Models —Example: Computational Catalyst Design

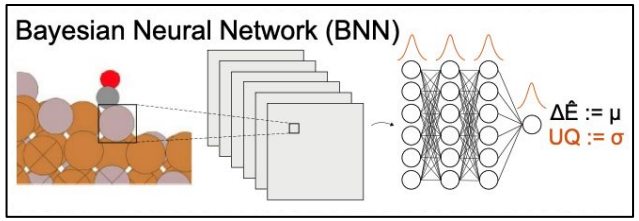


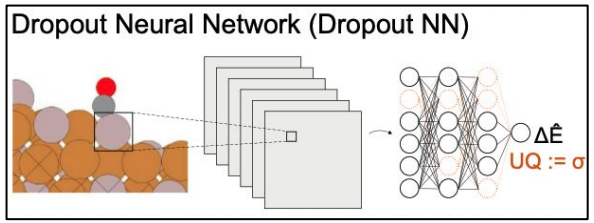
- Define some appropriate mean and kernel function on graph inputs.
- This yields a valid GP, which can make predictions on new inputs (new graphs)
- *Difficulty*: hard to hand-design good kernel function, & doesn't use NN.

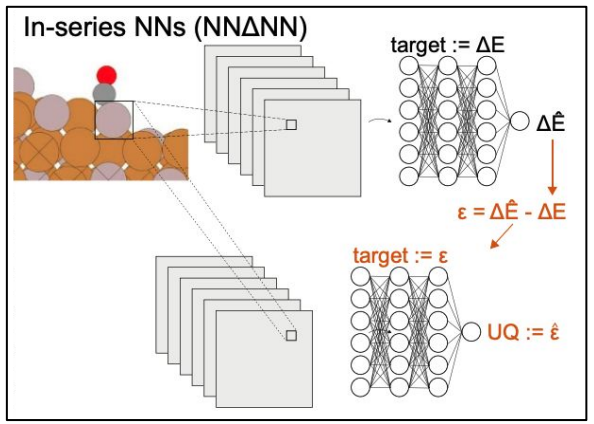
"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

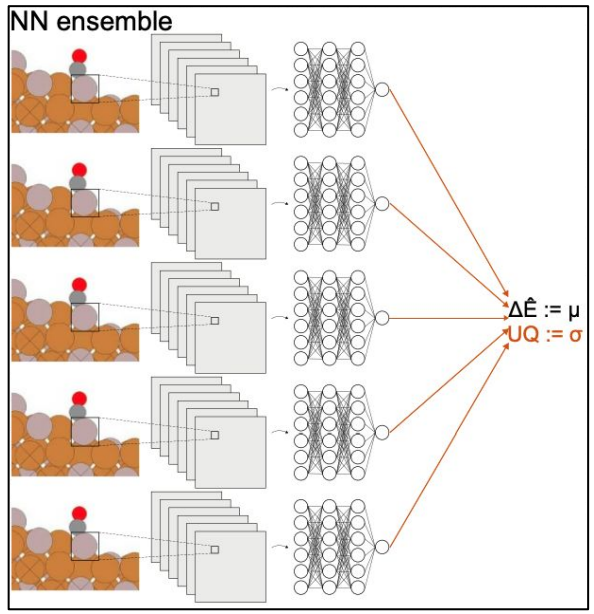
"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

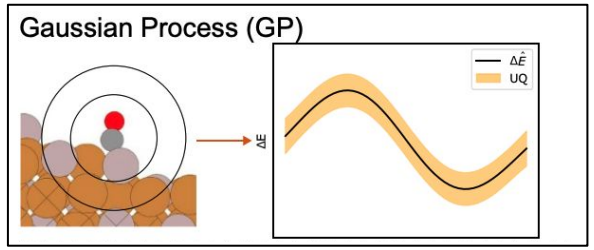
Deep Uncertainty Models —Example: Computational Catalyst Design

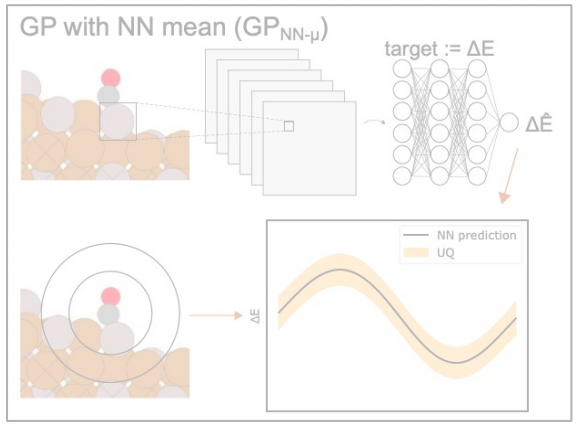


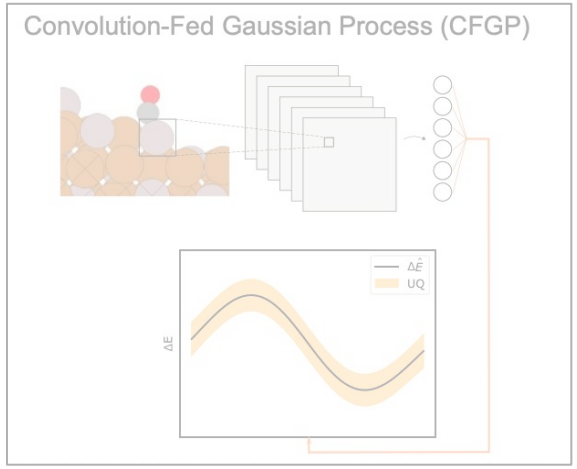








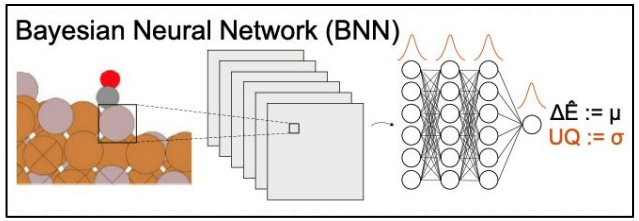


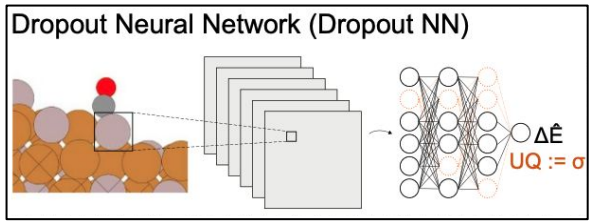


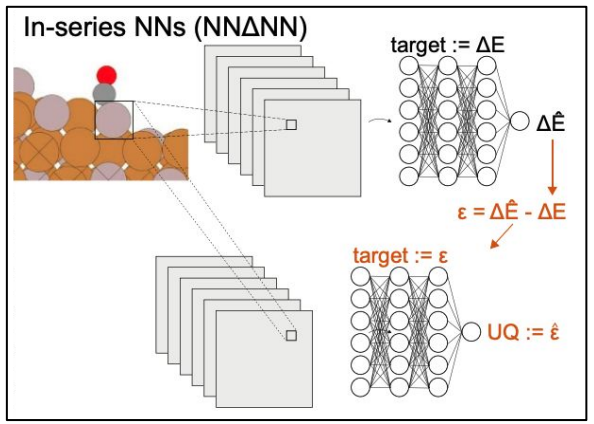
"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

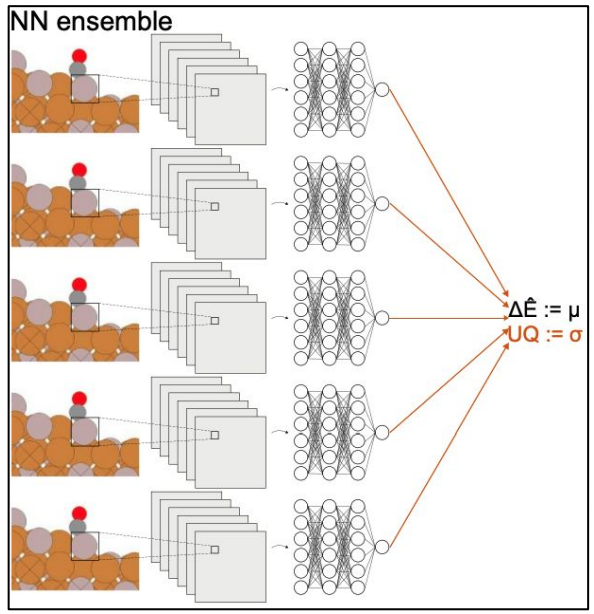
"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

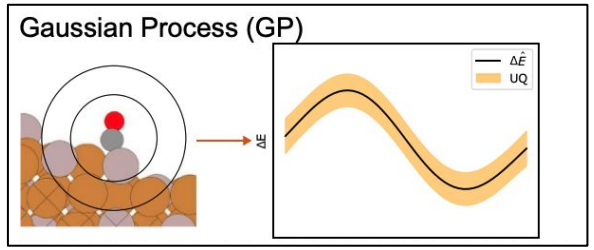
Deep Uncertainty Models —Example: Computational Catalyst Design

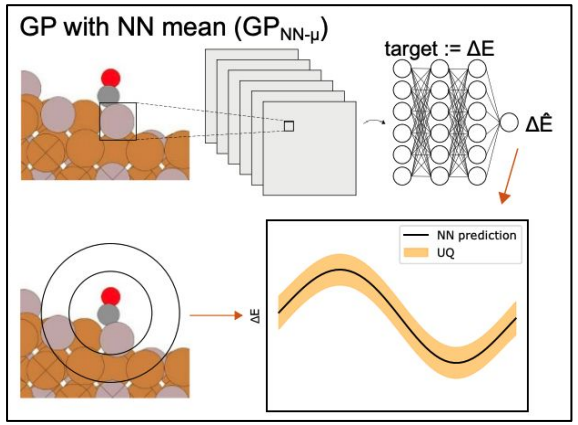


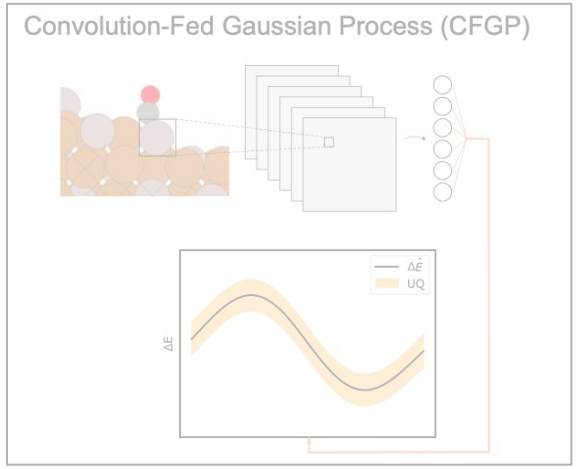








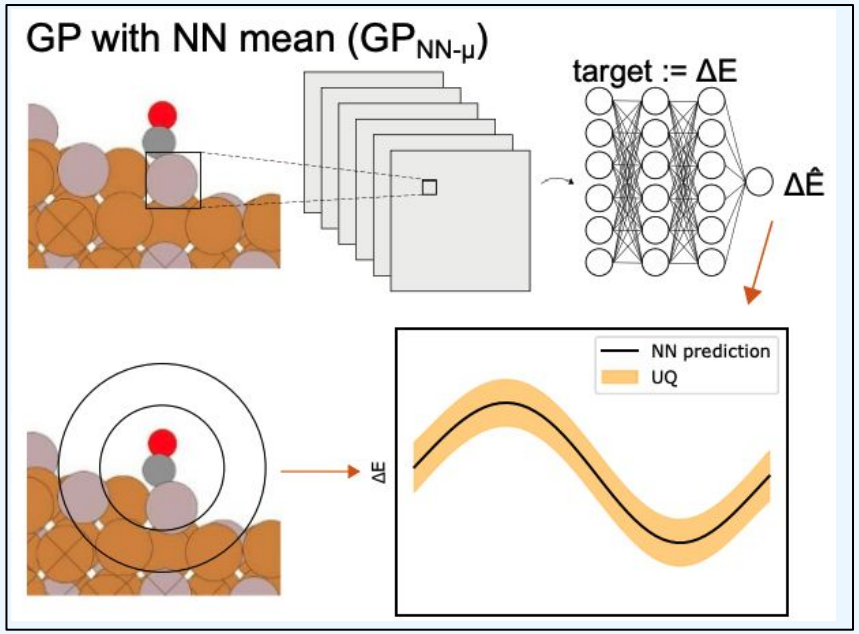




"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

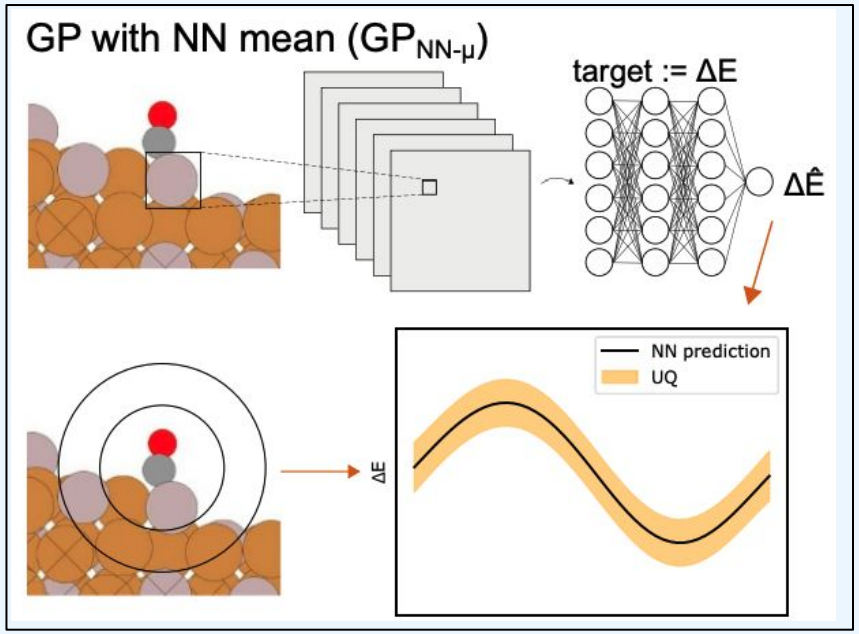
Deep Uncertainty Models —Example: Computational Catalyst Design



"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

Deep Uncertainty Models —Example: Computational Catalyst Design



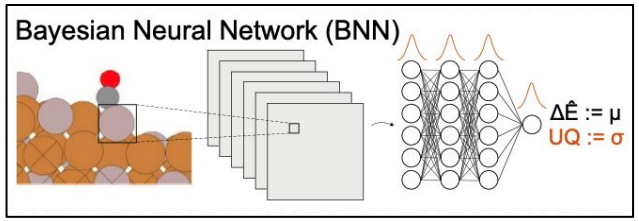
- In GPs, the prior mean fn is usually set to be **0.**
- But it could be anything…
- Here, we can set the prior mean function to be the output of a neural network model!

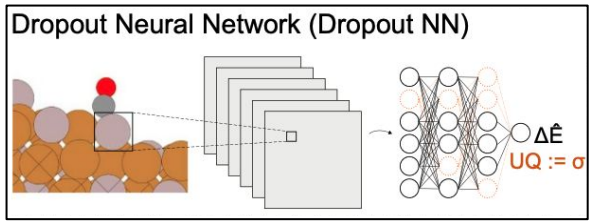
- Downside: still need to define a kernel function on graph inputs.

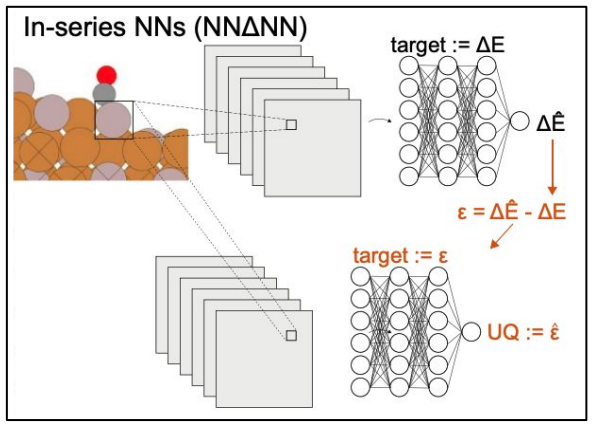
"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

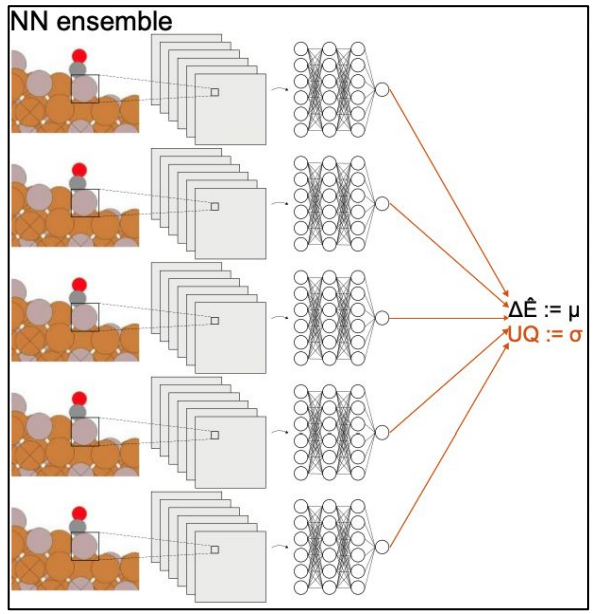
"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

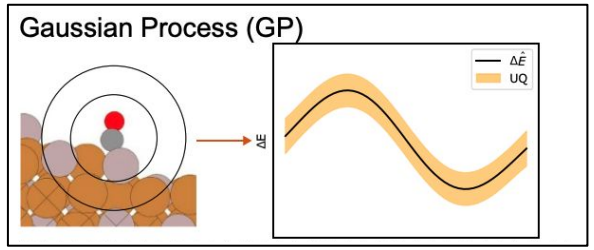
Deep Uncertainty Models —Example: Computational Catalyst Design

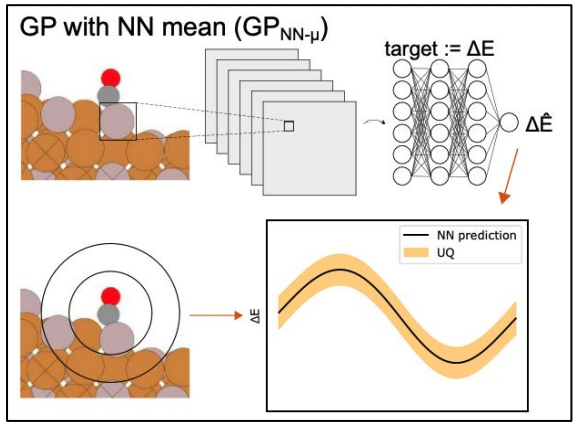


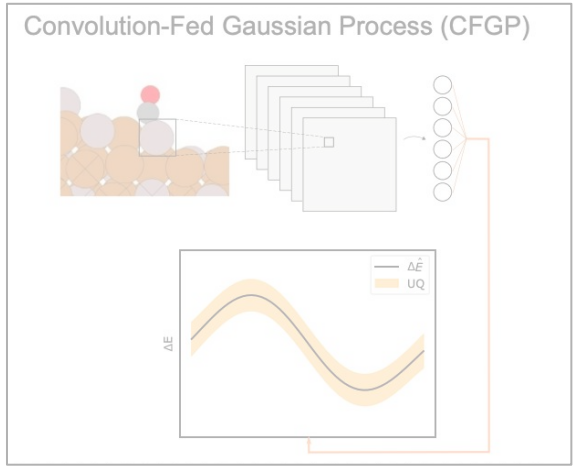








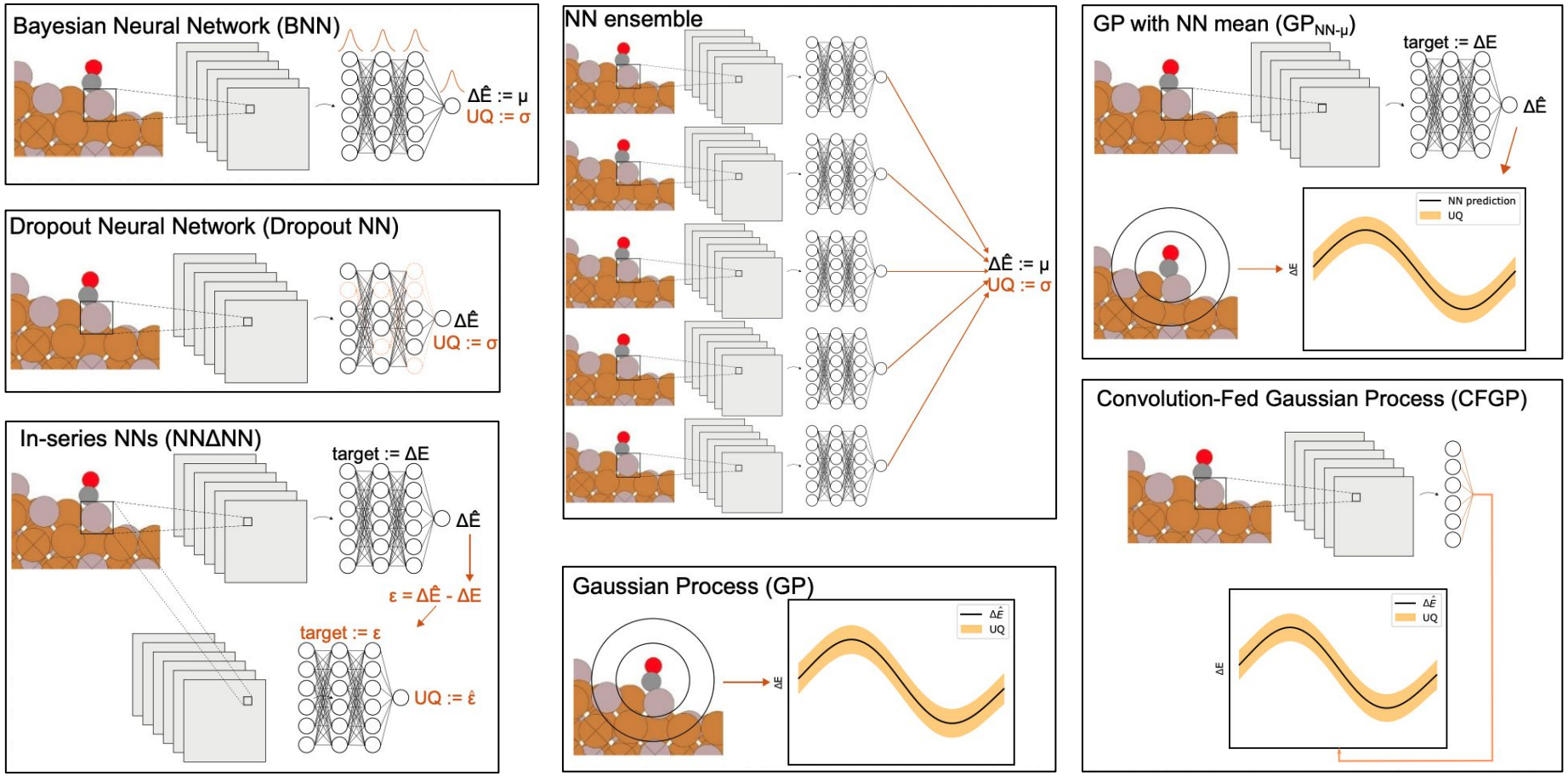




"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

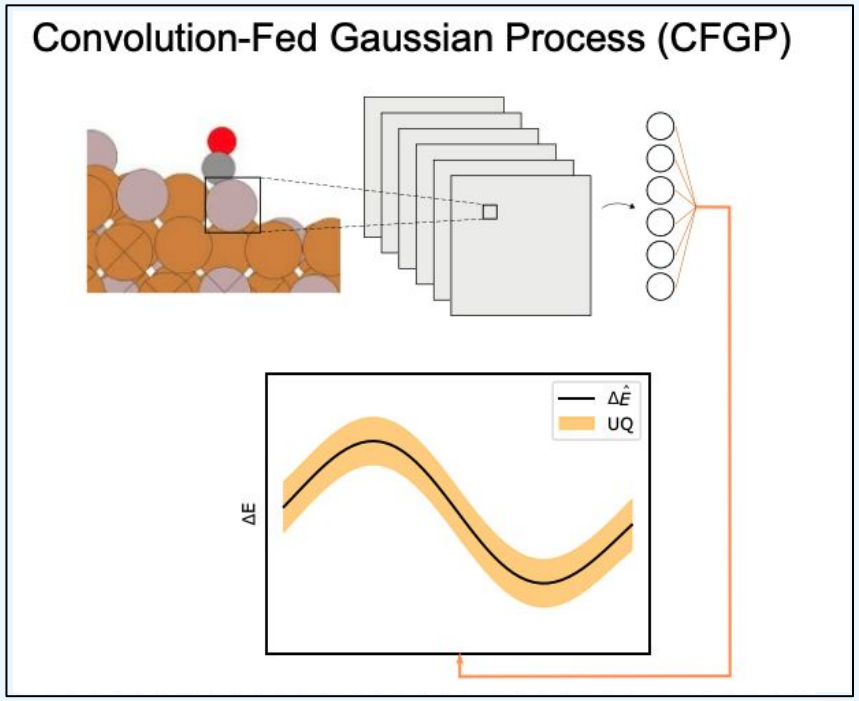
Deep Uncertainty Models —Example: Computational Catalyst Design



"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

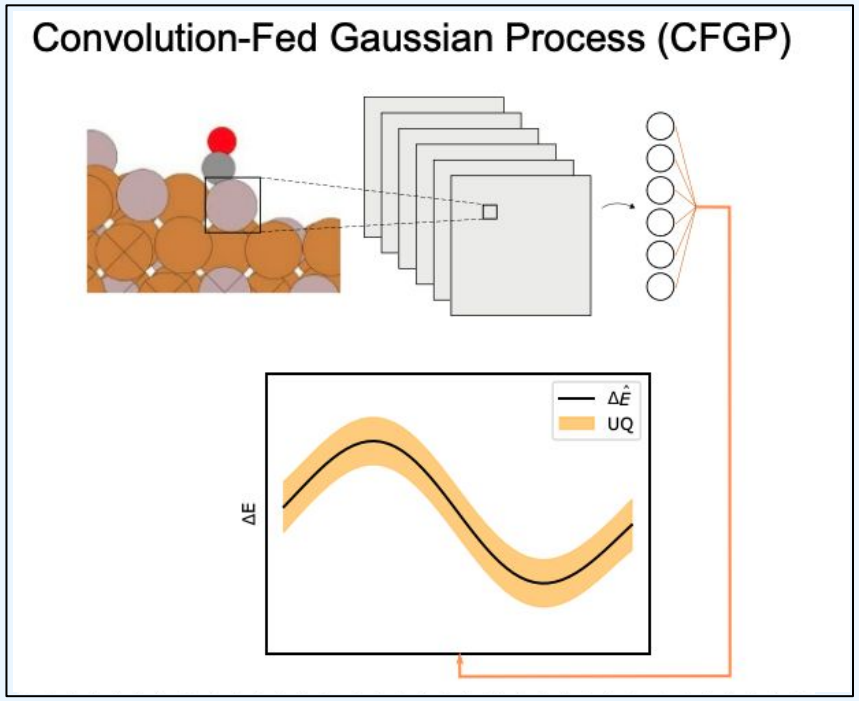
Deep Uncertainty Models —Example: Computational Catalyst Design



"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

Deep Uncertainty Models —Example: Computational Catalyst Design

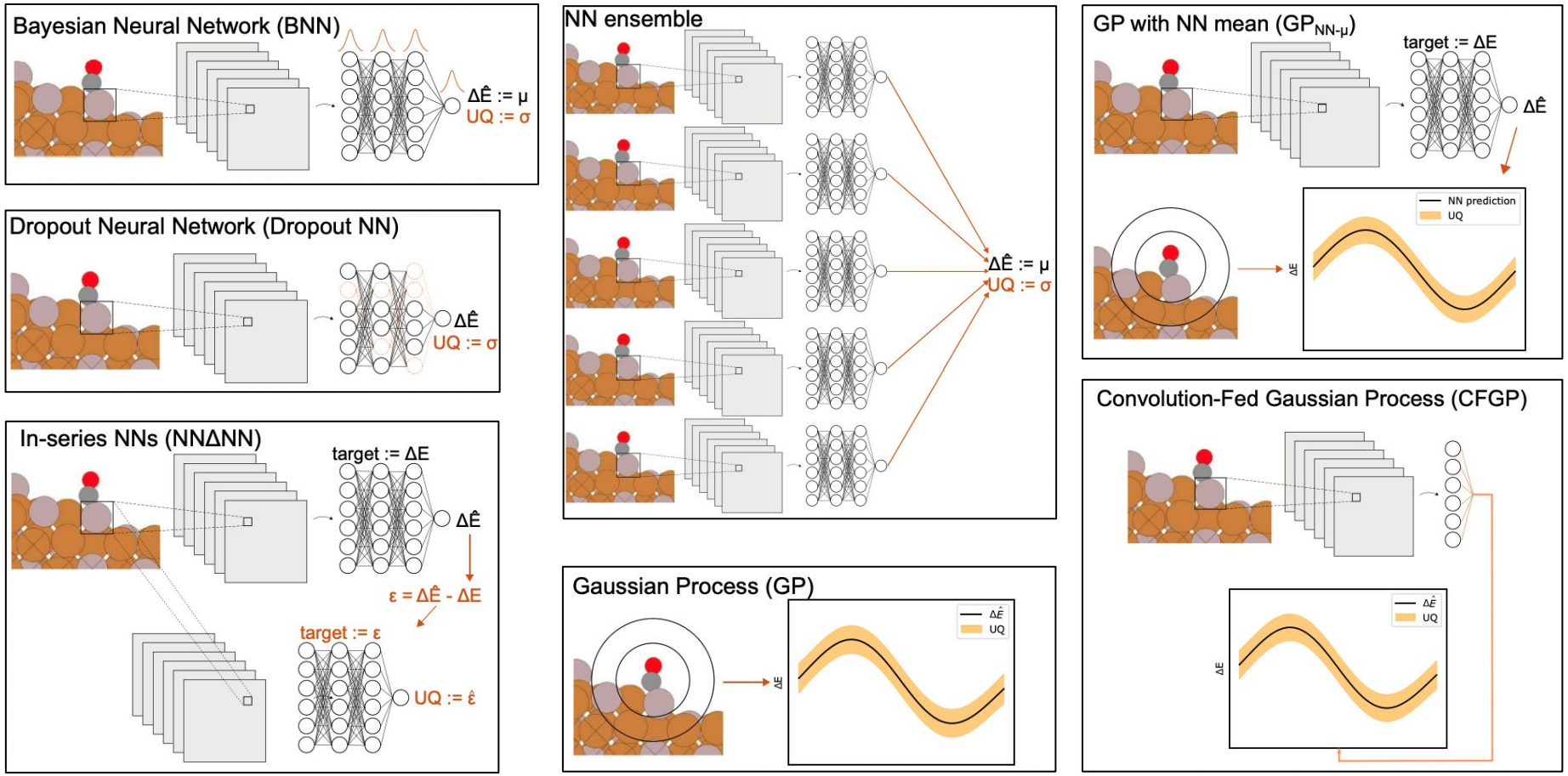


- Here, we use the neural network to learn good representations *(e.g., for a GP)*.
- For any input, first make a prediction w/ neural network.
- Use output at an intermediate layer as new transformed input (into a continuous space).
- Then define a Gaussian process on this new representation of inputs.

"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

Deep Uncertainty Models —Example: Computational Catalyst Design



"Methods for comparing uncertainty quantifications for material property predictions", *Tran, *Neiswanger, et al., MLST. 2020

"Computational catalyst discovery: Active classification through myopic multiscale sampling", *Tran, *Neiswanger, et al., J. Chem. Phys. 2021

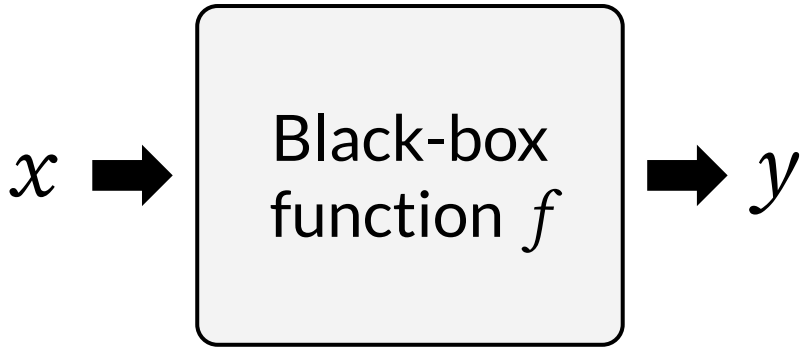
Bayesian Optimization

Expensive Black-box Functions

Expensive Black-box Functions

Unknown

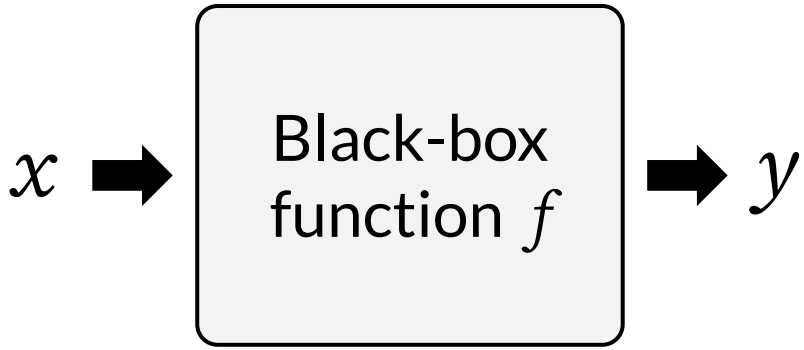
Many real-world systems can be modeled as an **expensive black-box function** f.



Expensive Black-box Functions

Many real-world systems can be modeled as an **expensive black-box function** f.

MATERIALS SCIENCE

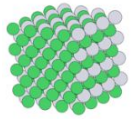


Expensive Black-box Functions

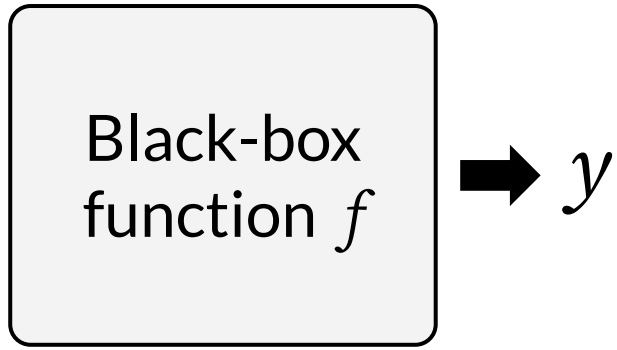
Many real-world systems can be modeled as an **expensive black-box function** f.

MATERIALS SCIENCE

Material Composition



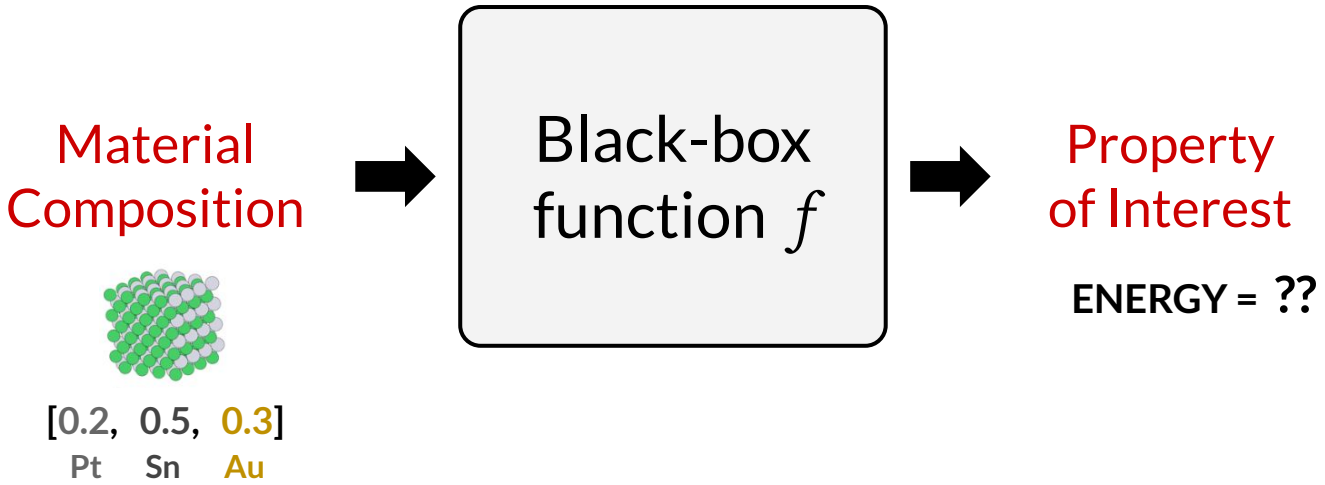
[0.2, 0.5, 0.3]
Pt Sn Au



Expensive Black-box Functions

Many real-world systems can be modeled as an **expensive black-box function** f.

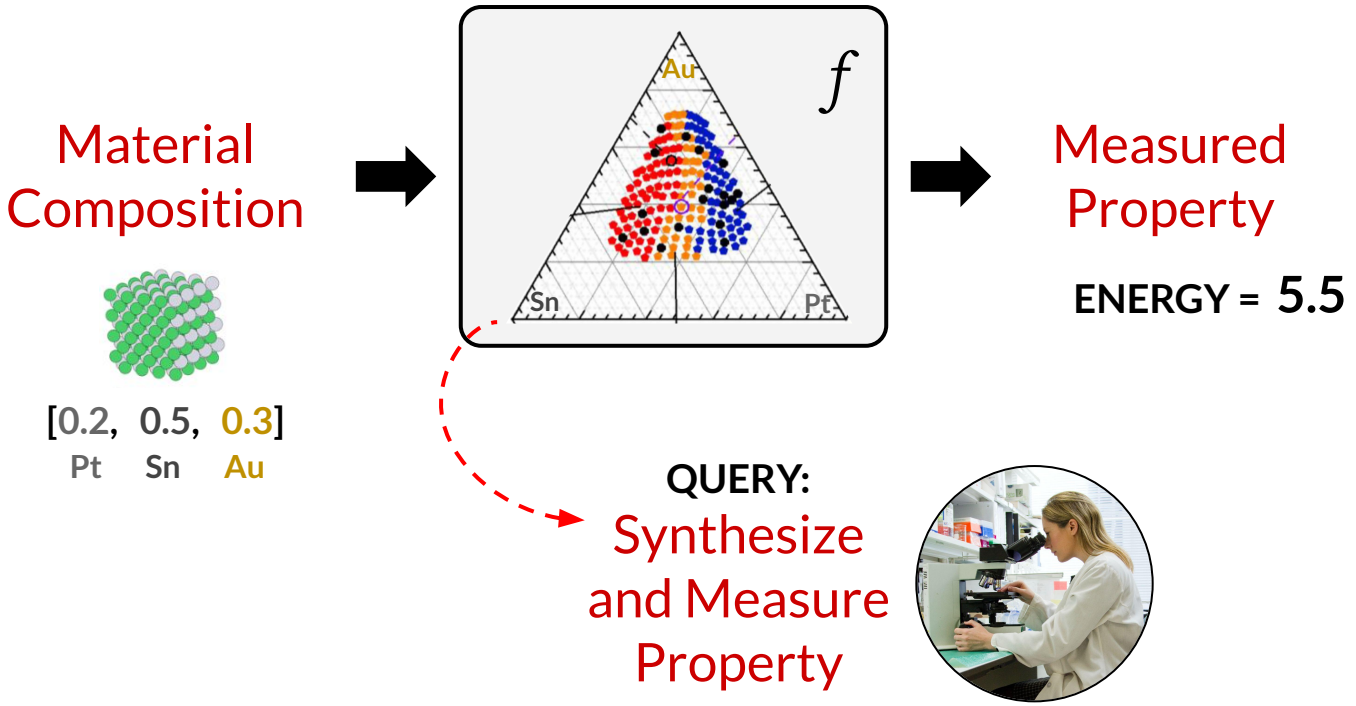
MATERIALS SCIENCE



Expensive Black-box Functions

Many real-world systems can be modeled as an **expensive black-box function** f.

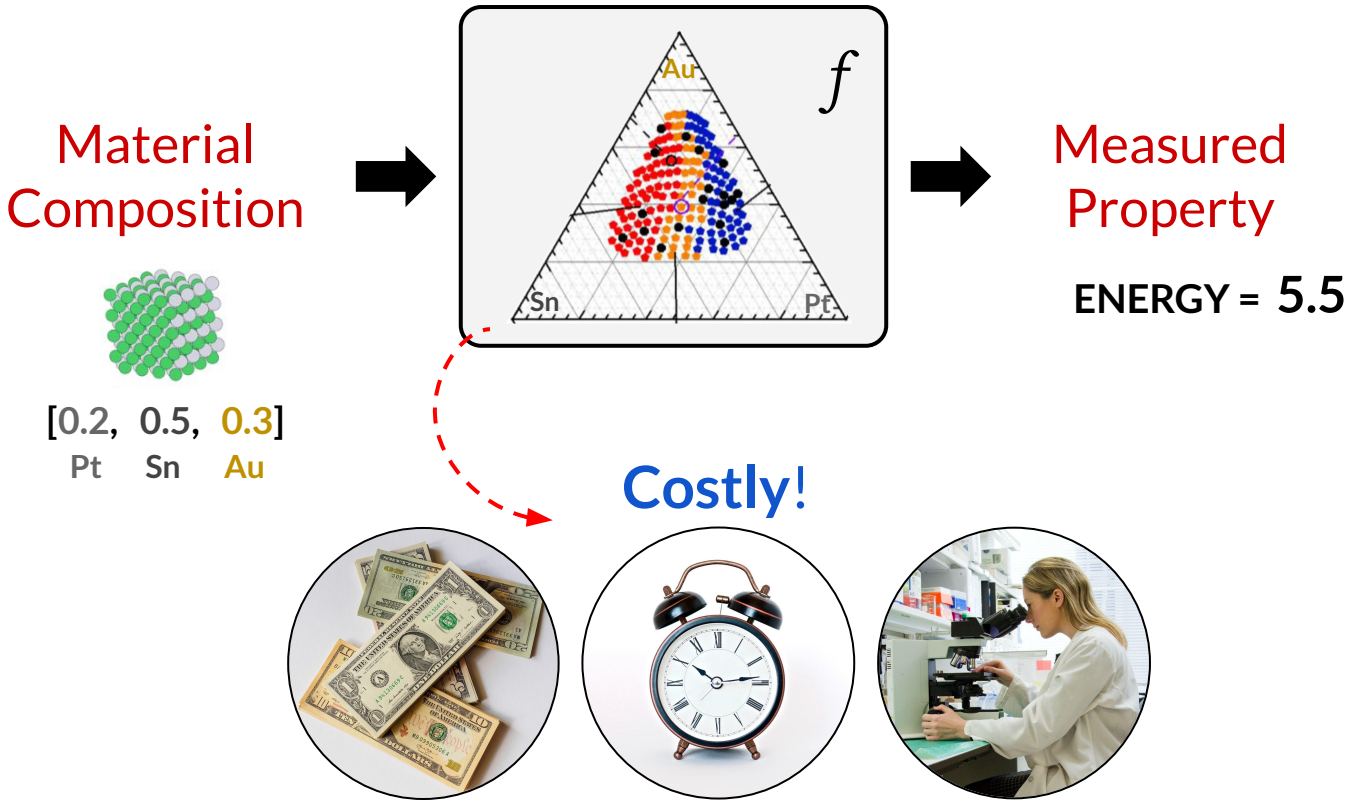
MATERIALS SCIENCE



Expensive Black-box Functions

Many real-world systems can be modeled as an **expensive black-box function** f.

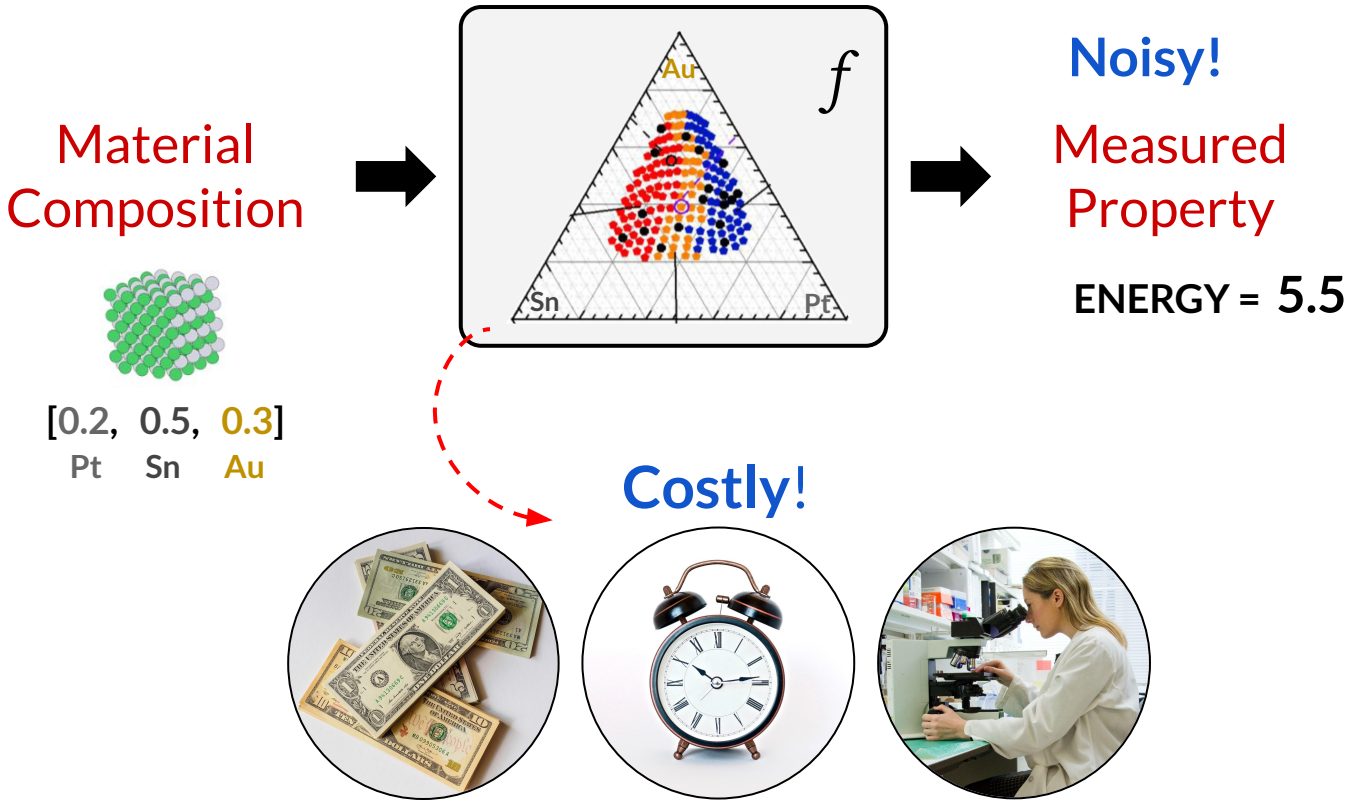
MATERIALS SCIENCE



Expensive Black-box Functions

Many real-world systems can be modeled as an **expensive black-box function** f.

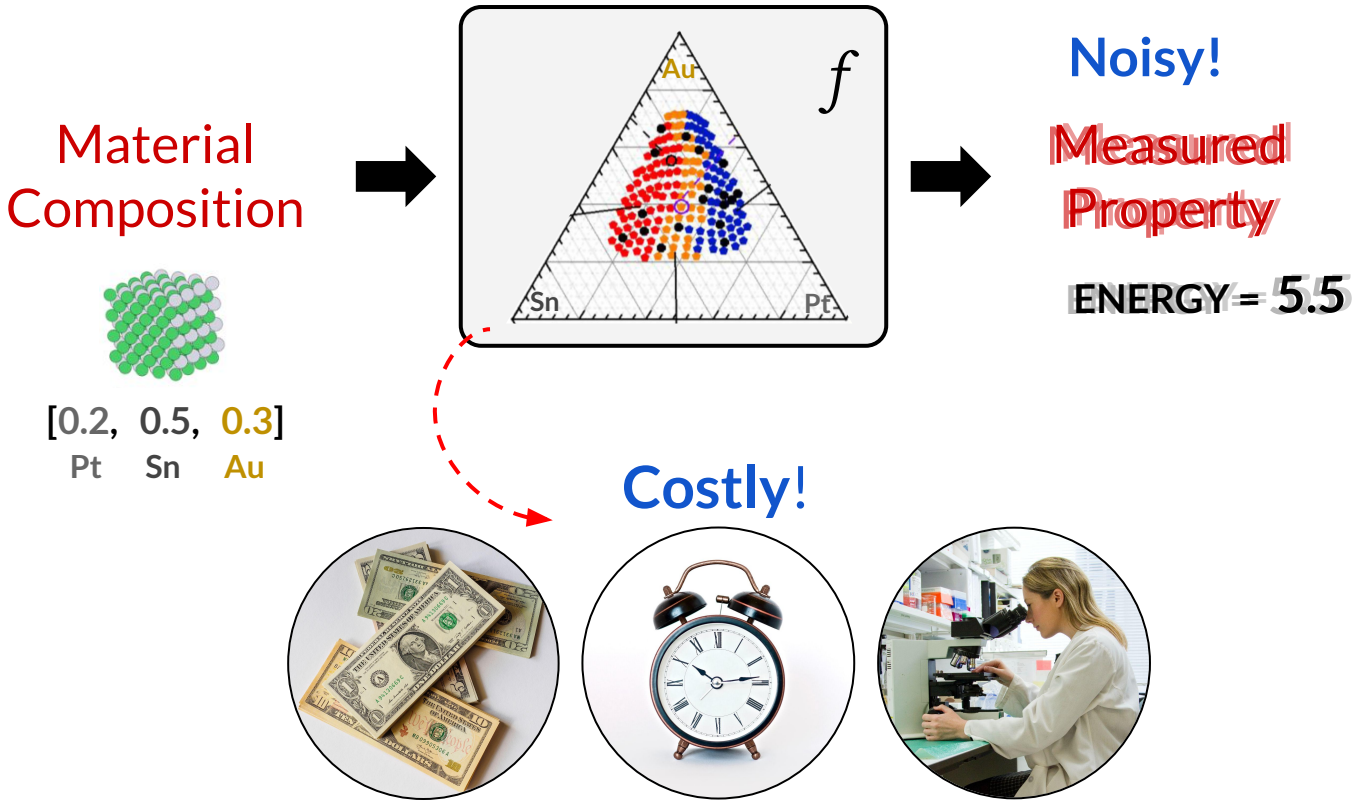
MATERIALS SCIENCE



Expensive Black-box Functions

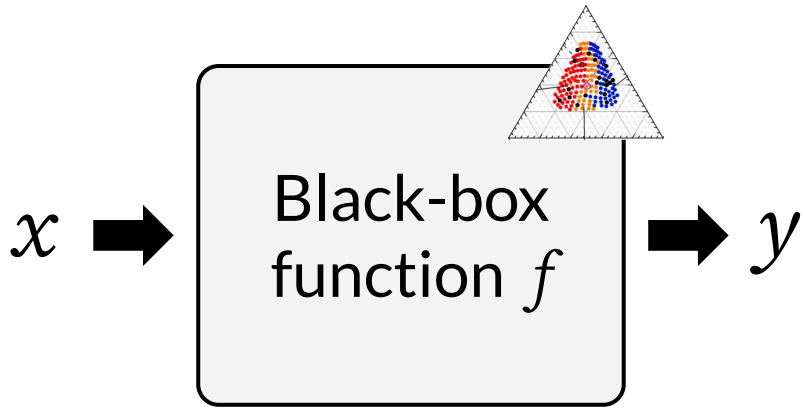
Many real-world systems can be modeled as an **expensive black-box function** f.

MATERIALS SCIENCE



Expensive Black-box Functions

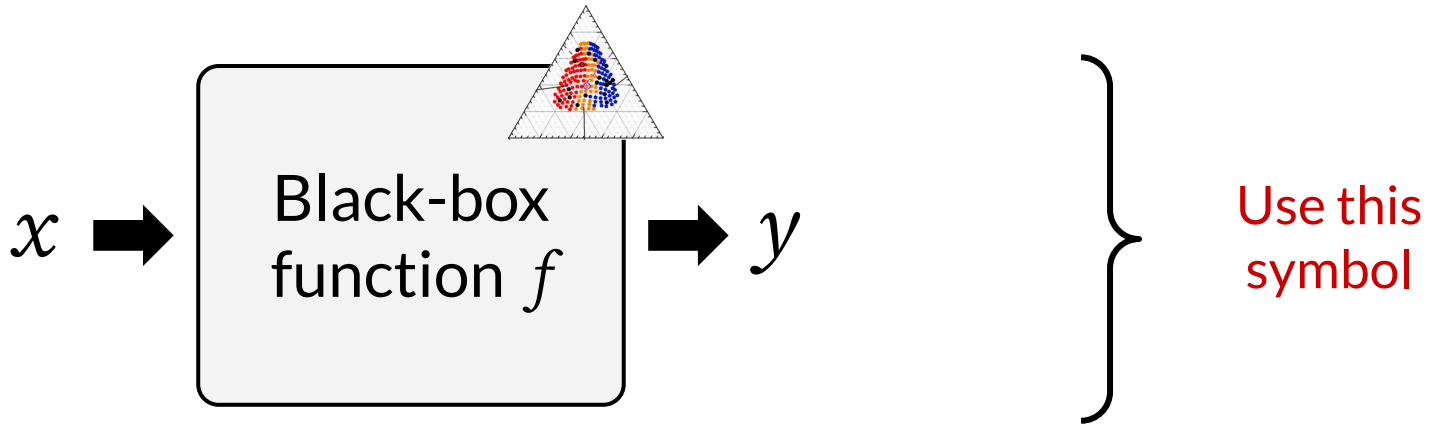
Many real-world systems can be modeled as an **expensive black-box function** f.



Use this symbol

Expensive Black-box Functions

Many real-world systems can be modeled as an **expensive black-box function** f.



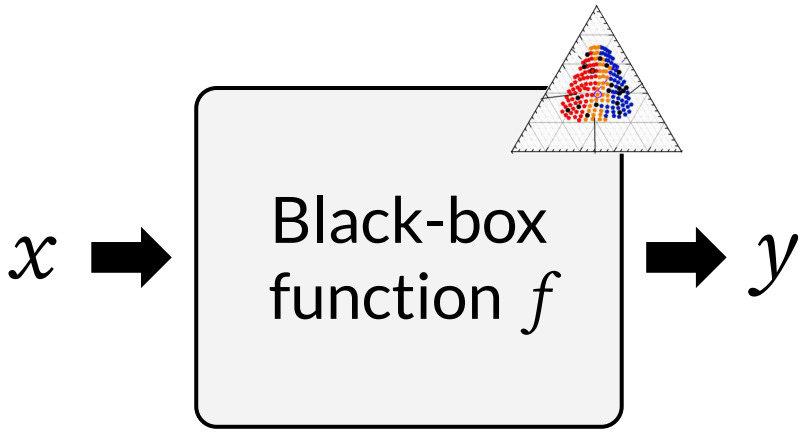
Leads to task of **black-box global optimization**.

Black-Box Global Optimization

Black-Box Global Optimization

Suppose we have an **expensive black-box function** $f : \mathcal{X} \to \mathbb{R}$.

Can ***query*** f

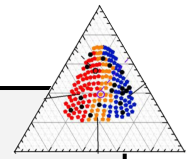


Observe $y \sim f(x) + \varepsilon$

Black-Box Global Optimization

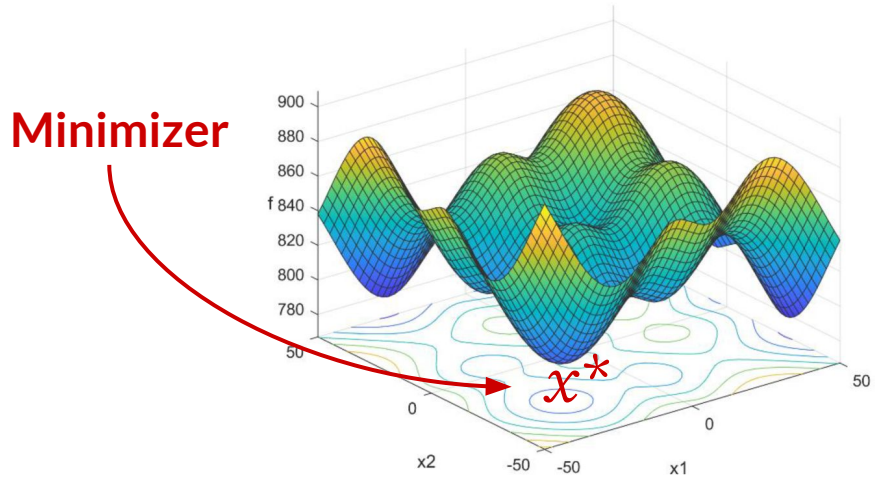
Suppose we have an **expensive black-box function** $f : \mathcal{X} \to \mathbb{R}$.

Can ***query*** f x → Black-box function f → y **Observe** $y \sim f(x) + \varepsilon$



Goal: estimate the location of a global optima.

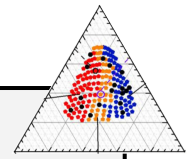
$$x^* = \arg\min_{x \in \mathcal{X}} f(x)$$



Black-Box Global Optimization

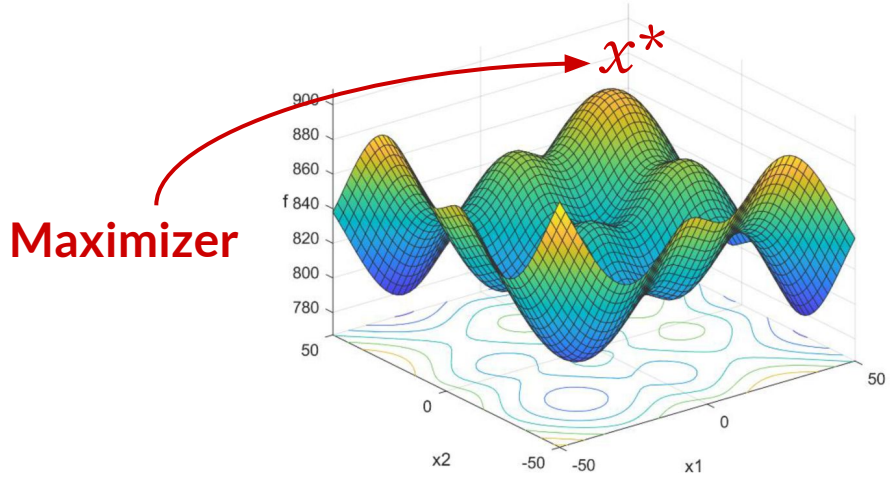
Suppose we have an **expensive black-box function** $f : \mathcal{X} \to \mathbb{R}$.

Can ***query*** f x → Black-box function f → y **Observe** $y \sim f(x) + \varepsilon$



Goal: estimate the location of a global optima.

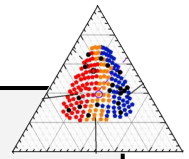
$$x^* = \arg\max_{x \in \mathcal{X}} f(x)$$



Black-Box Global Optimization

Suppose we have an **expensive black-box function** $f : \mathcal{X} \to \mathbb{R}$.

Can ***query*** f x ➡ Black-box function f ➡ y **Observe** $y \sim f(x) + \varepsilon$

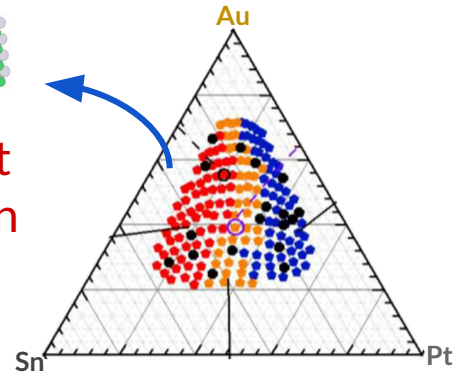


Goal: estimate the location of a global optima.

$$x^* = \arg\max_{x \in \mathcal{X}} f(x)$$

$x^* =$ 

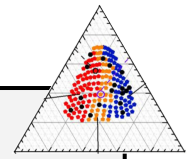
"Greatest activation energy"



Black-Box Global Optimization

Suppose we have an **expensive black-box function** $f : \mathcal{X} \to \mathbb{R}$.

Can ***query*** f x → Black-box function f → y **Observe** $y \sim f(x) + \varepsilon$

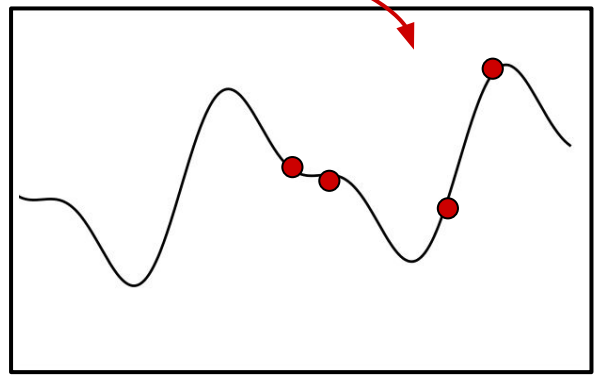


Goal: estimate the location of a global optima.

In a **limited budget** of T queries.

Few queries

No gradient information



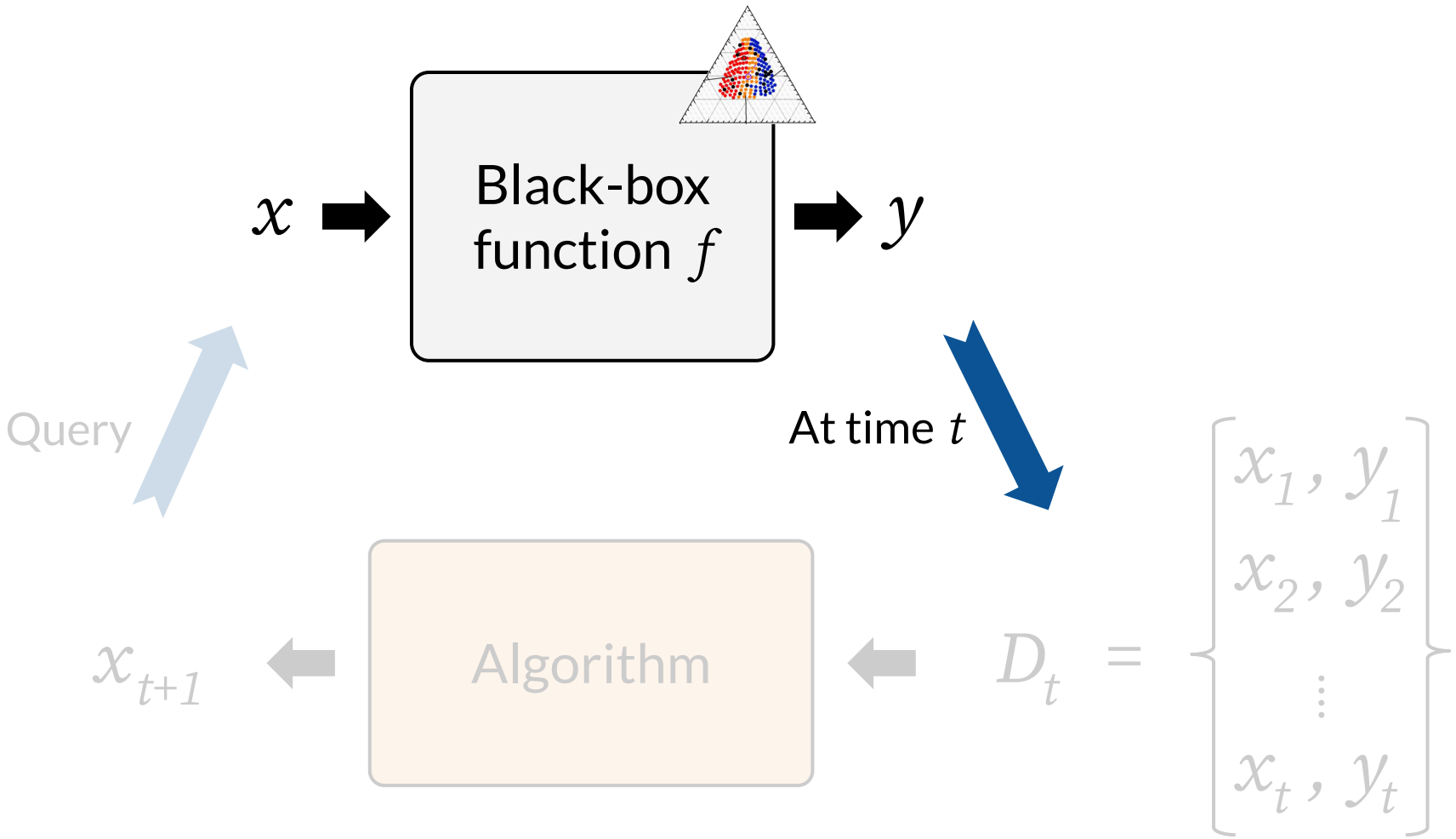
Black-Box Global Optimization — Usual Setup

Black-Box Global Optimization — Usual Setup

Sequence of time steps $t = 1, 2, \ldots$

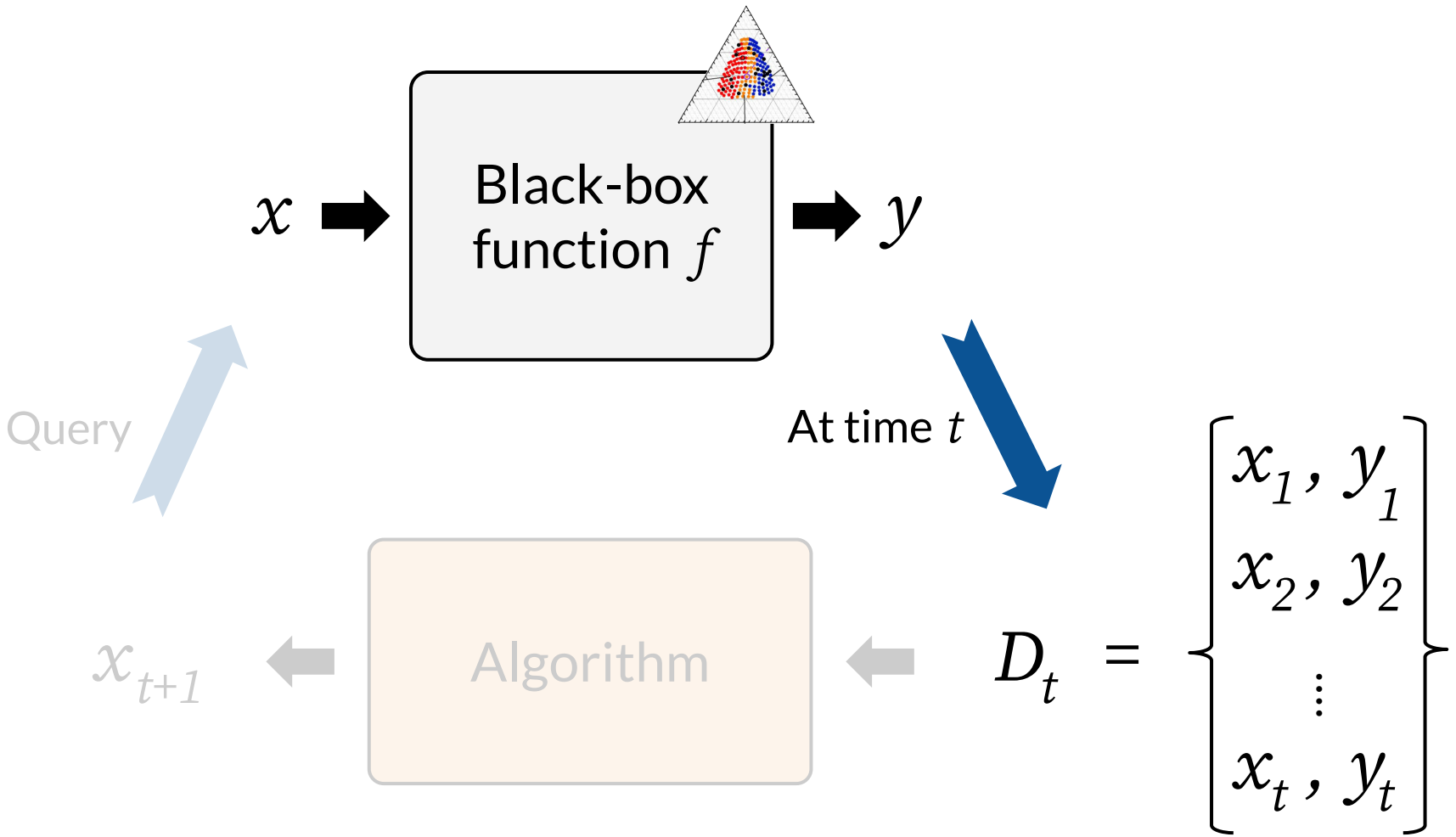
Black-Box Global Optimization — Usual Setup

Sequence of time steps $t = 1, 2, \ldots$



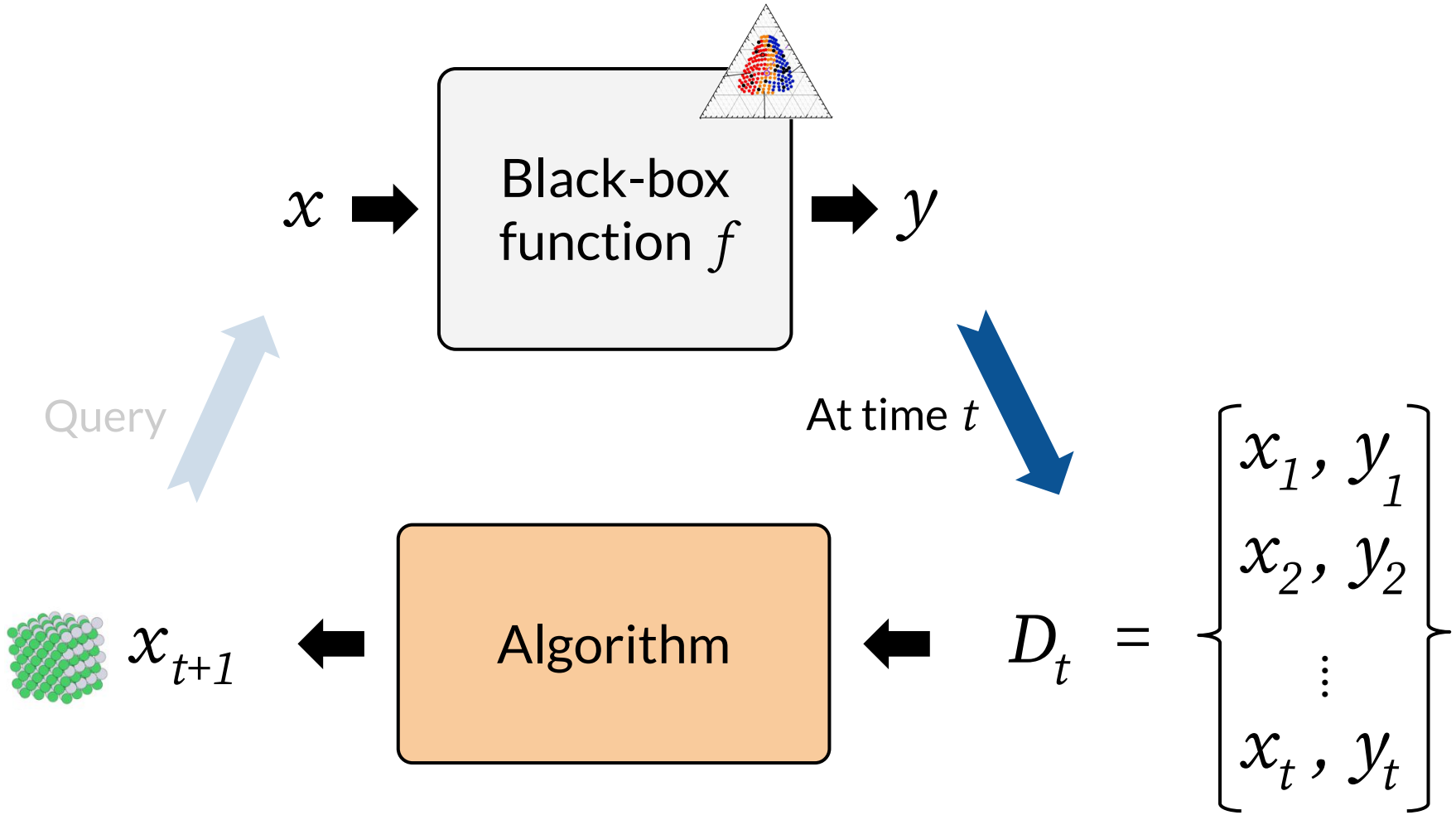
Black-Box Global Optimization — Usual Setup

Sequence of time steps $t = 1, 2, \ldots$



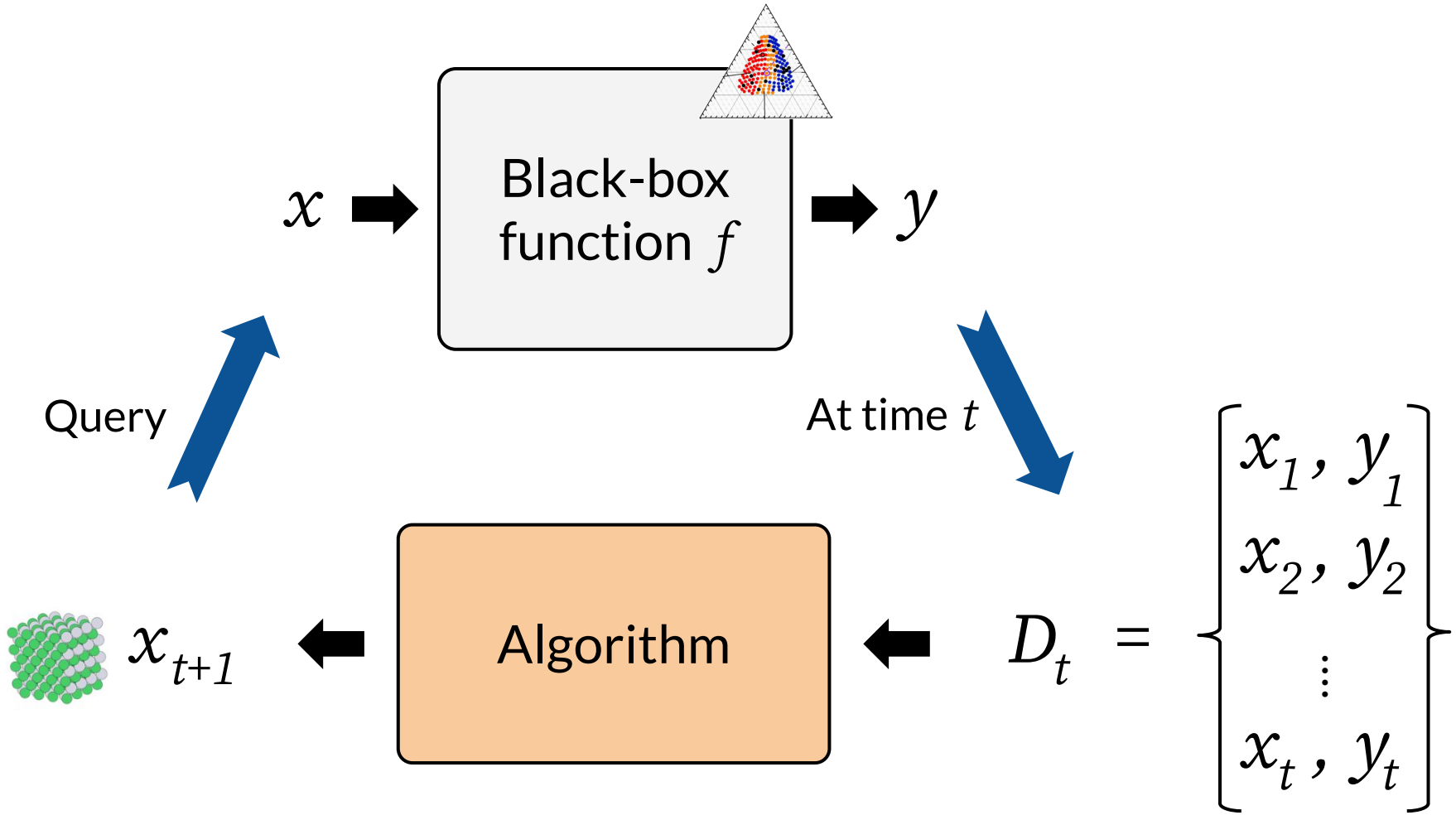
Black-Box Global Optimization — Usual Setup

Sequence of time steps $t = 1, 2, \ldots$



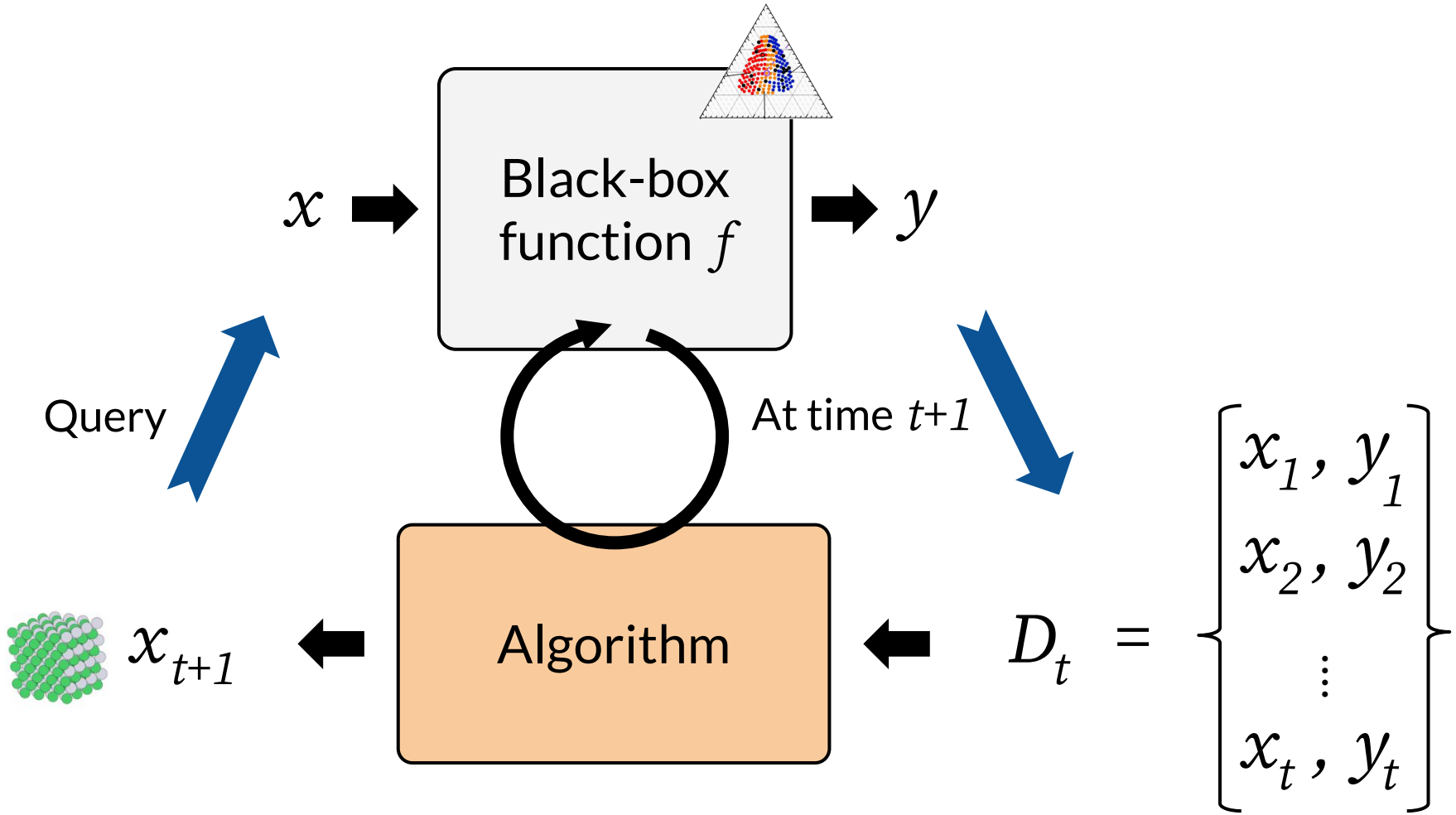
Black-Box Global Optimization — Usual Setup

Sequence of time steps $t = 1, 2, \ldots$



Black-Box Global Optimization — Usual Setup

Sequence of time steps $t = 1, 2, \ldots$



Bayesian Optimization — Motivation

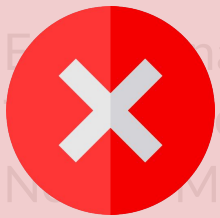
Derivative-free optimization:

- Evolutionary or genetic algorithms [FOW, 1966]
- Trust region methods [LM, 1944-1963]
- Nelder-Mead / simplex methods [NM, 1965]
- Simulated annealing [P, 1970; KGV, 1983]

Bayesian Optimization — Motivation

Derivative-free optimization:

- Random search [FOW, 1966]
- Pattern search methods [HJ, 1944-1963]
- Nelder-Mead / simplex methods [NM, 1965]
- Simulated annealing [P, 1970; KGV, 1983]



Can require 1000s of queries!

Need:

- Better sample-efficiency.
- Adaptive to history (data-driven).
- (Ideally) ability to incorporate any prior knowledge/beliefs about function.

Bayesian Optimization — Motivation

Derivative-free optimization:

- Evolutionary computation [FOW, 1966]
- Pattern search methods [HJ, 1944-1963]
- Nelder-Mead / Simplex methods [NM, 1965]
- Simulated annealing [P, 1970; KGV, 1983]

Can require 1000s of queries!

Need:

- Better sample-efficiency.
- Adaptive to history (data-driven).
- (Ideally) ability to incorporate any prior knowledge/beliefs about function.

What is Bayesian Opt? Uses a (probabilistic) predictive model to aid in optimization.

Bayesian Optimization — Motivation

Derivative-free optimization:

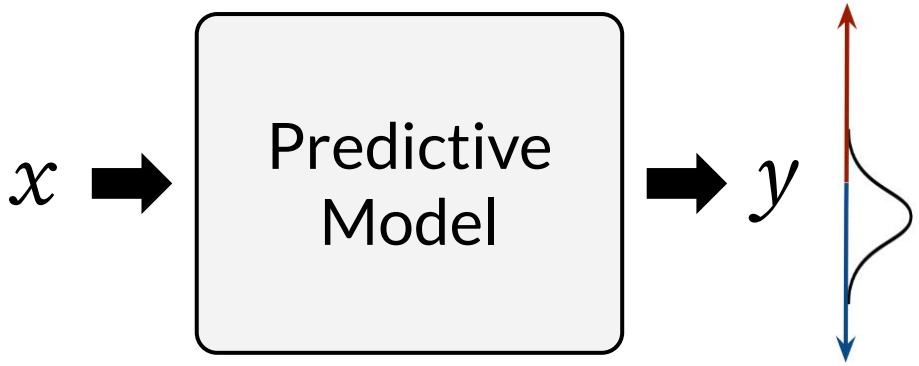
- Exhaustive search / grid search [FOW, 1966]
- Coordinate descent methods [LM, 1944-1963]
- Nelder-Mead / Simplex methods [N, 1965]
- Simulated annealing [P, 1970; KGV, 1983]

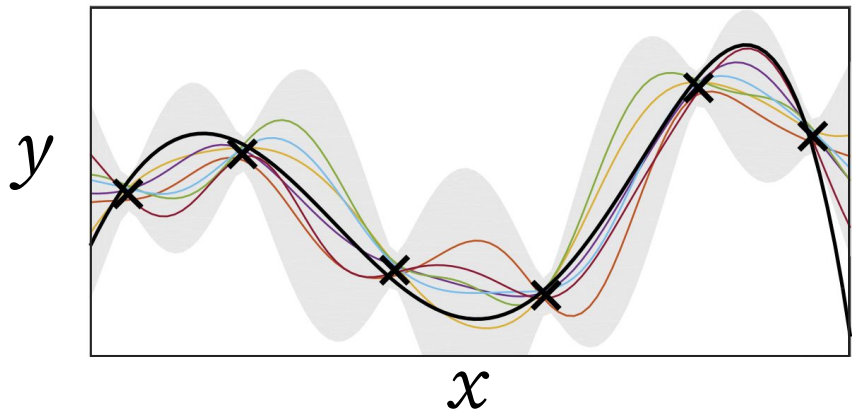
Can require 1000s of queries!

Need:

- Better sample-efficiency.
- Adaptive to history (data-driven).
- (Ideally) ability to incorporate any prior knowledge/beliefs about function.

What is Bayesian Opt? Uses a (probabilistic) predictive model to aid in optimization.





Bayesian Optimization — Motivation

Derivative-free optimization:

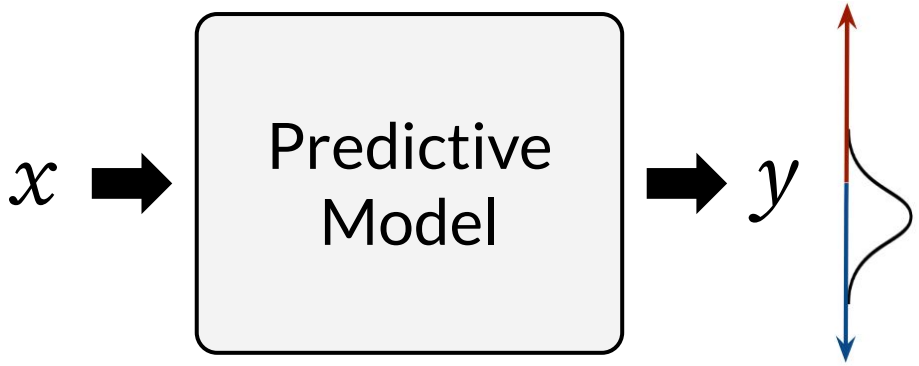
- Random search, Grid search, Factorial designs [FOW, 1966]
- Response surface methods [L, M, 1944-1963]
- Nelder-Mead, Simplex methods [N, M, 1965]
- Simulated annealing [P, 1970; KGV, 1983]

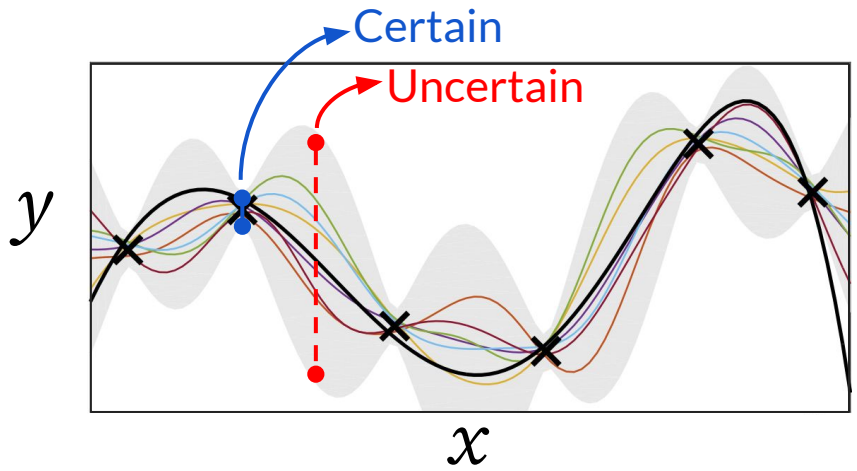
Can require 1000s of queries!

Need:

- Better sample-efficiency.
- Adaptive to history (data-driven).
- (Ideally) ability to incorporate any prior knowledge/beliefs about function.

What is Bayesian Opt? Uses a (probabilistic) predictive model to aid in optimization.





Bayesian Optimization — Motivation

Derivative-free optimization:

- Random search [FOW, 1966]
- Pattern search methods [LM, 1944-1963]
- Nelder-Mead, simplex methods [N, 1965]
- Simulated annealing [P, 1970; KGV, 1983]

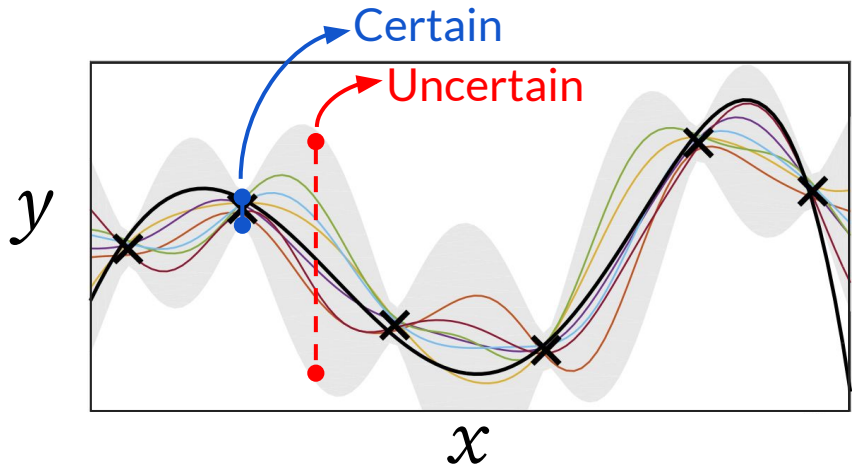
Can require 1000s of queries!

Need:

- Better sample-efficiency.
- Adaptive to history (data-driven).
- (Ideally) ability to incorporate any prior knowledge/beliefs about function.

What is Bayesian Opt? Uses a (probabilistic) predictive model to aid in optimization.

Model used for *intelligent* decision making ⇒ sample-efficient opt



Bayesian Optimization — Algorithm Overview

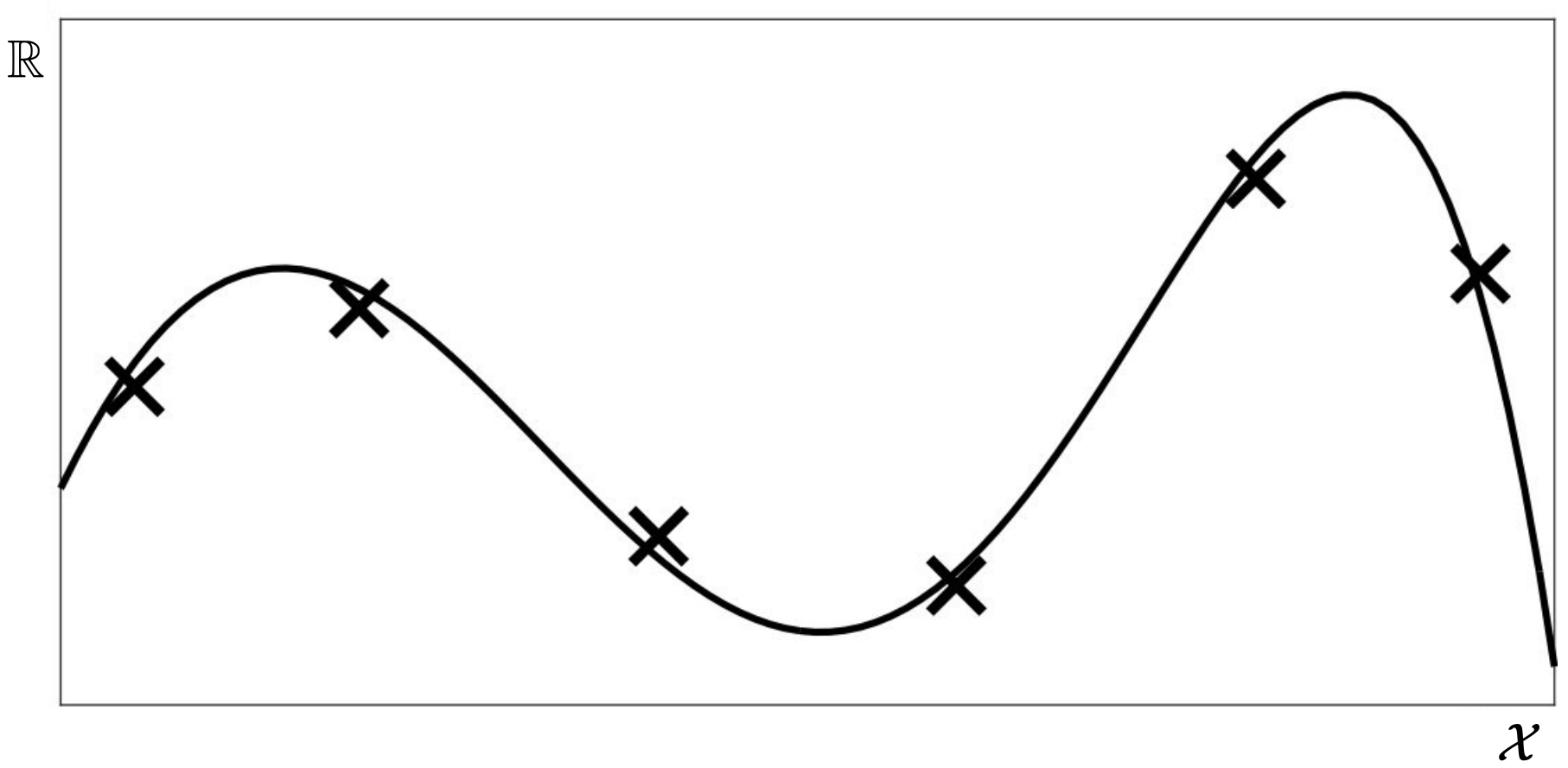
At each iteration of BO, three steps:

Bayesian Optimization — Algorithm Overview

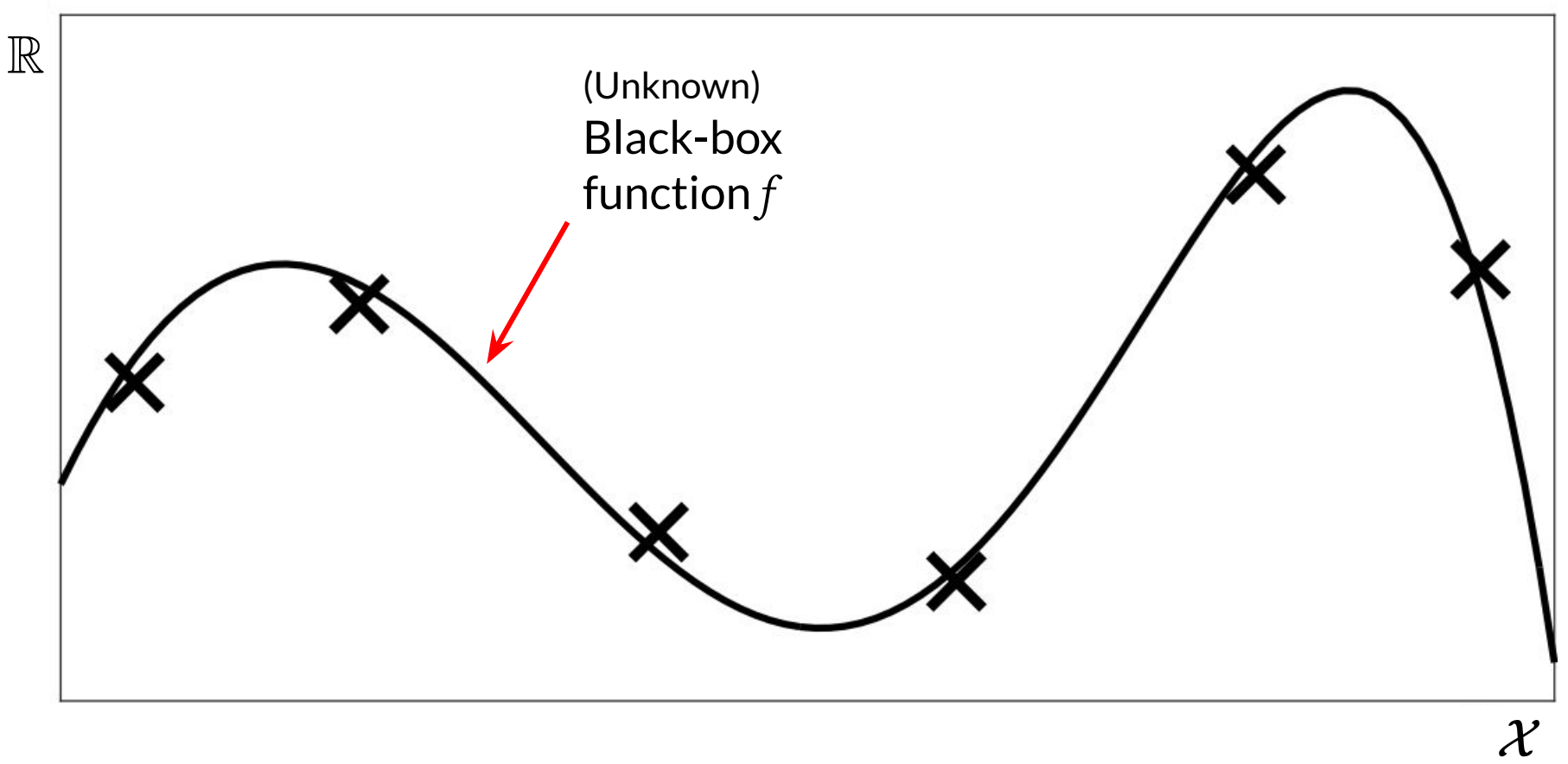
At each iteration of BO, three steps:

1. Fit model to dataset.
2. Optimize an *acquisition function* to select next point to query.
3. Query black-box function.

Bayesian Optimization — Visualization

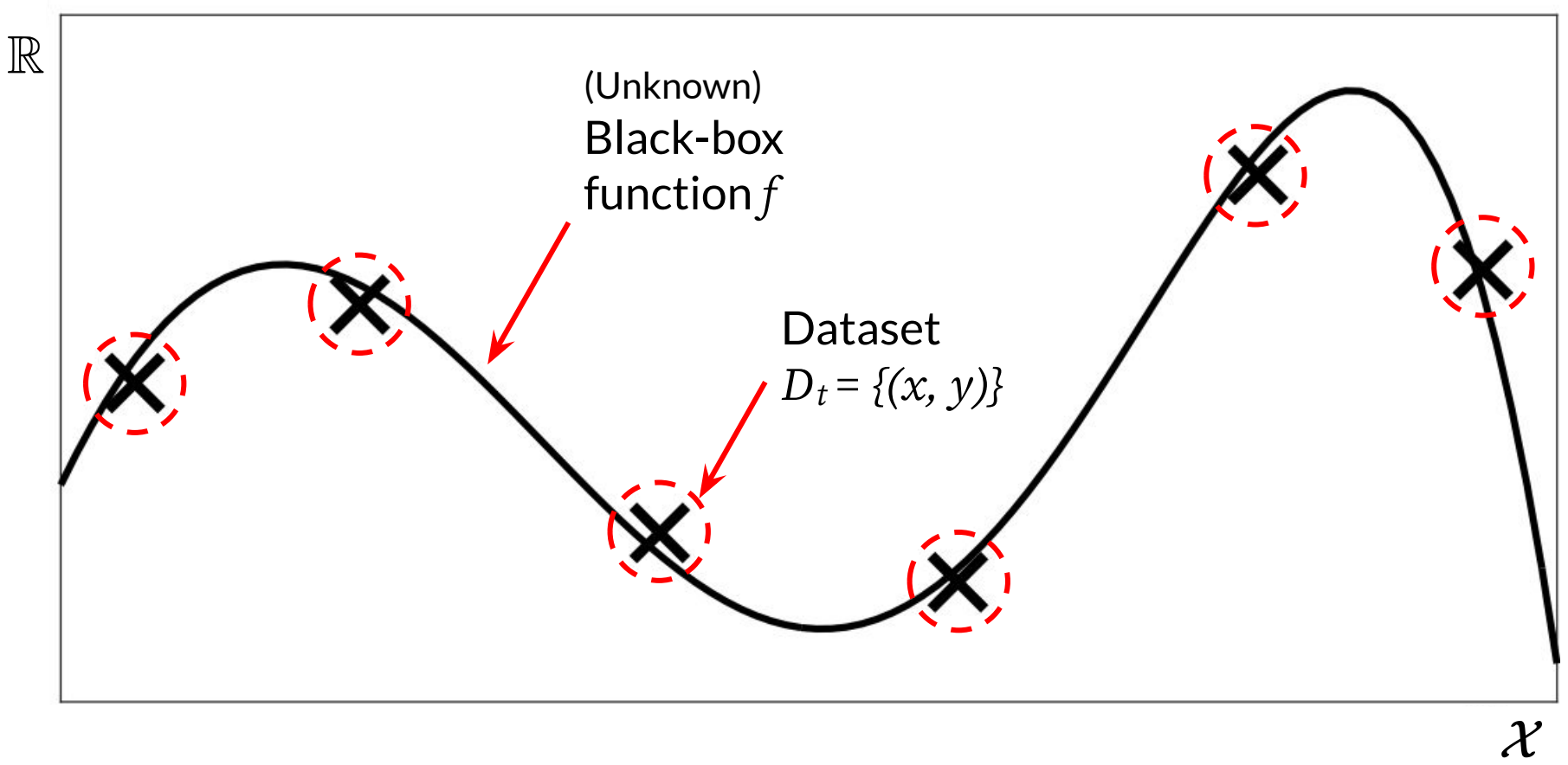


Bayesian Optimization — Visualization



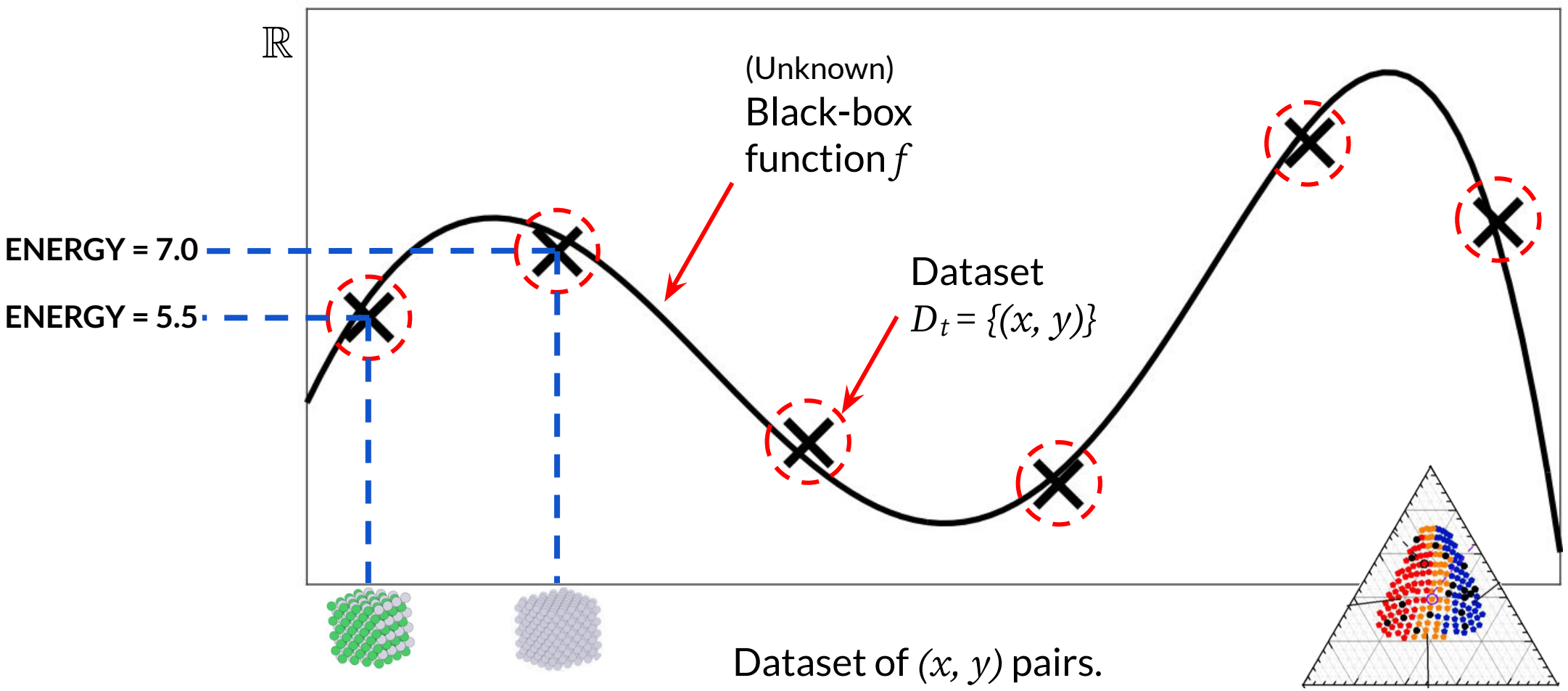
Unknown black-box f

Bayesian Optimization — Visualization

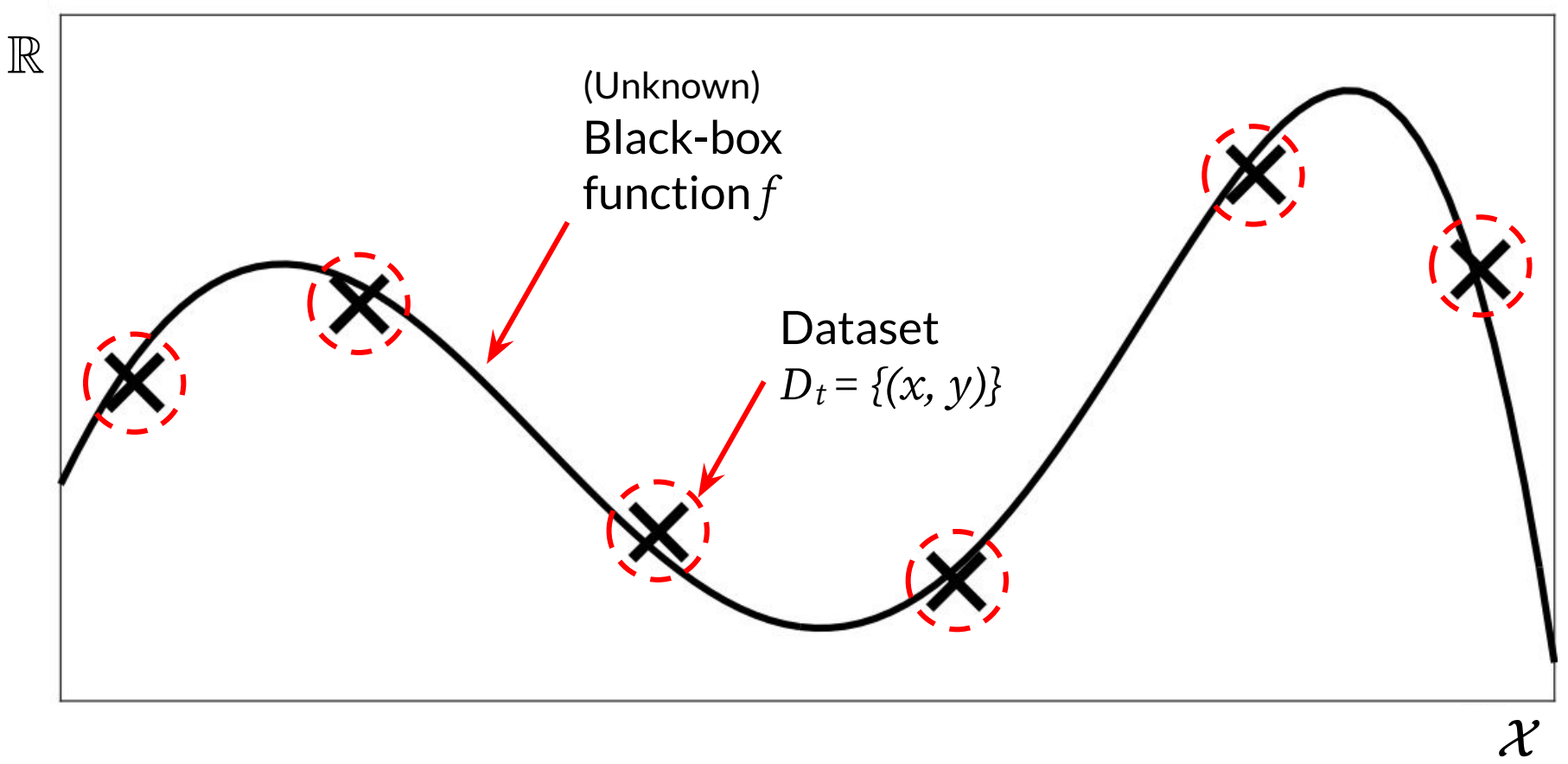


Unknown black-box f, and dataset of (x, y) pairs.

Bayesian Optimization — Visualization

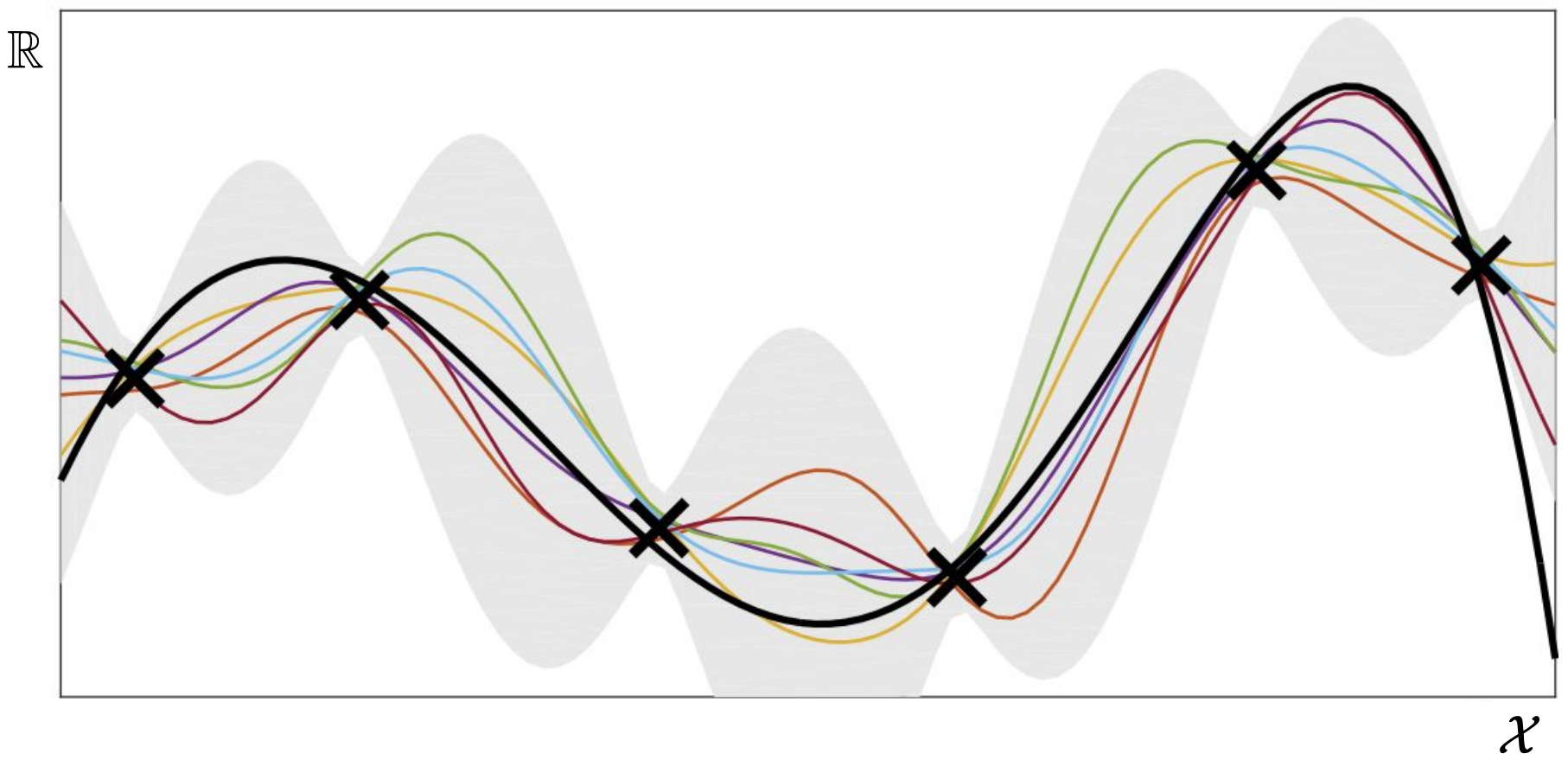


Bayesian Optimization — Visualization



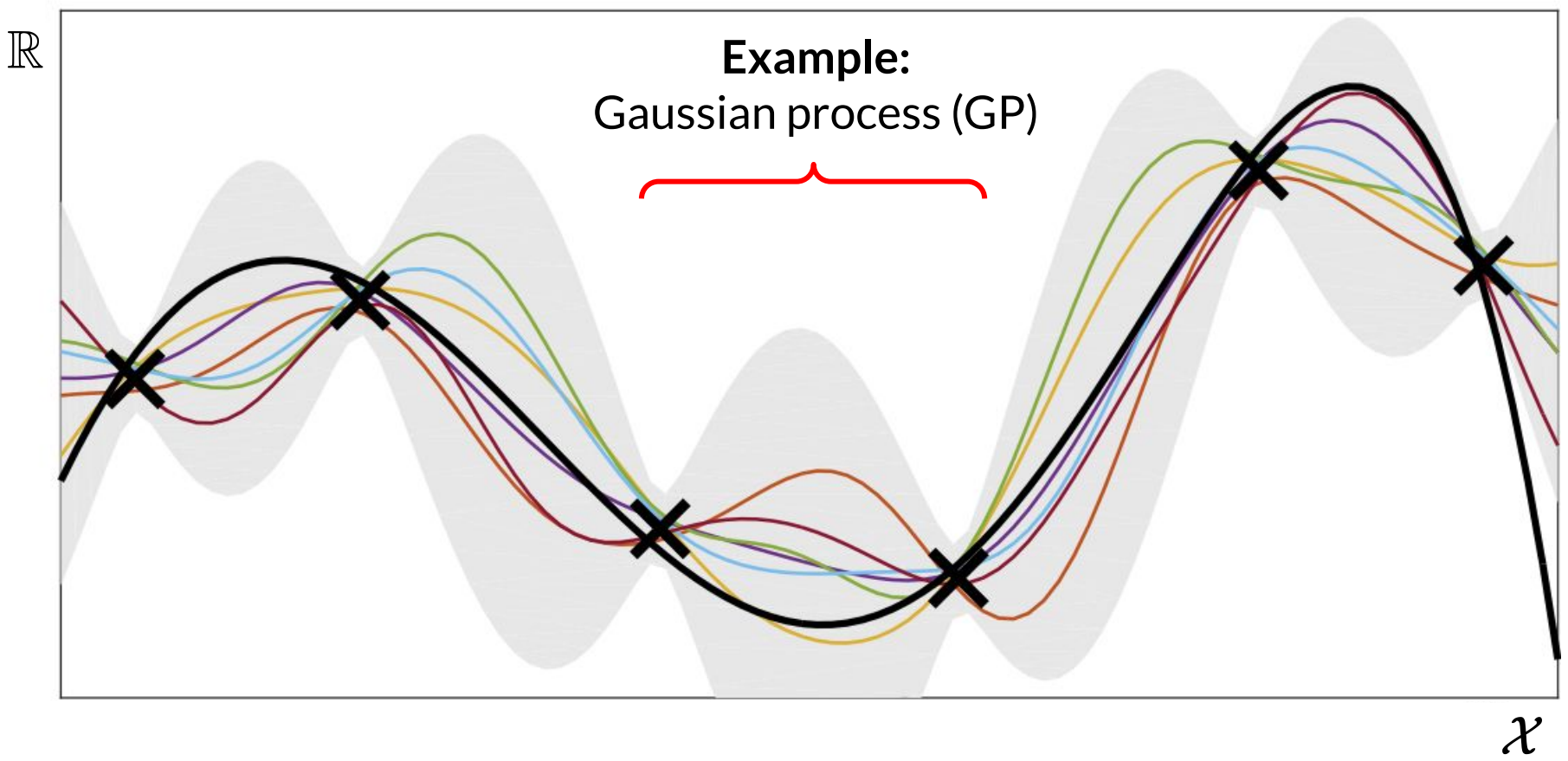
Dataset of (x, y) pairs.

Bayesian Optimization — Visualization



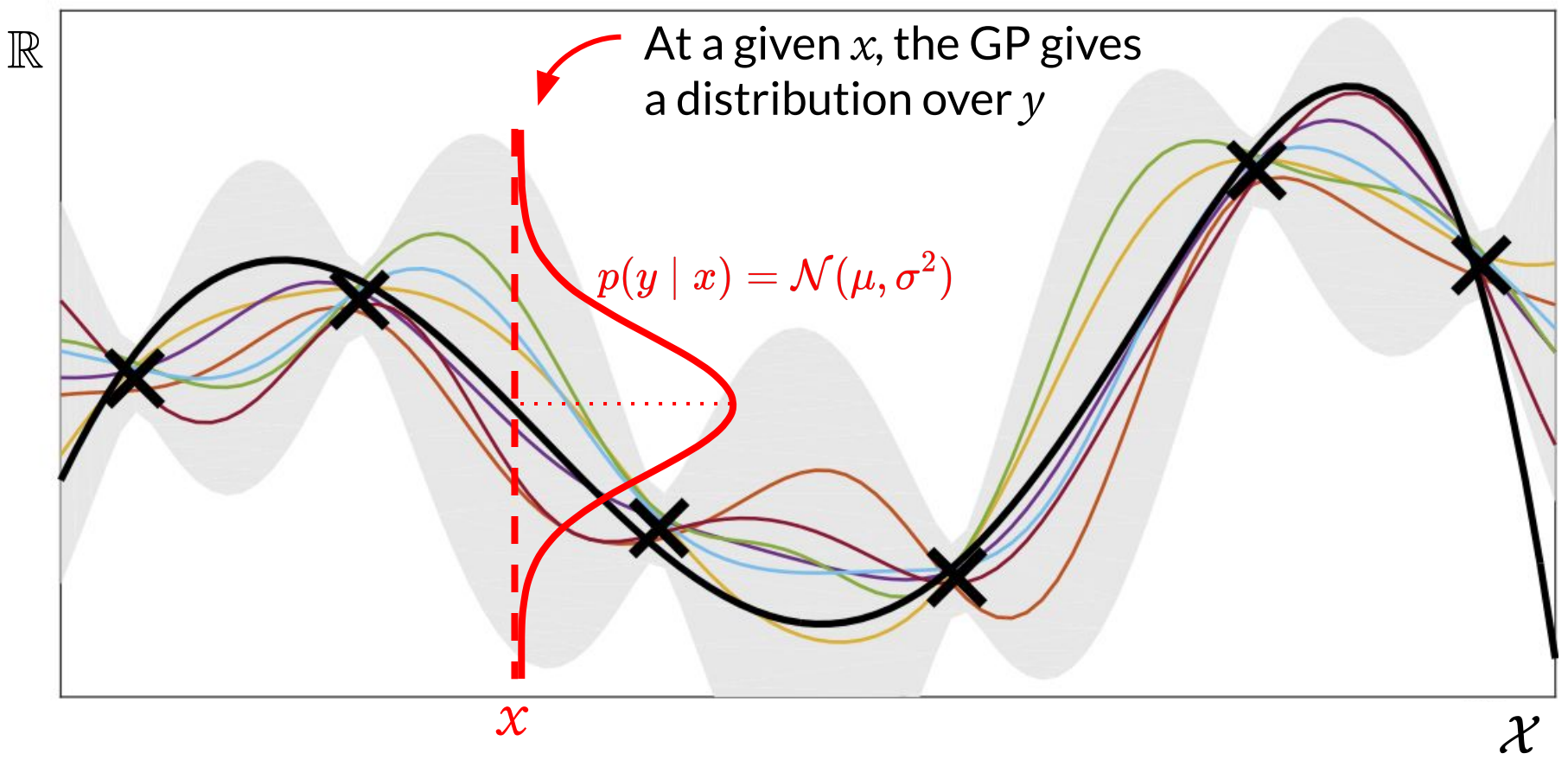
Given this dataset, can fit probabilistic model.

Bayesian Optimization — Visualization



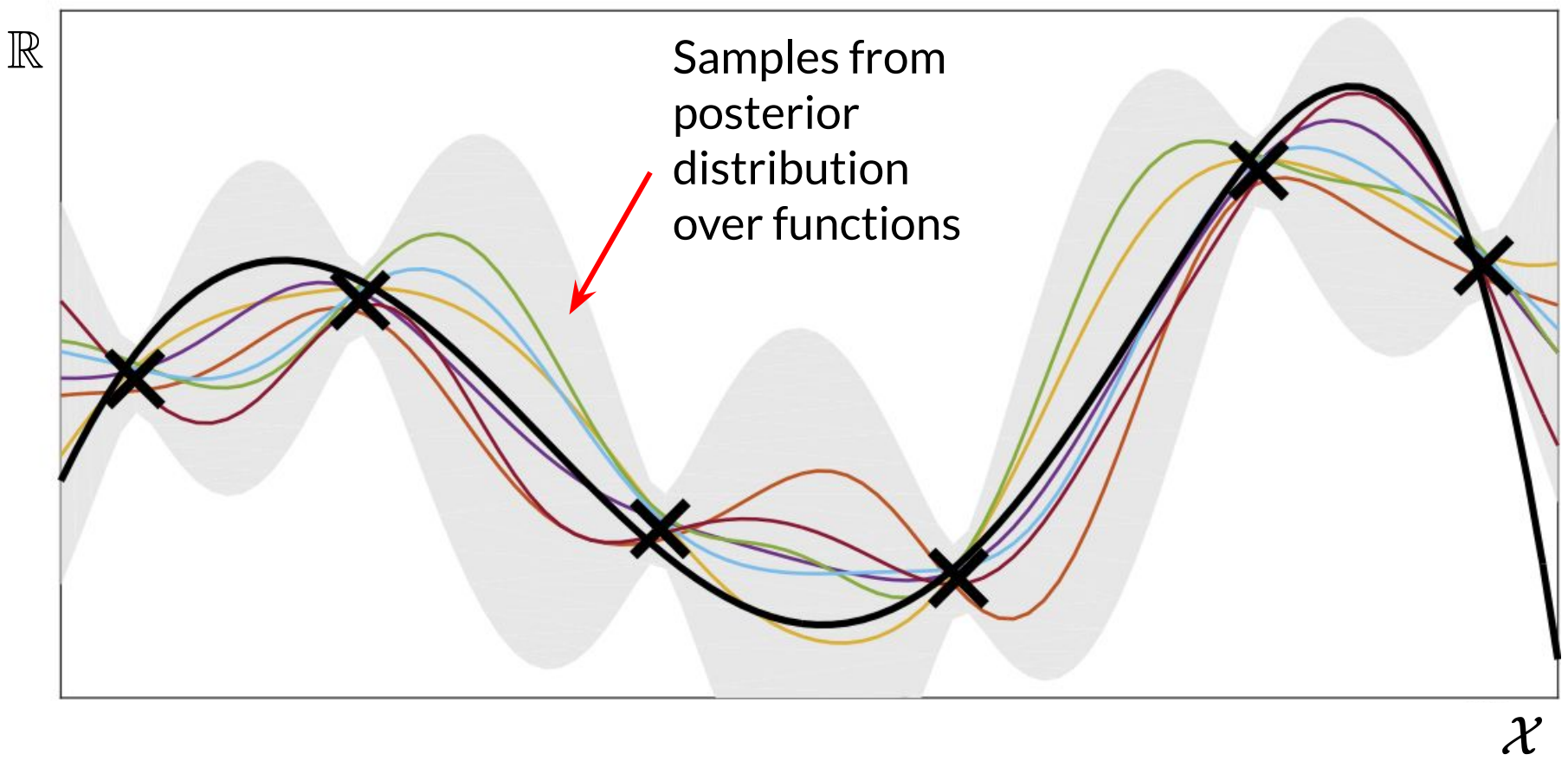
A popular choice for this model is a **Gaussian Process**.

Bayesian Optimization — Visualization



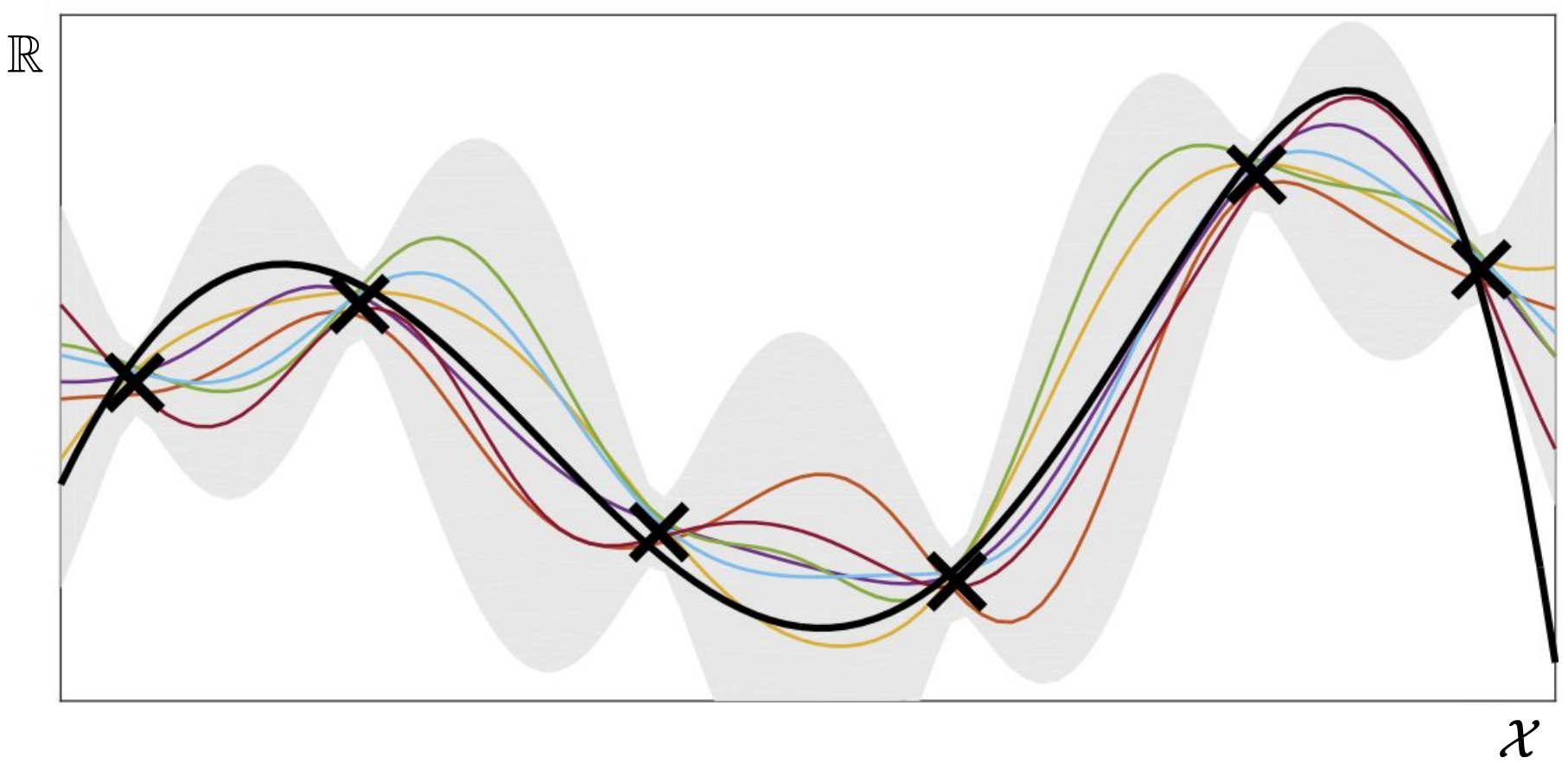
A popular choice for this model is a **Gaussian Process**.

Bayesian Optimization — Visualization



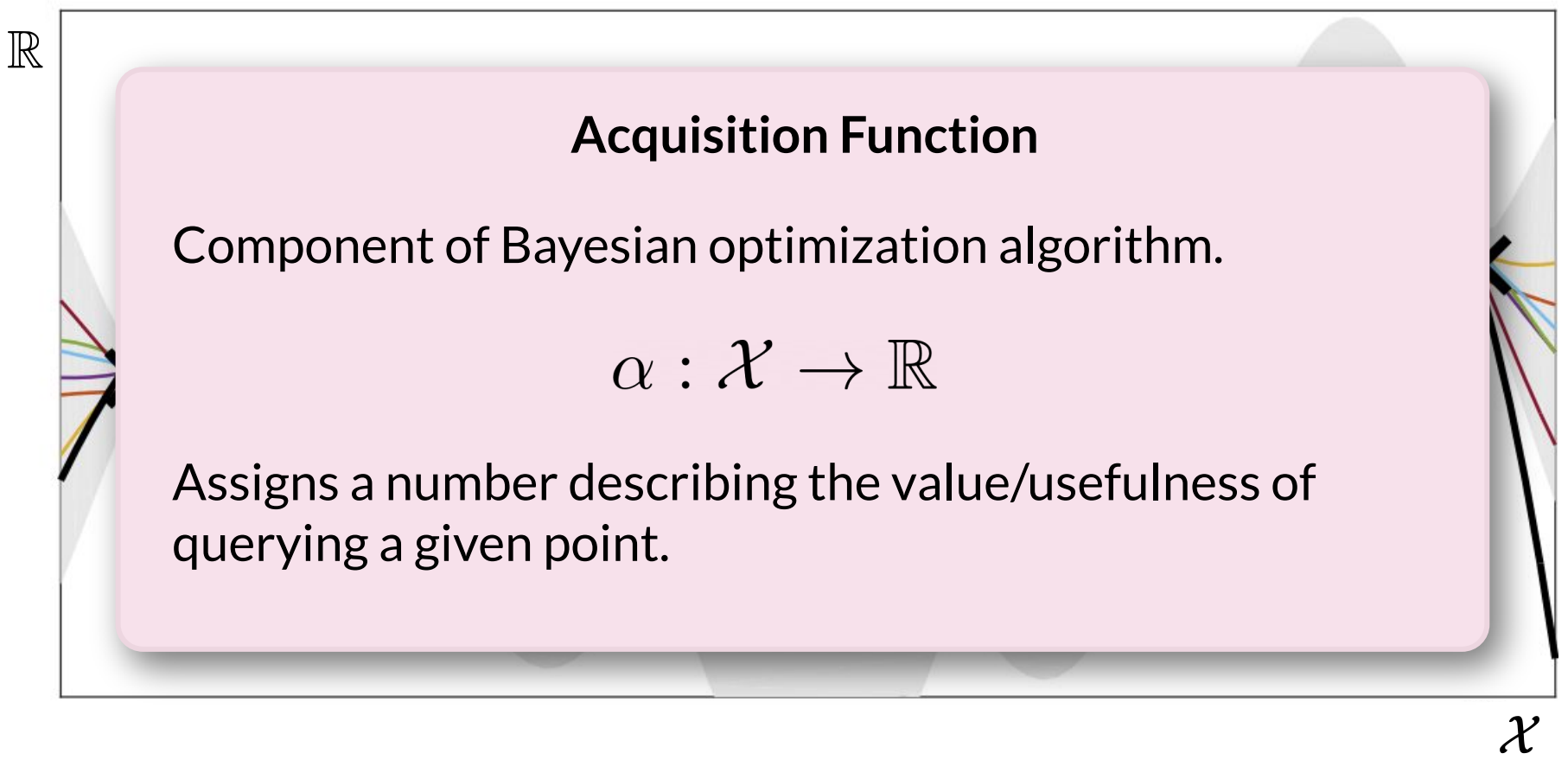
A popular choice for this model is a **Gaussian Process**.

Bayesian Optimization — Visualization



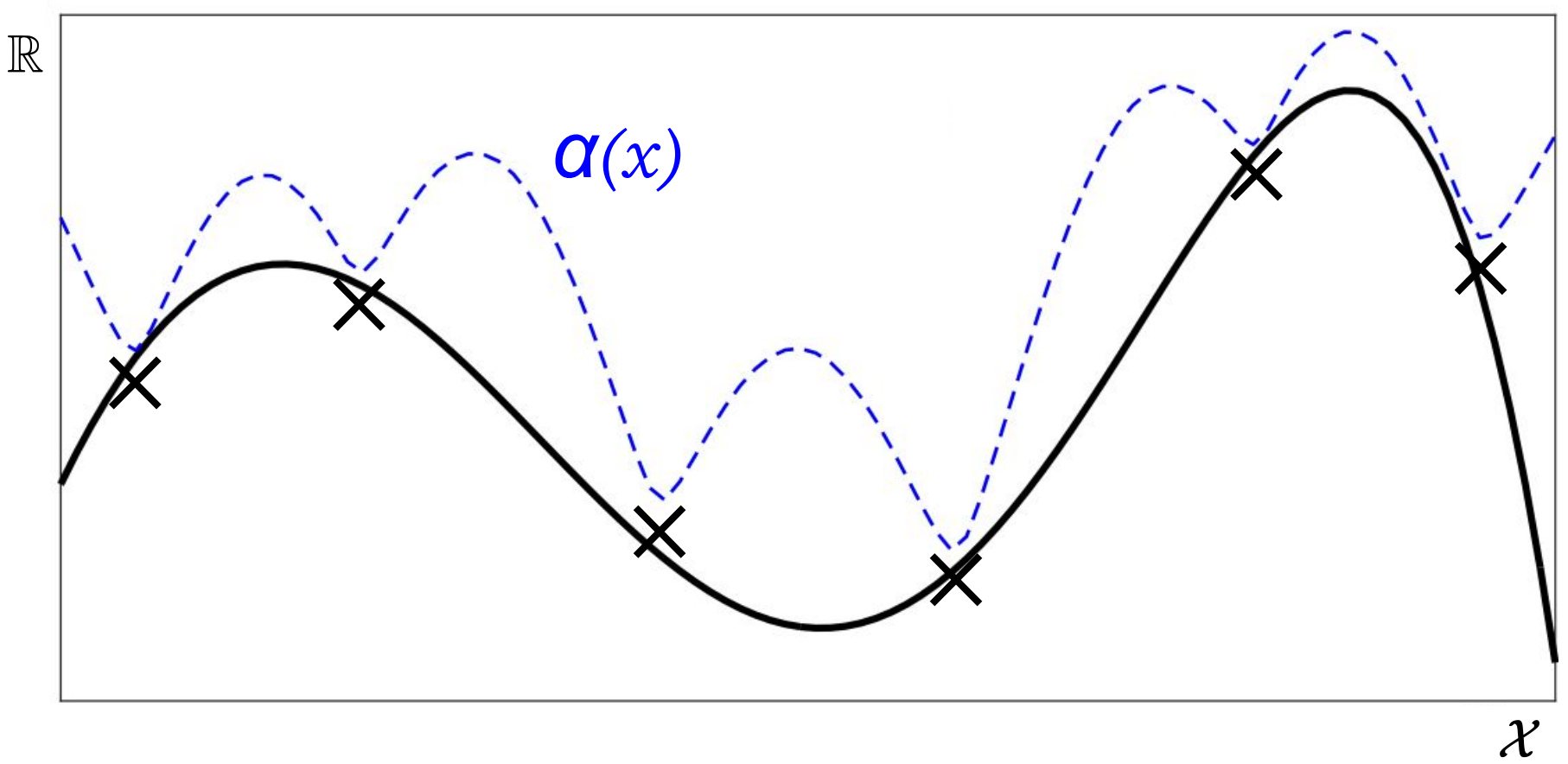
Define **acquisition function** using model.

Bayesian Optimization — Visualization



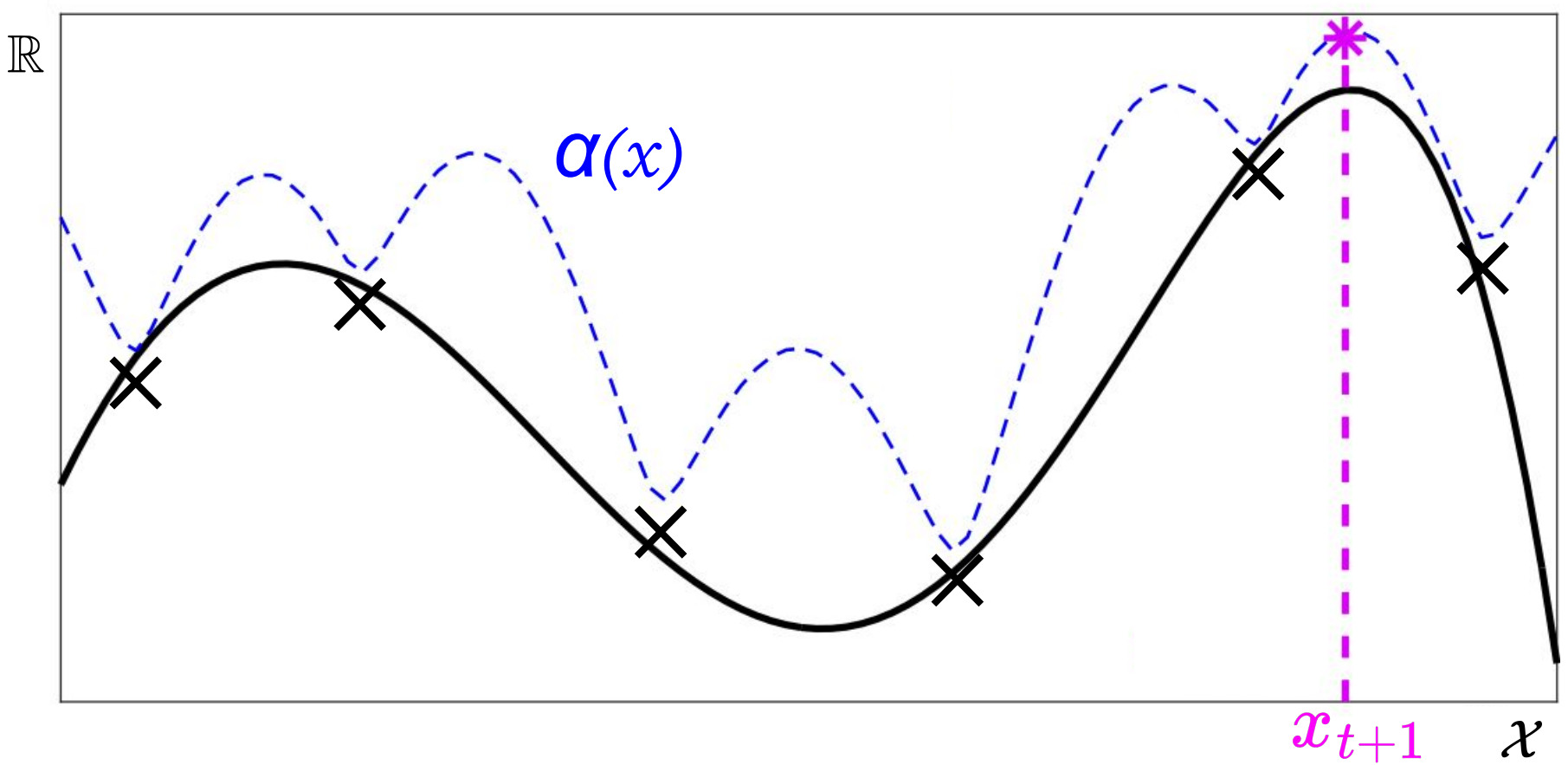
Define **acquisition function** using model.

Bayesian Optimization — Visualization



Define **acquisition function** using model.

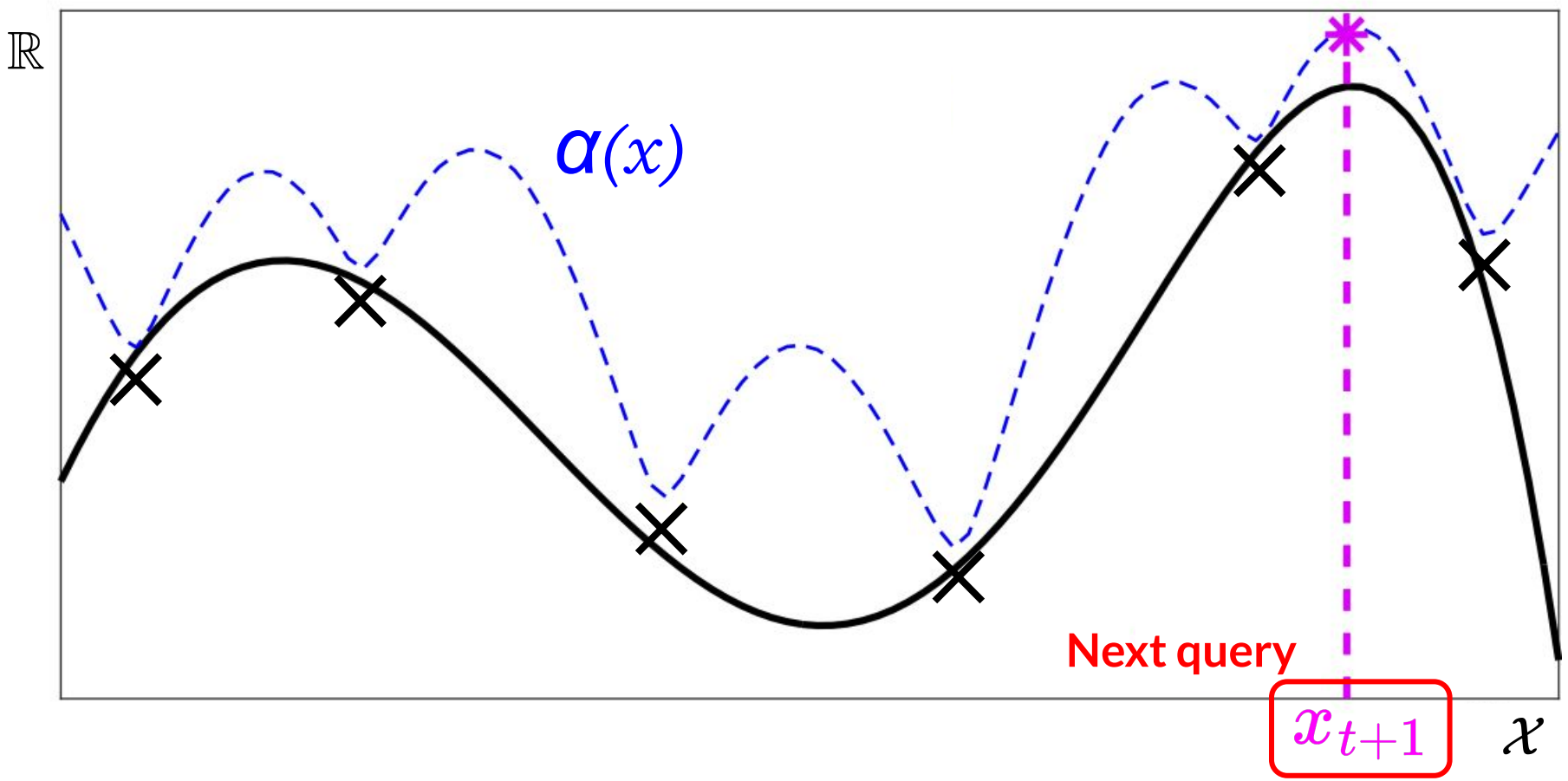
Bayesian Optimization — Visualization



Optimize acquisition function ⇒ yields next point to query.

$$x_{t+1} = \arg\max_{x \in \mathcal{X}} \alpha(x)$$

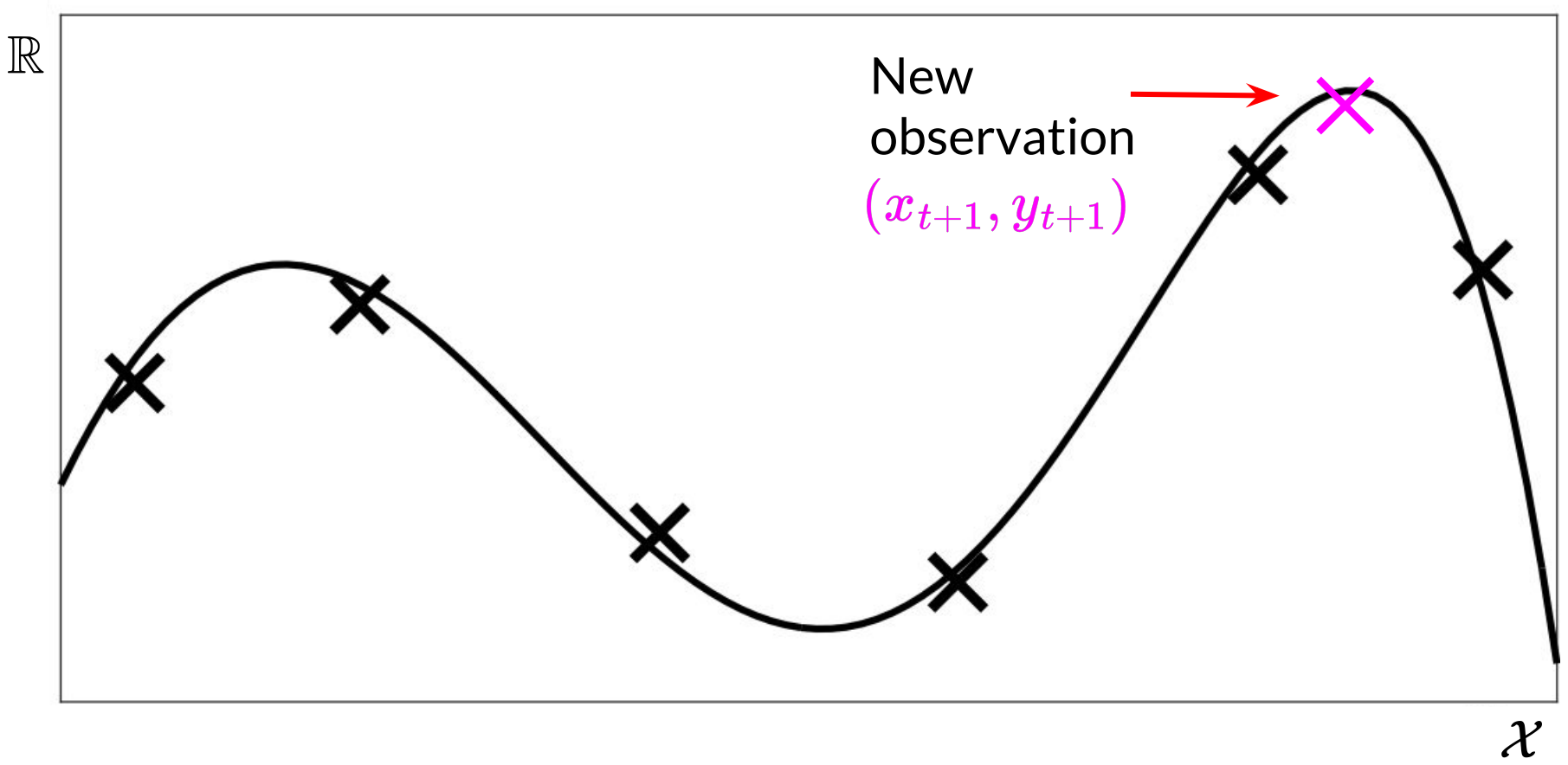
Bayesian Optimization — Visualization



Optimize acquisition function ⇒ yields next point to query.

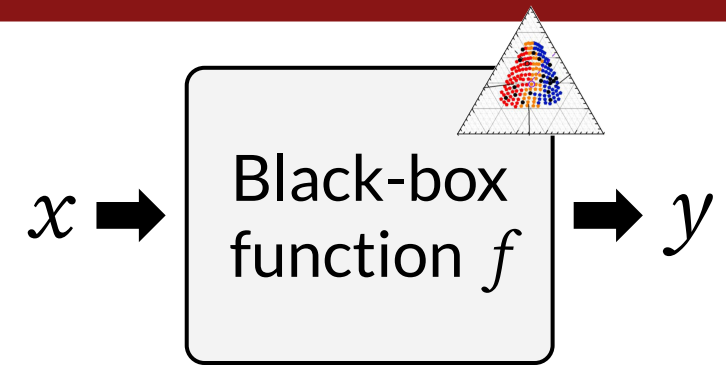
$$x_{t+1} = \arg\max_{x \in \mathcal{X}} \alpha(x)$$

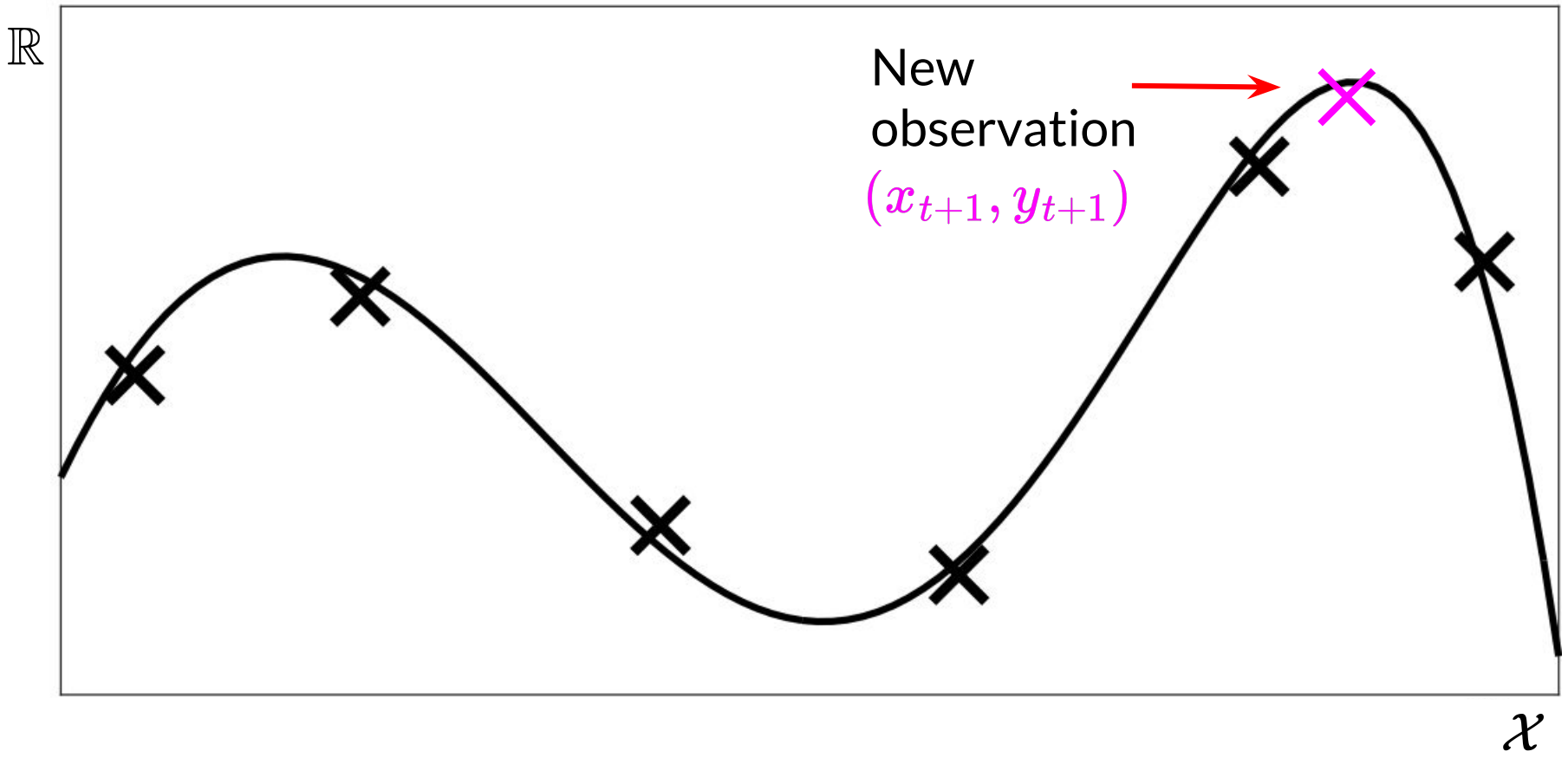
Bayesian Optimization — Visualization



Query black-box f at x, observe y, and update dataset.

Bayesian Optimization — Visualization





Query black-box f at x, observe y, and update dataset.

Acquisition Function — Key Component of BO

Acquisition Function — Key Component of BO

Acquisition functions are, effectively, (one of) the main things that change between different BO procedures.

⇒ they dictate the BO algorithm, and choose the point queried at each iteration.

Acquisition Function — Key Component of BO

Acquisition functions are, effectively, (one of) the main things that change between different BO procedures.

⇒ they dictate the BO algorithm, and choose the point queried at each iteration.

Goal when designing acquisition functions:

- Want to be “optimal” in some sense; to choose the point that is *most informative*.
- And/or: most-likely to accomplish goals of algorithm, *e.g.*, carry out optimization.

Acquisition Function — Key Component of BO

Next we will go through a few popular acquisition functions (for Bayes opt) in more detail:

Acquisition Function — Key Component of BO

Next we will go through a few popular acquisition functions (for Bayes opt) in more detail:

UCB (*upper confidence bound*) — which we saw visualized before.

Acquisition Function — Key Component of BO

Next we will go through a few popular acquisition functions (for Bayes opt) in more detail:

UCB (*upper confidence bound*) — which we saw visualized before.

PI (*probability of improvement*) — one of the simplest acquisition functions.

Acquisition Function — Key Component of BO

Next we will go through a few popular acquisition functions (for Bayes opt) in more detail:

UCB (*upper confidence bound*) — which we saw visualized before.

PI (*probability of improvement*) — one of the simplest acquisition functions.

EI (*expected improvement*) — perhaps the most popular of the acquisition functions.

Acquisition Function — Key Component of BO

Next we will go through a few popular acquisition functions (for Bayes opt) in more detail:

UCB (*upper confidence bound*) — which we saw visualized before.

PI (*probability of improvement*) — one of the simplest acquisition functions.

EI (*expected improvement*) — perhaps the most popular of the acquisition functions.

KG (*knowledge gradient*) — bit more complex, but performs well and is well-justified.

Acquisition Function — Key Component of BO

Next we will go through a few popular acquisition functions (for Bayes opt) in more detail:

UCB (*upper confidence bound*) — which we saw visualized before.

PI (*probability of improvement*) — one of the simplest acquisition functions.

EI (*expected improvement*) — perhaps the most popular of the acquisition functions.

KG (*knowledge gradient*) — bit more complex, but performs well and is well-justified.

ES (*entropy search*) — rooted in information theory and *Bayesian optimal experimental design (BOED)*.

Acquisition Function — Key Component of BO

Next we will go through a few popular acquisition functions (for Bayes opt) in more detail:

UCB (*upper confidence bound*) — which we saw visualized before.

PI (*probability of improvement*) — one of the simplest acquisition functions.

EI (*expected improvement*) — perhaps the most popular of the acquisition functions.

KG (*knowledge gradient*) — bit more complex, but performs well and is well-justified.

ES (*entropy search*) — rooted in information theory and *Bayesian optimal experimental design (BOED)*.

***But first*, a quick discussion on terminology!**

Terminology — Active Learning, Bayes Opt, Experimental Design, etc.

Terminology — Active Learning, Bayes Opt, Experimental Design, etc.

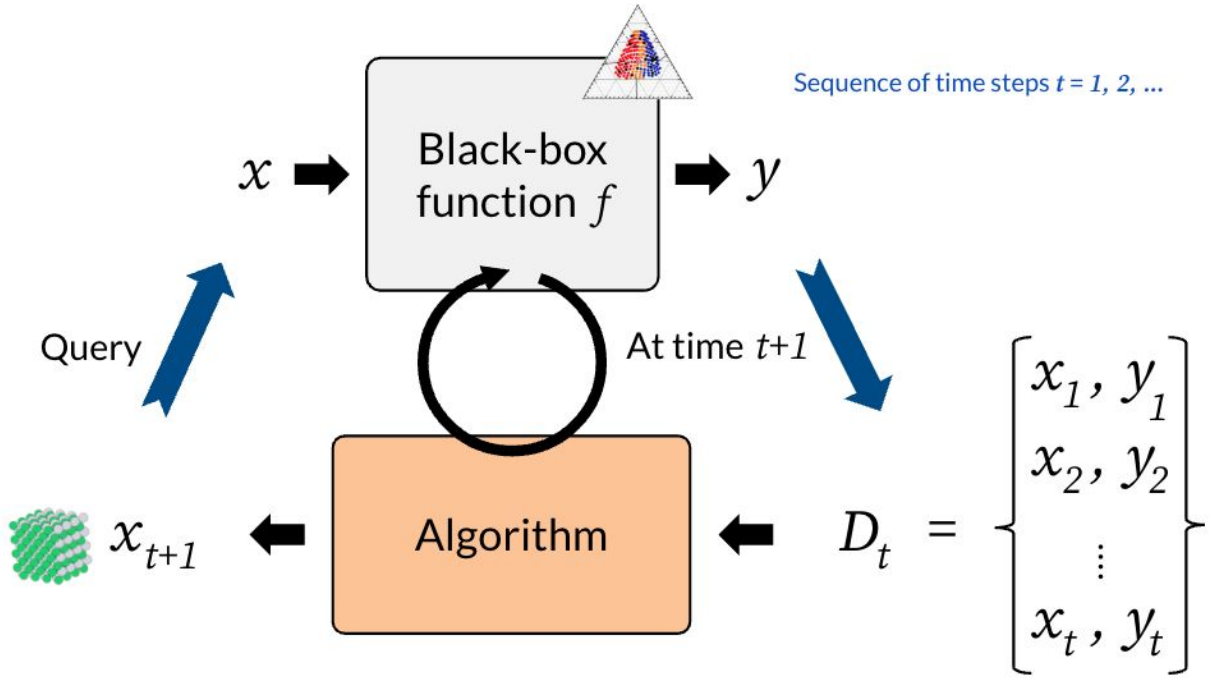
These terms all refer to related procedures, but there are slight differences between them…

Similarities: these all refer to sequential decision making (data acquisition) procedures.

Terminology — Active Learning, Bayes Opt, Experimental Design, etc.

These terms all refer to related procedures, but there are slight differences between them…

Similarities: these all refer to sequential decision making (data acquisition) procedures.



Terminology — Active Learning, Bayes Opt, Experimental Design, etc.

These terms all refer to related procedures, but there are slight differences between them…

Differences:

Terminology — Active Learning, Bayes Opt, Experimental Design, etc.

These terms all refer to related procedures, but there are slight differences between them…

Differences:

Active Learning

Originally from CS/ML.

Classically: iteratively choosing points to label from a set.

More recently: any learning method where you choose a data point to observe at each iteration .

Terminology — Active Learning, Bayes Opt, Experimental Design, etc.

These terms all refer to related procedures, but there are slight differences between them…

Differences:

Active Learning

Originally from CS/ML.

Classically: iteratively choosing points to label from a set.

More recently: any learning method where you choose a data point to observe at each iteration .

Experimental Design

Originally from statistics.

Classically: goal is to efficiently estimate a model parameter (often by reducing posterior entropy).

Can view parameter estimation as one strategy for active learning/optimization.

Terminology — Active Learning, Bayes Opt, Experimental Design, etc.

These terms all refer to related procedures, but there are slight differences between them…

Differences:

Active Learning

Originally from CS/ML.

Classically: iteratively choosing points to label from a set.

More recently: any learning method where you choose a data point to observe at each iteration .

Experimental Design

Originally from statistics.

Classically: goal is to efficiently estimate a model parameter (often by reducing posterior entropy).

Can view parameter estimation as one strategy for active learning/optimization.

Bayesian Optimization

Originally from operations research (among other fields).

Can view as: a special case of active learning, aiming to carry out the task of optimization.

(Which could potentially use an experimental design strategy, *e.g.*, in *entropy search*).

Terminology — Active Learning, Bayes Opt, Experimental Design, etc.

A bit more on experimental design:

Terminology — Active Learning, Bayes Opt, Experimental Design, etc.

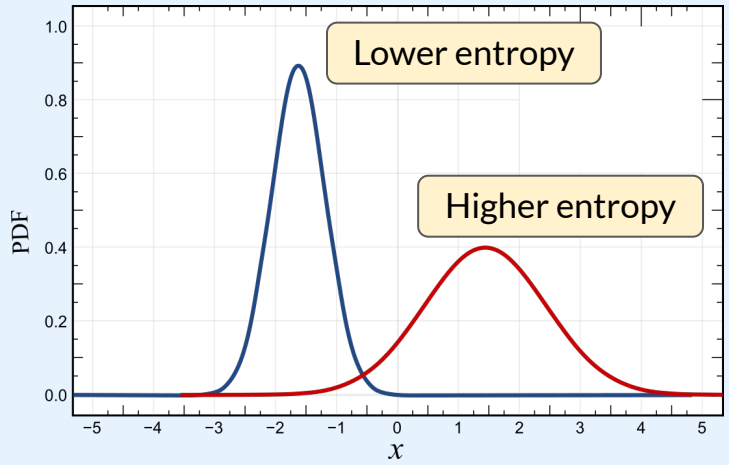
A bit more on experimental design: often referred to as *information-based experimental design* (if it involves reducing posterior entropy), or *Bayesian optimal experimental design (BOED)*.

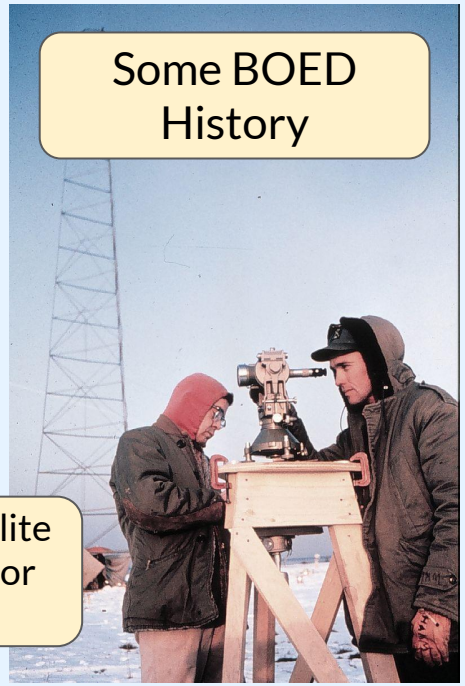
Terminology — Active Learning, Bayes Opt, Experimental Design, etc.

A bit more on experimental design: often referred to as *information-based experimental design* (if it involves reducing posterior entropy), or *Bayesian optimal experimental design (BOED)*.

BOED: have model with an (unknown) parameter of interest.

- Choose experiments that most reduce uncertainty about parameter.
- *Uncertainty*: **entropy** of posterior distribution over parameter.





Terminology — Active Learning, Bayes Opt, Experimental Design, etc.

A bit more on experimental design: often referred to as *information-based experimental design* (if it involves reducing posterior entropy), or *Bayesian optimal experimental design (BOED)*.

Relatedly — there is *information-based Bayesian optimization*:

Info-based BO: entropy search (ES)[1], predictive ES[2], max-value ES[3].

- Rooted in **Bayesian optimal experimental design (BOED)**.

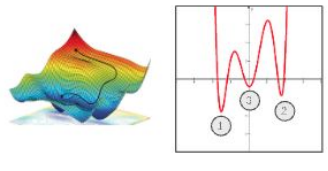
Which we will talk about a little later…

Terminology — Active Learning, Bayes Opt, Experimental Design, etc.

Side note: there are many other potential special cases of active learning beyond global optimization, as in BO…

Terminology — Active Learning, Bayes Opt, Experimental Design, etc.

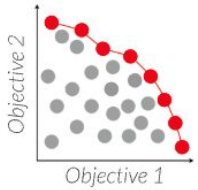
Optimization Variants
(global, local, top-k optima)



Applications

- *ML model training*
- *HPO / NAS*
- *Systems tuning*

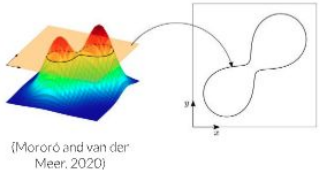
Multi-objective Opt.
(Pareto frontiers)



Applications

- *Process optimization*
- *Portfolio optimization*
- *Laboratory equipment / machines*

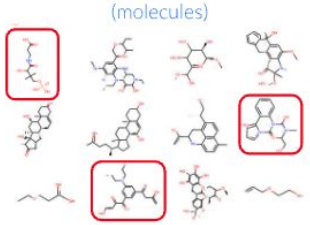
Level Set Estimation
(superlevel set, sublevel set)



Applications

- *Catalyst design*
- *Active learning / weak supervision*
- *Environmental monitoring*

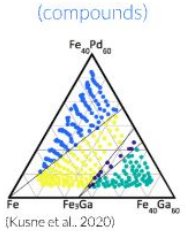
Search
(subset matching criteria)



Applications

- *Drug discovery*
- *Fraud detection*
- *Targeted opinion polling*

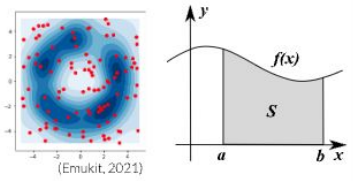
Phase Identification
(boundaries, partitions)



Applications

- *High throughput materials design/discovery*

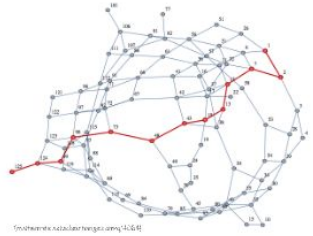
Quadrature, Integration
(integrals, expectations)



Applications

- *Probabilistic modeling (marginal distributions, normalization constants)*
- *Estimating center of mass (and centroids)*

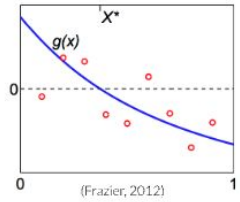
Graph Algorithms
(shortest paths)



Applications

- *Transportation networks*
- *Shipping networks*
- *Social networks*

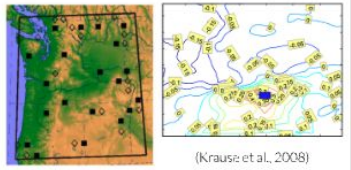
Root Finding, Bisection
(roots)



Applications

- *Target localization (airborne radar)*
- *Edge detection (computer vision)*
- *Biology / microbiology (phase shifts)*

Sensor Placement
(function value at locations)



Applications

- *Water distribution systems*
- *Outbreak detection in networks*
- *Weather monitoring*

Acquisition Functions — Back to Acquisition Functions

Acquisition Functions — Back to Acquisition Functions

Acquisition Function

Component of Bayesian optimization algorithm.

$$\alpha : \mathcal{X} \to \mathbb{R}$$

Assigns a number describing the value/usefulness of querying a given point.

Acquisition Functions — Upper Confidence Bound (UCB)

Acquisition Functions — Upper Confidence Bound (UCB)

The UCB acquisition function is often written:

$$\alpha_t(x) = \mu(x) + \beta_t \sigma(x)$$

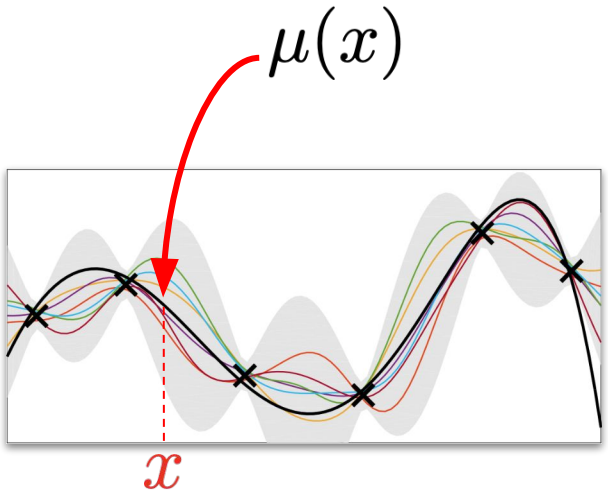
Acquisition Functions — Upper Confidence Bound (UCB)

The UCB acquisition function is often written:

$$\alpha_t(x) = \mu(x) + \beta_t \sigma(x)$$

where:

$\mu(x)$ - Mean of *marginal posterior* at input x.



Acquisition Functions — Upper Confidence Bound (UCB)

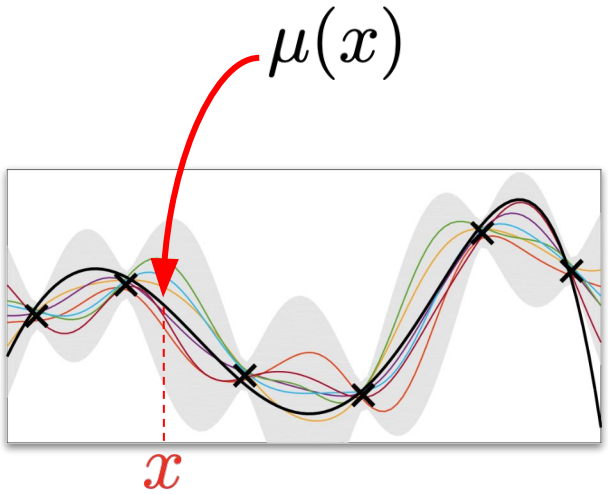
The UCB acquisition function is often written:

$$\alpha_t(x) = \mu(x) + \beta_t \sigma(x)$$

where:

$p(f(x) \mid \mathcal{D}_t)$

$\mu(x)$ - Mean of *marginal posterior* at input x.



Acquisition Functions — Upper Confidence Bound (UCB)

The UCB acquisition function is often written:

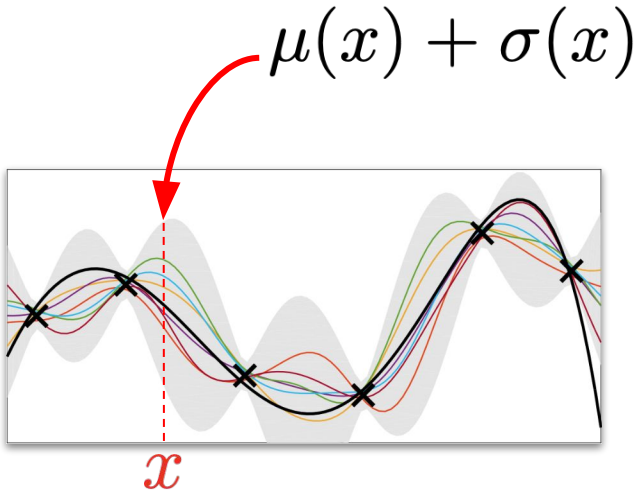
$$\alpha_t(x) = \mu(x) + \beta_t \sigma(x)$$

where:

$p(f(x) \mid \mathcal{D}_t)$

$\mu(x)$ - Mean of *marginal posterior* at input x.

$\sigma(x)$ - Standard deviation of *marginal posterior* at input x.



Acquisition Functions — Upper Confidence Bound (UCB)

The UCB acquisition function is often written:

$$\alpha_t(x) = \mu(x) + \beta_t \sigma(x)$$

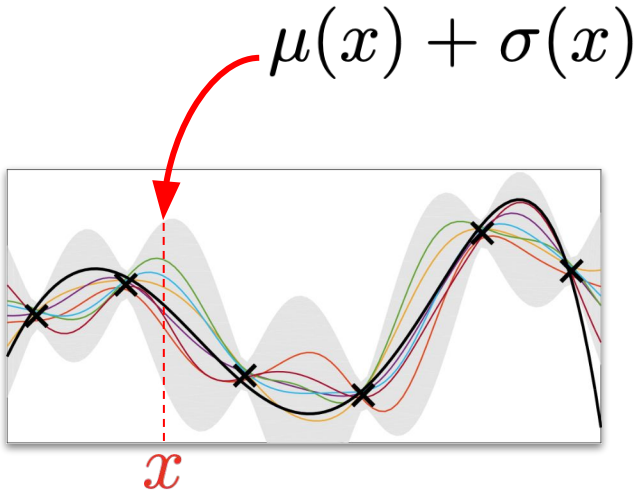
where:

$p(f(x) \mid \mathcal{D}_t)$

$\mu(x)$ - Mean of *marginal posterior* at input x.

$\sigma(x)$ - Standard deviation of *marginal posterior* at input x.

β_t - Hyperparameter controlling tradeoff between exploration and exploitation.

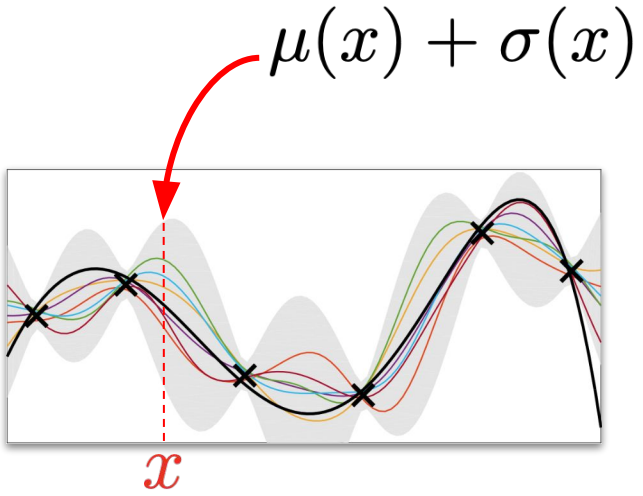


Acquisition Functions — Upper Confidence Bound (UCB)

The UCB acquisition function is often written:

$$\alpha_t(x) = \mu(x) + \beta_t \sigma(x)$$

Intuition:



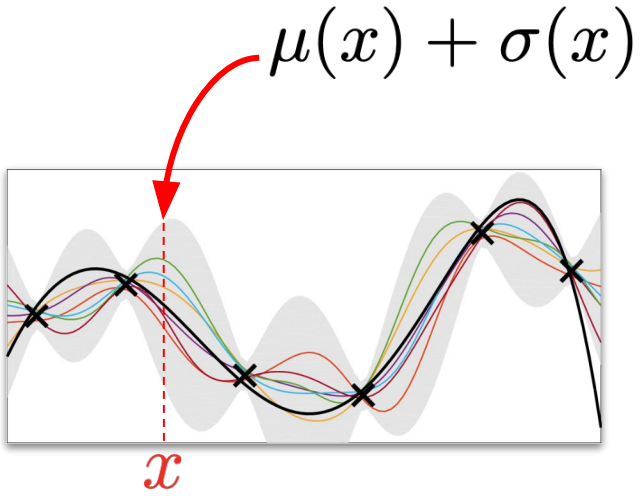
Acquisition Functions — Upper Confidence Bound (UCB)

The UCB acquisition function is often written:

$$\alpha_t(x) = \mu(x) + \beta_t \sigma(x)$$

Intuition:

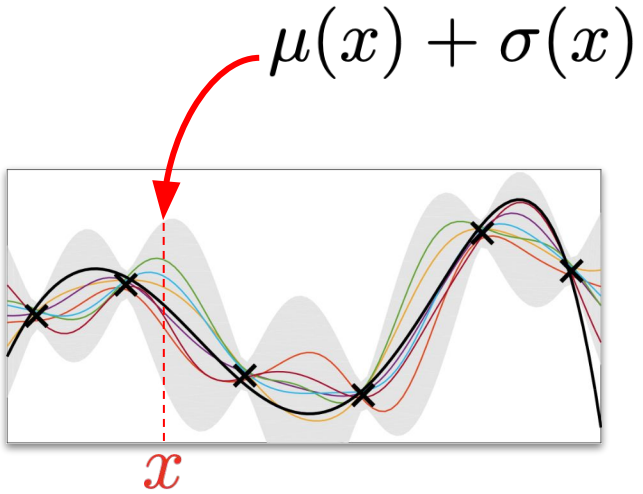
Maximizing variance → queries *most unknown* part of function
⇒ most "informative".



Acquisition Functions — Upper Confidence Bound (UCB)

The UCB acquisition function is often written:

$$\alpha_t(x) = \mu(x) + \beta_t \sigma(x)$$



Intuition:

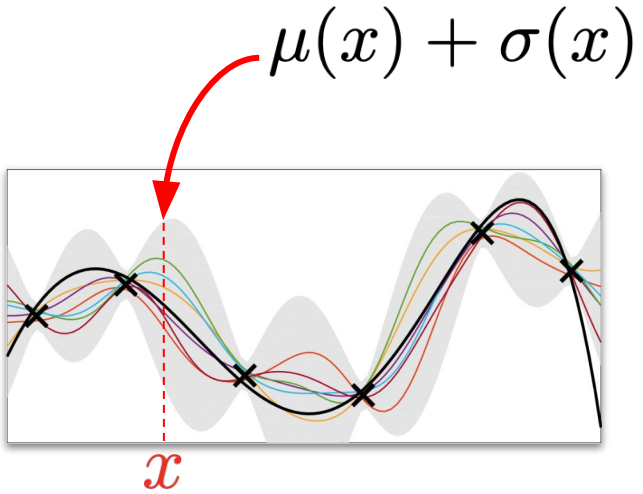
Maximizing variance → queries *most unknown* part of function ⇒ most "informative".

Maximizing mean → queries *best guess of highest value*, based on current knowledge of function.

Acquisition Functions — Upper Confidence Bound (UCB)

The UCB acquisition function is often written:

$$\alpha_t(x) = \mu(x) + \beta_t \sigma(x)$$



Intuition:

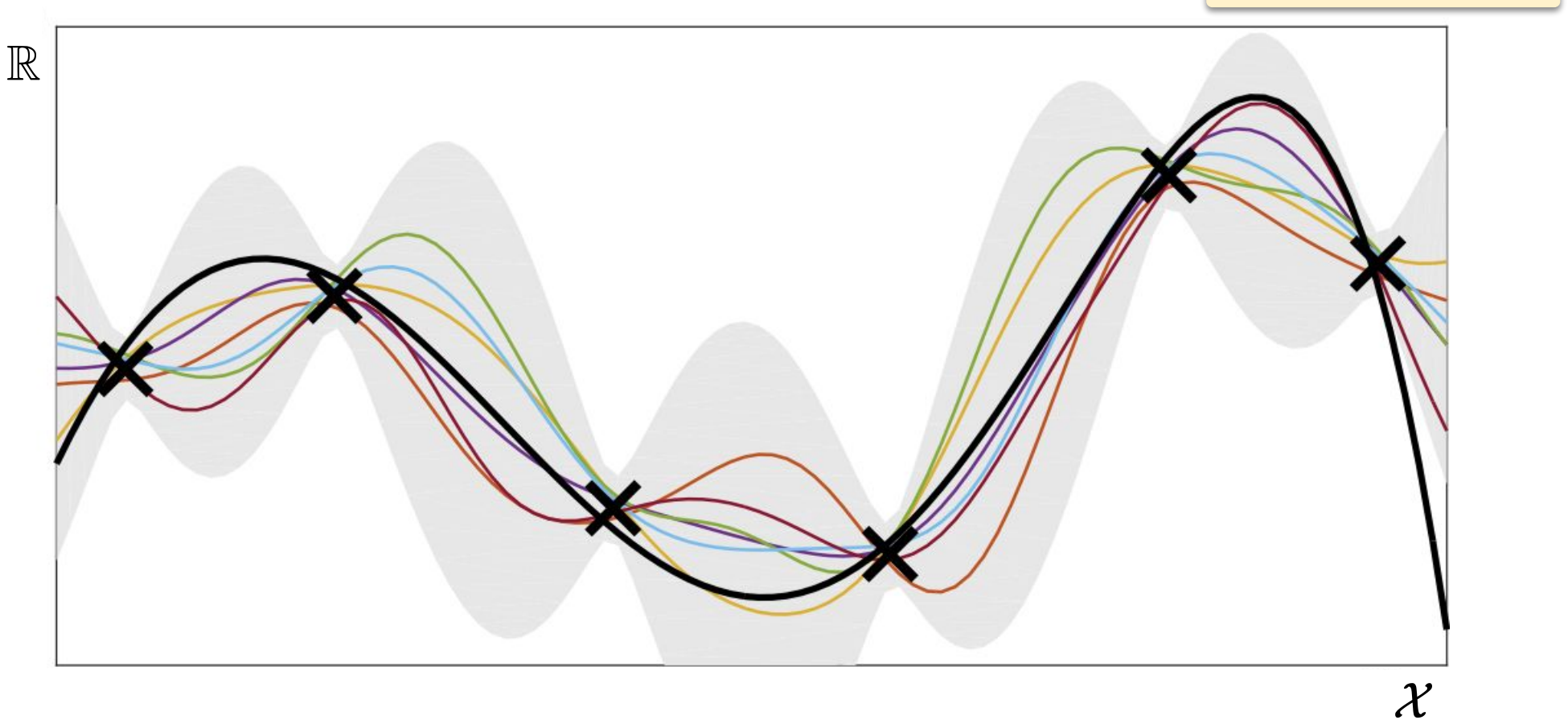
Maximizing variance → queries *most unknown* part of function ⇒ most "informative".

Maximizing mean → queries *best guess of highest value*, based on current knowledge of function.

We want to trade off between these two quantities.

Acquisition Functions — Upper Confidence Bound (UCB)

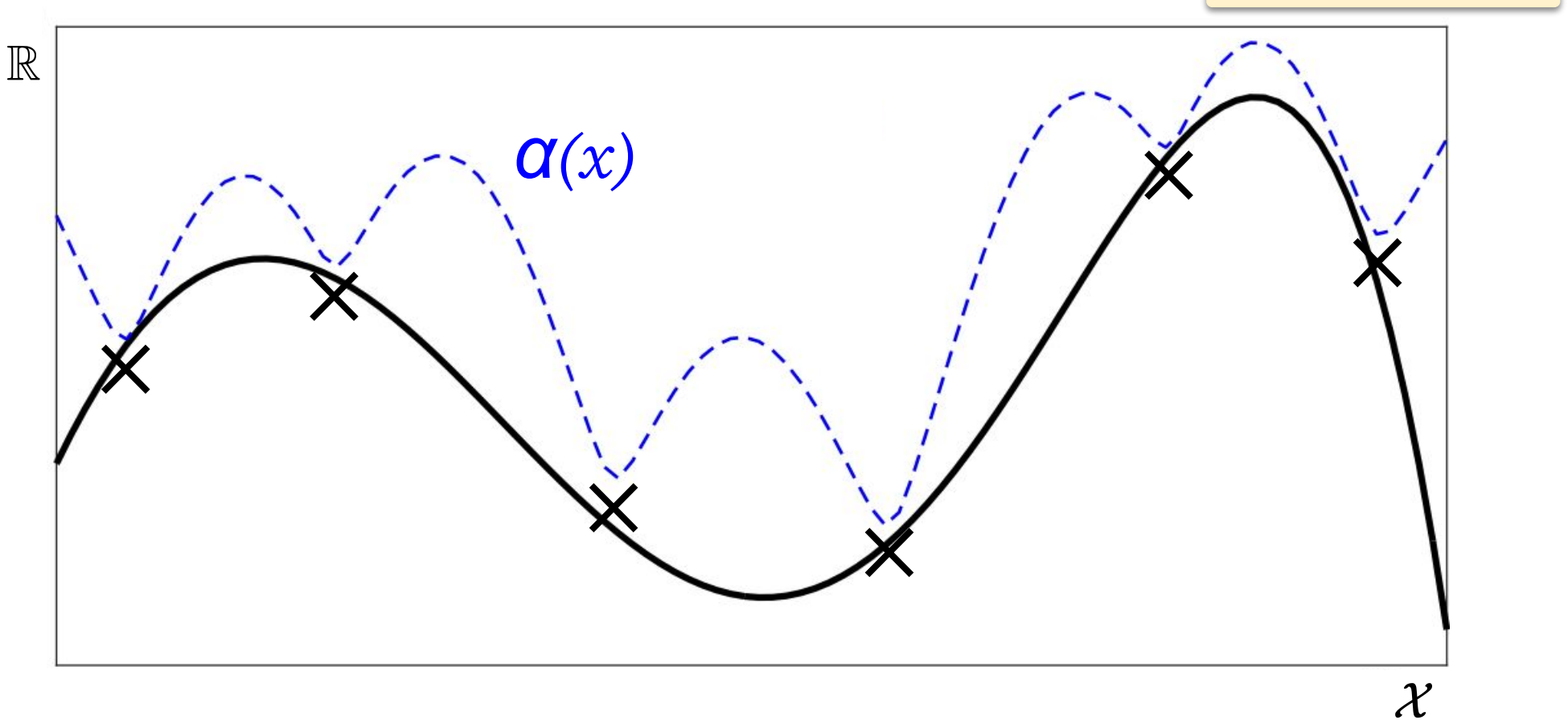
Visualizing UCB



Given our probabilistic model…

Acquisition Functions — Upper Confidence Bound (UCB)

Visualizing UCB



… define the **UCB acquisition function**.

Acquisition Functions — Experimental Design (EIG about f)

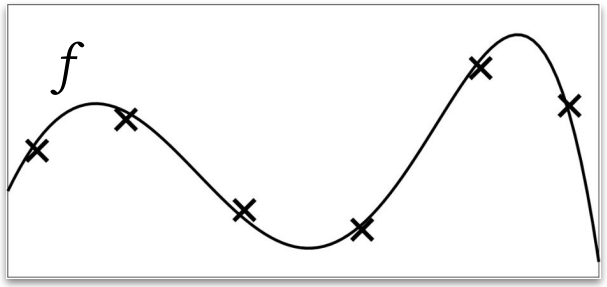
Quick aside on experimental design, to give some perspective on UCB:

Acquisition Functions — Experimental Design (EIG about f)

Quick aside on experimental design, to give some perspective on UCB:

Suppose you don't want to do optimization, but instead want to learn the full function landscape (*i.e.*, learn "all of f").

⇒ technically speaking, this is not *optimization* anymore.

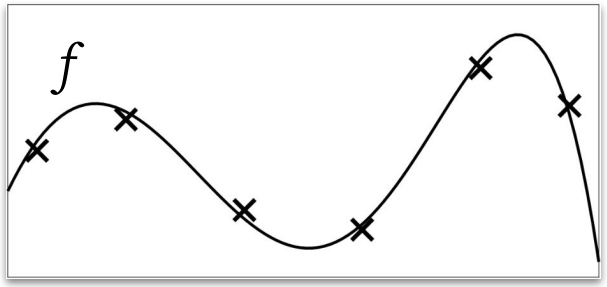


Acquisition Functions — Experimental Design (EIG about f)

Quick aside on experimental design, to give some perspective on UCB:

Suppose you don't want to do optimization, but instead want to learn the full function landscape (*i.e.*, learn "all of f").

⇒ technically speaking, this is not *optimization* anymore.



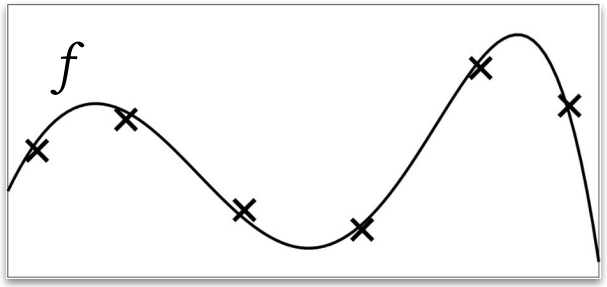
An *experimental design approach* would be to query the point that **maximally reduces the posterior entropy about** f**.**

Acquisition Functions — Experimental Design (EIG about f)

Quick aside on experimental design, to give some perspective on UCB:

Suppose you don't want to do optimization, but instead want to learn the full function landscape (*i.e.*, learn "all of f").

⇒ technically speaking, this is not *optimization* anymore.



An *experimental design approach* would be to query the point that **maximally reduces the posterior entropy about** f**.**

(This is equivalently also commonly called: *expected information gain*, or *EIG*, *about* f.)

Acquisition Functions — Experimental Design (EIG about f)

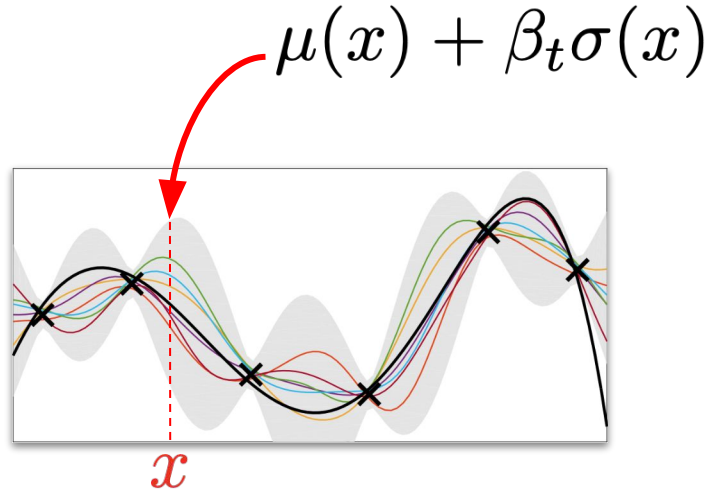
You can prove (at least for GP models) that maximally reducing posterior entropy is equivalent to querying point with maximum standard deviation (of posterior marginal).

Acquisition Functions — Experimental Design (EIG about f)

You can prove (at least for GP models) that maximally reducing posterior entropy is equivalent to querying point with maximum standard deviation (of posterior marginal).

So in **UCB** ⇒

The second term (standard deviation term) is in fact balancing exploration (EIG about f) with exploitation (best expected input x).



Acquisition Functions — Probability of Improvement (PI)

Acquisition Functions — Probability of Improvement (PI)

Next is one of the simplest acquisition functions (though not used a lot in practice).

Acquisition Functions — Probability of Improvement (PI)

Next is one of the simplest acquisition functions (though not used a lot in practice).

The probability of improvement (PI) acquisition function is written:

Acquisition Functions — Probability of Improvement (PI)

Next is one of the simplest acquisition functions (though not used a lot in practice).

The probability of improvement (PI) acquisition function is written:

$$\alpha_t(x) = p(f(x) > f(x^*) \mid \mathcal{D}_t)$$

Acquisition Functions — Probability of Improvement (PI)

Next is one of the simplest acquisition functions (though not used a lot in practice).

The probability of improvement (PI) acquisition function is written:

$$\alpha_t(x) = p(f(x) > f(x^*) \mid \mathcal{D}_t)$$

where $f(x^*)$ is the function value of the best point observed so far in $\mathcal{D}_t$.

I.e., the posterior probability that the f(x) is greater than the best point observed so far.

Acquisition Functions — Probability of Improvement (PI)

Next is one of the simplest acquisition functions (though not used a lot in practice).

The probability of improvement (PI) acquisition function is written:

$$\alpha_t(x) = p(f(x) > f(x^*) \mid \mathcal{D}_t)$$

where $f(x^*)$ is the function value of the best point observed so far in $\mathcal{D}_t$.

I.e., the posterior probability that the f(x) is greater than the best point observed so far.

You can prove this is equal to …

Acquisition Functions — Probability of Improvement (PI)

Next is one of the simplest acquisition functions (though not used a lot in practice).

The probability of improvement (PI) acquisition function is written:

$$\begin{aligned}\alpha_t(x) &= p(f(x) > f(x^*) \mid \mathcal{D}_t) \\ &= \Phi\left(\frac{\mu(x) - f(x^*)}{\sigma(x)}\right)\end{aligned}$$

Acquisition Functions — Probability of Improvement (PI)

Next is one of the simplest acquisition functions (though not used a lot in practice).

The probability of improvement (PI) acquisition function is written:

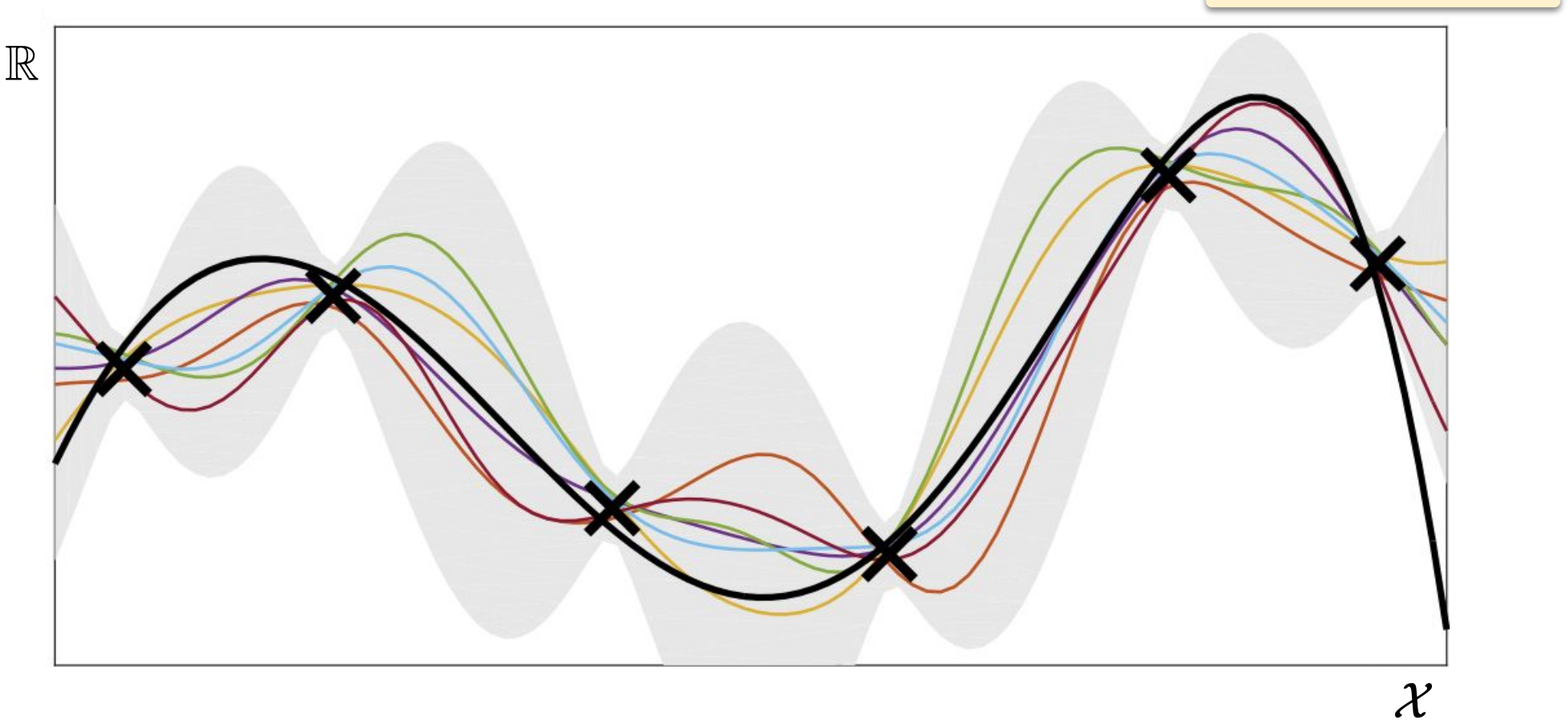
$$\begin{aligned}\alpha_t(x) &= p(f(x) > f(x^*) \mid \mathcal{D}_t) \\ &= \Phi\left(\frac{\mu(x) - f(x^*)}{\sigma(x)}\right)\end{aligned}$$

where $\Phi(\cdot)$ denotes the CDF of the standard normal distribution.

(And, as before, $\mu(x)$ and $\sigma(x)$ are the mean and SD of marginal posterior at x.)

Acquisition Functions — Probability of Improvement (PI)

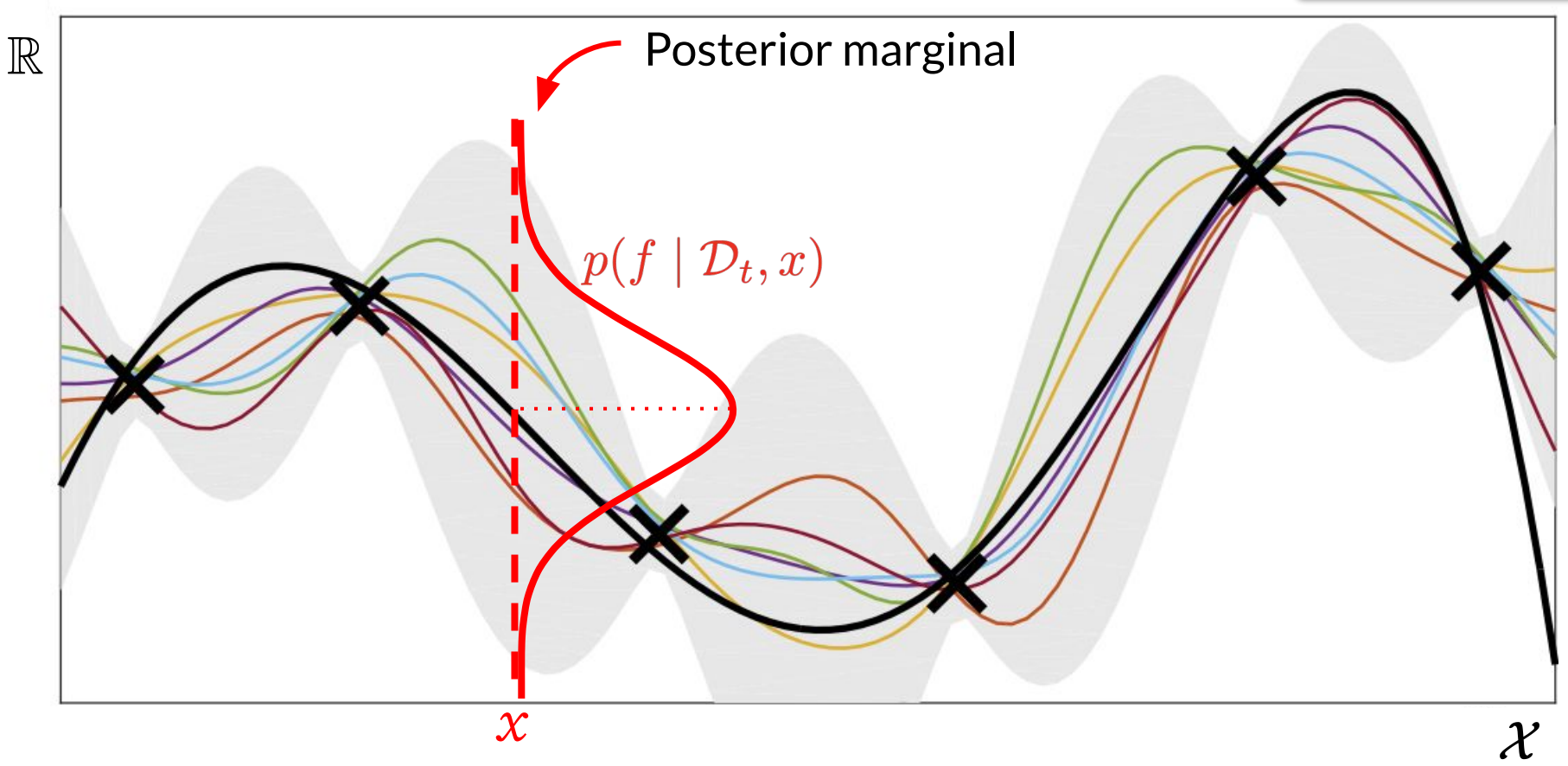
Visualizing PI



Given our probabilistic model…

Acquisition Functions — Probability of Improvement (PI)

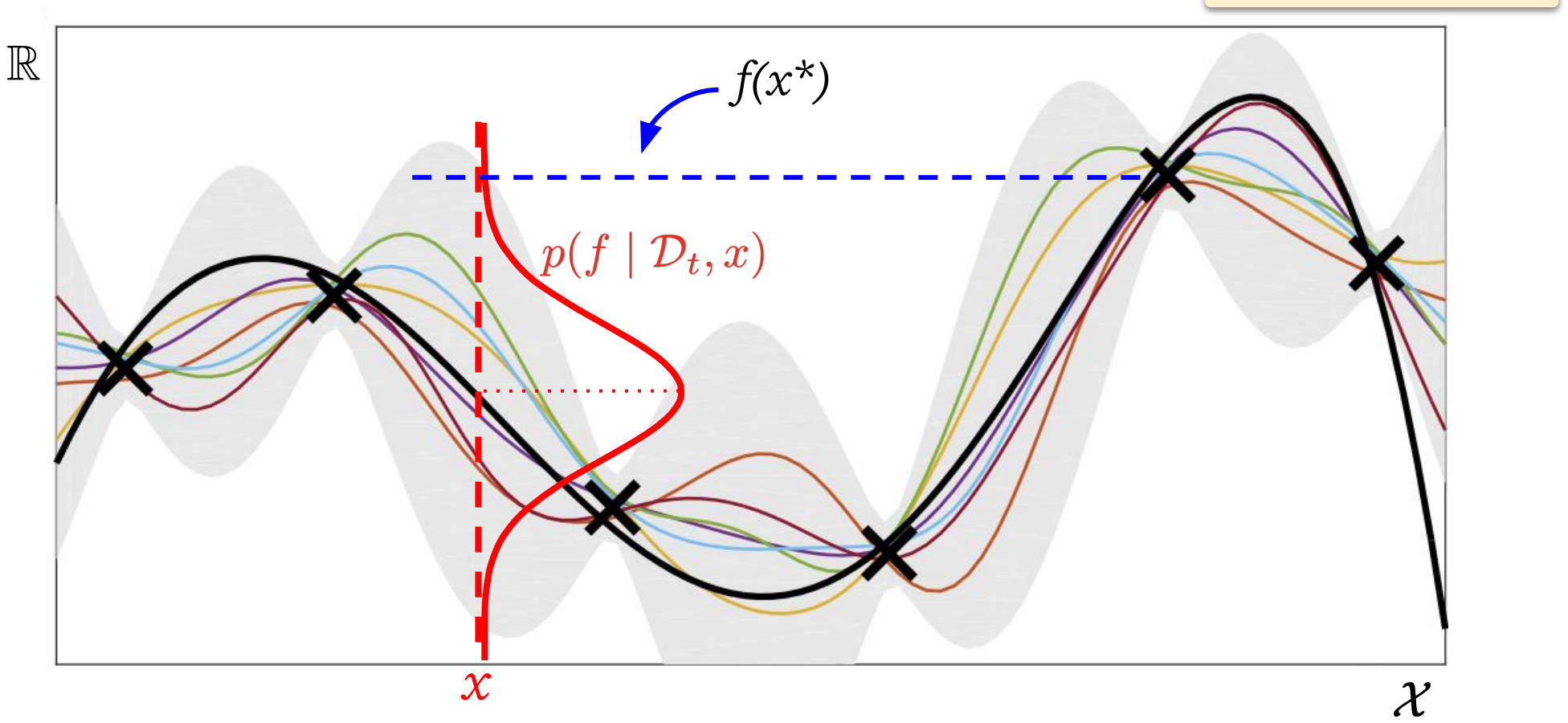
Visualizing PI



Given our probabilistic model…

Acquisition Functions — Probability of Improvement (PI)

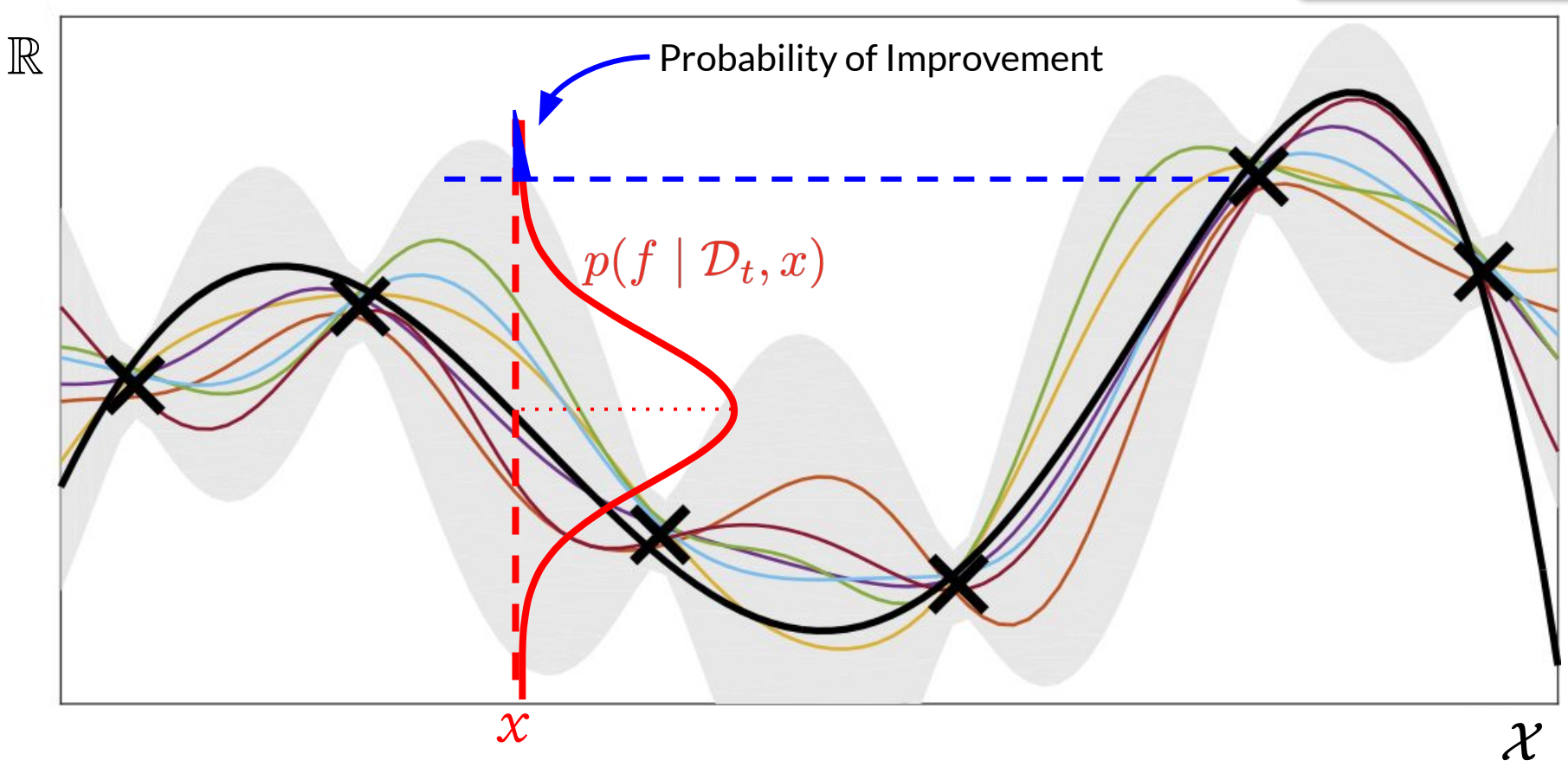
Visualizing PI



Given our probabilistic model…

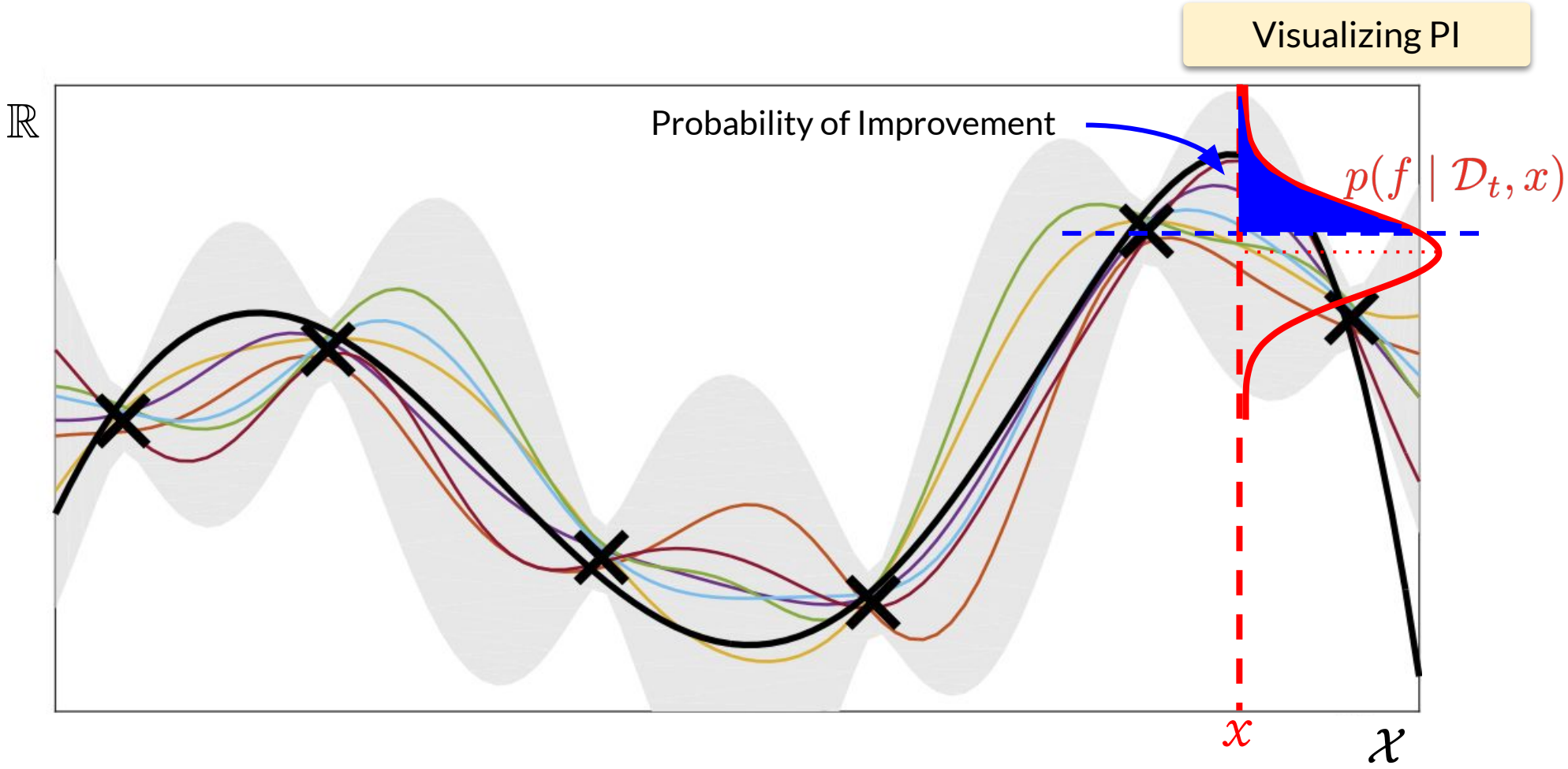
Acquisition Functions — Probability of Improvement (PI)

Visualizing PI



… define the **PI acquisition function**.

Acquisition Functions — Probability of Improvement (PI)



… define the **PI acquisition function**.

Acquisition Functions — Probability of Improvement (PI)

However, note that:

Acquisition Functions — Probability of Improvement (PI)

However, note that:

PI maximizes the **probability** that $f(x)$ is greater than then best point so far.

Does *not* try to maximize the **amount** that $f(x)$ is greater than the best point so far.

Acquisition Functions — Probability of Improvement (PI)

However, note that:

PI maximizes the **probability** that $f(x)$ is greater than then best point so far.

Does *not* try to maximize the **amount** that $f(x)$ is greater than the best point so far.

(*i.e.*, it might choose an x that is only *slightly better* than best point so far, if it is more confident about it — rather than a point that is *significantly greater in expectation*, but it is less confident about it).

Acquisition Functions — Probability of Improvement (PI)

However, note that:

PI maximizes the **probability** that $f(x)$ is greater than then best point so far.

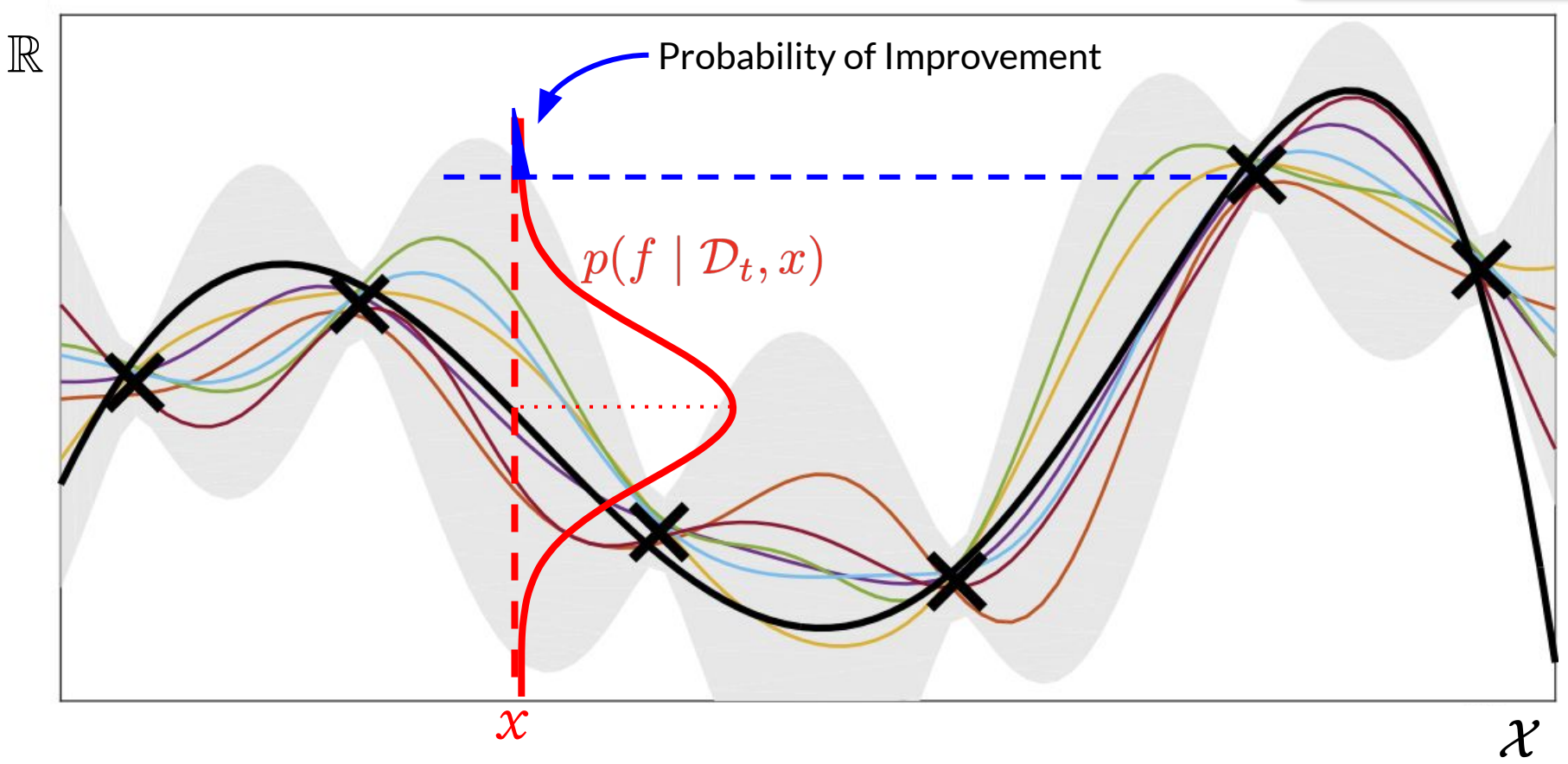
Does *not* try to maximize the **amount** that $f(x)$ is greater than the best point so far.

(*i.e.*, it might choose an x that is only *slightly better* than best point so far, if it is more confident about it — rather than a point that is *significantly greater in expectation*, but it is less confident about it).

Visualizing this…

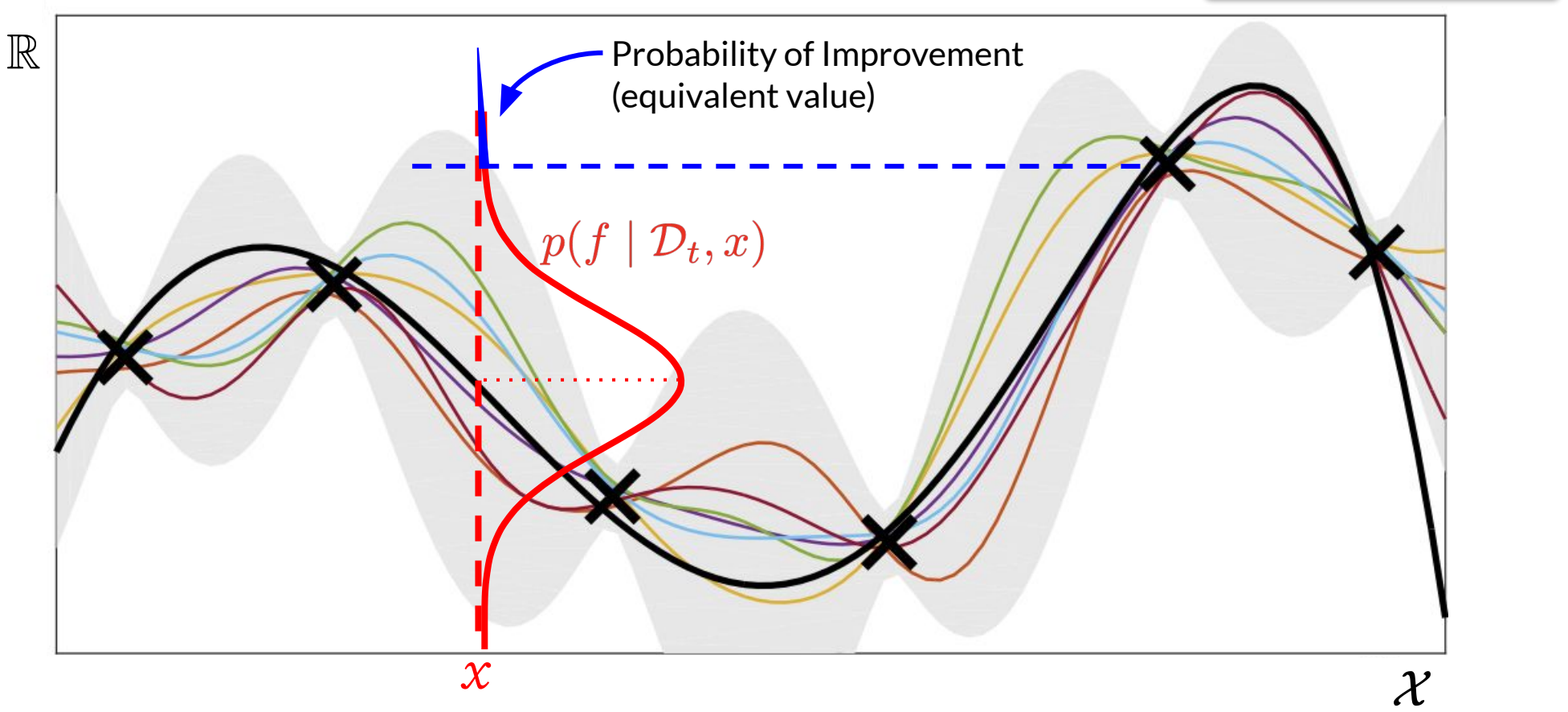
Acquisition Functions — Probability of Improvement (PI)

Visualizing PI



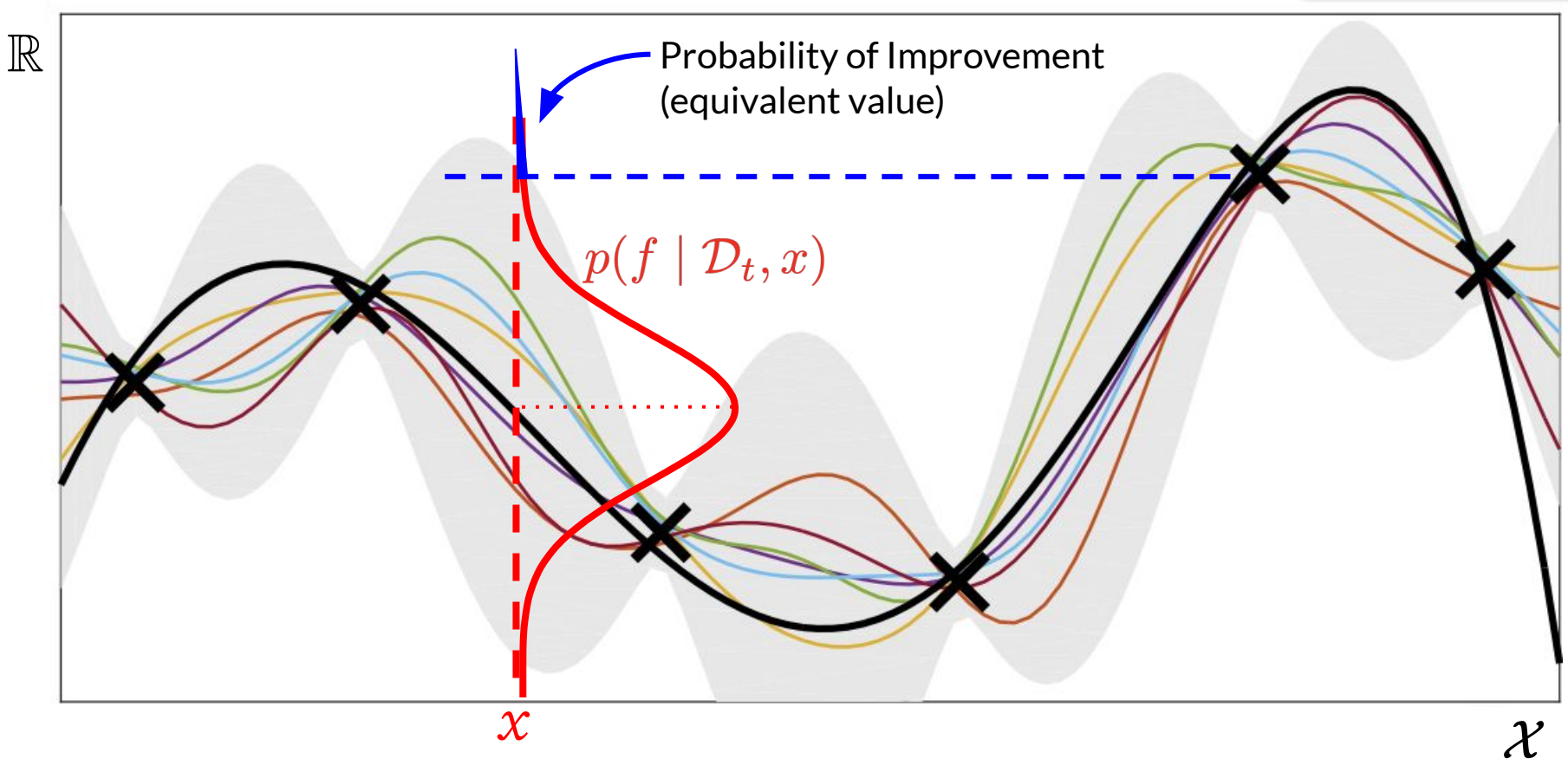
Acquisition Functions — Probability of Improvement (PI)

Visualizing PI



Acquisition Functions — Probability of Improvement (PI)

Visualizing PI



In contrast, we could try to maximize the *expected **amount** of improvement* over the best point so far…

Acquisition Functions — Expected Improvement (EI)

Acquisition Functions — Expected Improvement (EI)

This leads us to one of the most famous and widely used BO acquisition functions (except possibly for UCB).

Acquisition Functions — Expected Improvement (EI)

This leads us to one of the most famous and widely used BO acquisition functions (except possibly for UCB).

The **expected improvement (EI)** acquisition function is written:

Acquisition Functions — Expected Improvement (EI)

This leads us to one of the most famous and widely used BO acquisition functions (except possibly for UCB).

The **expected improvement (EI)** acquisition function is written:

$$\alpha_t(x) = \mathbb{E}_{p(f|\mathcal{D}_t)}\left[\max\{0, f(x) - f(x^*)\}\right]$$

Acquisition Functions — Expected Improvement (EI)

This leads us to one of the most famous and widely used BO acquisition functions (except possibly for UCB).

The **expected improvement (EI)** acquisition function is written:

$$\alpha_t(x) = \mathbb{E}_{p(f|\mathcal{D}_t)}\left[\max\{0, f(x) - f(x^*)\}\right]$$

where $f(x^*)$ is the function value of the best point observed so far in $\mathcal{D}_t$.

Acquisition Functions — Expected Improvement (EI)

This leads us to one of the most famous and widely used BO acquisition functions (except possibly for UCB).

The **expected improvement (EI)** acquisition function is written:

$$\alpha_t(x) = \mathbb{E}_{p(f|\mathcal{D}_t)}\left[\max\{0, f(x) - f(x^*)\}\right]$$

where $f(x^*)$ is the function value of the best point observed so far in $\mathcal{D}_t$.

I.e., the expected amount that $f(x)$ is greater than the best point observed so far (while ignoring the magnitude if $f(x)$ is less than the best point so far).

Acquisition Functions — Expected Improvement (EI)

This leads us to one of the most famous and widely used BO acquisition functions (except possibly for UCB).

The **expected improvement (EI)** acquisition function is written:

$$\alpha_t(x) = \mathbb{E}_{p(f|\mathcal{D}_t)}\left[\max\{0, f(x) - f(x^*)\}\right]$$

where $f(x^*)$ is the function value of the best point observed so far in $\mathcal{D}_t$.

I.e., the expected amount that f(x) is greater than the best point observed so far (while ignoring the magnitude if f(x) is less than the best point so far).

You can prove this is equal to …

Acquisition Functions — Expected Improvement (EI)

This leads us to one of the most famous and widely used BO acquisition functions (except possibly for UCB).

The **expected improvement (EI)** acquisition function is written:

$$\begin{aligned}\alpha_t(x) &= \mathbb{E}_{p(f|\mathcal{D}_t)}\left[\max\{0, f(x) - f(x^*)\}\right] \\ &= (\mu(x) - f(x^*))\Phi\left(\frac{\mu(x) - f(x^*)}{\sigma(x)}\right) + \sigma(x)\phi\left(\frac{\mu(x) - f(x^*)}{\sigma(x)}\right)\end{aligned}$$

Acquisition Functions — Expected Improvement (EI)

This leads us to one of the most famous and widely used BO acquisition functions (except possibly for UCB).

The **expected improvement (EI)** acquisition function is written:

$$\begin{aligned}\alpha_t(x) &= \mathbb{E}_{p(f|\mathcal{D}_t)}\left[\max\{0, f(x) - f(x^*)\}\right] \\ &= (\mu(x) - f(x^*))\Phi\left(\frac{\mu(x) - f(x^*)}{\sigma(x)}\right) + \sigma(x)\phi\left(\frac{\mu(x) - f(x^*)}{\sigma(x)}\end{aligned}$$

where $\Phi(\cdot)$ denotes the CDF of the standard normal distribution and $\phi(\cdot)$ denotes the PDF of the standard normal distribution.

(And, as before, $\mu(x)$ and $\sigma(x)$ are the mean and SD of marginal posterior at x.)

Acquisition Functions — Expected Improvement (EI)

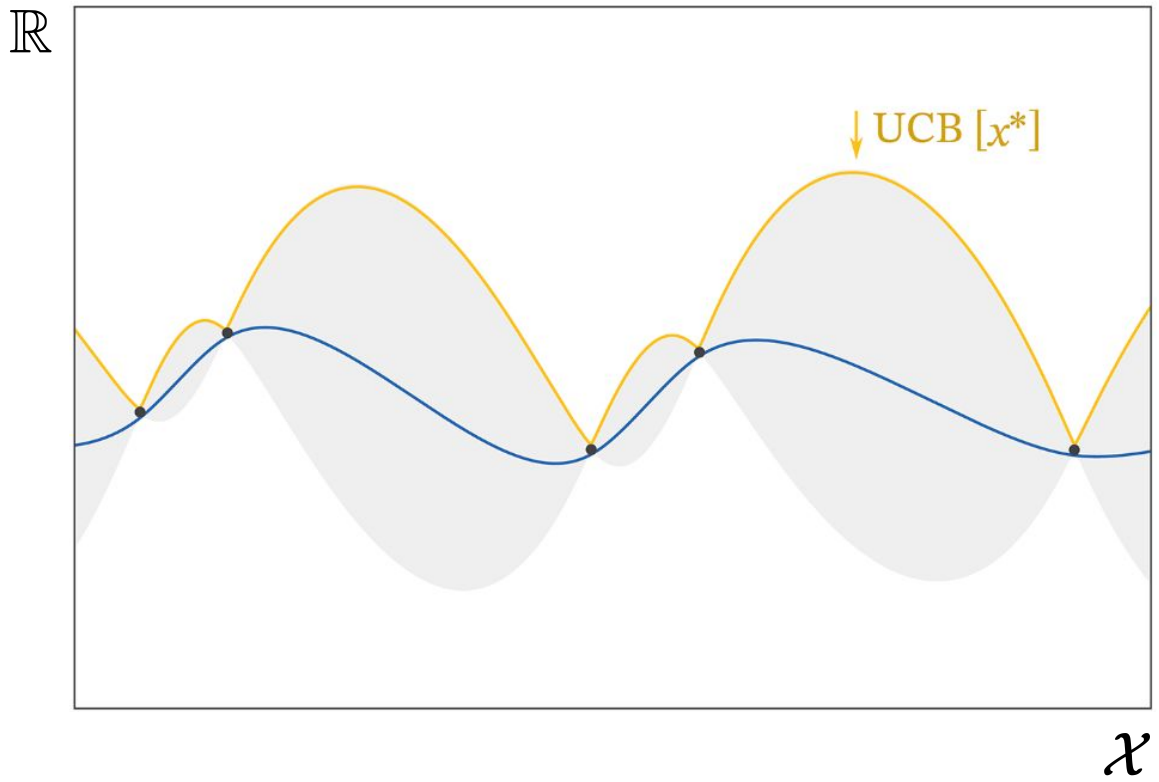
Nice visual comparing UCB *vs.* PI *vs.* EI:

Source: M. O. Ahmed, S. Prince, “Bayesian optimization”

Acquisition Functions — Expected Improvement (EI)

Nice visual comparing UCB *vs.* PI *vs.* EI:

Source: M. O. Ahmed, S. Prince, "Bayesian optimization"

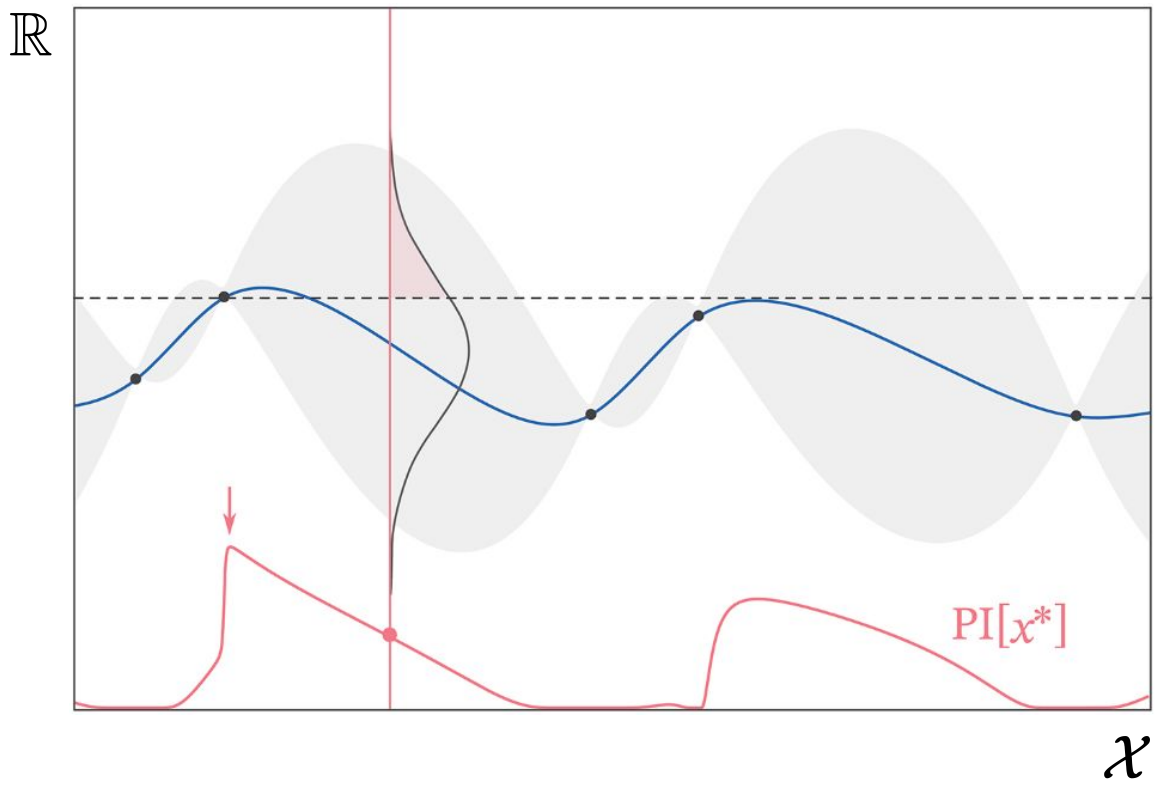


UCB: upper confidence bound of posterior

Acquisition Functions — Expected Improvement (EI)

Nice visual comparing UCB *vs.* PI *vs.* EI:

Source: M. O. Ahmed, S. Prince, "Bayesian optimization"

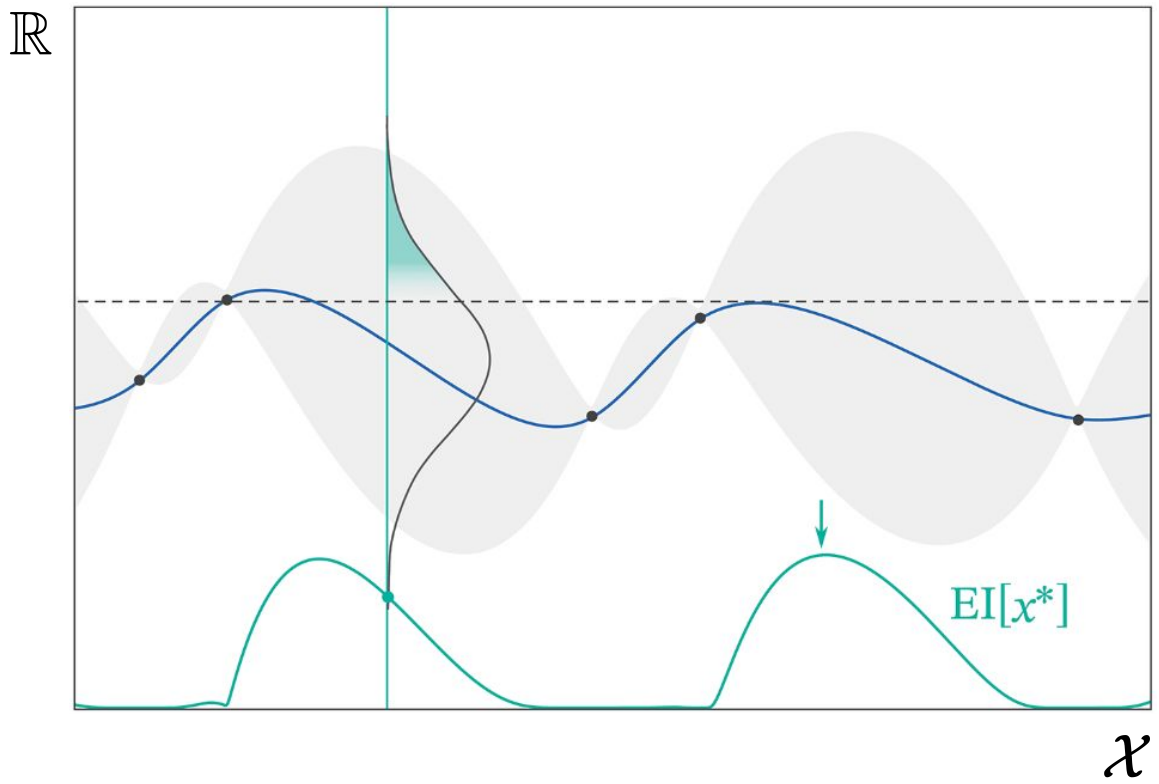


PI: probability that $f(x)$ is greater than best point

Acquisition Functions — Expected Improvement (EI)

Nice visual comparing UCB *vs.* PI *vs.* EI:

Source: M. O. Ahmed, S. Prince, "Bayesian optimization"



EI: *expected amount* $f(x)$ is greater than best point

(and ignoring amount that it's less than best point)

Acquisition Functions — Entropy Search (ES)

Acquisition Functions — Entropy Search (ES)

Another strategy is the “experimental design” strategy, but *applied to BO setting*.

Acquisition Functions — Entropy Search (ES)

Another strategy is the “experimental design” strategy, but *applied to BO setting*.

Specifically, this means:

Reducing posterior uncertainty (entropy) over quantity of interest — in this case, the location x^* of the function maximizer — rather than, e.g., the full landscape of f.

Acquisition Functions — Entropy Search (ES)

Another strategy is the “experimental design” strategy, but *applied to BO setting*.

Specifically, this means:

Reducing posterior uncertainty (entropy) over quantity of interest — in this case, the location x^* of the function maximizer — rather than, e.g., the full landscape of f.

As mentioned, this is an *info-based BO* acquisition function called **entropy search**.

Acquisition Functions — Entropy Search (ES)

Another strategy is the "experimental design" strategy, but *applied to BO setting*.

Specifically, this means:

Reducing posterior uncertainty (entropy) over quantity of interest — in this case, the location x^* of the function maximizer — rather than, e.g., the full landscape of f.

As mentioned, this is an *info-based BO* acquisition function called **entropy search**.

Why would you want to use this acquisition function?

⇒ Intuitively: if you get rewarded for *how good your guess of* x^* *is* — rather than, the value of the maximizer, $f(x^*)$ — ES should be more optimal.

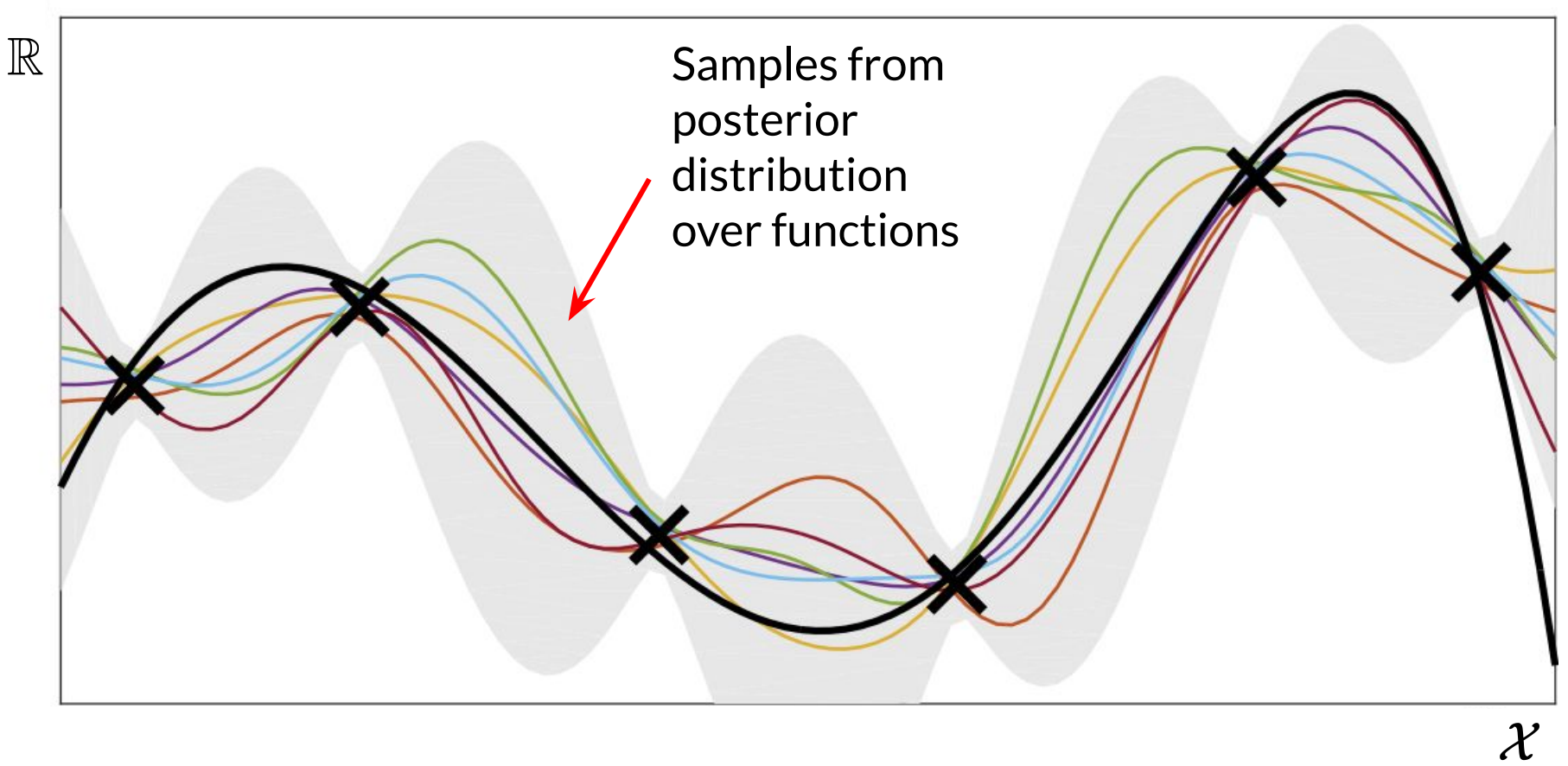
Acquisition Functions — Entropy Search

Visualizing ES

To explain this acquisition function, let's visualize…

Acquisition Functions — Entropy Search

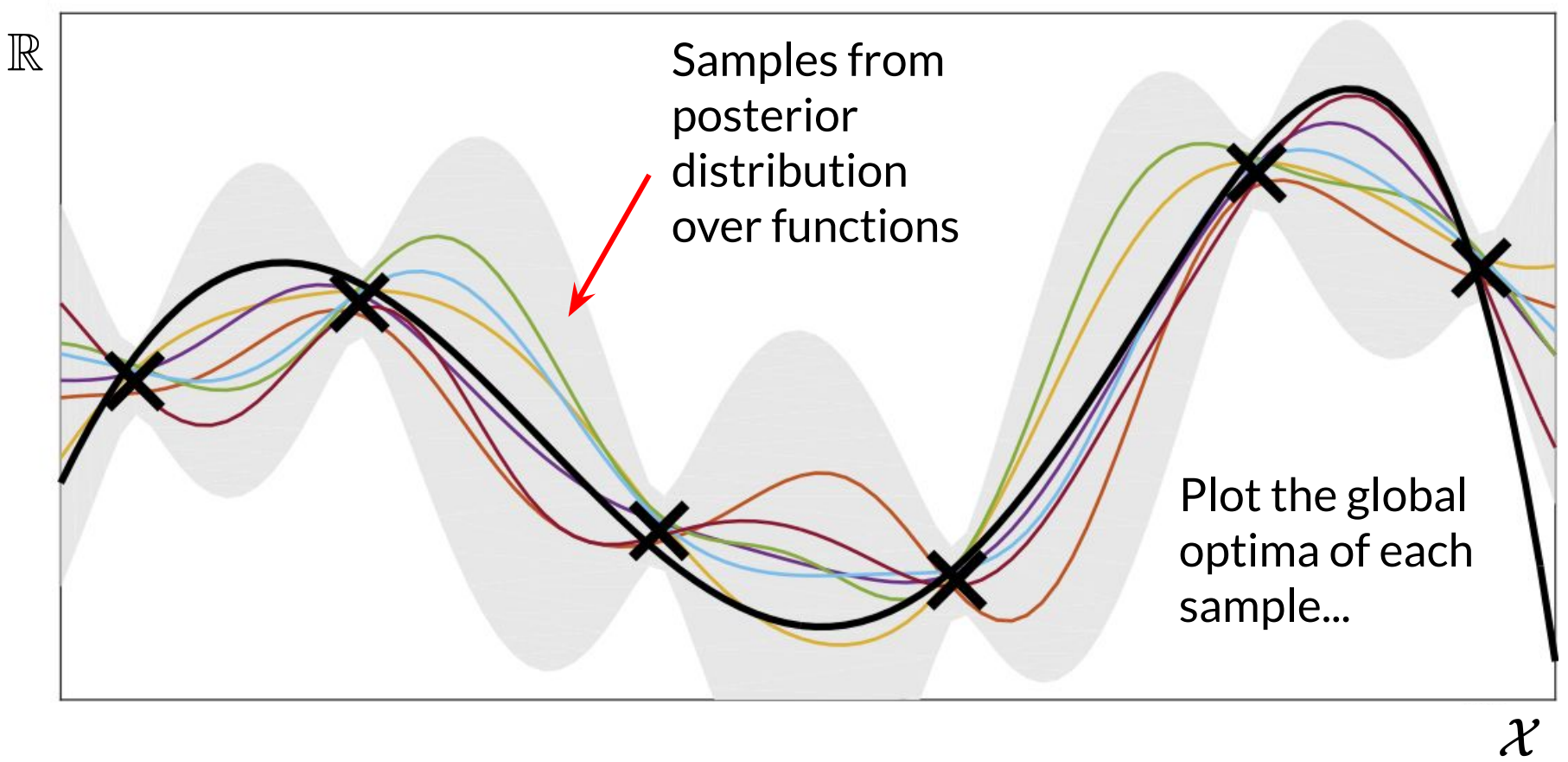
Visualizing ES



Probabilistic model of f, given dataset.

Acquisition Functions — Entropy Search

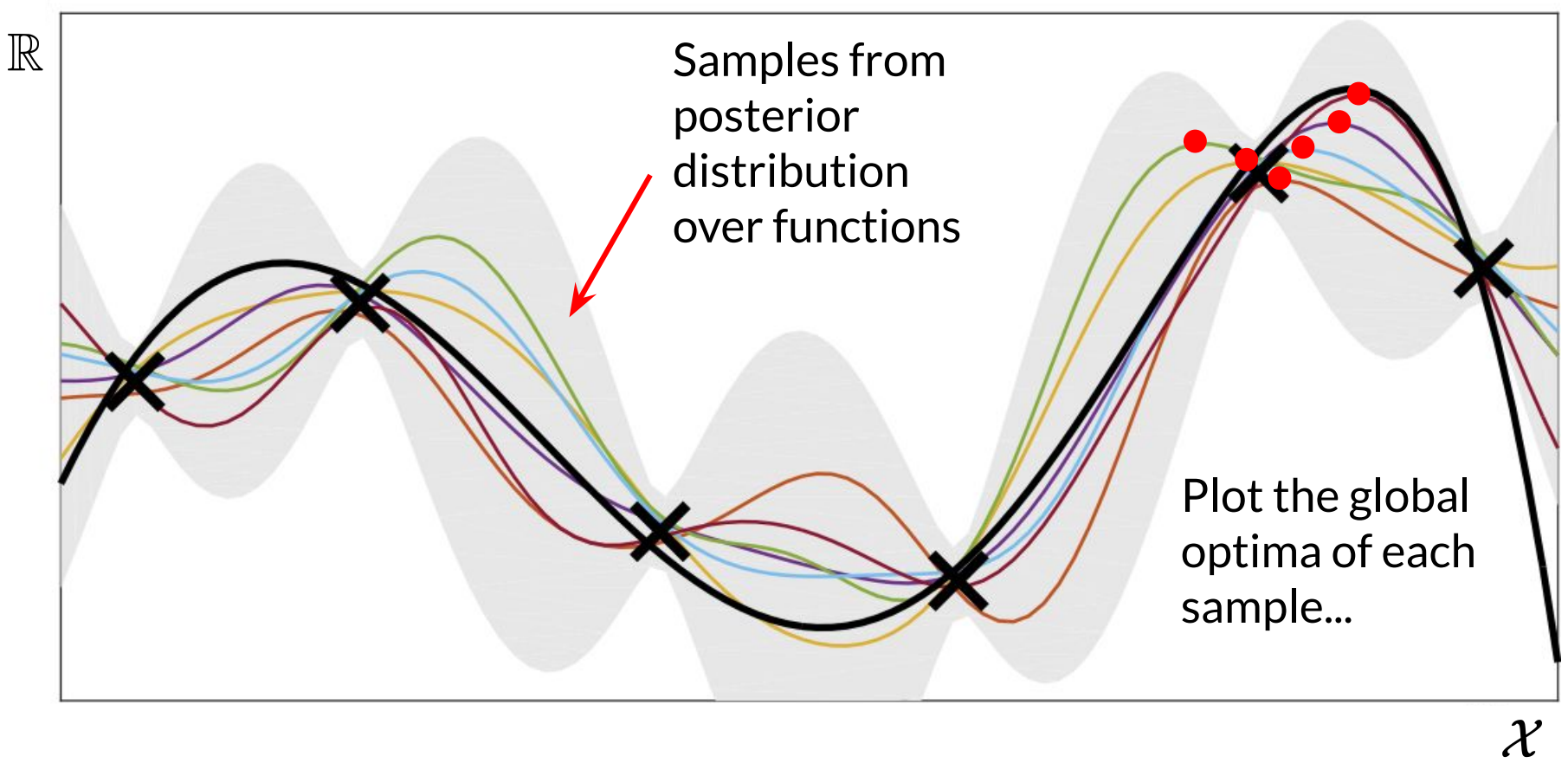
Visualizing ES



Probabilistic model of f, given dataset.

Acquisition Functions — Entropy Search

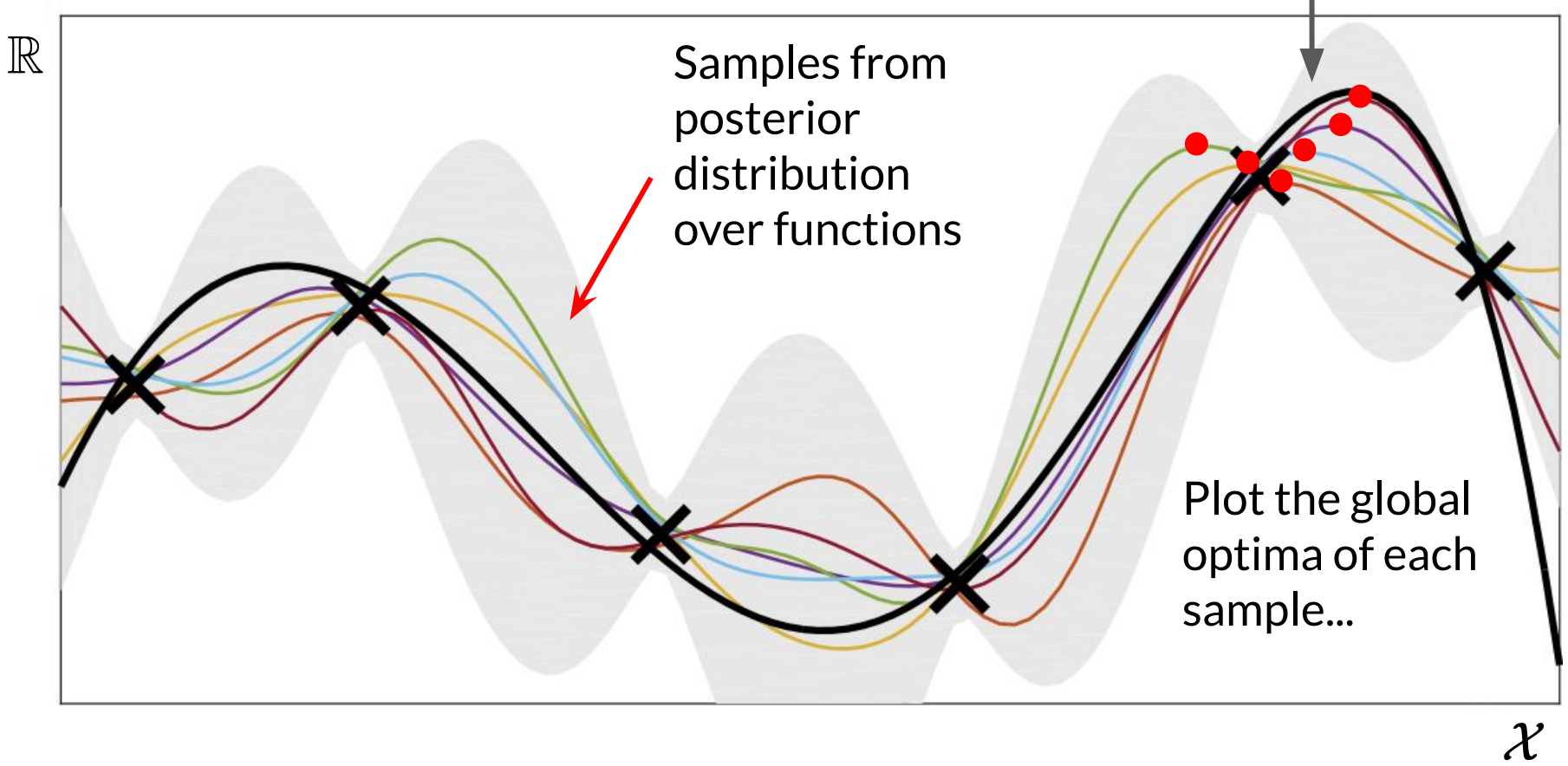
Visualizing ES



Probabilistic model of f, given dataset.

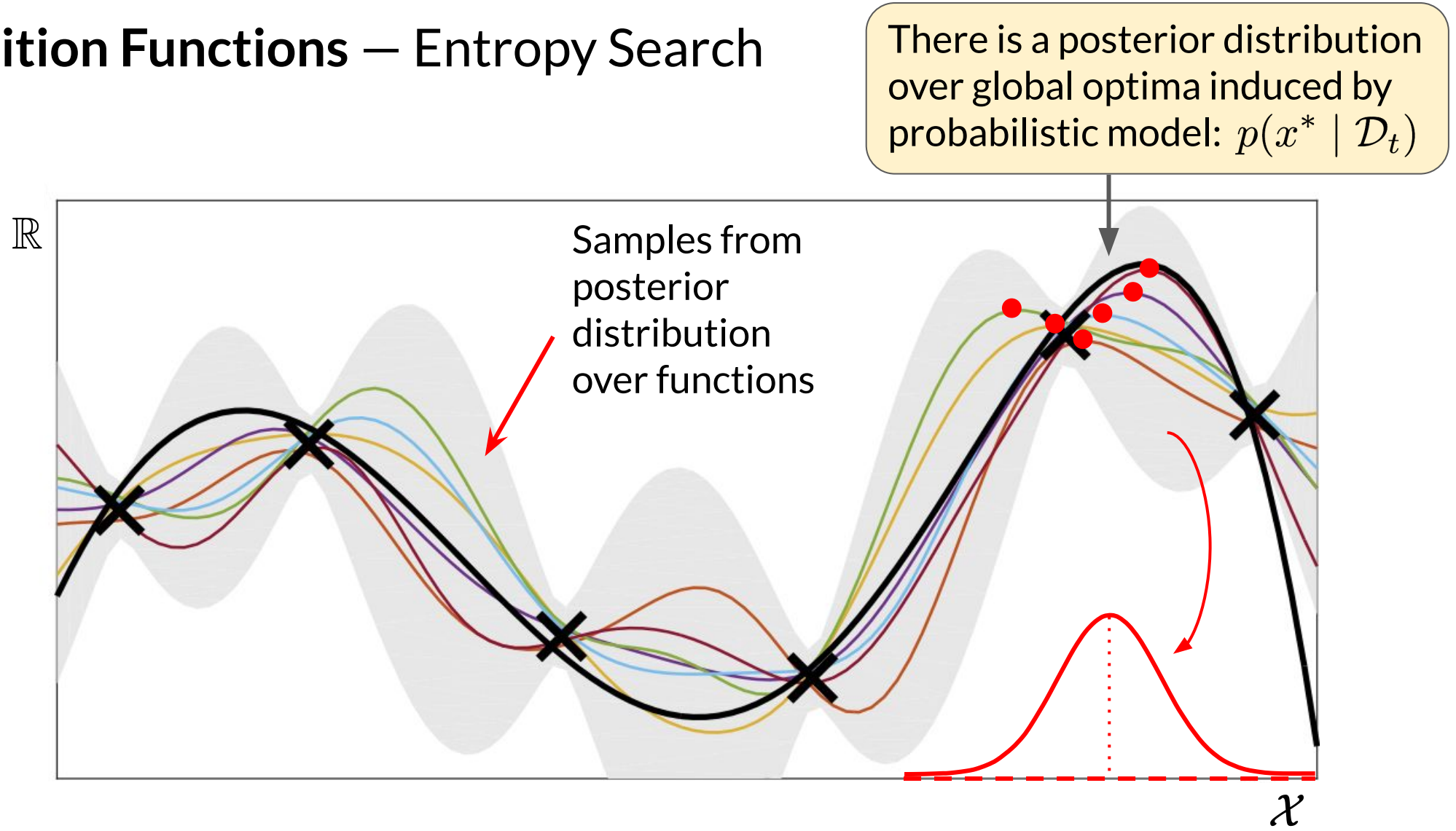
Acquisition Functions — Entropy Search

There is a posterior distribution over global optima induced by probabilistic model: $p(x^* \mid \mathcal{D}_t)$



Probabilistic model of f, given dataset.

Acquisition Functions — Entropy Search



Probabilistic model of f, given dataset.

Acquisition Functions — Entropy Search

This leads us to the acquisition function:

e.g. used in entropy search (ES), predictive entropy search (PES)

"Entropy Search for Information-Efficient Global Optimization", Hennig and Schuler, **JMLR 2012**

"Predictive Entropy Search for Efficient Global Optimization of Black-box Functions", Hernández-Lobato, Hoffman, Ghahramani, **NeurIPS 2014**

Acquisition Functions — Entropy Search

This leads us to the acquisition function:

e.g. used in entropy search (ES), predictive entropy search (PES)

$$\alpha_t(x) = H[p(x^* \mid \mathcal{D}_t)] - \mathbb{E}_{p(y_x \mid \mathcal{D}_t)}[H[p(x^* \mid \mathcal{D}_t \cup \{(x, y_x)\})]]$$

entropy of posterior distribution over global optima x^*

minus

expected entropy of posterior distribution over global optima x^*,
… if we were to make a query at x

"Entropy Search for Information-Efficient Global Optimization", Hennig and Schuler, **JMLR 2012**

"Predictive Entropy Search for Efficient Global Optimization of Black-box Functions", Hernández-Lobato, Hoffman, Ghahramani, **NeurIPS 2014**

Acquisition Functions — Entropy Search

This leads us to the acquisition function:

e.g. used in entropy search (ES), predictive entropy search (PES)

$$\alpha_t(x) = H[p(x^* \mid \mathcal{D}_t)] - \mathbb{E}_{p(y_x|\mathcal{D}_t)}[H[p(x^* \mid \mathcal{D}_t \cup \{(x, y_x)\})]]$$

entropy of posterior distribution over global optima x^*

minus

expected entropy of posterior distribution over global optima x^*,
… if we were to make a query at x

"Entropy Search for Information-Efficient Global Optimization", Hennig and Schuler, **JMLR 2012**
"Predictive Entropy Search for Efficient Global Optimization of Black-box Functions", Hernández-Lobato, Hoffman, Ghahramani, **NeurIPS 2014**

Acquisition Functions — Entropy Search

This leads us to the acquisition function:

e.g. used in entropy search (ES), predictive entropy search (PES)

$$\alpha_t(x) = H[p(x^* \mid \mathcal{D}_t)] - \mathbb{E}_{p(y_x \mid \mathcal{D}_t)}[H[p(x^* \mid \mathcal{D}_t \cup \{(x, y_x)\})]]$$

entropy of posterior distribution over global optima x^*

minus

expected entropy of posterior distribution over global optima x^*,
… if we were to make a query at x

"Entropy Search for Information-Efficient Global Optimization", Hennig and Schuler, **JMLR 2012**

"Predictive Entropy Search for Efficient Global Optimization of Black-box Functions", Hernández-Lobato, Hoffman, Ghahramani, **NeurIPS 2014**

Acquisition Functions — Entropy Search

This leads us to the acquisition function:

e.g. used in entropy search (ES), predictive entropy search (PES)

$$\alpha_t(x) = H[p(x^* \mid \mathcal{D}_t)] - \mathbb{E}_{p(y_x \mid \mathcal{D}_t)}[H[p(x^* \mid \mathcal{D}_t \cup \{(x, y_x)\})]]$$

entropy of posterior distribution over global optima x^*

minus

expected entropy of posterior distribution over global optima x^*,
… if we were to make a query at x

"Entropy Search for Information-Efficient Global Optimization", Hennig and Schuler, **JMLR 2012**

"Predictive Entropy Search for Efficient Global Optimization of Black-box Functions", Hernández-Lobato, Hoffman, Ghahramani, **NeurIPS 2014**

Acquisition Functions — Entropy Search

This leads us to the acquisition function:

e.g. used in entropy search (ES), predictive entropy search (PES)

$$\alpha_t(x) = H[p(x^* \mid \mathcal{D}_t)] - \mathbb{E}_{p(y_x|\mathcal{D}_t)}[H[p(x^* \mid \mathcal{D}_t \cup \{(x, y_x)\})]]$$

entropy of posterior distribution over global optima x^*

minus

expected entropy of posterior distribution over global optima x^*,
… if we were to make a query at x

"Entropy Search for Information-Efficient Global Optimization", Hennig and Schuler, **JMLR 2012**
"Predictive Entropy Search for Efficient Global Optimization of Black-box Functions", Hernández-Lobato, Hoffman, Ghahramani, **NeurIPS 2014**

Acquisition Functions — Entropy Search

This leads us to the acquisition function:

e.g. used in entropy search (ES), predictive entropy search (PES)

$$\alpha_t(x) = H[p(x^* \mid \mathcal{D}_t)] - \mathbb{E}_{p(y_x \mid \mathcal{D}_t)}[H[p(x^* \mid \mathcal{D}_t \cup \{(x, y_x)\})]]$$

entropy of posterior distribution over global optima x^*

minus

expected entropy of posterior distribution over global optima x^*,
… if we were to make a query at x

Expected information gain (EIG) — expected decrease in entropy if we were to query f at x.

Acquisition Functions — Additional

Acquisition Functions — Additional

A few acquisition functions we didn’t get to today:

Acquisition Functions — Additional

A few acquisition functions we didn't get to today:

- **Knowledge gradient** — somewhat complex, but one of the *most-optimal* (both theoretically and in practice) BO acquisition functions.

Acquisition Functions — Additional

A few acquisition functions we didn't get to today:

- **Knowledge gradient** — somewhat complex, but one of the *most-optimal* (both theoretically and in practice) BO acquisition functions.
- **Thompson sampling** — a stochastic acquisition function (*i.e.*, draw one posterior, and use it within an acquisition function)

Acquisition Functions — Additional

A few acquisition functions we didn't get to today:

- **Knowledge gradient** — somewhat complex, but one of the *most-optimal* (both theoretically and in practice) BO acquisition functions.
- **Thompson sampling** — a stochastic acquisition function (*i.e.*, draw one posterior, and use it within an acquisition function)
- **Non-BO acq fns.** — acquisition functions for non-optimization tasks, such as level sets, quadrature, phase boundaries, and more-general tasks.

Next Time

Next Time

Finish up our discussion of active learning methods!