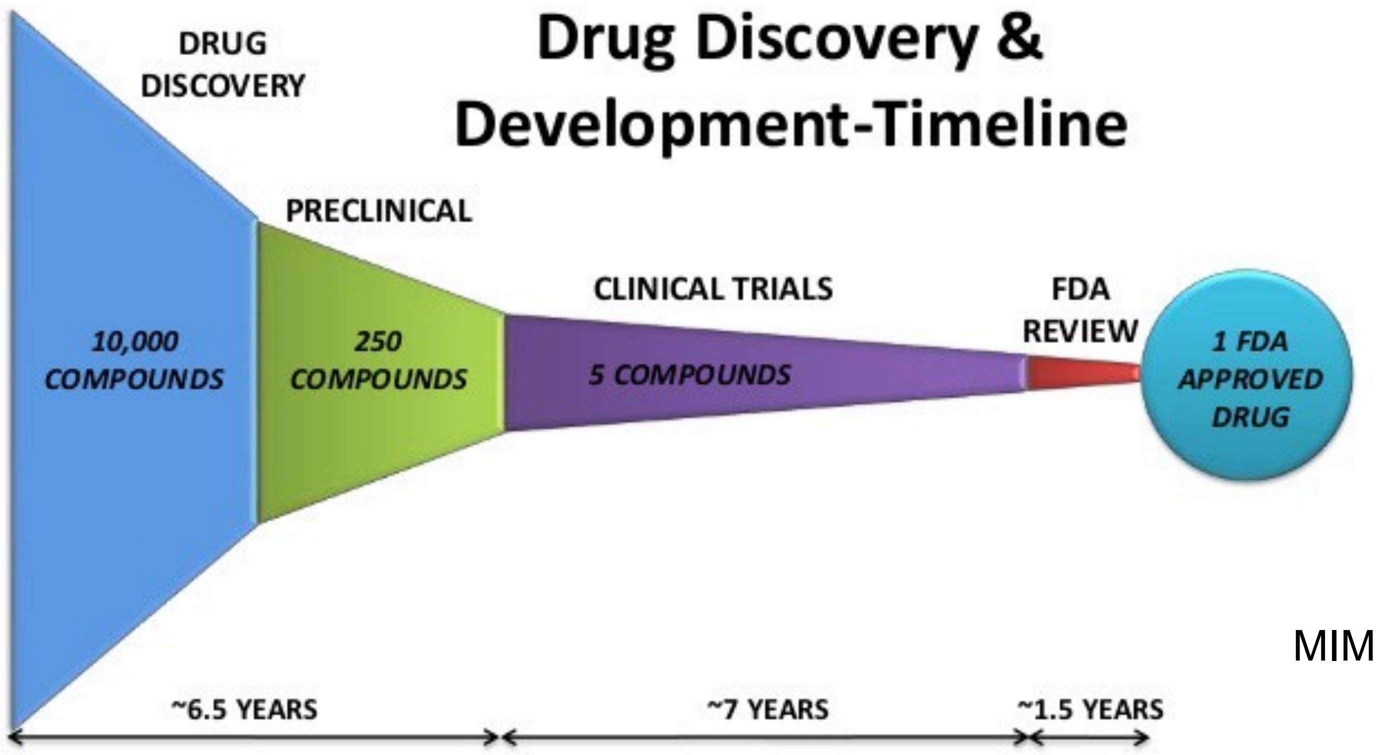
Recent Advances in Bayesian Optimization

@deshwal_aryan

@syrineblk

@janadoppa

Drug/Vaccine Design

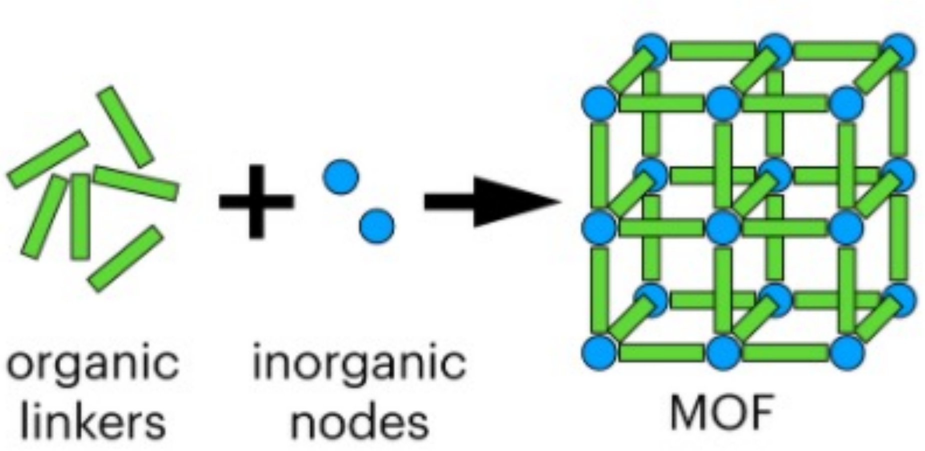


Credit:

MIMA healthcare

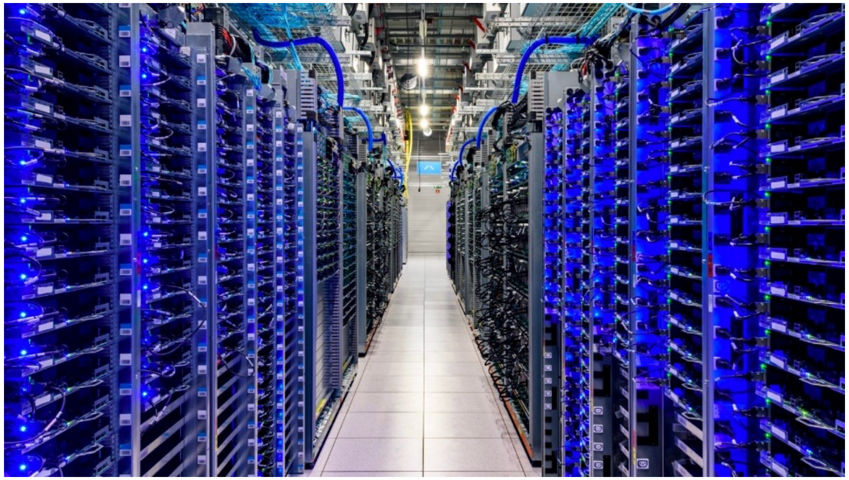
- Accelerate the discovery of promising designs

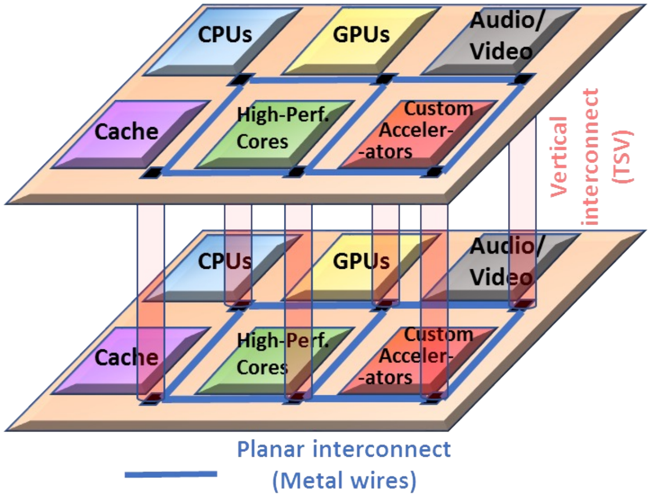
Nanoporous Materials Design



- **Sustainability applications**
  - Storing gases (e.g., hydrogen powered cars)
  - Separating gases (e.g., carbon dioxide from flue gas of coalfired power plants)
  - Detecting gases (e.g., detecting pollutants in outdoor air)

Sustainable Hardware Design for Data Centers





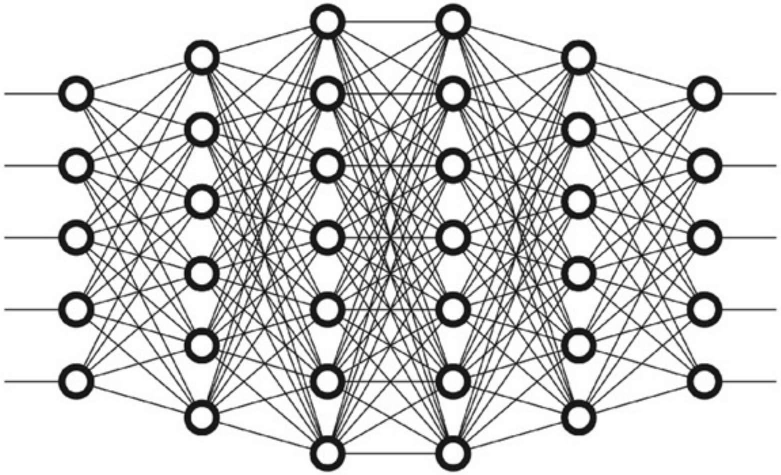
America’s Data Centers Are Wasting Huge Amounts of Energy

By 2020, data centers are projected to consume roughly 140 billion kilowatt-hours annually, costing American businesses $13 billion annually in electricity bills and emitting nearly 150 million metric tons of carbon pollution

High-performance and Energy-efficient manycore chips

Report from Natural Resources Defense Council:.
https://www.nrdc.org/sites/default/files/data-center-efficiency-assessment-IB.pdf

Auto ML and Hyperparameter Tuning

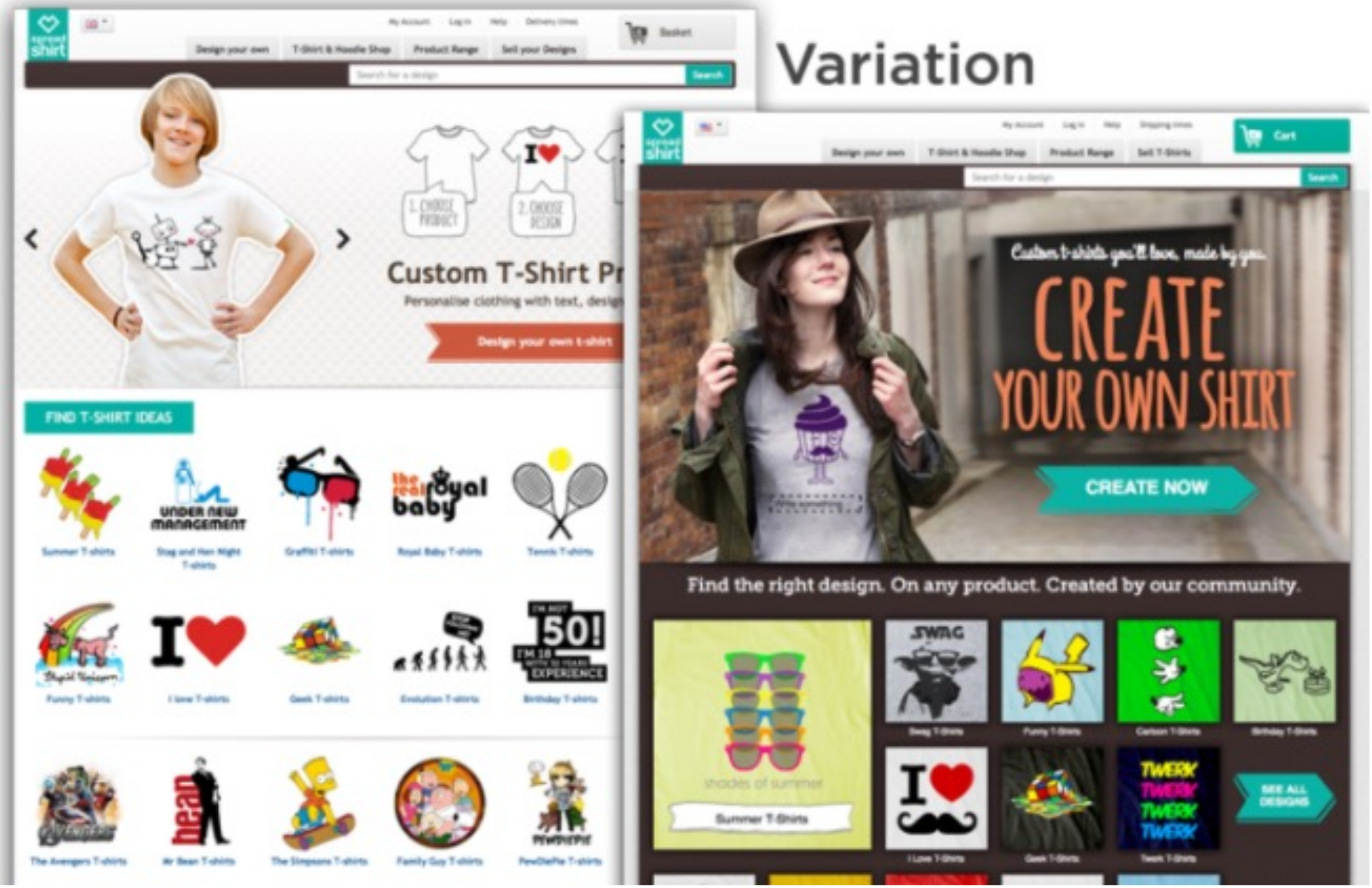


- Accuracy of models critically depends on hyper-parameters
  - Optimization algorithm, learning rates, momentum, batch normalization, batch sizes, dropout rates, weight decay, data augmentation, …

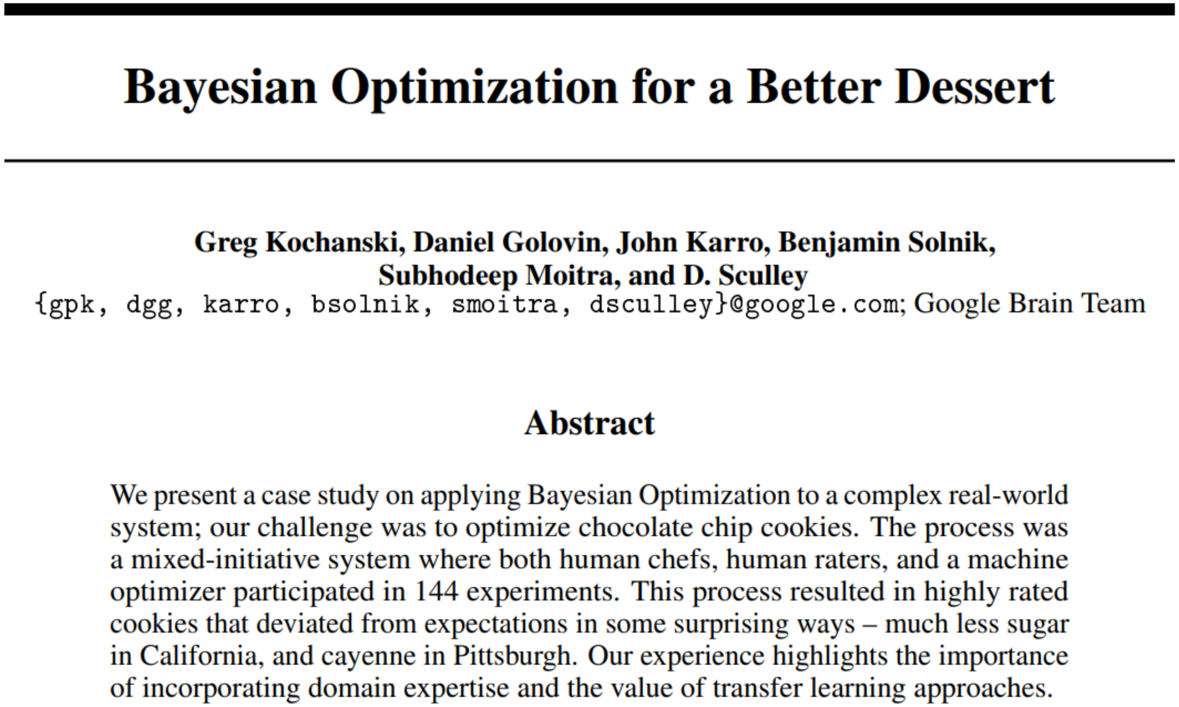
A/B Testing to Configure Websites

Original

Variation



Making Delicious Cookies



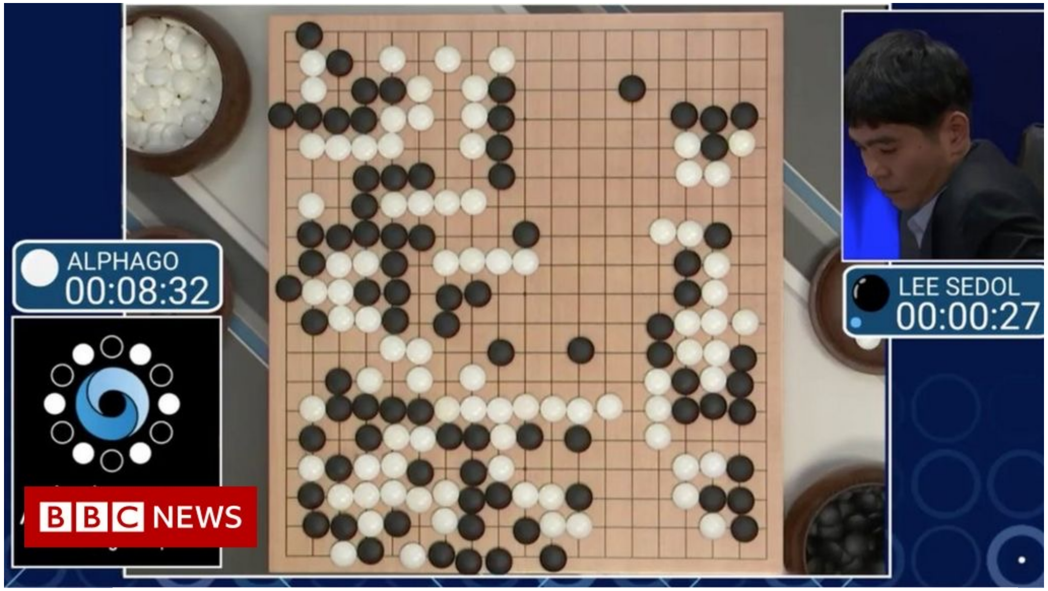
Bayesian Optimization for a Better Dessert

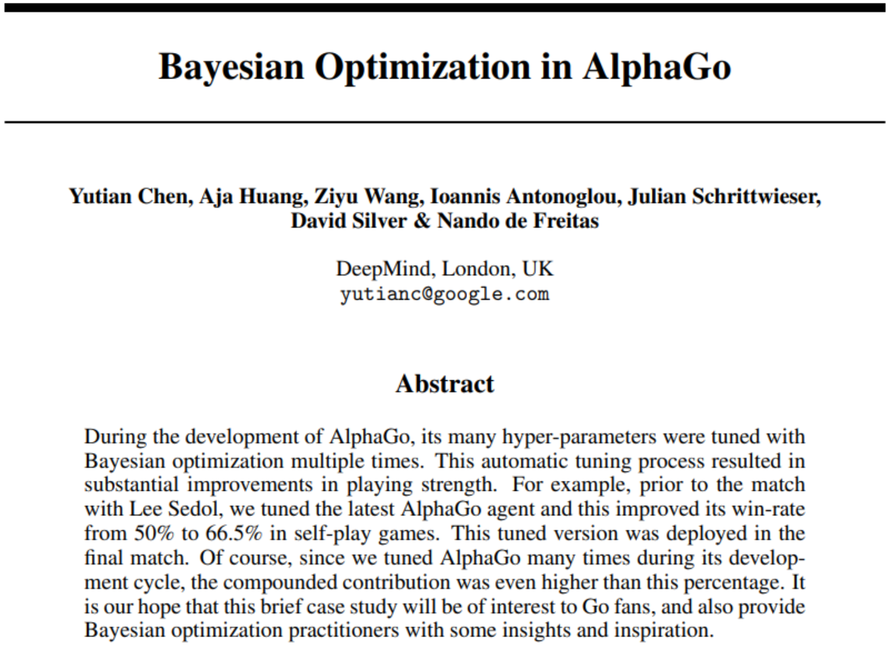
Greg Kochanski, Daniel Golovin, John Karro, Benjamin Solnik,
Subhodeep Moitra, and D. Sculley
`{gpk, dgg, karro, bsolnik, smoitra, dsculley}@google.com`; Google Brain Team

Abstract

We present a case study on applying Bayesian Optimization to a complex real-world system; our challenge was to optimize chocolate chip cookies. The process was a mixed-initiative system where both human chefs, human raters, and a machine optimizer participated in 144 experiments. This process resulted in highly rated cookies that deviated from expectations in some surprising ways – much less sugar in California, and cayenne in Pittsburgh. Our experience highlights the importance of incorporating domain expertise and the value of transfer learning approaches.

Making AlphaGo Better





Bayesian Optimization in AlphaGo

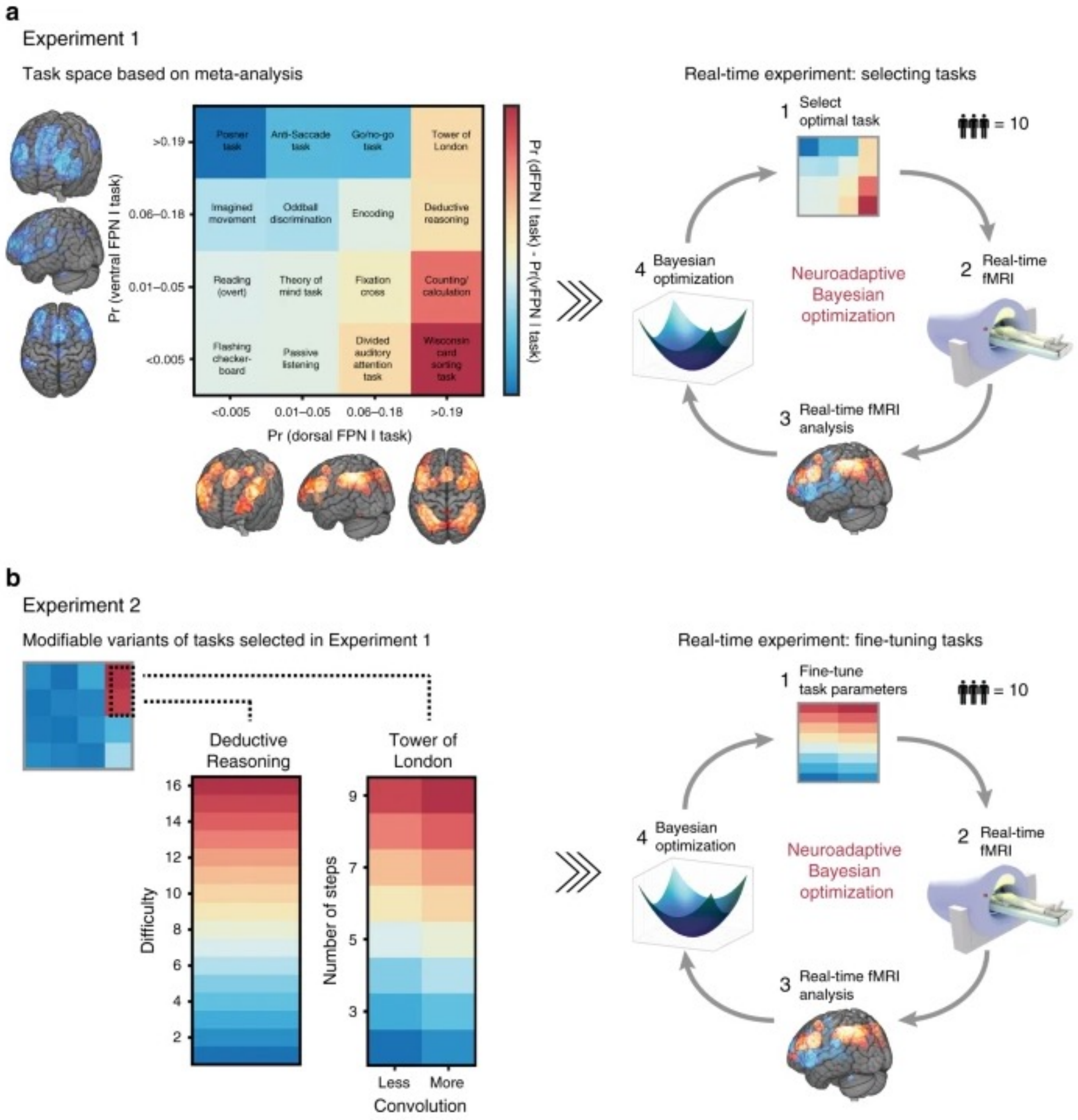
Yutian Chen, Aja Huang, Ziyu Wang, Ioannis Antonoglou, Julian Schrittwieser, David Silver & Nando de Freitas

DeepMind, London, UK
yutianc@google.com

Abstract

During the development of AlphaGo, its many hyper-parameters were tuned with Bayesian optimization multiple times. This automatic tuning process resulted in substantial improvements in playing strength. For example, prior to the match with Lee Sedol, we tuned the latest AlphaGo agent and this improved its win-rate from 50% to 66.5% in self-play games. This tuned version was deployed in the final match. Of course, since we tuned AlphaGo many times during its development cycle, the compounded contribution was even higher than this percentage. It is our hope that this brief case study will be of interest to Go fans, and also provide Bayesian optimization practitioners with some insights and inspiration.

Neuroscience and Brain Analytics



Credit: https://www.nature.com/articles/s41467-018-03657-3

Common Attributes of the Search Problem

- Search Space: Many candidate choices (inputs)

- Objective function: Need to perform an expensive experiment to evaluate the objective value of any input

- Optimization problem: find the candidate input with highest objective function value

Common Attributes of the Search Problem

- Search Space: Many candidate choices (inputs)
- Objective function: Need to perform an expensive experiment to evaluate the objective value of any input
- Optimization problem: find the candidate input with highest objective function value

Cannot afford exhaustive search

Common Attributes of the Search Problem

- Search Space: Many candidate choices (inputs)
- Objective function: Need to perform an expensive experiment to evaluate the objective value of any input
- Optimization problem: find the candidate input with highest objective function value

Trial and Error?

Common Attributes of the Search Problem

- Search Space: Many candidate choices (inputs)

- Objective function: Need to perform an expensive experiment to evaluate the objective value of any input

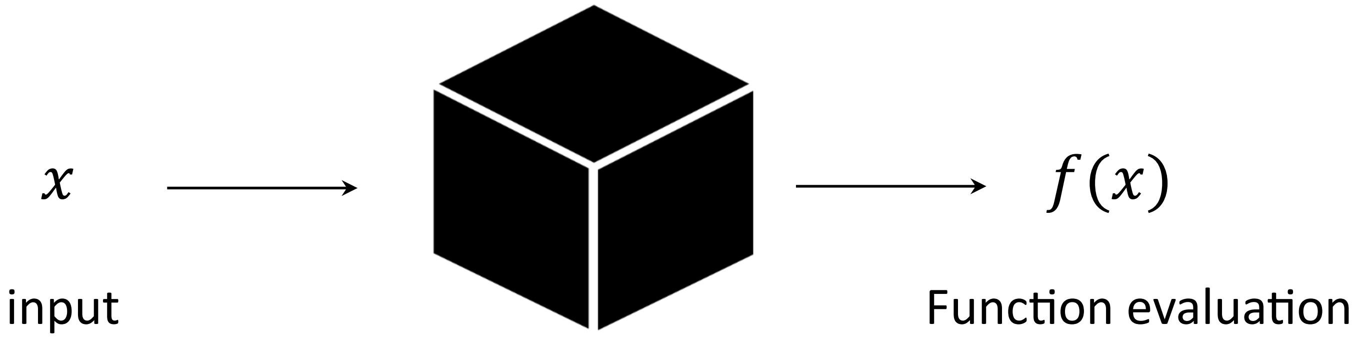
- Optimization problem: find the candidate input with highest objective function value

Can we do better than trial-and-error?

Accelerate Search via Bayesian Optimization

- Efficiently optimize expensive black-box functions

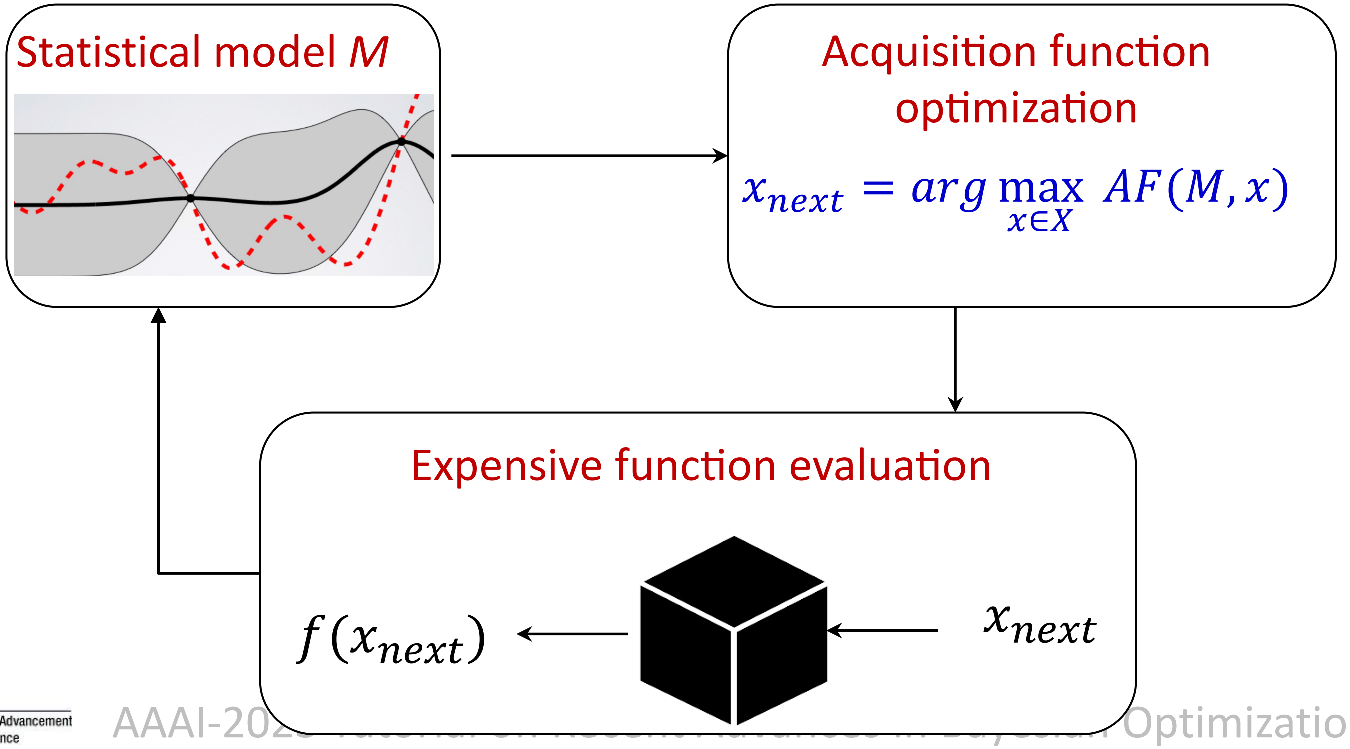
$$x^* = arg\max_{x \in X} f(x)$$



- Black-box queries (aka experiments) are expensive

Bayesian Optimization: Key Idea

- Build a surrogate statistical model and use it to intelligently search the space
  - Replace expensive queries with cheaper queries
  - Use uncertainty of the model to select expensive queries



Statistical model *M*

Acquisition function optimization

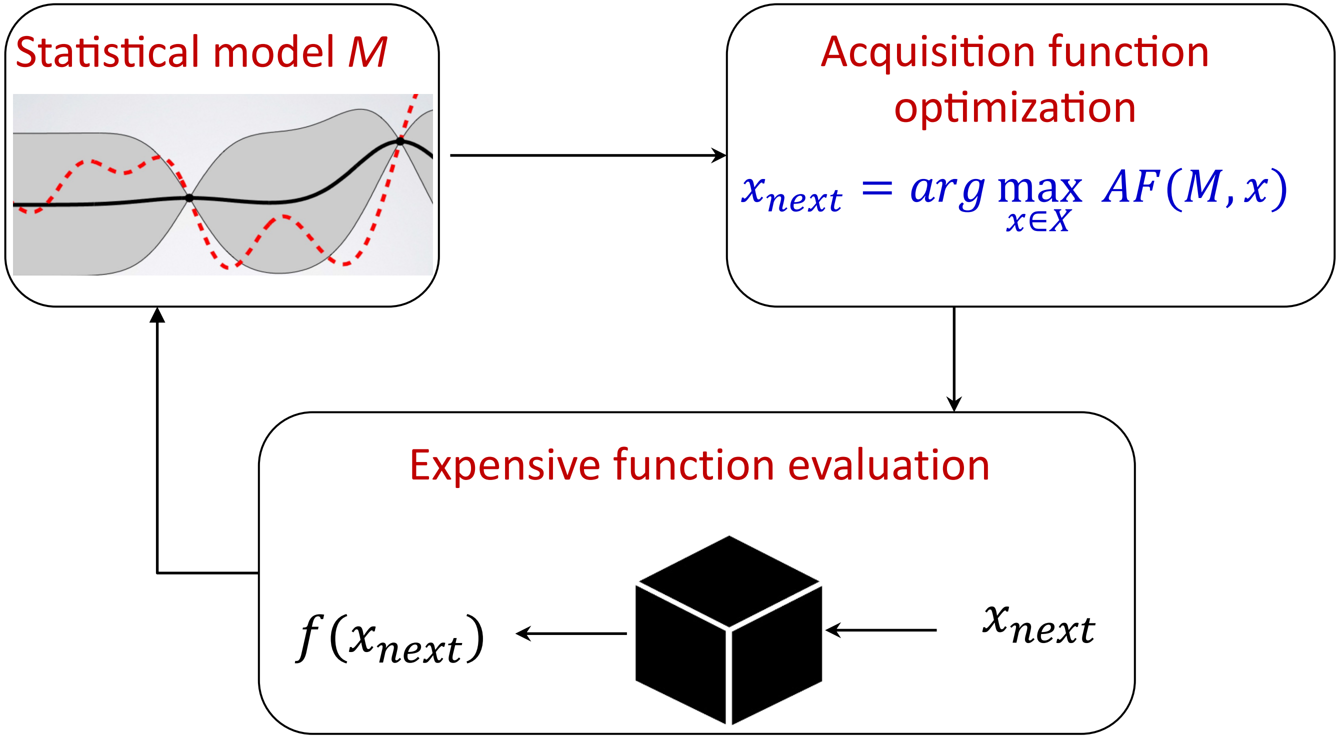
$$x_{next} = arg \max_{x \in X} AF(M, x)$$

Expensive function evaluation

$f(x_{next})$

x_{next}

Bayesian Optimization: Three Key Elements



Statistical model *M*

Acquisition function optimization

$$x_{next} = arg\max_{x \in X} AF(M, x)$$

Expensive function evaluation

$f(x_{next})$

x_{next}

- Statistical model (e.g., Gaussian process)
- Acquisition function (e.g., Expected improvement)
- Acquisition function optimizer (e.g., local search)

BO Dimensions: Input Space

- **Continuous space**
  - All variables of input x are continuous

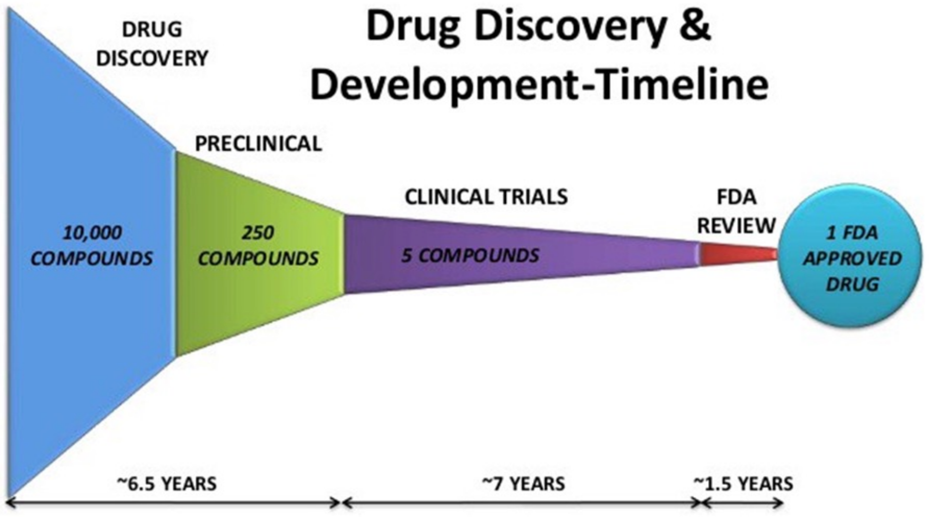
- **Discrete / Combinatorial space**
  - Sequences, trees, graphs, sets, permutations etc.

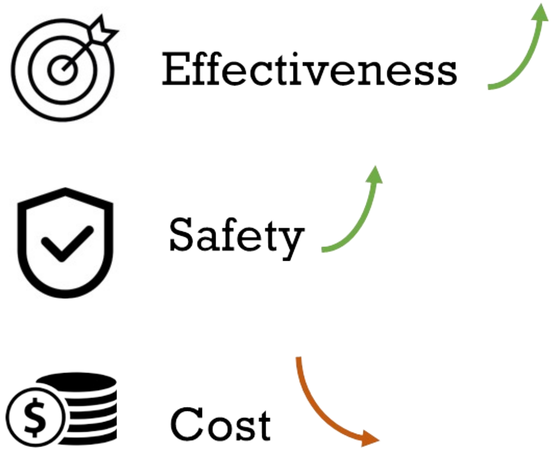
- **Hybrid space**
  - x = mixture of x_d (discrete) and x_c (continuous) variables

BO Dimensions: No. of Objectives

- **Single objective**
  - For example, finding hyperparameters to optimize accuracy

- **Multiple objectives**



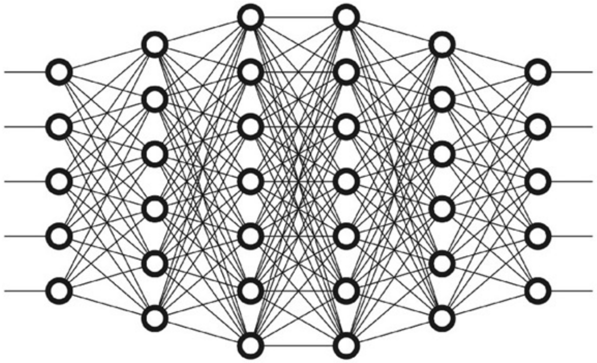


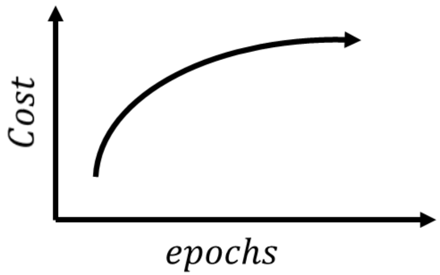
Credit: MIMA healthcare

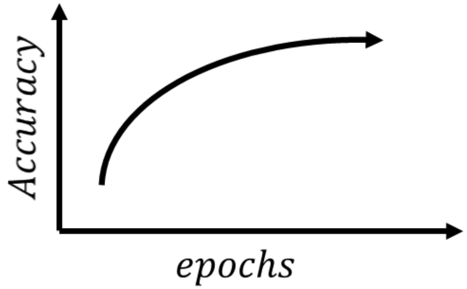
BO Dimensions: No. of Fidelities

- **Single-fidelity setting**
  - Most expensive and accurate function evaluation

- **Multi-fidelity setting**
  - Function evaluations with varying trade-offs in cost and accuracy



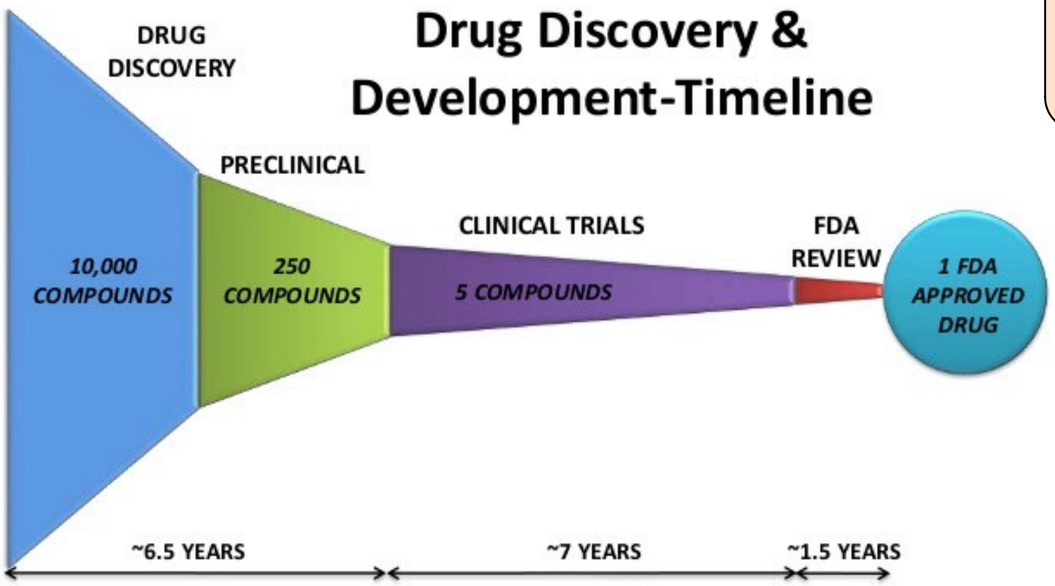




BO Dimensions: Constraints

- **Unconstrained setting**
  - all inputs are valid

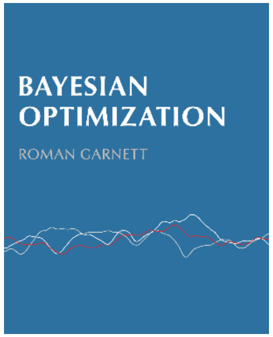
- **Constrained setting**



Drugs/Vaccines that are safe

Important References and Software

- **Bayesian Optimization Book by Roman Garnett**
  - https://bayesoptbook.com/
  - Excellent contribution by Roman to both AI researchers and practitioners! Thanks Roman!!



BAYESIAN OPTIMIZATION

ROMAN GARNETT

- **BoTorch Software**
  - https://botorch.org/
  - Excellent contribution by Meta's Adaptive Experimentation team!

Outline of the Tutorial

- Overview of the BO Framework, GPs, advances in GPs and acquisition functions, and BoTorch demo

- Bayesian Optimization over Discrete/Hybrid Spaces

- Multi-fidelity Bayesian Optimization

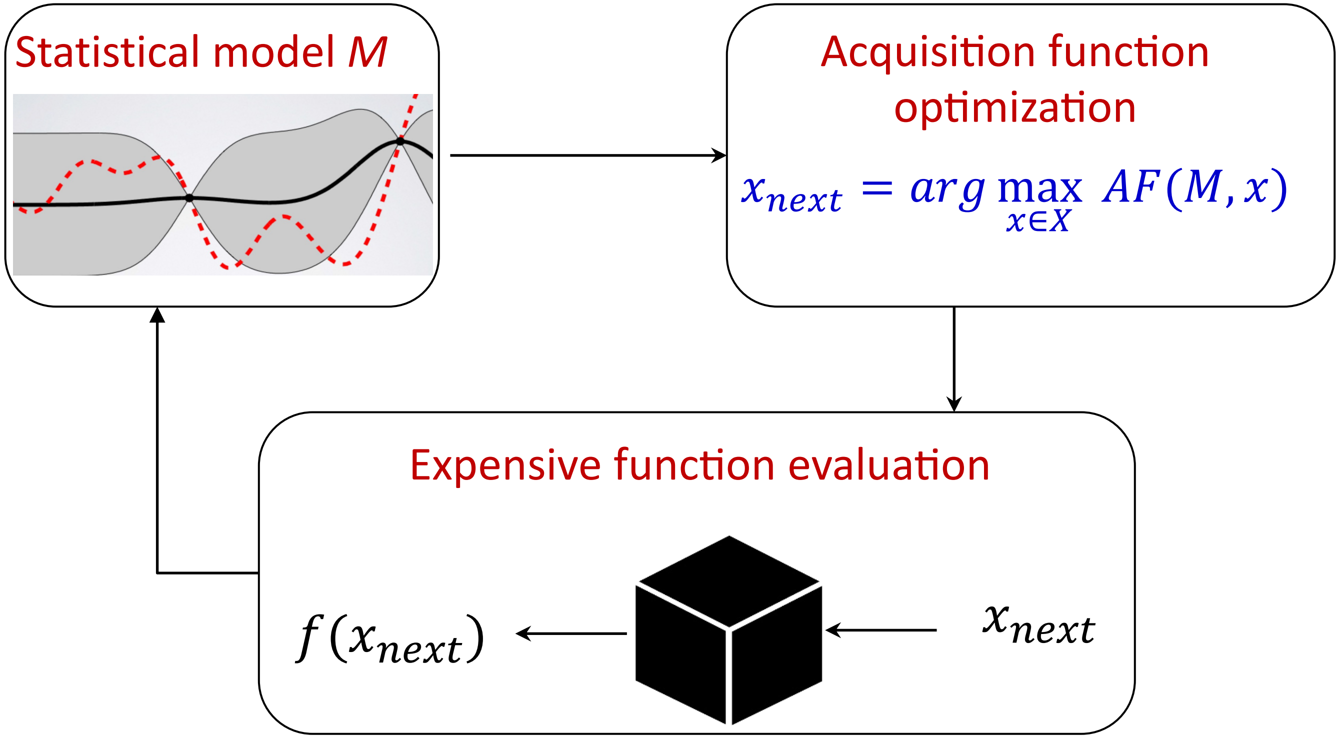
30 mins Break

- High-Dimensional BO and BoTorch Hands-on demo

- Multi-Objective BO and BoTorch Hands-on demo

- Summary and Outstanding Challenges in BO

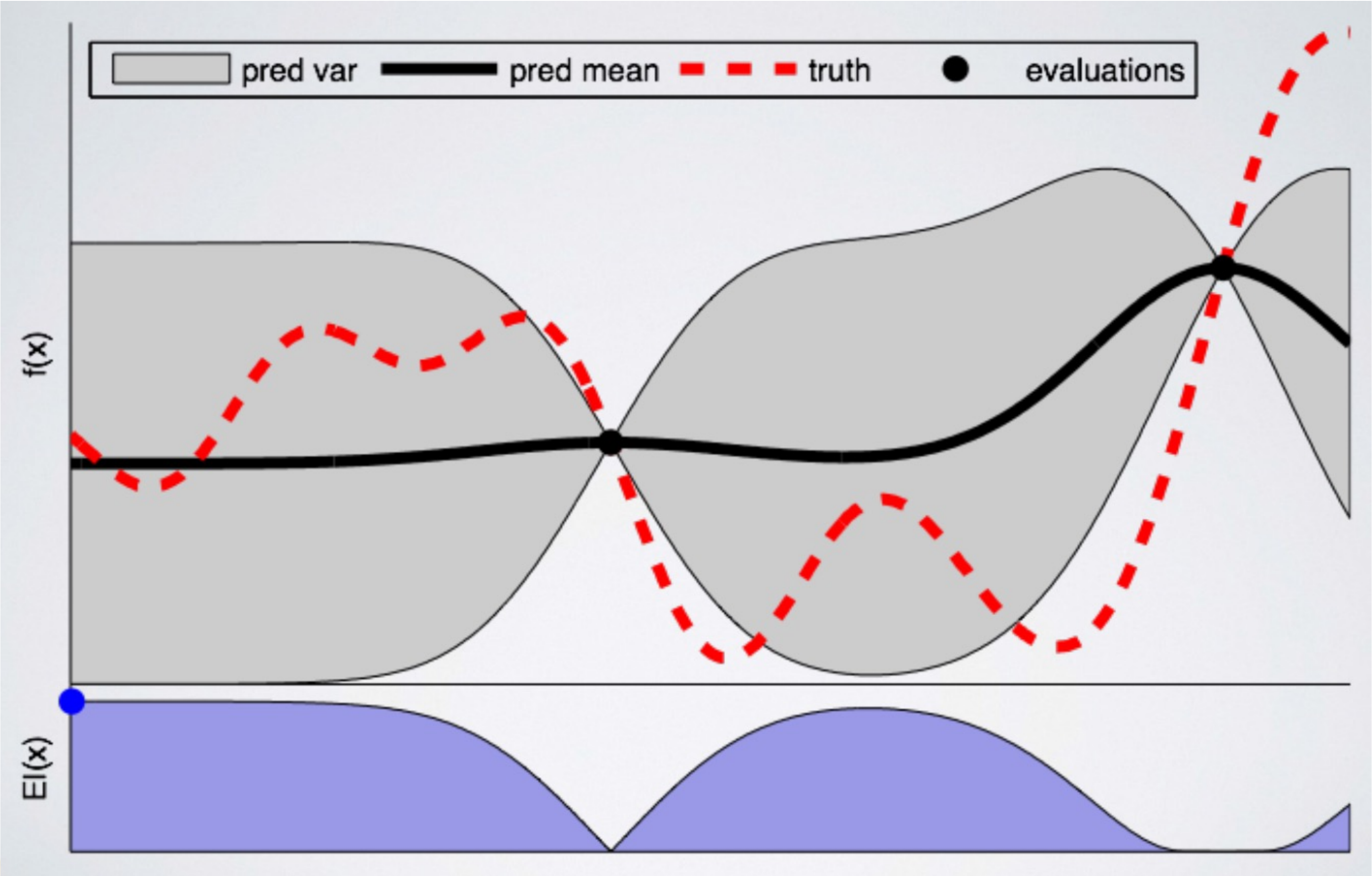
Bayesian Optimization Framework



Statistical model M
Acquisition function optimization
$x_{next} = arg \max_{x \in X} AF(M, x)$
Expensive function evaluation
$f(x_{next})$
x_{next}

- Statistical model (e.g., Gaussian process)
- Acquisition function (e.g., Expected improvement)
- Acquisition function optimizer (e.g., local search)

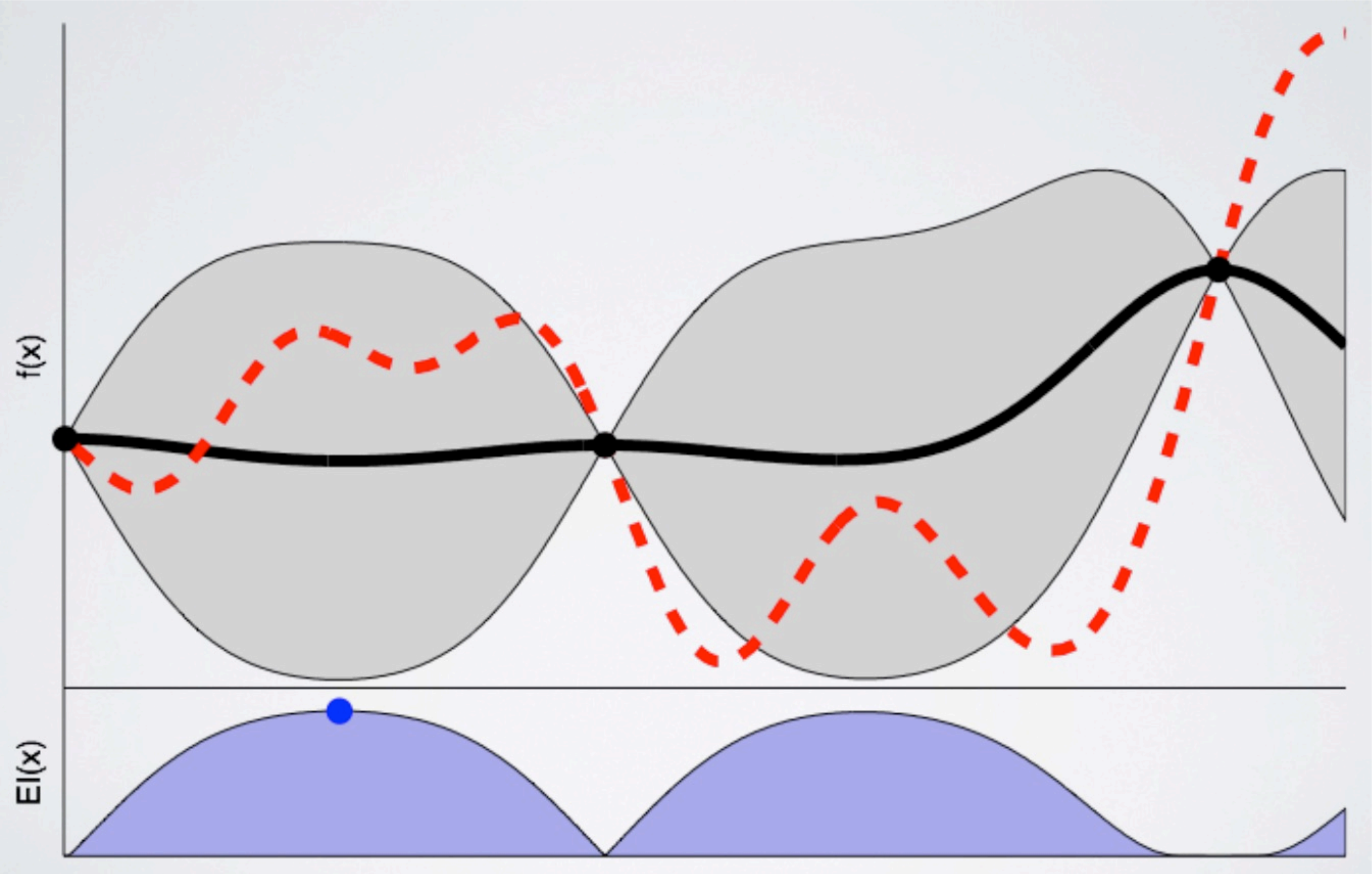
Bayesian Optimization: Illustration



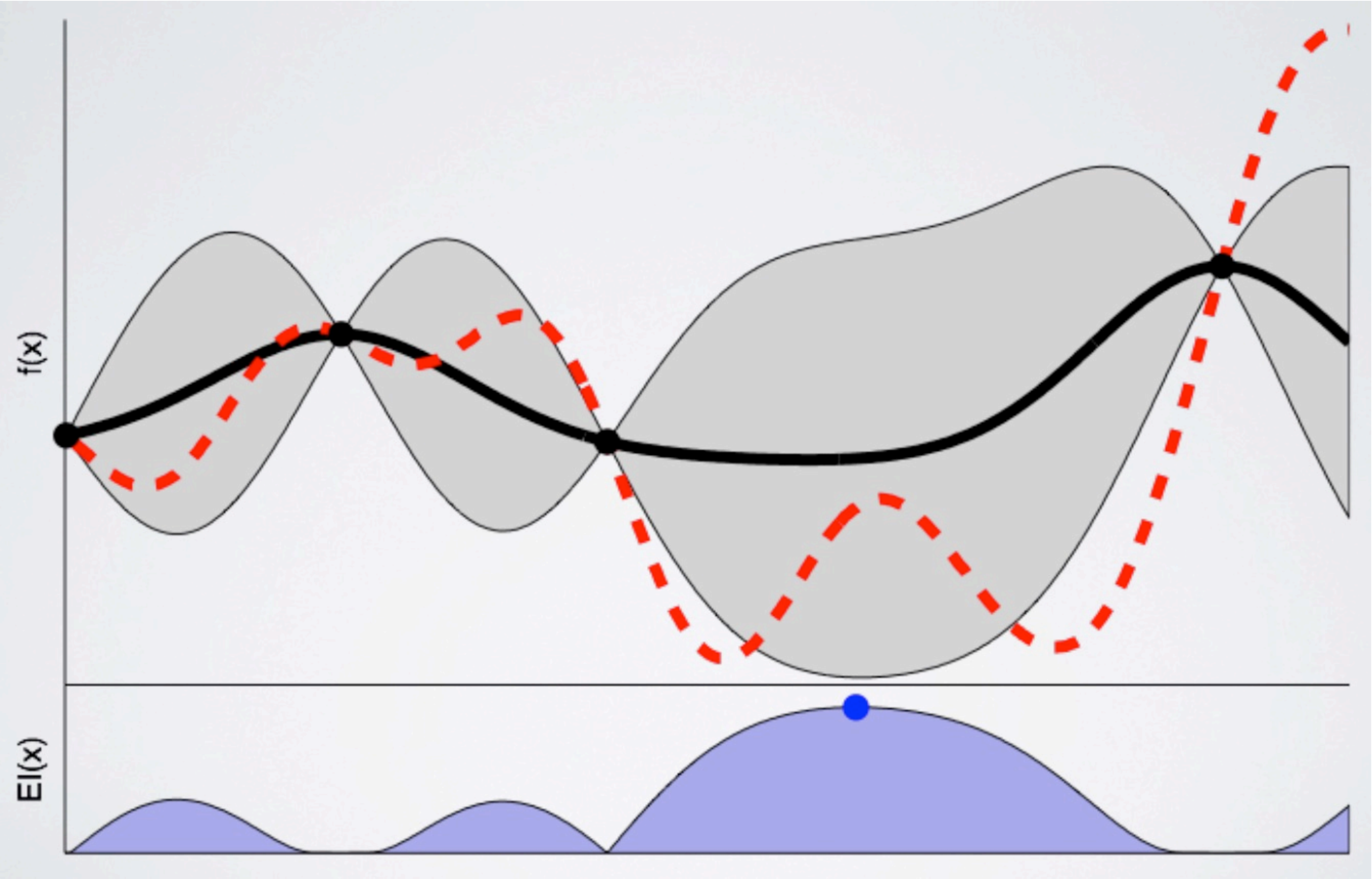
Credit: Ryan Adams

https://www.cs.toronto.edu/~rgrosse/courses/csc411_f18/tutorials/tut8_adams_slides.pdf

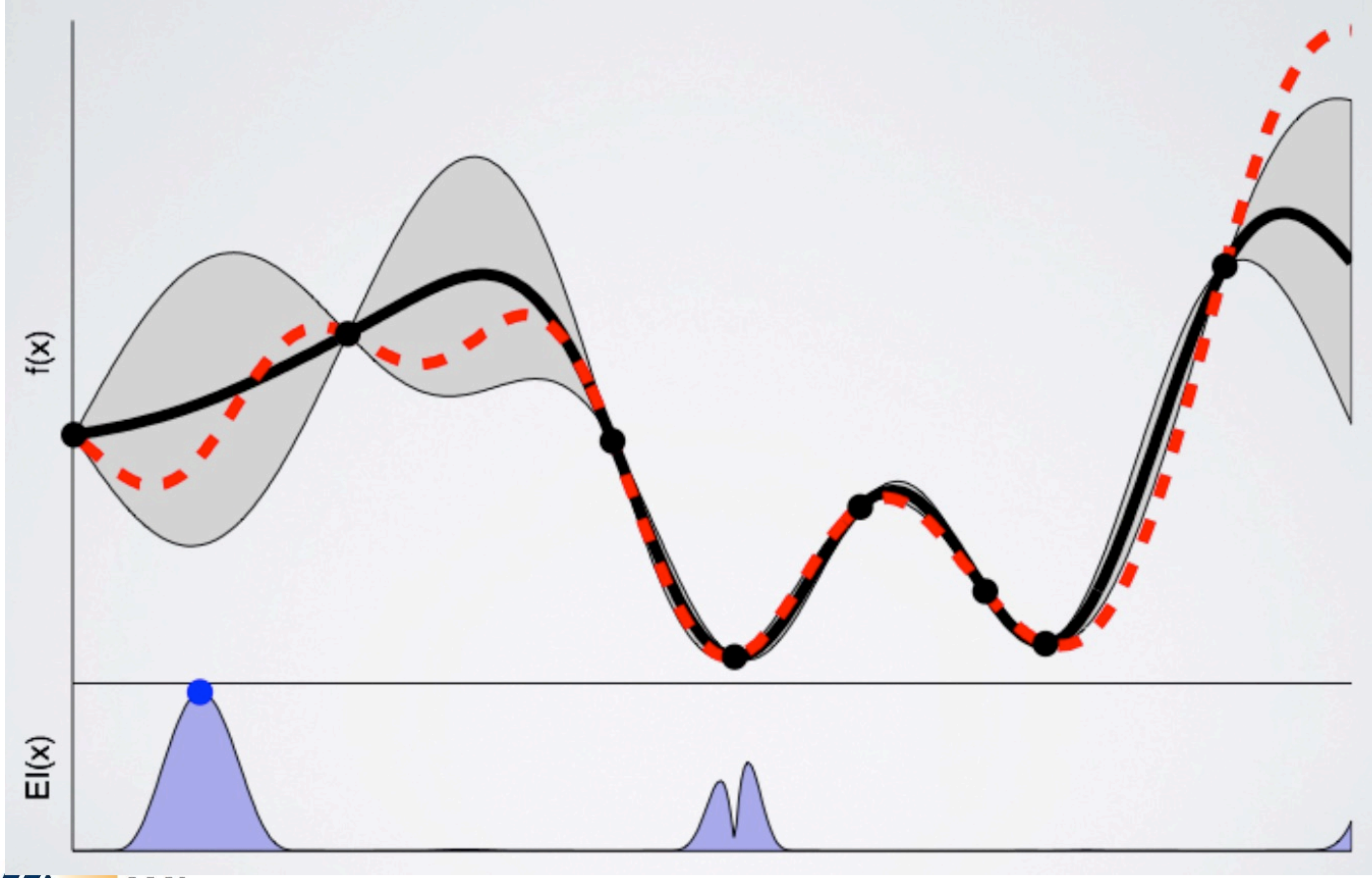
Bayesian Optimization: Illustration



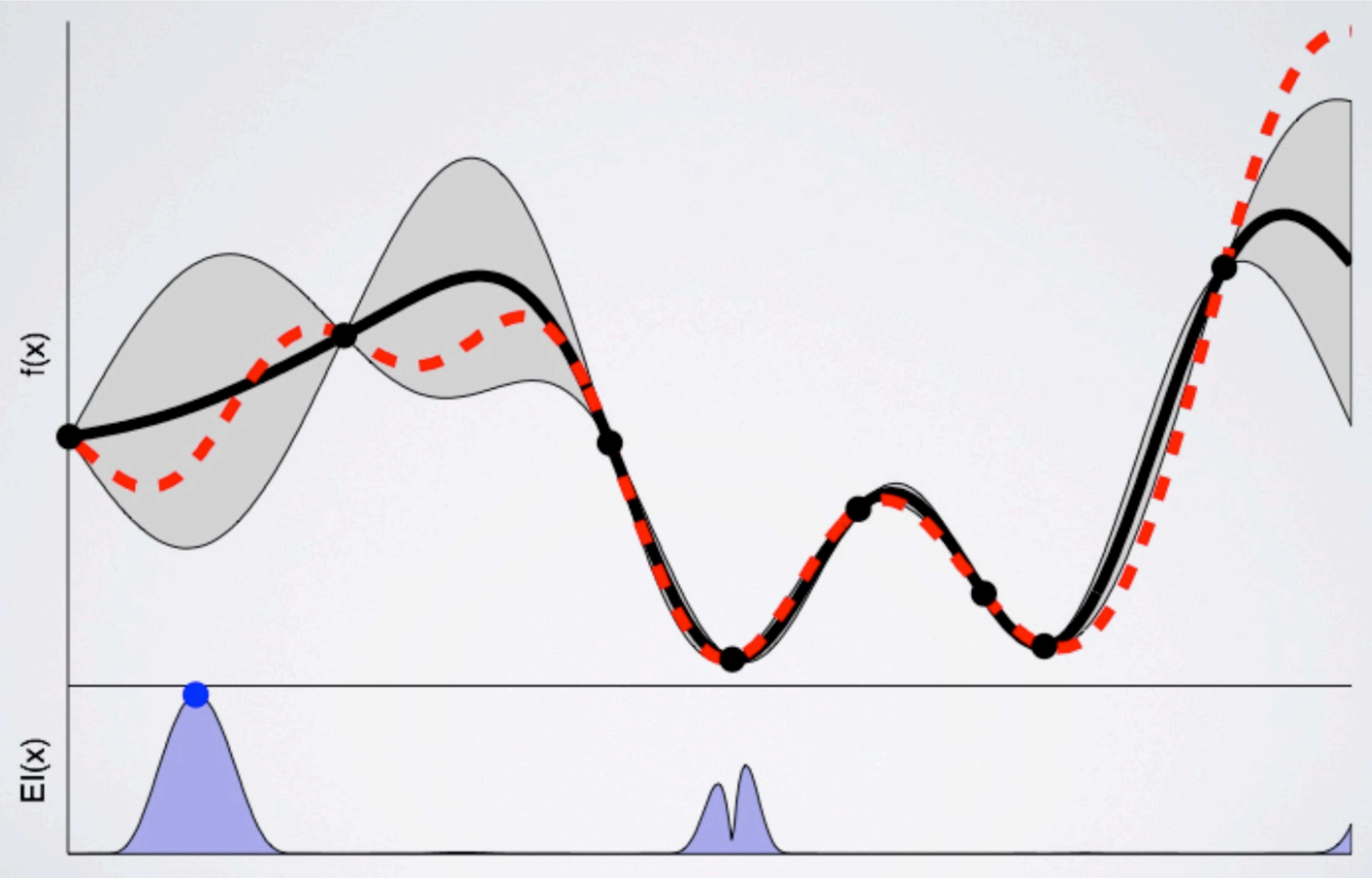
Bayesian Optimization: Illustration



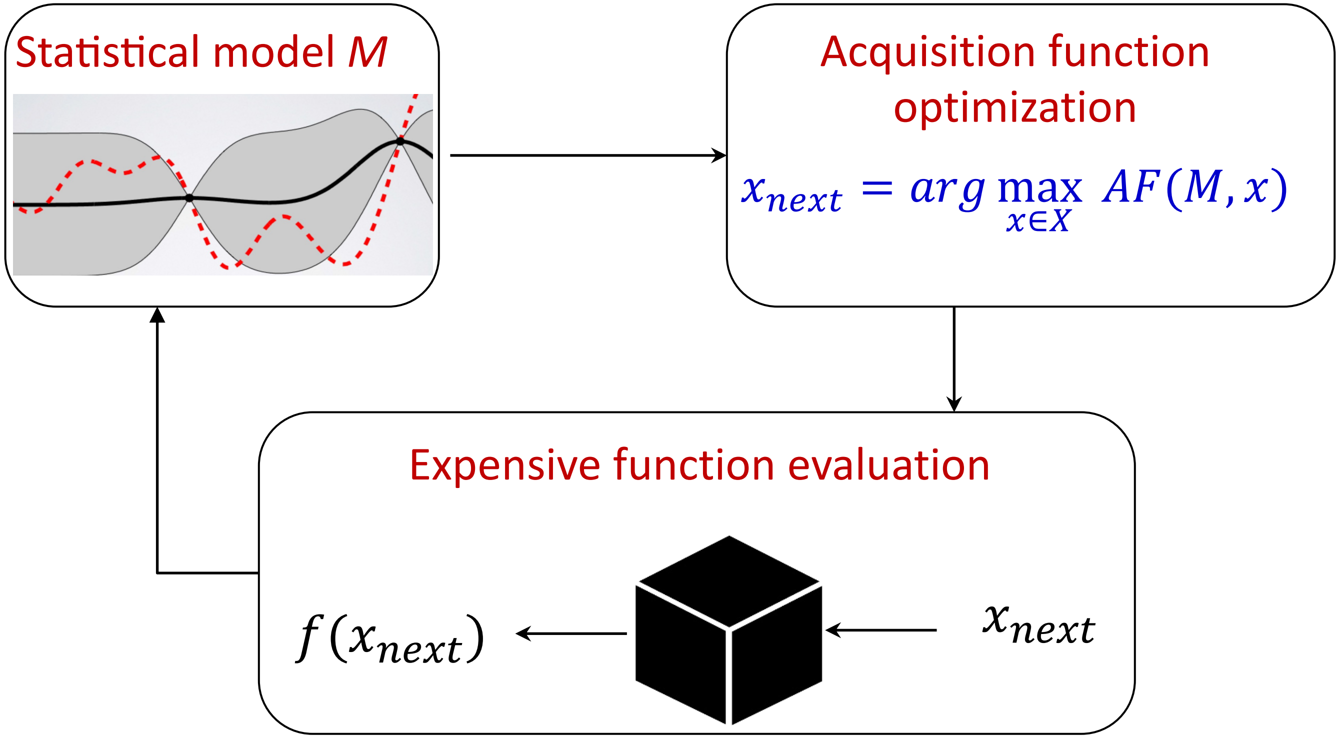
Bayesian Optimization: Illustration



Bayesian Optimization: Illustration



Bayesian Optimization: Three Key Elements

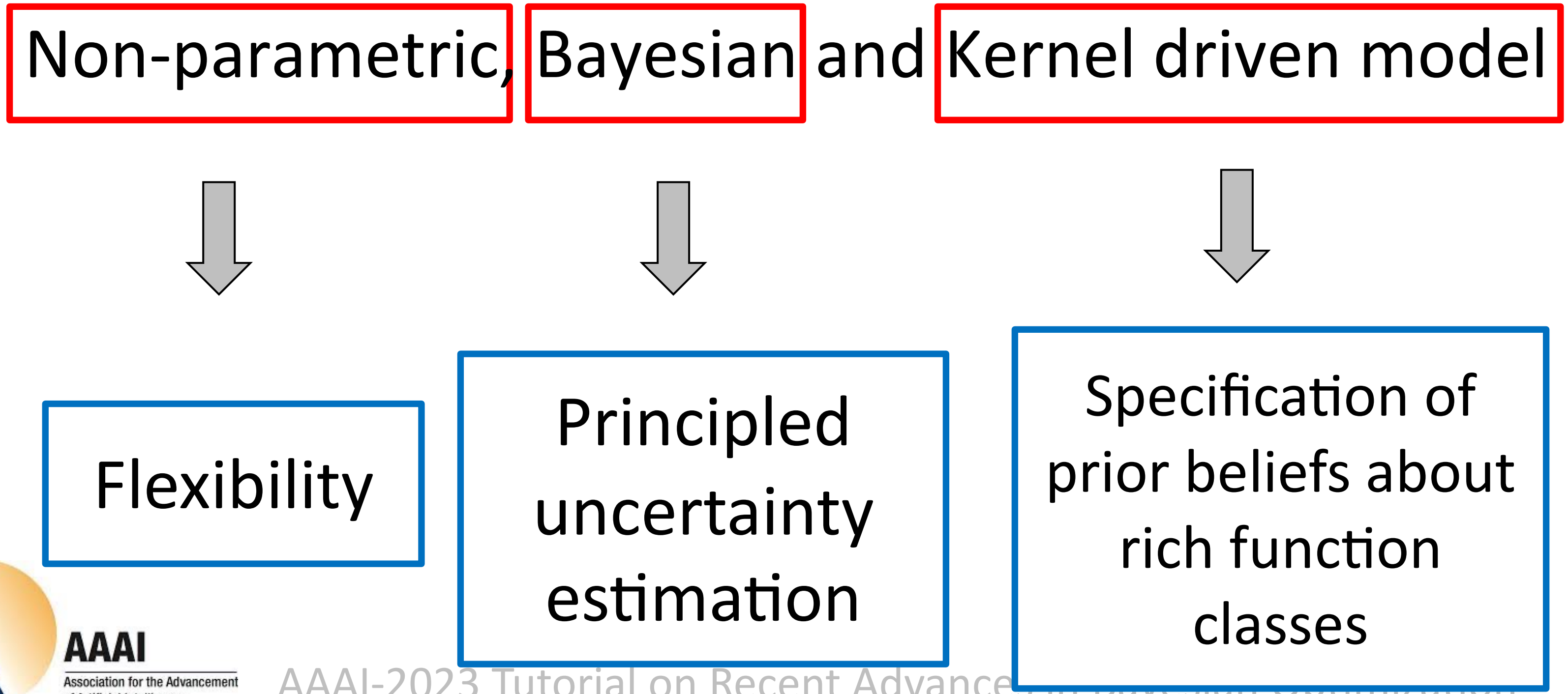

Statistical model M
Acquisition function optimization
$x_{next} = arg \max_{x \in X} AF(M, x)$
Expensive function evaluation
$f(x_{next})$
x_{next}

- Statistical model (e.g., Gaussian process)
- Acquisition function (e.g., Expected improvement)
- Acquisition function optimizer (e.g., local search)

BO needs a Probabilistic Model

- To make predictions on unknown input
- To quantify the uncertainty in predictions

- One popular class of such models are Gaussian Processes (also called GPs)



Non-parametric, Bayesian and Kernel driven model

Flexibility

Principled uncertainty estimation

Specification of prior beliefs about rich function classes

Gaussian Process

- **Stochastic process definition**
  - Given any set of input points $\{x_1, x_2, \ldots, x_m\}$, the output values follows a multi-variate Gaussian distribution

$$[f(x_1), f(x_2), f(x_3), \ldots, f(x_m)] \sim \mathcal{N}(0, \Sigma)$$

- The covariance matrix Σ is given by a kernel function $k(x, x')$, i.e., $\Sigma_{ij} = k\left(x_i, x_j\right)$
  - Kernel captures the similarity between x and x'
- Choice of kernel $k(x, x')$ is critical for good performance
  - Allows to incorporate domain knowledge (e.g., Morgan fingerprints in chemistry)
  - Matern kernel is a popular choice for continuous spaces

Gaussian Process: inference and training

- Given training data $\{(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots\ (x_m, y_m)\}$, the prediction for an unseen point x'

$$\boxed{f(x') \sim \mathcal{N}(y', \sigma')}$$

$$\boxed{\begin{gathered} y' = \boldsymbol{k}^* \boldsymbol{K}^{-1} \boldsymbol{y} \\ \sigma' = k(x', x') - \boldsymbol{k}^* K^{-1}\ k^* \end{gathered}}$$

$$\boxed{\begin{aligned} \boldsymbol{k}^* &= [k(x', x_1), k(x', x_2), \ldots, k(x', x_m)] \\ \boldsymbol{K_{ij}} &= k(x_i, x_j) \end{aligned}}$$

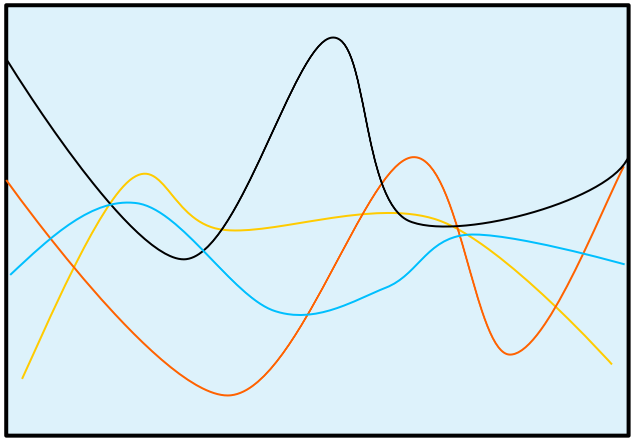
- Training procedure: searching for hyper-parameters by optimizing the marginal log-likelihood

$$\boxed{\log p(Y) = -\frac{1}{2} Y^T \boldsymbol{K}^{-1} Y - \frac{1}{2} \log \det(\boldsymbol{K}) - \frac{n}{2} \log 2\pi}$$

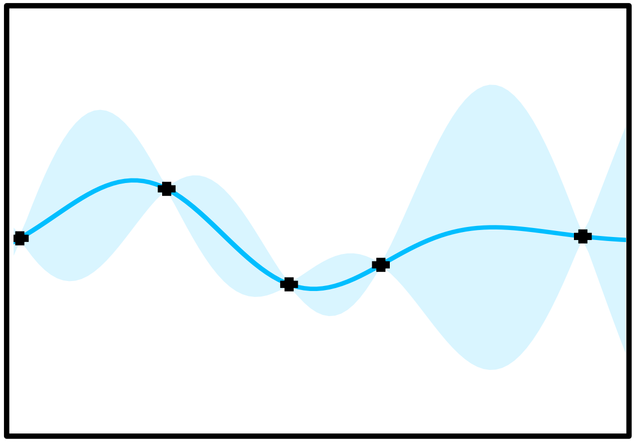
Gaussian Process: Two Views

- Function space view: distribution over functions
  - Function class is characterized by kernel

Prior



Posterior



- Weight space view: Bayesian linear regression in kernel's feature space

$$f(x) = w^T \tau(x)$$

$$k(x, x') = < \tau(x), \tau(x') >$$

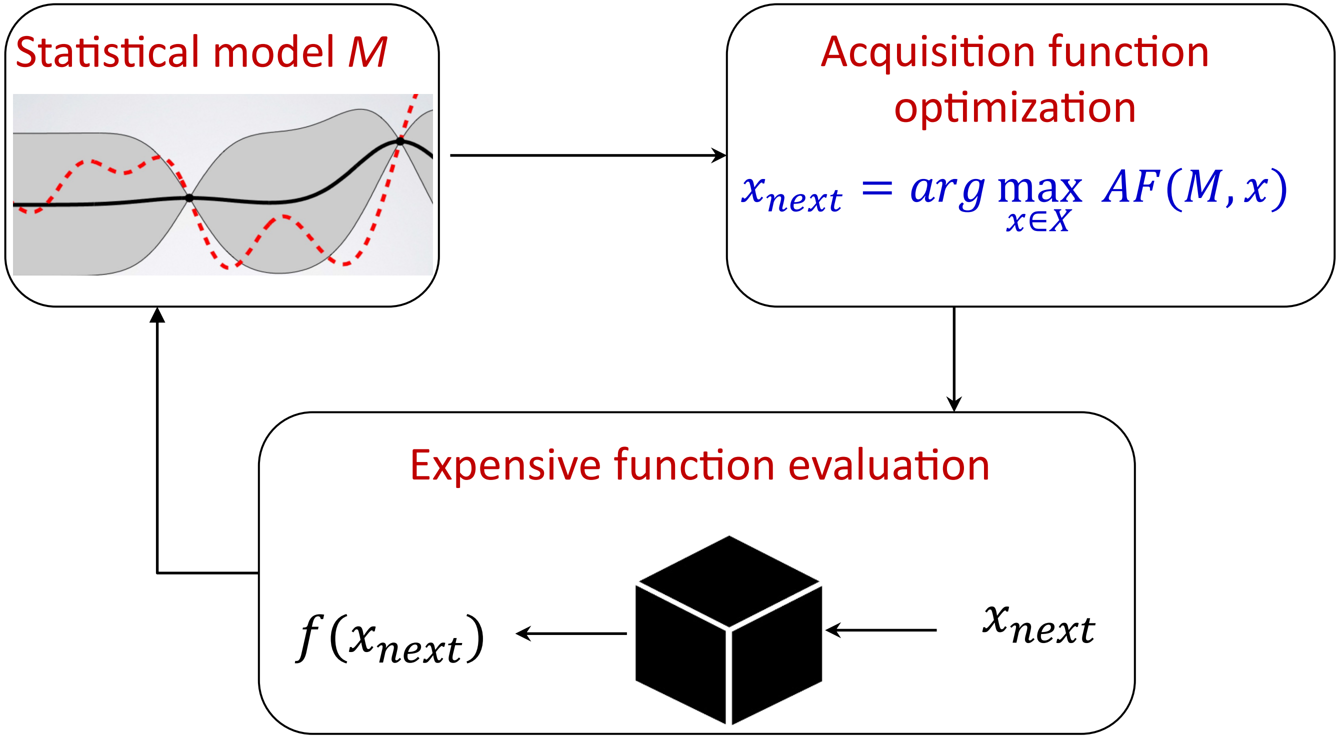
Alternative Surrogate Model Choices

- Random forest [Hutter et al., 2010, 2011]

- Bayesian neural networks

- Classification models
  - BORE [Tiao et al., 2021] and LFBO [Song et al., 2022]
  - Can work with any input space (continuous, discrete, hybrid)

Bayesian Optimization: Three Key Elements



Statistical model *M*

Acquisition function optimization

$$x_{next} = arg \max_{x \in X} AF(M, x)$$

Expensive function evaluation

$f(x_{next})$

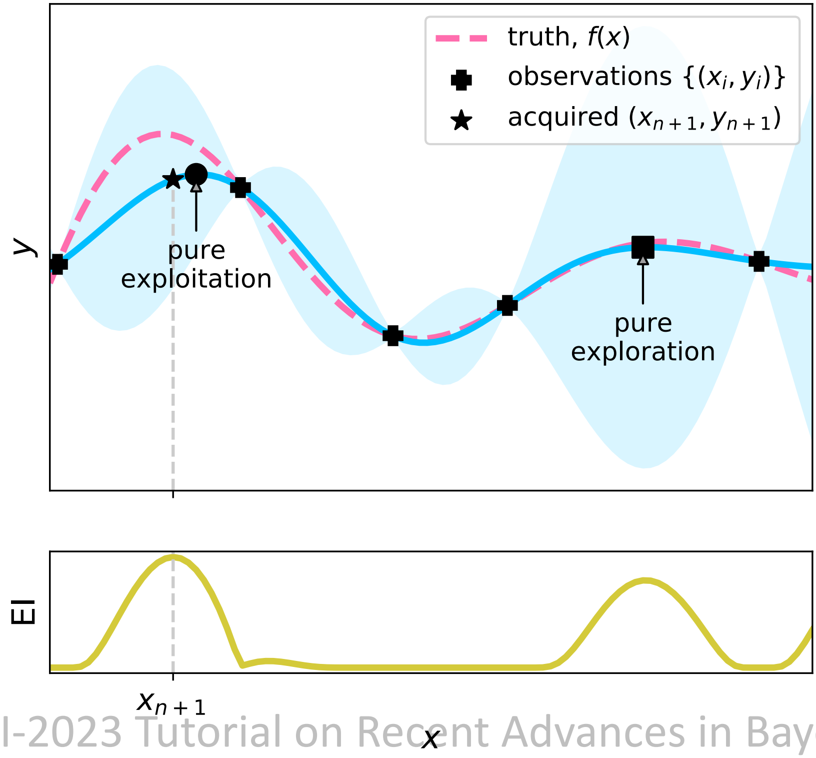
x_{next}

- Statistical model (e.g., Gaussian process)
- Acquisition function (e.g., Expected improvement)
- Acquisition function optimizer (e.g., local search)

Acquisition Function

- Intuition: captures utility of evaluating an input

- Challenge: trade-off exploration and exploitation
  - Exploration: seek inputs with high variance
  - Exploitation: seek inputs with high mean



Acquisition Function: Examples

- Upper Confidence Bound (UCB)
  - Selects input that maximizes upper confidence bound

$$AF(x) = y^*(x) + \beta\, \sigma^*(x)$$

- Expected Improvement (EI)
  - Selects input with highest expected improvement over the incumbent
- Thompson Sampling (TS)
  - Selects optimizer of a function sampled from the surrogate model's posterior
- Knowledge Gradient

Information-Theoretic Acquisition Functions

- Key principle: select inputs for evaluation which provide maximum information about the optimum

- Concretely, pick observations which quickly decrease the entropy of distribution over the optimum

$$AF(x) = \text{ Expected decrease in entropy}$$
$$AF(x) = H(\alpha \mid D) - E_y[H(\alpha|D \cup \{x, y\}$$
$$= \text{Information Gain}(\alpha; y)$$

- Design choices of α leads to different algorithms

Information-Theoretic Acquisition Functions

- Design choices of α leads to different algorithms

$$AF(x) = \text{Expected decrease in entropy}$$

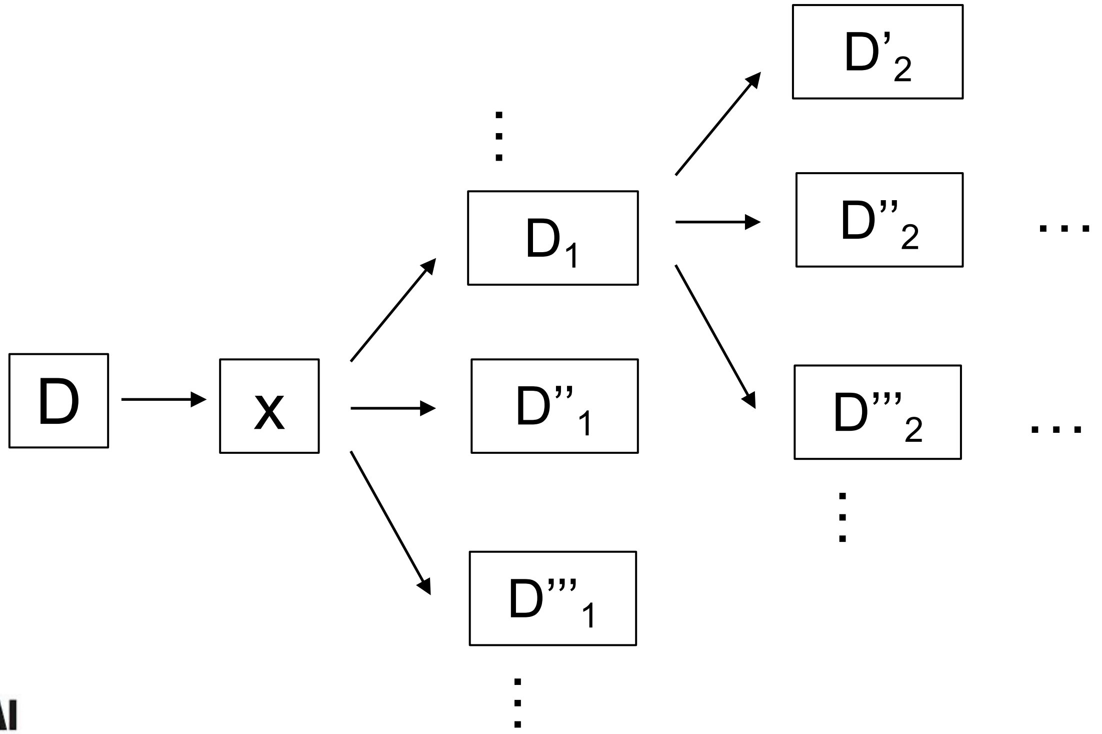
$$AF(x) = H(\alpha \mid D) - E_y[H(\alpha | D \cup \{x, y\}$$

$$= \text{Information Gain}(\alpha; y)$$

- α as input location of optima x^*
  - Entropy Search (ES) / Predictive Entropy Search (PES)
  - Intuitive but requires expensive approximations
- α as output value of optima y^*
  - Max-value Entropy Search (MES) and it's variants
  - Computationally cheaper and more robust
- Generalization via decision-theoretic entropies [Neiswanger et al., 2022]

Non-Myopic / Lookahead Acquisition Functions

- Myopic acquisition functions (e.g., EI) reason about immediate utility
- Non-myopic variants consider BO as a MDP and reason about longer decision horizons



EI and KG as One-step Lookahead

- **Idea**: Expected marginal gain in some utility $u(D)$

$$\boxed{AF = \int [u(D') - u(D)]p(y|x, D)dy}$$

- For Expected Improvement (EI), utility is the simple reward
  - $u(D) = \max \mu_D(x)$
  - maximum picked from points evaluated during BO search
  - Closed-form expression

- For Knowledge Gradient (KG), utility is the global reward
  - $u(D) = max_{x \in X}\, \mu_D(x)$
  - maximum picked from the entire search space X

Non-Myopic / Lookahead Acquisition Functions

- Non-myopic variants consider BO as MDP and reason about longer decision horizons

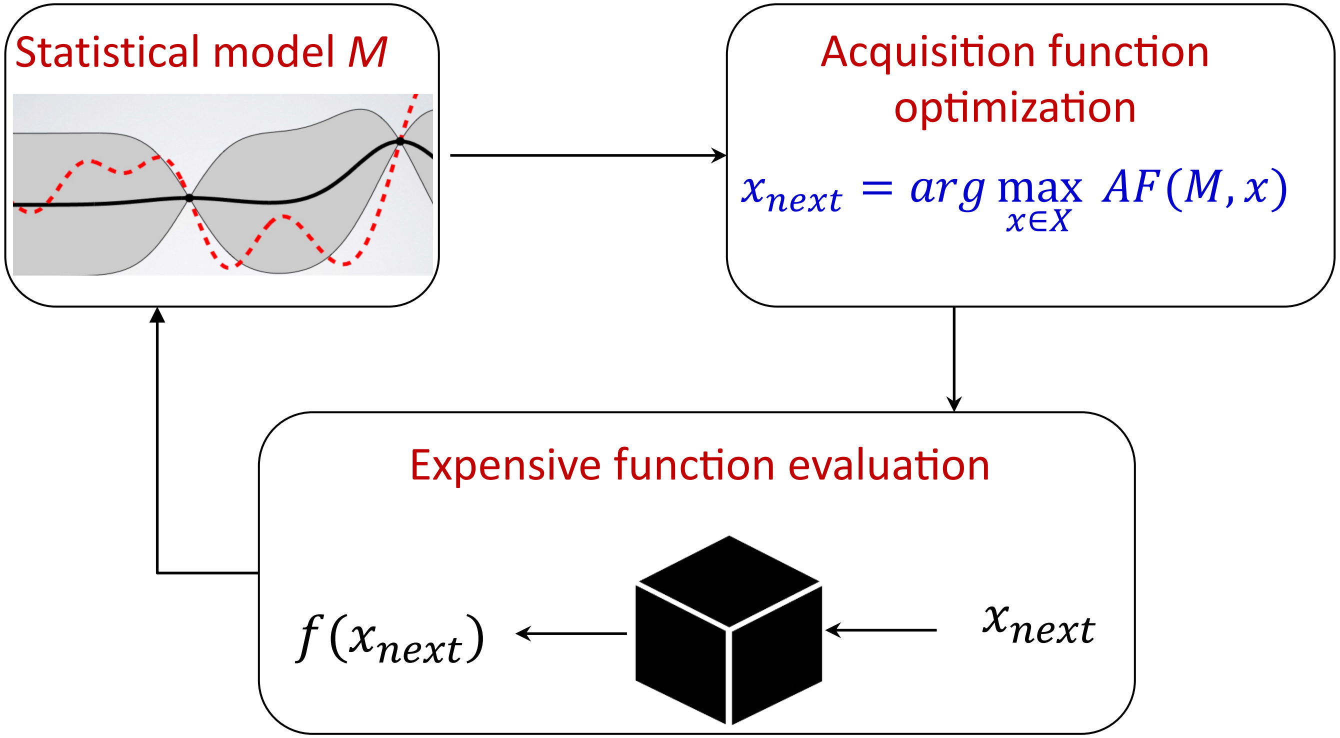
$$u_k(x|D) = u_1(x|D) + E_y\left[\max_{x'} u_{t-1}(x'|D \cup \{x,y\})\right]$$

- Challenge: curse of dimensionality

$$u_k(x|D) = u_1(x|D) + E_y\left[\max_{x1}\{u(x_1|D_1) + E_{y1}[\max_{x2}\{u(x_2|D_2) \dots .\}]\}\right]$$

- Some solutions [Lam et al., 2016, Lee et al., 2020, Gonzalez et al 2016, Jiang et al 2020]
  - Multi-step lookahead policies with approximations
  - Rollout based approximate dynamic programming

Bayesian Optimization: Three Key Elements



Statistical model *M*

Acquisition function optimization

$x_{next} = arg \max_{x \in X} AF(M, x)$

Expensive function evaluation

$f(x_{next})$

x_{next}

- Statistical model (e.g., Gaussian process)
- Acquisition function (e.g., Expected improvement)
- Acquisition function optimizer (e.g., local search)

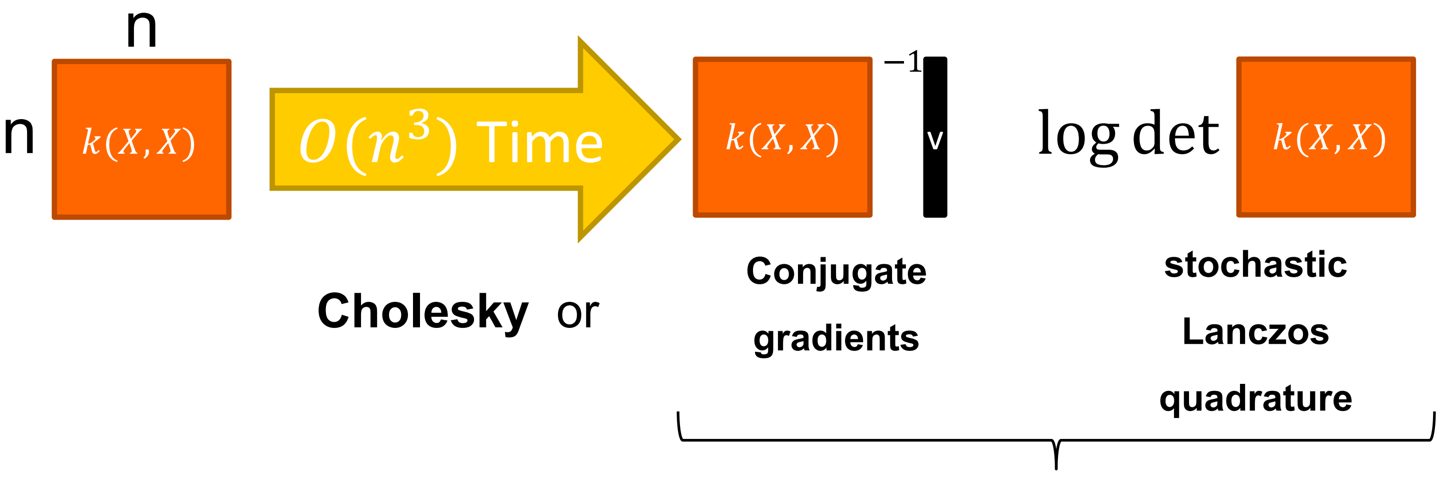
Acquisition Function Optimizer

- Challenge: non-convex/multi-modal optimization problem

- Commonly used approaches for continuous spaces
  - Space partitioning methods (e.g., DIRECT, LOGO)
  - Gradient based methods (e.g., Gradient descent)
  - Evolutionary search (e.g., CMA-ES)

Scaling GPs for Bayesian Optimization

GPs require:



n

n $k(X,X)$

$O(n^3)$ Time

$k(X,X)^{-1}$ v

$\log\det$ $k(X,X)$

Up to 10,000s:

(More is possible, but BayesOpt requires training **many** GPs)

Cholesky or

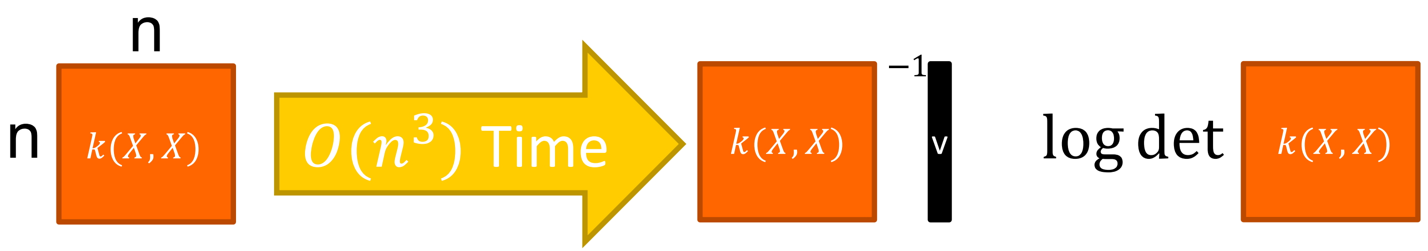
Conjugate gradients

stochastic Lanczos quadrature

Approximate **iterative methods** (Cutajar et al., 2016; Gardner et al., 2018; Wang et al., 2019; Burt et al., 2021; Wenger et al., 2022)

Scaling GPs for Bayesian Optimization

GPs require:



n $k(X,X)$ n $\xrightarrow{O(n^3)\text{ Time}}$ $k(X,X)^{-1}$ v $\log\det k(X,X)$

Up to 10,000s:

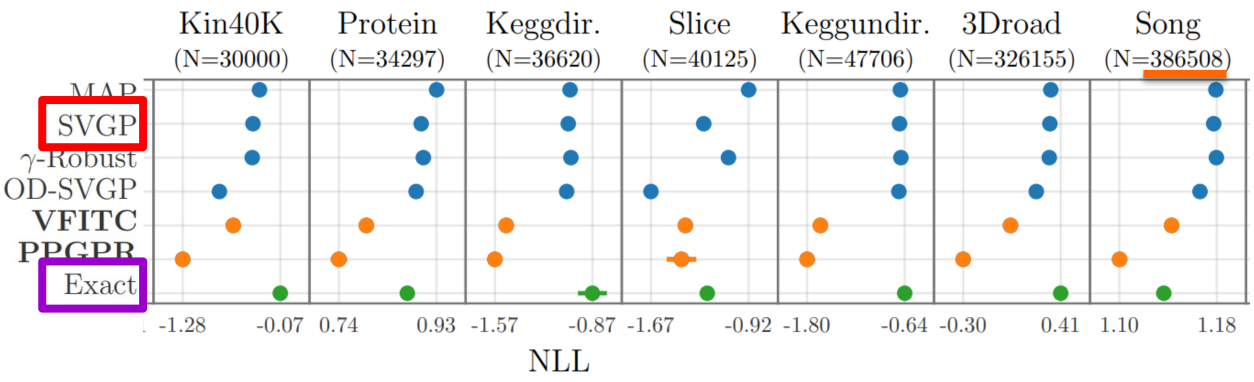
(More is possible, but BayesOpt requires training **many** GPs)

Cholesky or **Conjugate gradients** / **stochastic Lanczos quadrature**

Approximate **iterative methods**
(Cutajar et al., 2016; Gardner et al., 2018; Wang et al., 2019; Burt et al., 2021; Wenger et al., 2022)

More data: use **Stochastic Variational Inference**

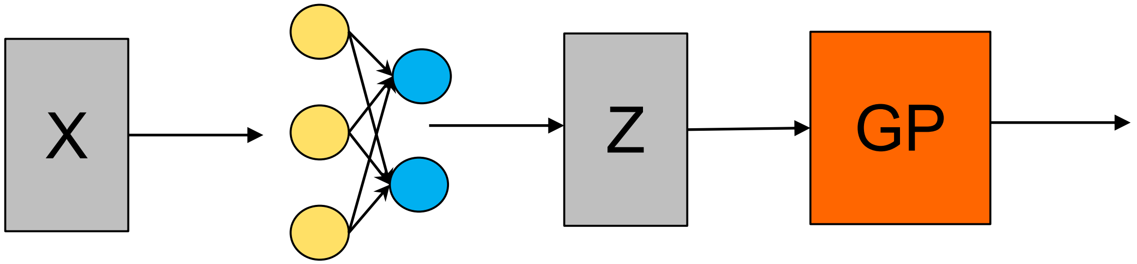
(Hensman et al., 2013; Salimbeni et al., 2018; Jankowiak et al., 2020)



(Lower is better)

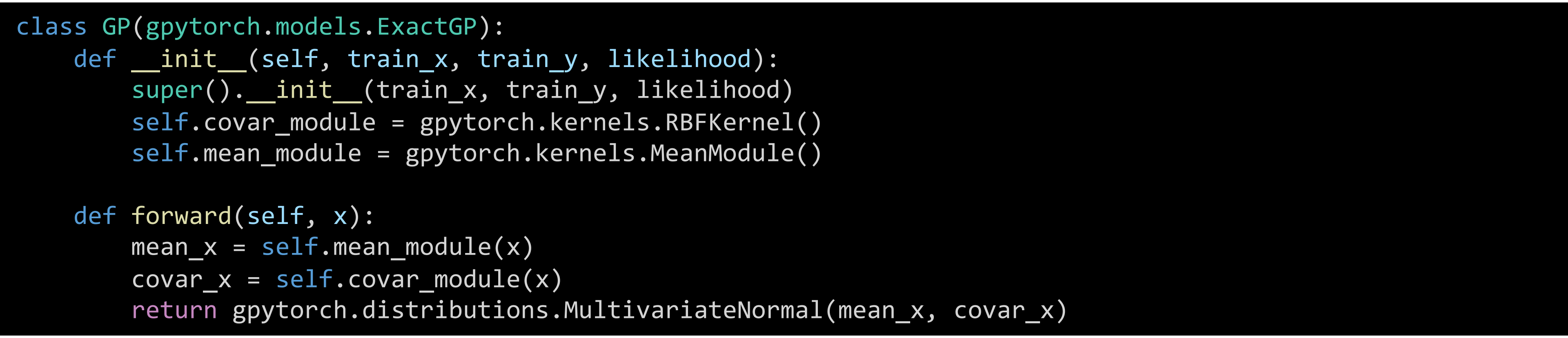
More Expressive Models for More Data

GP Model with Deep Kernel:



$\mathbb{E}[f(z)], \mathrm{Cov}[f(z), f(z')]$

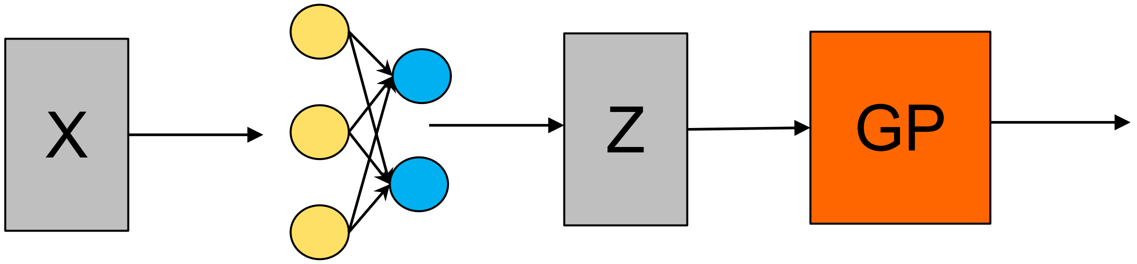
GPyTorch model implementation:



```
class GP(gpytorch.models.ExactGP):
    def __init__(self, train_x, train_y, likelihood):
        super().__init__(train_x, train_y, likelihood)
        self.covar_module = gpytorch.kernels.RBFKernel()
        self.mean_module = gpytorch.kernels.MeanModule()

    def forward(self, x):
        mean_x = self.mean_module(x)
        covar_x = self.covar_module(x)
        return gpytorch.distributions.MultivariateNormal(mean_x, covar_x)
```

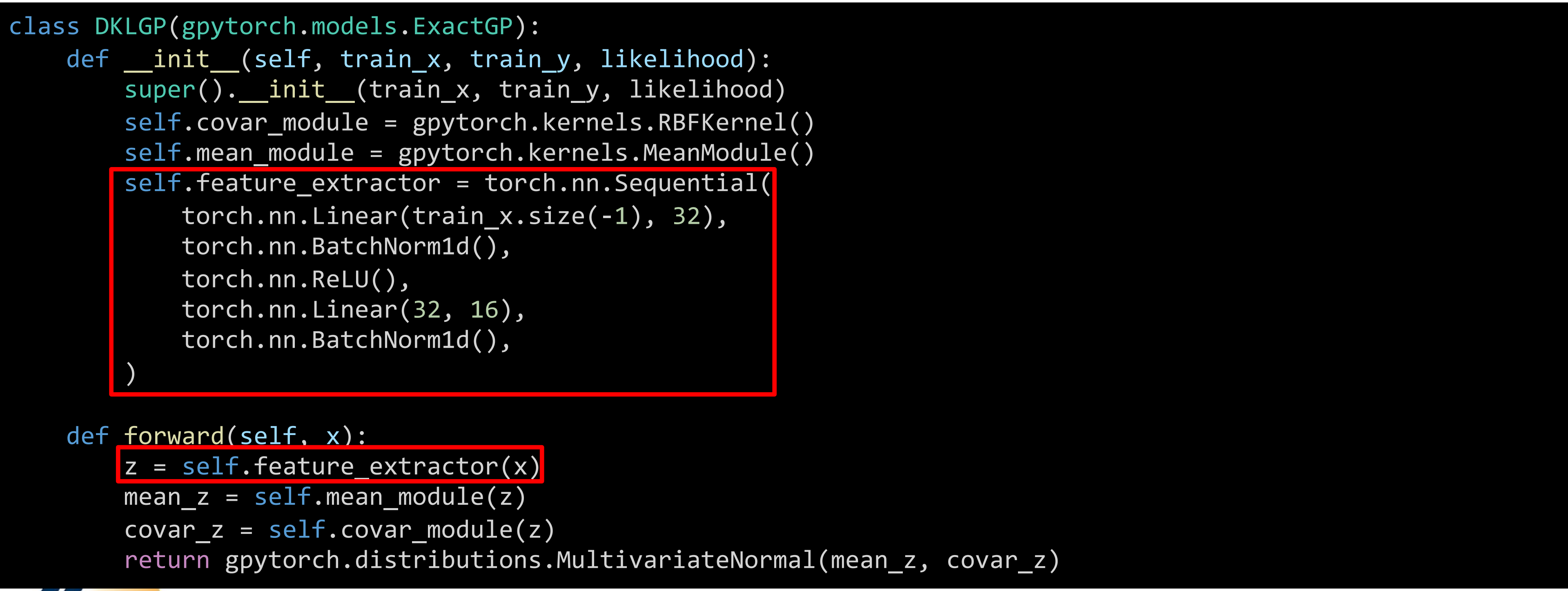
More Expressive Models for More Data

GP Model with Deep Kernel:



$\mathbb{E}[f(z)], \mathrm{Cov}[f(z), f(z')]$

GPyTorch model implementation:



```
class DKLGP(gpytorch.models.ExactGP):
    def __init__(self, train_x, train_y, likelihood):
        super().__init__(train_x, train_y, likelihood)
        self.covar_module = gpytorch.kernels.RBFKernel()
        self.mean_module = gpytorch.kernels.MeanModule()
        self.feature_extractor = torch.nn.Sequential(
            torch.nn.Linear(train_x.size(-1), 32),
            torch.nn.BatchNorm1d(),
            torch.nn.ReLU(),
            torch.nn.Linear(32, 16),
            torch.nn.BatchNorm1d(),
        )

    def forward(self, x):
        z = self.feature_extractor(x)
        mean_z = self.mean_module(z)
        covar_z = self.covar_module(z)
        return gpytorch.distributions.MultivariateNormal(mean_z, covar_z)
```

BO Software: BoTorch

- Scalability via automatic differentiation
  - PyTorch/GpyTorch
- Monte-Carlo acquisition functions
  - Express acquisition functions as expectations of utility functions
  - Compute expectations via Monte-Carlo sampling
  - Use the reparameterization trick to make acquisition functions differentiable
- Other software: Trieste (based on TensorFlow)
- Not actively maintained: GPyOpt, Spearmint

Outline of the Tutorial

- Overview of the BO Framework, GPs, advances in GPs and acquisition functions, and BoTorch demo

- Bayesian Optimization over Discrete/Hybrid Spaces

- Multi-fidelity Bayesian Optimization

30 mins Break

- High-Dimensional BO and BoTorch Hands-on demo

- Multi-Objective BO and BoTorch Hands-on demo

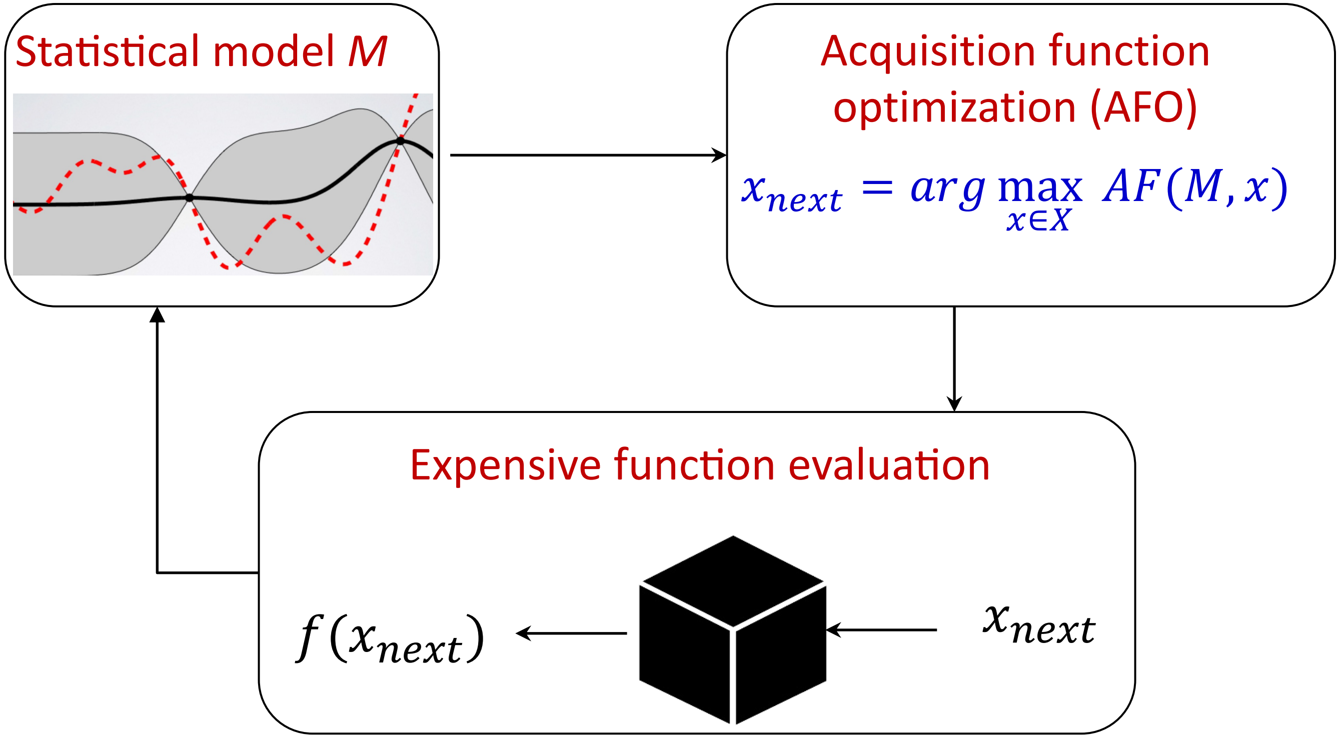
- Summary and Outstanding Challenges in BO

Combinatorial/Discrete BO

- **Given:** a combinatorial space of structures X (e.g., sequences, trees, graphs)
- **Find:** optimized combinatorial structure $x^* \in X$

- **Space of binary structures** $X = \{0,1\}^n$
  - Each structure $x \in X$ be represented using n binary variables $x_1, x_2, \ldots, x_n$
- **Categorical variables**
  - x_i can take more than two candidate values
- **How to deal with categorical variables?**
  - Option 1: Encode them as binary variables (a common practice)
  - Option 2: Modeling and reasoning over categorical variables

Discrete BO: Technical Challenges


Statistical model *M*

Acquisition function optimization (AFO)

$$x_{next} = arg \max_{x \in X} AF(M, x)$$

Expensive function evaluation

$f(x_{next})$ ← ← x_{next}

- Effective modeling over combinatorial structures (e.g., sequences, graphs)
- Solving hard combinatorial optimization problem to select next structure
  - Not an issue if we are searching over a fixed database of structures

Discrete BO: Summary of Approaches

- Trade-off complexity of model and tractability of AFO

- Simple statistical models and tractable search for AFO
  - BOCS [Baptista et al., 2018]

- Complex statistical models and heuristic search for AFO
  - SMAC [Hutter et al., 2011] and COMBO [Oh et al., 2019]

- Complex statistical models and tractable/accurate AFO
  - L2S-DISCO [Deshwal et al., 2020] and MerCBO [Deshwal et al., 2021]
  - Reduction to continuous BO [Gómez-Bombarelli et al., 2018]…

Aside: Discrete BO vs. Structured Prediction

- **Structured prediction (SP)** [Lafferty et al., 2001] [Bakir et al., 2007]
  - Generalization of classification to structured outputs (e.g., sequences, trees, and graphs)
  - CRFs, Structured Perceptron, Structured SVM

- Complexity of cost function vs. tractability of inference
  - Simple cost functions (e.g., first-order) and tractable inference
  - Complex cost functions (e.g., higher-order) and heuristic inference
  - Learning to search for SP [Daume' et al., 2009] [Doppa et al., 2014]

Key Difference:

Small data vs. big data setting

Surrogate Models: BOCS [Baptista et al., 2018]

- Linear surrogate model over binary structures
  - $f(x \in X) = \theta^T . \phi(x)$
  - $\phi(x)$ consists of up to Quadratic (second-order) terms
  - $\phi(x) = [x_1, x_2, \dots, x_d, x_1 . x_2, x_1 . x_3, \dots, x_{d-1} . x_d]$

May not be sufficient to capture desired dependencies

Surrogate Models: SMAC [Hutter et al., 2010, 2011]

- Random forest as surrogate model
  - works naturally for categorical and mixed variables
  - Prediction/Uncertainty (= empirical mean/variance over trees)

Uncertainty estimates can be poor

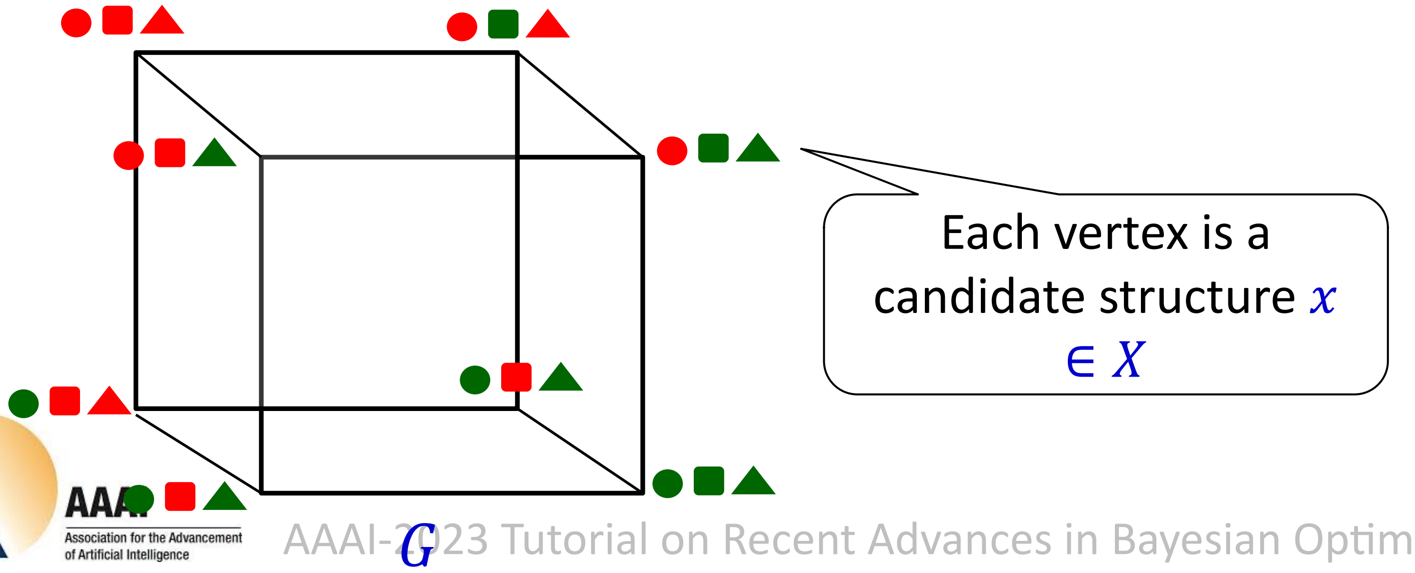
- Improvements for better uncertainty estimates
  - Bagging with oversampling [Kim and Choi, 2022]

Surrogate Models: COMBO [Oh et al., 2019]

- GP with diffusion kernel [Kondor and Lafferty 2002]
  - Requires a graph representation of the input space X

$$\boxed{K(V,V) = exp(-\beta L(G))}$$

- **Combinatorial graph representation** [Oh et al., 2019]

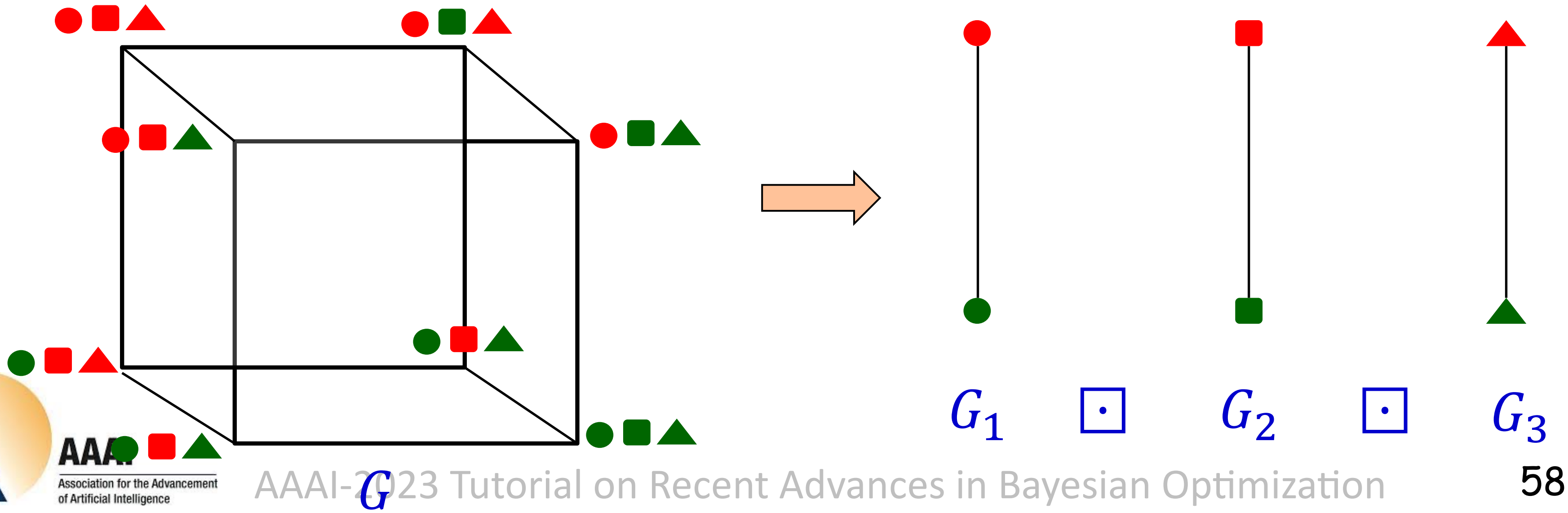


Surrogate Models: COMBO [Oh et al., 2019]

- GP with diffusion kernel [Kondor and Lafferty 2002]
  - Requires a graph representation of the input space X

$$K(V,V) = exp(-\beta L(G))$$

- **Combinatorial graph representation** [Oh et al., 2019]



Surrogate Models: GP via Structured Kernels

- Leverage prior work on kernels over structured data [Gartner, 2003] to build GP surrogate models

- Some examples from BO literatures
  - String kernels [Moss et al., 2020]
  - Kernels for protein design [Gruver et al., 2021]
  - Permutation kernels [Deshwal et al., 2022; Oh et al., 2021]
  - Weisfeiler-Lehman kernels for NAS [Ru et al., 2021]

Acquisition Functions

- Can use acquisition functions that don’t require sampling from the posterior model
  - EI and UCB

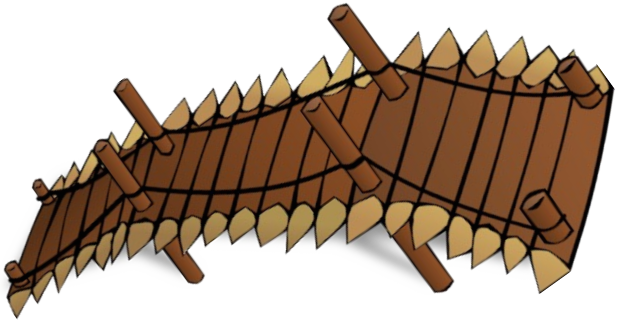
- Many advanced acquisition functions require sampling functions from posterior
  - Thompson sampling
  - Predictive entropy search
  - Max-value entropy search
  - …

Acquisition Function: Mercer Features [Deshwal et al., 2021]

- Mercer features allow sampling functions from GP posterior

- Missing puzzle to leverage prior acquisition functions
  - Thompson Sampling (TS)
  - Predictive Entropy Search (PES)
  - Max-value Entropy Search (MES)
  - …

BO for continuous spaces



BO for discrete spaces

Acquisition Function: Mercer Features [Deshwal et al., 2021]

- COMBO surrogate model: GP with discrete diffusion kernel and graph representation G

- **Key Idea:** exploit the structure of combinatorial graph G to compute its eigenspace in closed-form

- **Eigenvalue set:** $\{0, 2, \ldots, 2n\}$
  - j^{th} eigenvalue occurs with $\binom{n}{j}$ multiplicity

- **Eigenvector set:** Hadamard matrix (H) of order 2^n

$$H_{ij} = (-1)^{\langle r_i, r_j \rangle}$$

Acquisition Function: Mercer Features [Deshwal et al., 2021]

$$K(x_1, x_2) = \sum_{i=0}^{2^n-1} e^{-\beta\lambda_i} \; -1^{<r_i, x_1>} \; -1^{<r_i, x_2>}$$

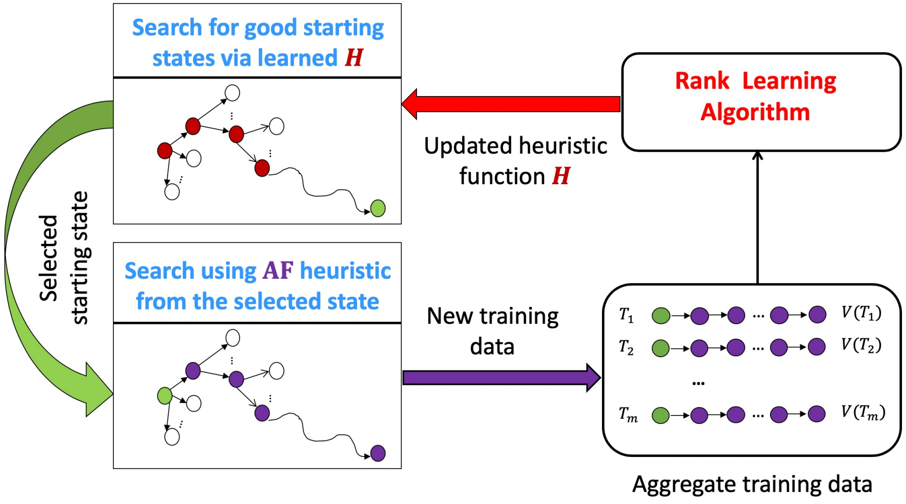
$$K(x_1, x_2) = \phi(x_1)^T \phi(x_2)$$

$$\phi(x)_i = \{\sqrt{e^{-\beta\lambda_i}} \; -1^{<r_i, x>}\}$$

j^{th} order Mercer features: first j distinct eigenvalues

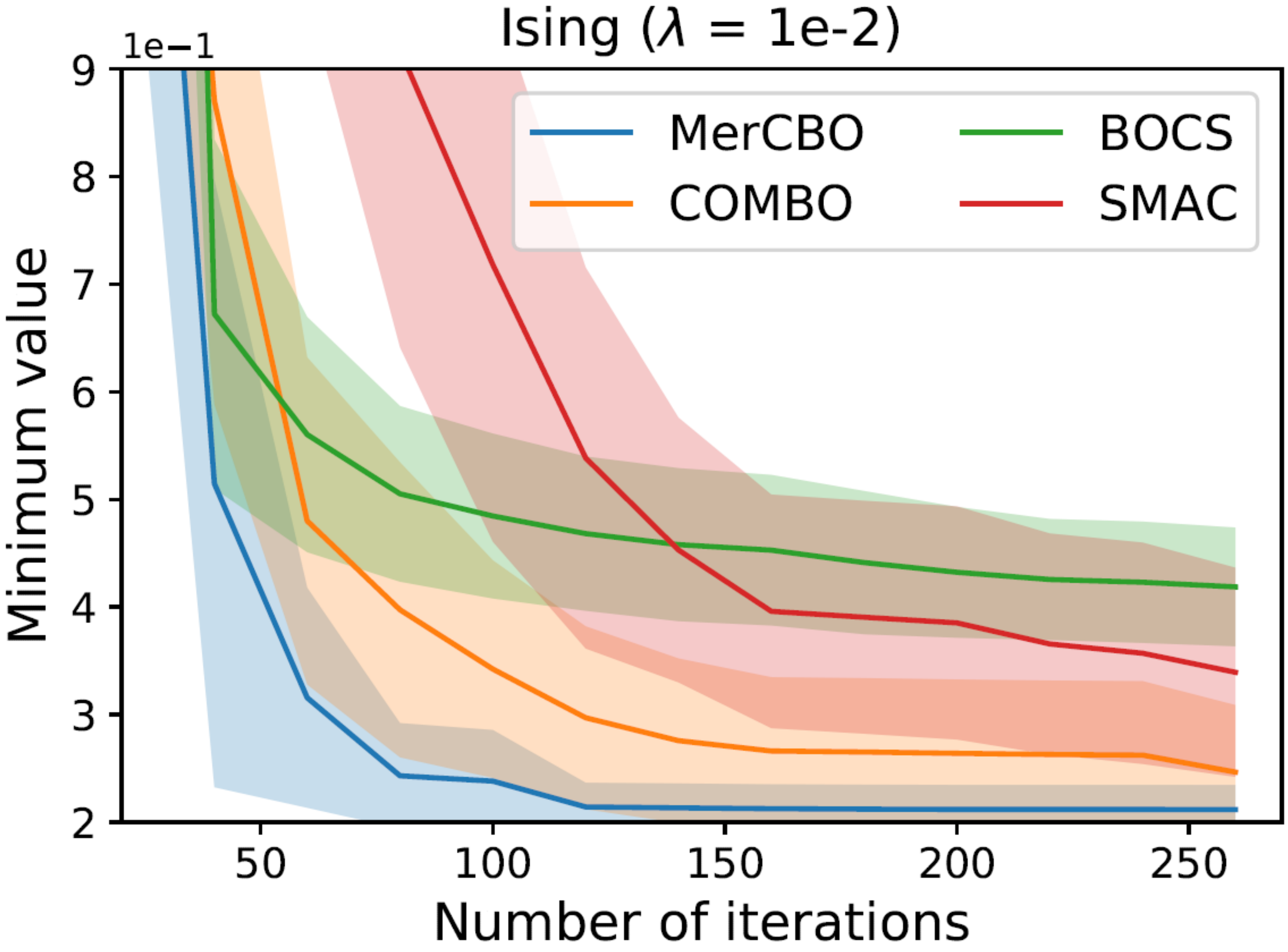
Acquisition Function Optimization (AFO)

- Tractable search for AFO via Thompson sampling
  - BOCS [Baptista et al., 2018]
  - MerCBO [Deshwal et al., 2021]
  - Neural network + MILP [Papalexopoulos et al., 2022]
- Heuristic search (local search with restarts) for AFO
  - SMAC [Hutter et al., 2011] and COMBO [Oh et al., 2019]
- Learning-to-search framework for AFO [Deshwal et al., 2020]

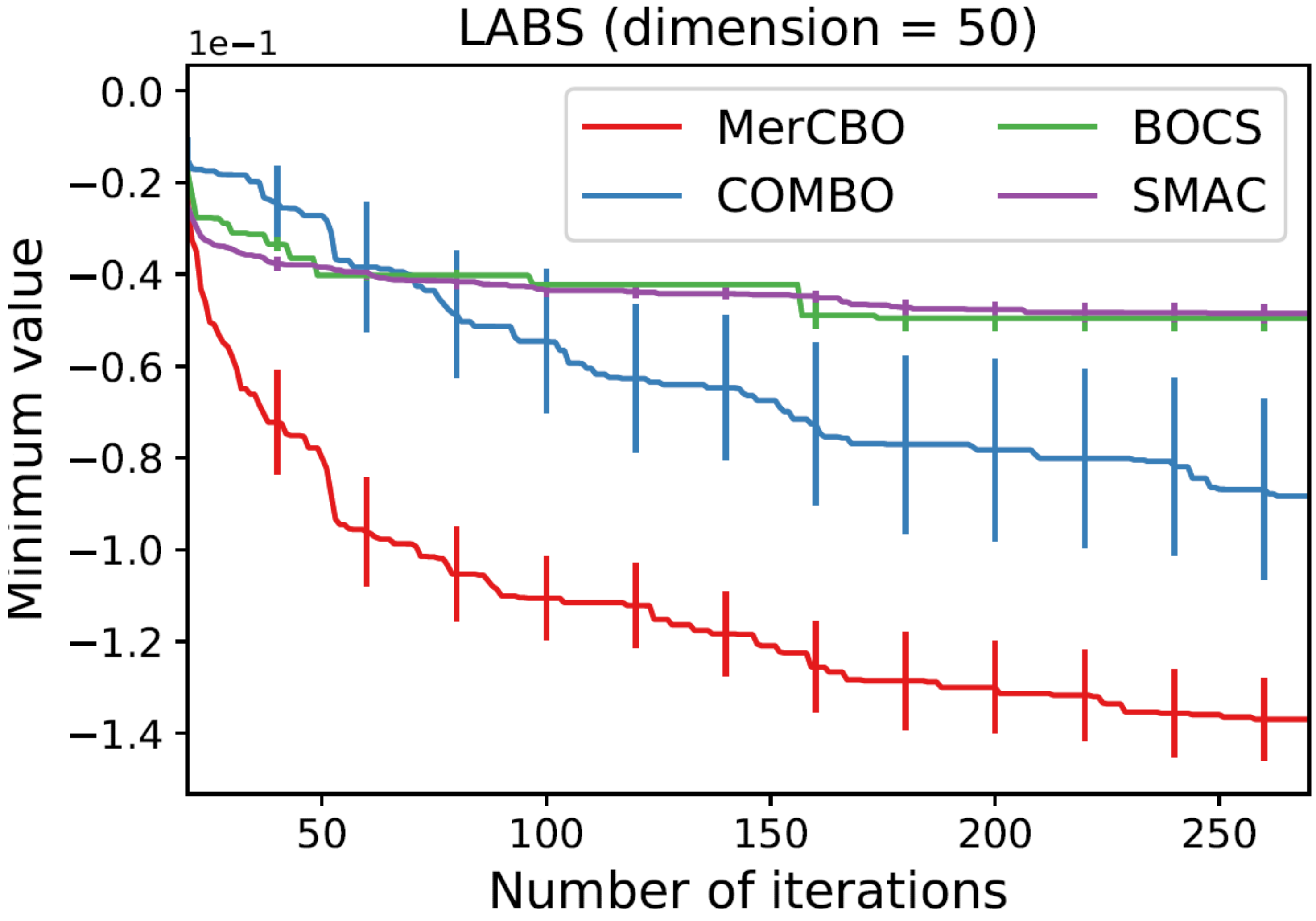


Use machine learning to improve accuracy of search

Discrete BO: Experimental Results #1



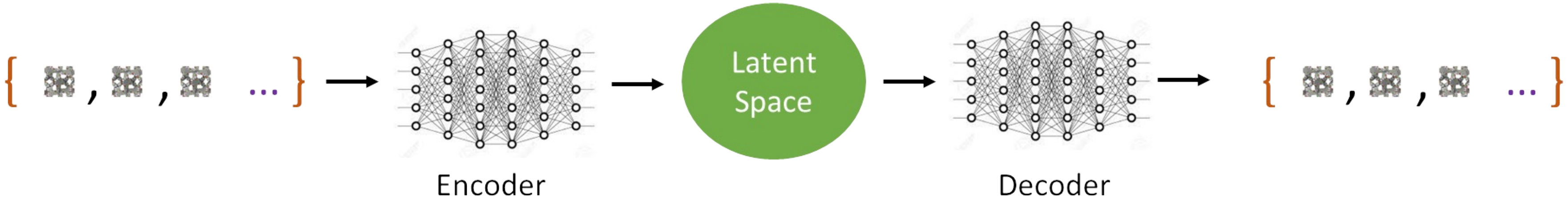
Discrete BO: Experimental Results #2



Reduction to Continuous BO [Gómez-Bombarelli et al., 2018]…

- **Key Idea:** Convert discrete space into continuous space

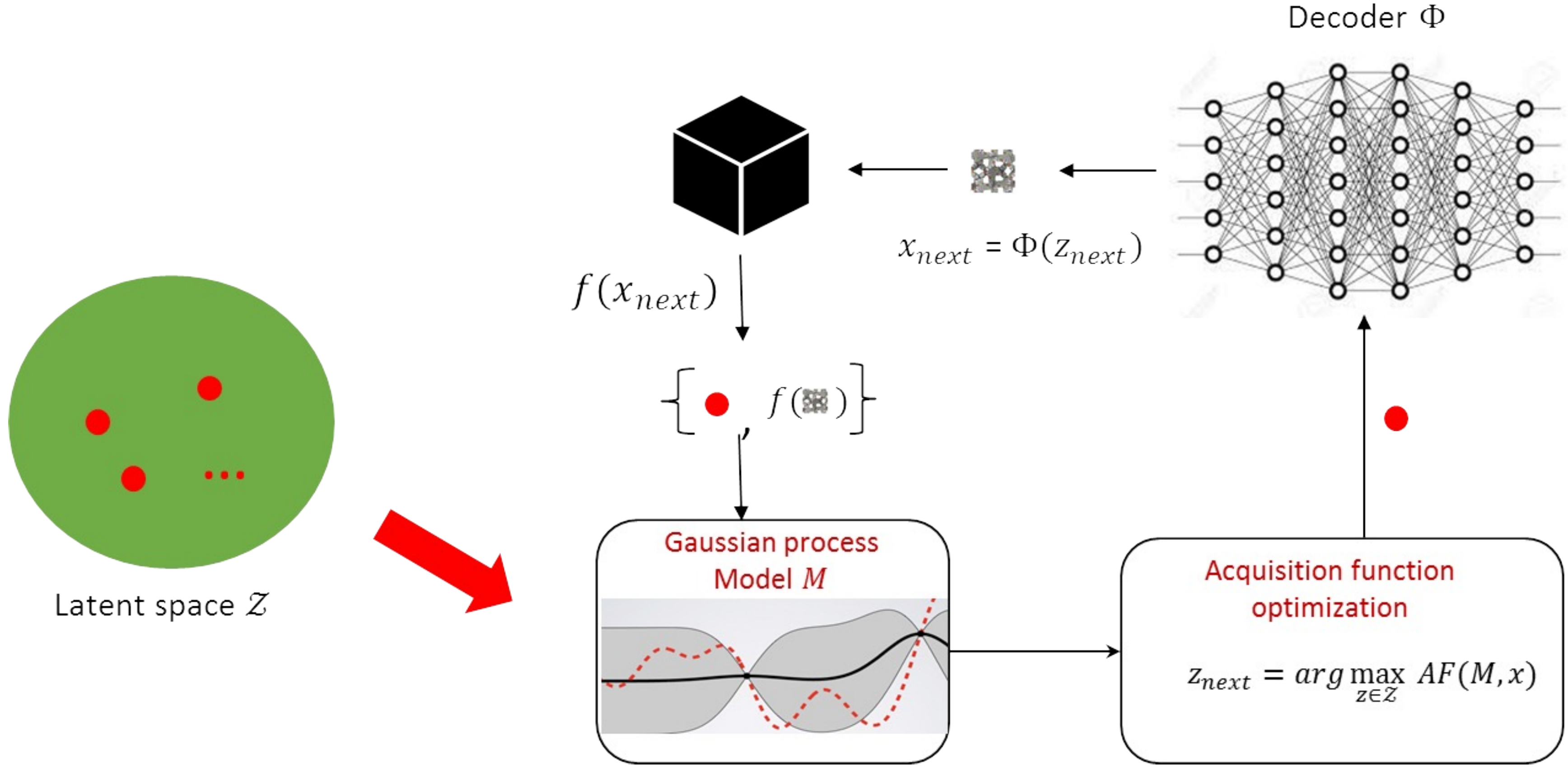
- Train a deep generative model (VAE) using unsupervised structures



- Perform BO in the learned continuous latent space
  - Surrogate modeling and acquisition function optimization in latent space (vs. combinatorial space)

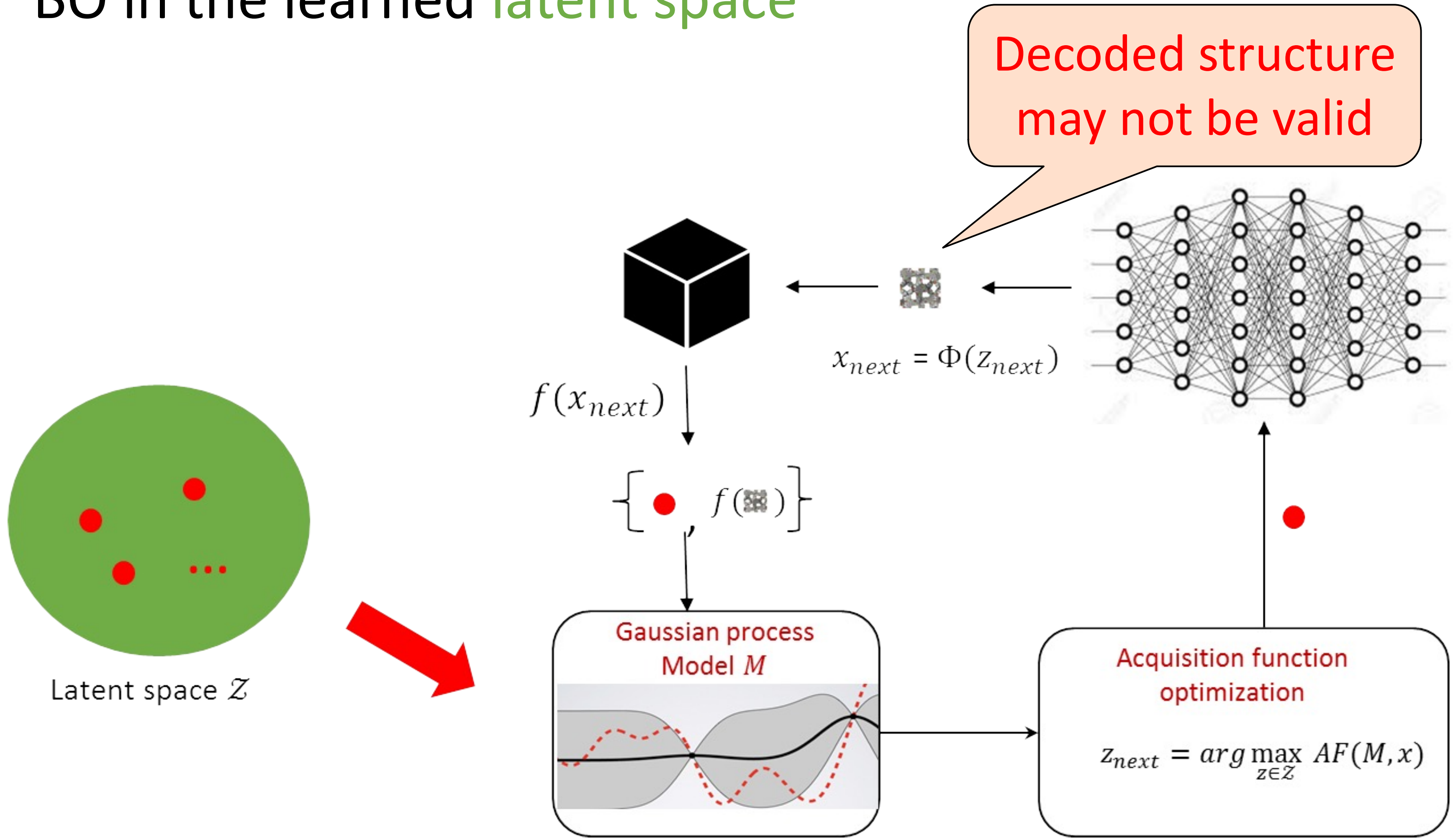
Reduction to Continuous BO [Gómez-Bombarelli et al., 2018]…

- BO in the learned latent space


Decoder Φ
$x_{next} = \Phi(z_{next})$
$f(x_{next})$
$\{ \bullet , f(\cdot) \}$
...
Latent space $\mathcal{Z}$
Gaussian process Model M
Acquisition function optimization
$z_{next} = arg \max_{z \in \mathcal{Z}} AF(M,x)$

Reduction to Continuous BO [Gómez-Bombarelli et al., 2018]…

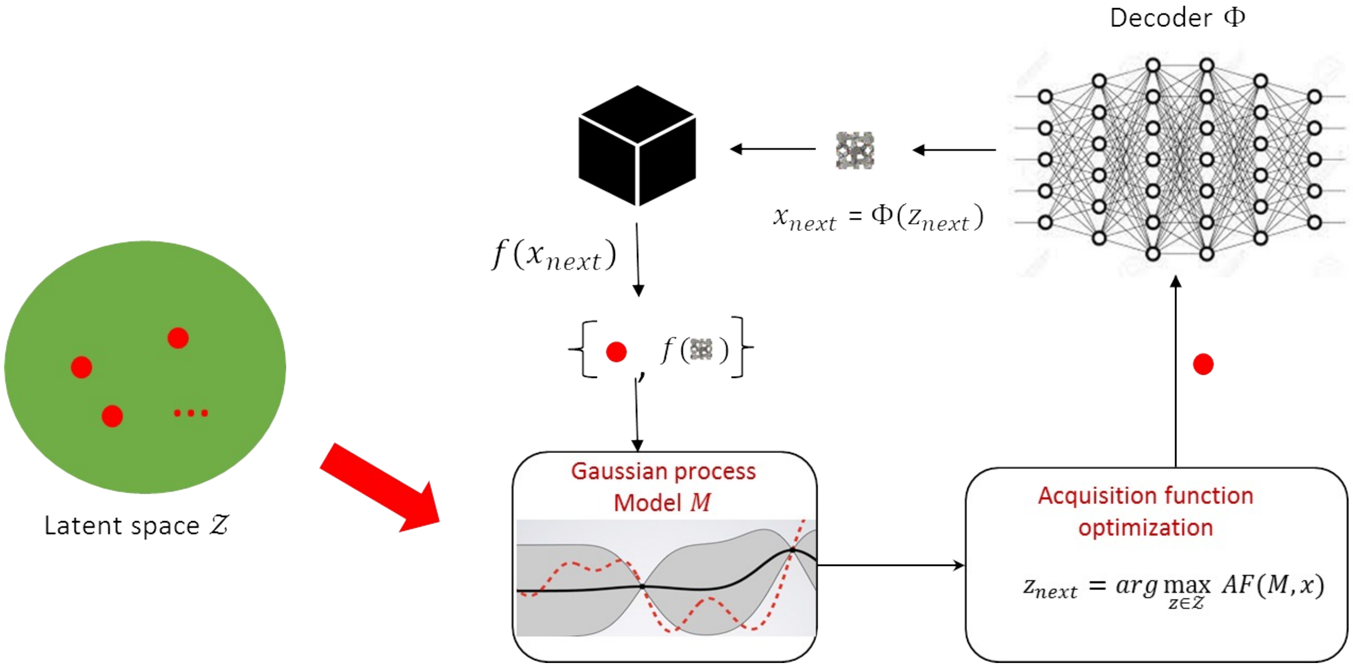
- BO in the learned latent space


Decoded structure may not be valid
$x_{next} = \Phi(z_{next})$
$f(x_{next})$
Latent space $\mathcal{Z}$
Gaussian process Model M
Acquisition function optimization
$z_{next} = arg\max_{z \in \mathcal{Z}} AF(M, x)$

Griffiths R.-R. and Hernández-Lobato J. M.: Constrained Bayesian optimization for Automatic Chemical Design Using Variational Autoencoders, Chemical Science, 2019

Reduction to Continuous BO [Gómez-Bombarelli et al., 2018]…

- BO in the learned latent space


Decoder Φ
$x_{next} = \Phi(z_{next})$
$f(x_{next})$
$\{ \bullet , f(\cdot) \}$
Latent space $\mathcal{Z}$
Gaussian process Model M
Acquisition function optimization
$z_{next} = \arg\max_{z \in \mathcal{Z}} AF(M, x)$

- Challenges
  - Doesn't (explicitly) incorporate information about decoded structures
  - Surrogate model may not generalize well for small data setting

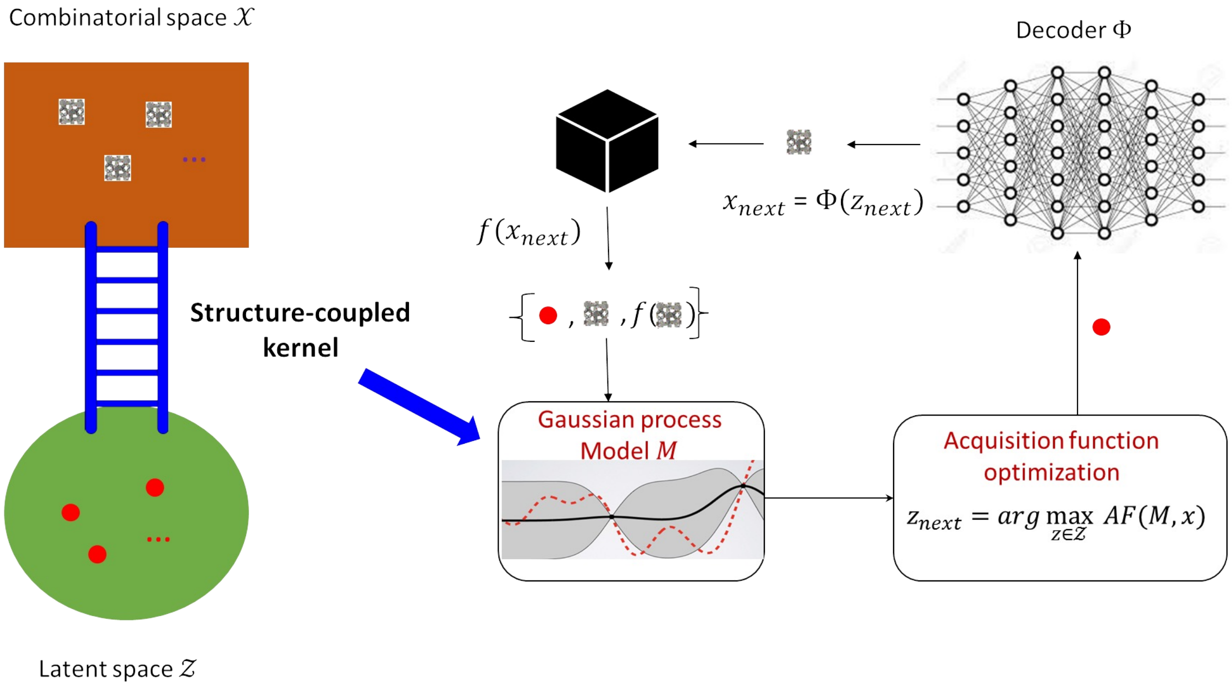
Improvements to Latent Space BO

- Weighted retraining [Tripp et al., 2020]
  - Periodically retrain the deep generative model
  - Assign importance weights to training data proportional to their objective function value

- Uncertainty-guided latent space BO [Notin et al., 2021]
  - Leverage the epistemic uncertainty of the decoder to guide the optimization process
  - No retraining of deep generative model is needed

Improvements to Latent Space BO

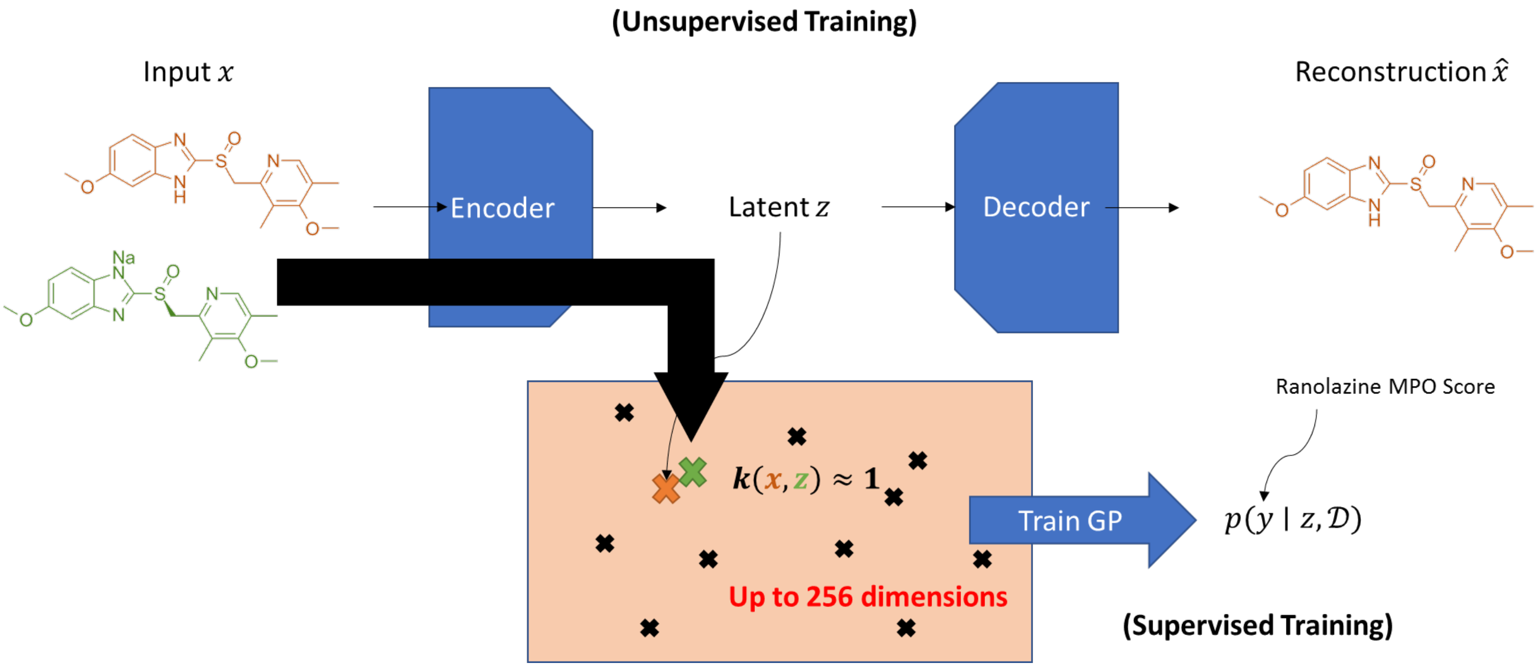
- LADDER algorithm [Deshwal and Doppa, 2021]
  - Incorporate domain knowledge via (hand-designed) structured kernels


Combinatorial space $\mathcal{X}$
Decoder Φ
$x_{next} = \Phi(z_{next})$
$f(x_{next})$
$f($ $)$
Structure-coupled kernel
Gaussian process Model M
Acquisition function optimization
$z_{next} = arg \max_{z \in Z} AF(M, x)$
Latent space $\mathcal{Z}$

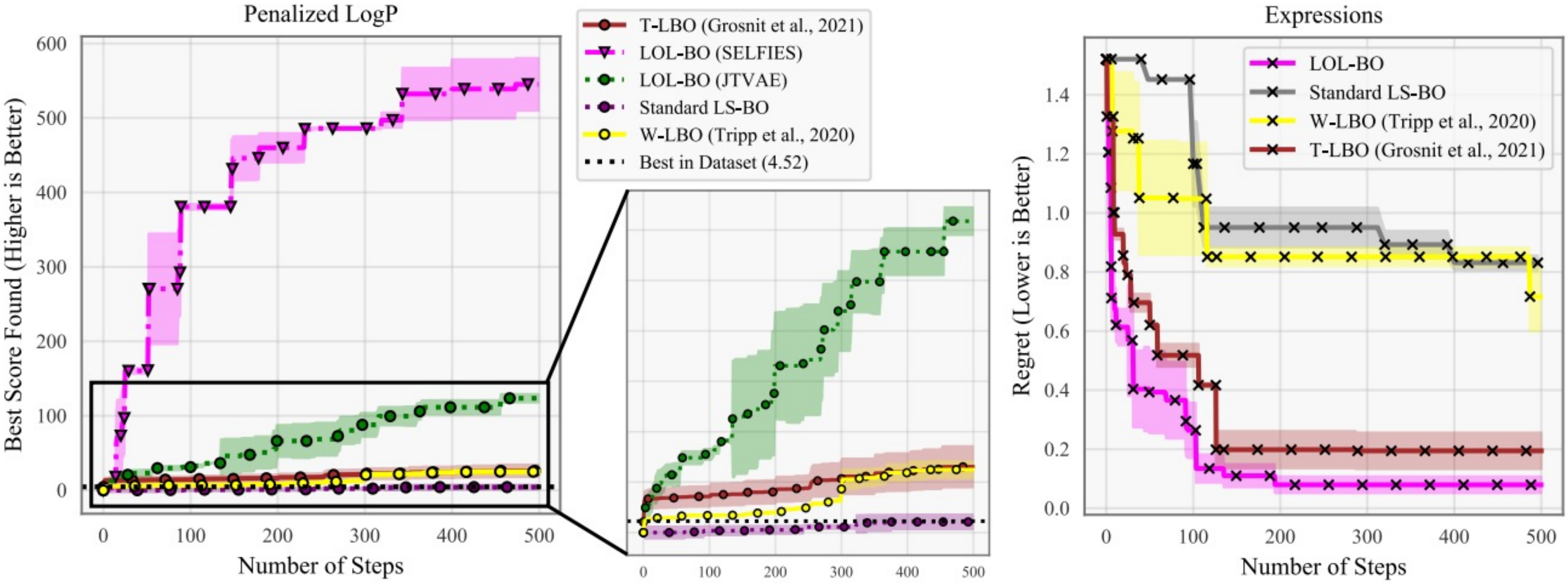
- Key Idea
  - ○ Extrapolate eigenfunctions of the latent space kernel matrix **L** with basis functions from the structured kernel k

Improvements to Latent Space BO

- LOL-BO [Maus et al., 2022]
  - Idea 1: train GP and VAE jointly
  - Idea 2: adapt high-dimensional BO methods in continuous spaces



Latent Space BO: Experimental Results



Outline of the Tutorial

- Overview of the BO Framework, GPs, advances in GPs and acquisition functions, and BoTorch demo
- Bayesian Optimization over Discrete/Hybrid Spaces
- Multi-fidelity Bayesian Optimization

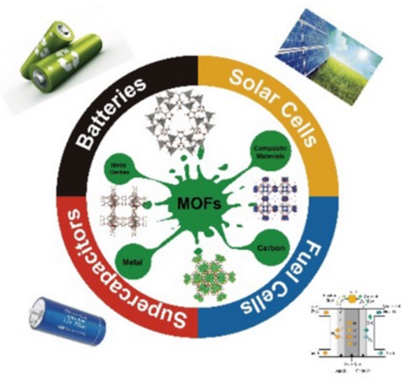
30 mins Break

- High-Dimensional BO and BoTorch Hands-on demo
- Multi-Objective BO and BoTorch Hands-on demo
- Summary and Outstanding Challenges in BO

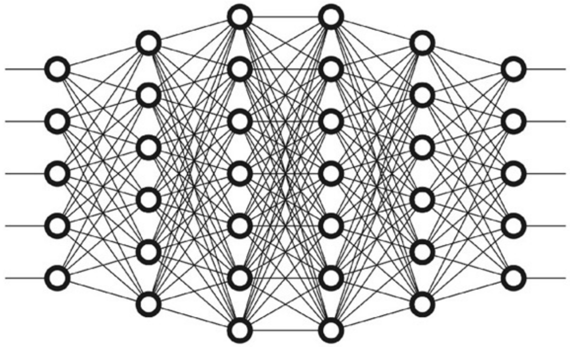
BO Over Hybrid Spaces: The Problem

- **Goal:** find optimized hybrid structures via expensive experiments
  - x = mixture of x_d (discrete) and x_c (continuous) variables

Microbiome design



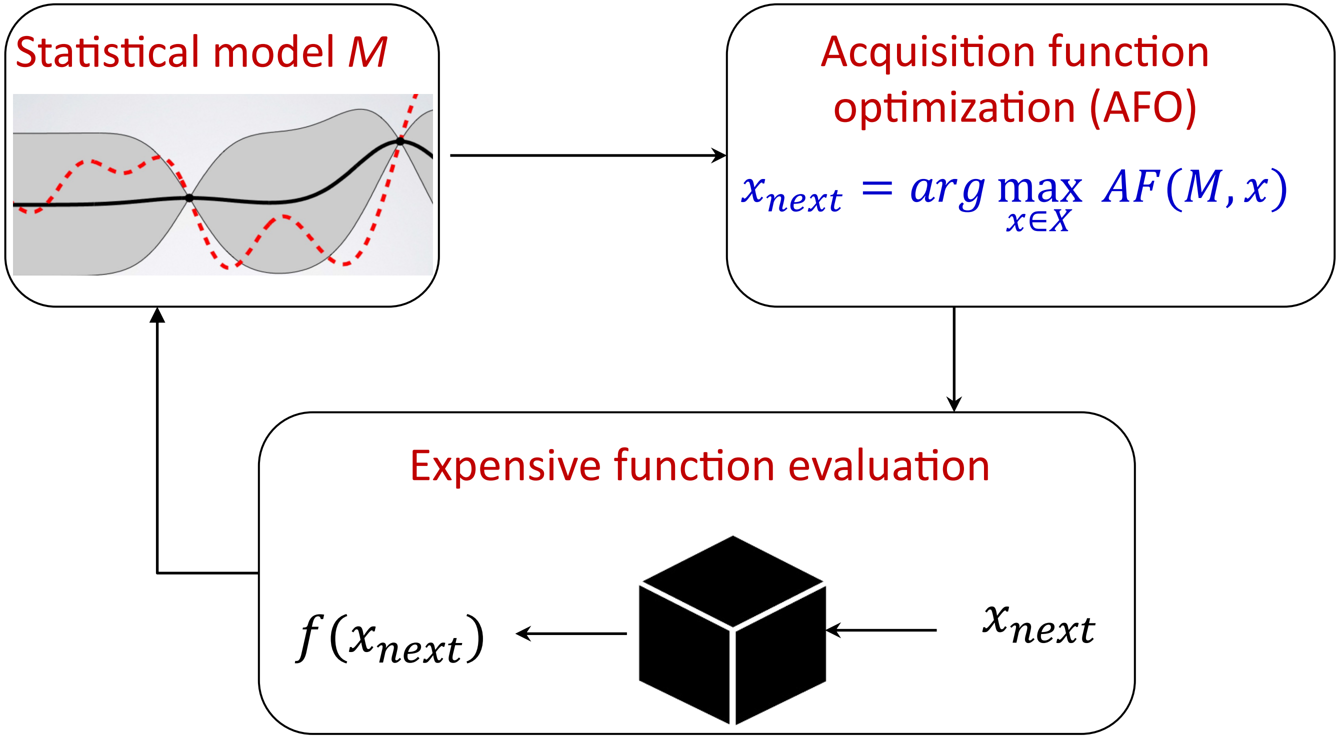
Material design



Hyper-parameter tuning / Auto ML

- Many other science, engineering, industrial applications

Hybrid BO: Technical Challenges



Statistical model *M*

Acquisition function optimization (AFO)

$$x_{next} = arg \max_{x \in X} AF(M, x)$$

Expensive function evaluation

$f(x_{next})$

x_{next}

- Effective modeling over hybrid structures (capture complex interactions among discrete and continuous variables)
- Solving hard optimization problem over hybrid spaces for AFO

Surrogate Models: MiVaBO [Daxberger et al., 2019]

- Linear surrogate model over binary structures
  - $f(x \in X) = \theta^T . \phi(x)$
  - $\phi(x)$ consists of continuous (random Fourier features), discrete (BOCS representation for binary variables), and mixed (products of all pairwise combinations) features

May not be sufficient to capture desired dependencies

Surrogate Models: SMAC [Hutter et al., 2010, 2011]

- Random forest as surrogate model
  - works naturally for categorical and mixed variables
  - Prediction/Uncertainty (= empirical mean/variance over trees)

Uncertainty estimates can be poor

- Improvements for better uncertainty estimates
  - Bagging with oversampling [Kim and Choi, 2022]

Surrogate Models: GP with Sum-and-Product Kernels

[Ru et al., 2021; Wan et al., 2021]

$$k(z,z') = (1-\lambda) * (\overbrace{k_d(x_d,x'_d) + k_c(x_c,x'_c)}^{\text{Sum}})$$
$$+ \quad \lambda * \underbrace{k_d(x_d,x'_d)k_c(x_c,x'_c)}_{\text{Product}}$$

- CoCaBO [Ru et al., 2021]
  - $k_d(x_d,x'_d) = \sum(x_d^i == x_d^i)$ // Hamming similarity
  - k_c is Matern Kernel
- Casmopolitan [Wan et al., 2021]
  - $k_d(x_d,x'_d) = \exp(\sum l_i(x_d^i == x_d^i))$ // exponentiated Hamming
  - k_c is Matern Kernel

Surrogate Models: GP w/ Diffusion Kernels [Deshwal et al., 2021]

- GP surrogate model with additive diffusion kernels
  - Exploits the general recipe of additive kernels [Duvenaud et al., 2011]
  - Instantiation w/ discrete & continuous diffusion kernels
  - Bayesian treatment of the hyper-parameters

$$\mathcal{K}_{HYB} = \boxed{\sum_{p=1}^{m+n} (\theta_p^2 \sum_{i_1,\cdots,i_p} \prod_{d=1}^{p} k_{i_d}(x_{i_d}, x'_{i_d}))}$$

Surrogate Models:
GP w/ Frequency Modulation Kernels [Oh et al., 2021]

- Key idea: Generalize the COMBO kernel [Oh et al., 2019] by parametrizing via a function of continuous variables

$$\boxed{K = exp(-\beta L(G))}$$

$$\boxed{K = U^T exp(-\beta \Sigma) U}$$

$$\boxed{K = U^T f(\Sigma, X_c, X_{c'}) U}$$

Remember the COMBO kernel

- Requirement on f for K to be a positive definite kernel
  - f should be positive definite w.r.t $X_c, X_{c'}$

Acquisition Functions

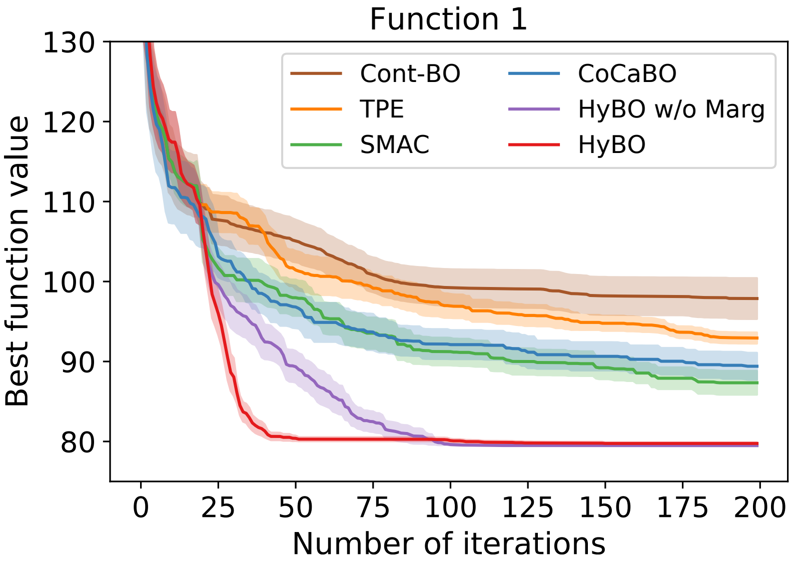
- Thompson Sampling
  - MiVaBO: linear surrogate model $f(x \in X) = \theta^T.\phi(x)$

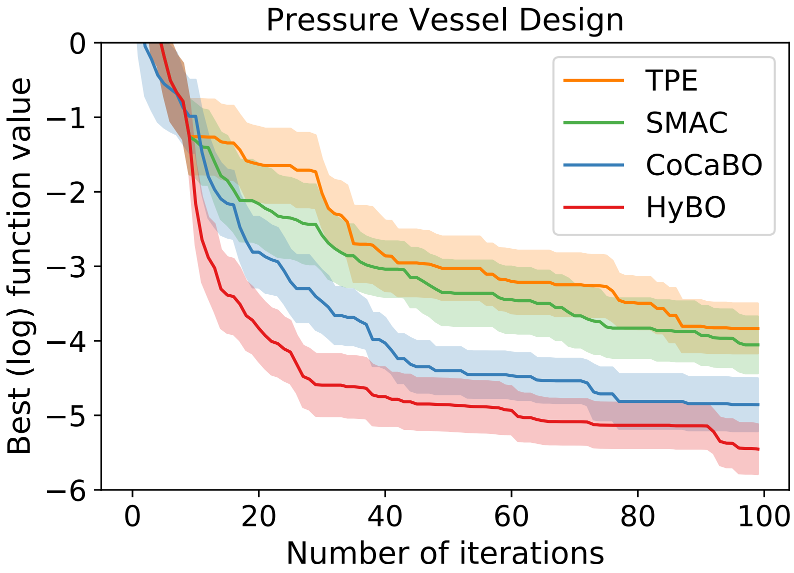
- Expected Improvement
  - SMAC: random forest model
  - HyBO: GP with hybrid diffusion kernel
  - BO-FM: GP with frequency modulated kernel
  - CoCaBO: GP with sum and product kernel over subspaces

Acquisition Function Optimization

- MiVaBO, HyBO, BO-FM: Alternating search
  - Step 1: Search over continuous sub-space
  - Step 2: Search over discrete sub-space using output of Step #1
  - Repeat if needed
- SMAC
  - Hand-designed local search with restarts
- CoCaBO
  - Continuous variables via gradient-based search
  - Categorical variables via multi-armed bandit algorithms
- Casmopolitan
  - Trust region-based AFO to scale to high-dimensional spaces

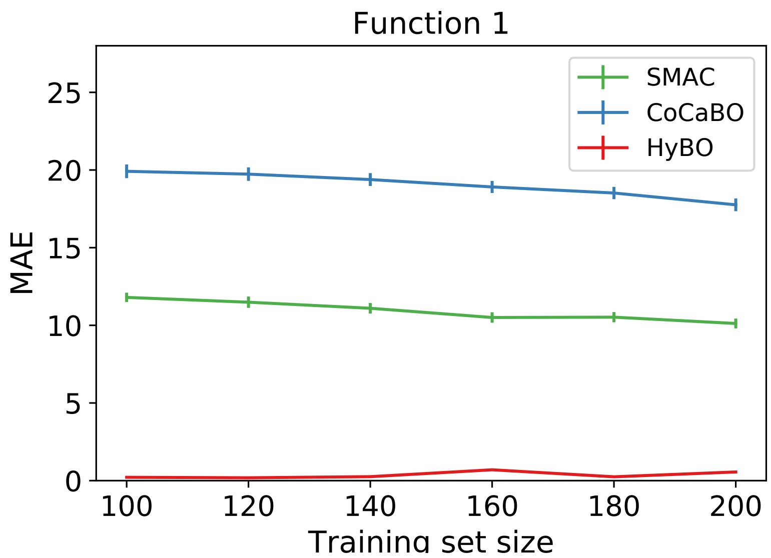
Hybrid BO: Experimental Results #1

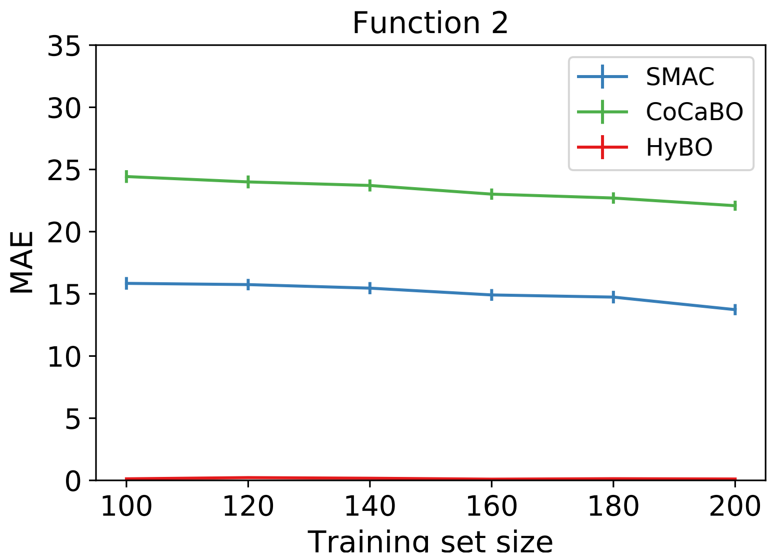




- HyBO performs significantly better than prior methods

Hybrid BO: Experimental Results #2





- HyBO’s better BO performance is due to better surrogate model

Outline of the Tutorial

- Overview of the BO Framework, GPs, advances in GPs and acquisition functions, and BoTorch demo

- Bayesian Optimization over Discrete/Hybrid Spaces

- Multi-fidelity Bayesian Optimization

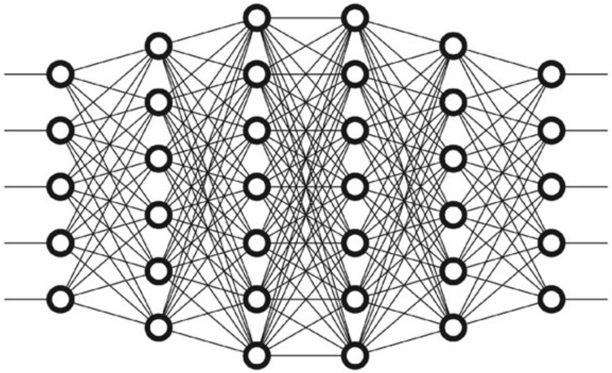
30 mins Break

- High-Dimensional BO and BoTorch Hands-on demo

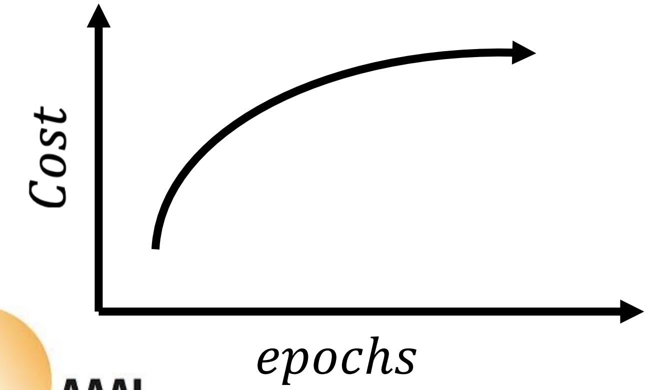
- Multi-Objective BO and BoTorch Hands-on demo

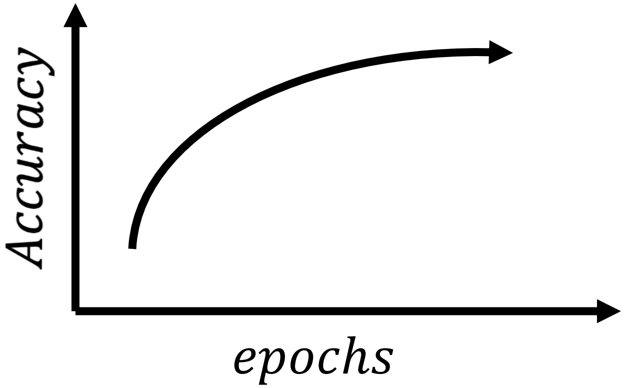
- Summary and Outstanding Challenges in BO

Application #1: Auto ML and Hyperparameter Tuning

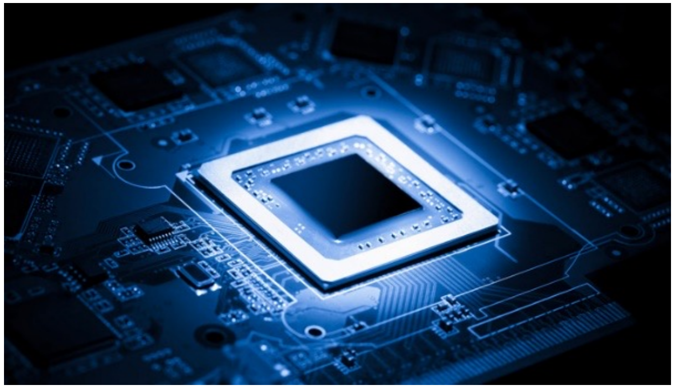


Cost vs. Accuracy trade-offs in evaluating hyperparameter configurations

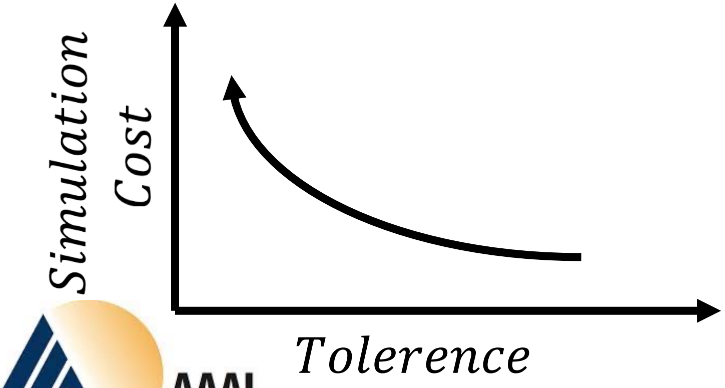


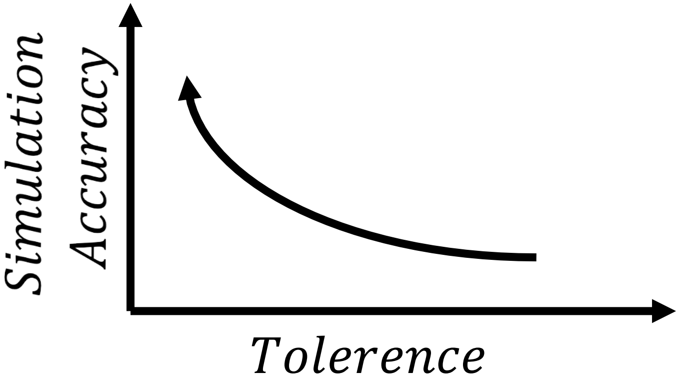


Application #2: Hardware Design via Simulations

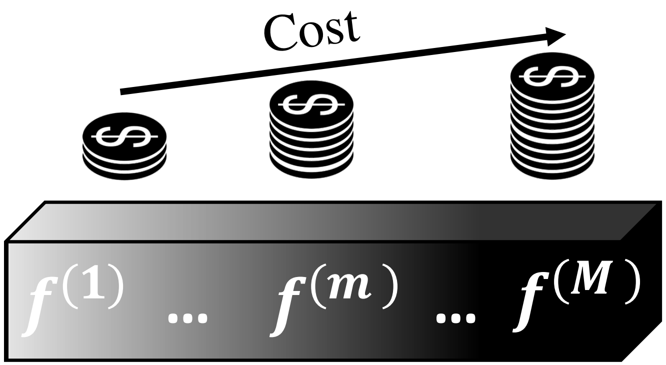


Cost vs. Accuracy trade-offs in evaluating hardware designs

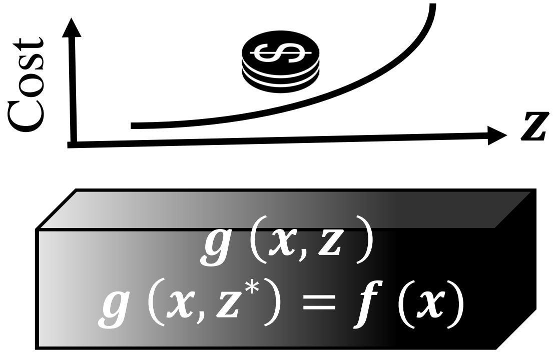




Multi-Fidelity BO: The Problem



Discrete fidelity



Continuous fidelity

- Cost vs. accuracy trade-offs for function approximations
- Continuous-fidelity is the most general case
  - Discrete-fidelity is a special case
- Goal: Optimize the highest-fidelity function by minimizing the resource cost of experiments

Multi-Fidelity BO: Key Challenges

- Intuition: use cheap (low-fidelity) experiments to gain information and prune the input space; and use costly (high-fidelity) experiments on promising candidates

- Modeling challenge: How to model multi-fidelity functions to allow information sharing?
- Reasoning challenge:
  - How to select the input design and fidelity pair in each BO iteration?
  - How to progressively select higher fidelity experiments?

Multi-Fidelity GPs for Modeling

- Desiderata: model relationship/information sharing between different fidelities

- Solution: multi-output GPs with vector-valued kernels
  - Provides a prediction μ and uncertainty σ for each input and fidelity pair

$$k : (\chi \times Z)^2 \rightarrow \mathbb{R}$$
$$k(\{x, z\}, \{x', z'\}) = k_{\chi}(x, x')k_{Z}(z, z')$$

$$\mu_f(x) = \mu_g(x, z^*) \text{ and } \sigma_f(x) = \sigma_g(x, z^*)$$

EI Extension for Multi-Fidelity BO

- Multi-fidelity expected improvement (MF-EI)
  - Extension of EI for multi-fidelity setting
  - Applicable for discrete-fidelity setting

$$EI(x,z) = E\left[\max\left(\tau - y^{z^*}\right)\right] cov\left[y^z, y^{z^*}\right] C_{z^*}/C_z$$

- Acquisition function optimization
  - Enumerate each fidelity z and find the best x while fixing z

Information-Theoretic Extensions for Multi-Fidelity BO

$$AF(x) = H(\alpha \mid D) - E_y[H(\alpha|D \cup \{x, y\})]$$
$$= \text{Information Gain}(\alpha; y)$$

- Design choices of α leads to different algorithms

- α as input location of optima x^*
  - Entropy Search (ES) / Predictive Entropy Search (PES)
  - Intuitive but requires expensive approximations

- α as output value of optima y^*
  - Max-value Entropy Search (MES) and it's variants
  - Computationally cheaper and more robust

Information-Theoretic Extensions for Multi-Fidelity BO

$$AF(x,z) = [H(\alpha \mid D) \ - E_y[H(\alpha|D \cup \{x,z,y\})]]/C(x,z)$$
$$= \text{Information Gain per Unit Cost}(\alpha; y)$$

- Design choices of α leads to different algorithms

- α as input location of optima x^*
  - MF-Predictive Entropy Search (MF-PES)
  - Intuitive but requires expensive approximations

- α as output value of optima y^*
  - MF Max-value Entropy Search (MF-MES)
  - Computationally cheaper and more robust

Continuous-Fidelity BO: BOCA Algorithm

- Two step procedure to select input x and fidelity z separately

- **Selection of input $\boldsymbol{x}$**
  - Optimize the AF with respect to highest fidelity

$$\text{UCB}(\text{x}, z^*) = y^{z^*}(x) + \beta\, \sigma^{z^*}(x)$$

- **Selection of fidelity $\boldsymbol{z}$**
  - Reducing fidelity space:

Select higher fidelities

Avoid close neighborhood of the highest fidelity

$$Z_t = \{z{:}\, \boxed{\sigma^{z}\left(x_{opt}\right) \geq \gamma(z)}, \boxed{\xi(z) > \beta \;\| \, \xi \, \|_{\infty}}\} \cup \{z^*\}$$

Multi-Fidelity BO for HPO: taKG Algorithm

- Trace aware Knowledge Gradient for Hyperparameter optimization:
  - Use exponential decay kernel for fidelity space
  - Select the input and fidelity (x, z) maximizing:

$$taKG(x,z) = \frac{L(\emptyset) - L(x,z)}{C(x,z)}$$

$$= \frac{\min_{x'} \mathbb{E}[g(x', z^*)] - \mathbb{E}[\min_{x'} \mathbb{E}[g(x', z^*)|y(x,z)]}{C(x,z)}$$

Code and Software

- Multi-fidelity modeling
  - https://mlatcl.github.io/mlphysical/lectures/05-02-multifidelity.html
- BOTorch
  - https://botorch.org/tutorials/discrete_multi_fidelity_bo

Outline of the Tutorial

- Overview of the BO Framework, GPs, advances in GPs and acquisition functions, and BoTorch demo

- Bayesian Optimization over Discrete/Hybrid Spaces

- Multi-fidelity Bayesian Optimization

30 mins Break

- High-Dimensional BO and BoTorch Hands-on demo

- Multi-Objective BO and BoTorch Hands-on demo

- Summary and Outstanding Challenges in BO

High-Dimensional Bayesian Optimization

Slides credit: Jake Gardner @ University of Pennsylvania

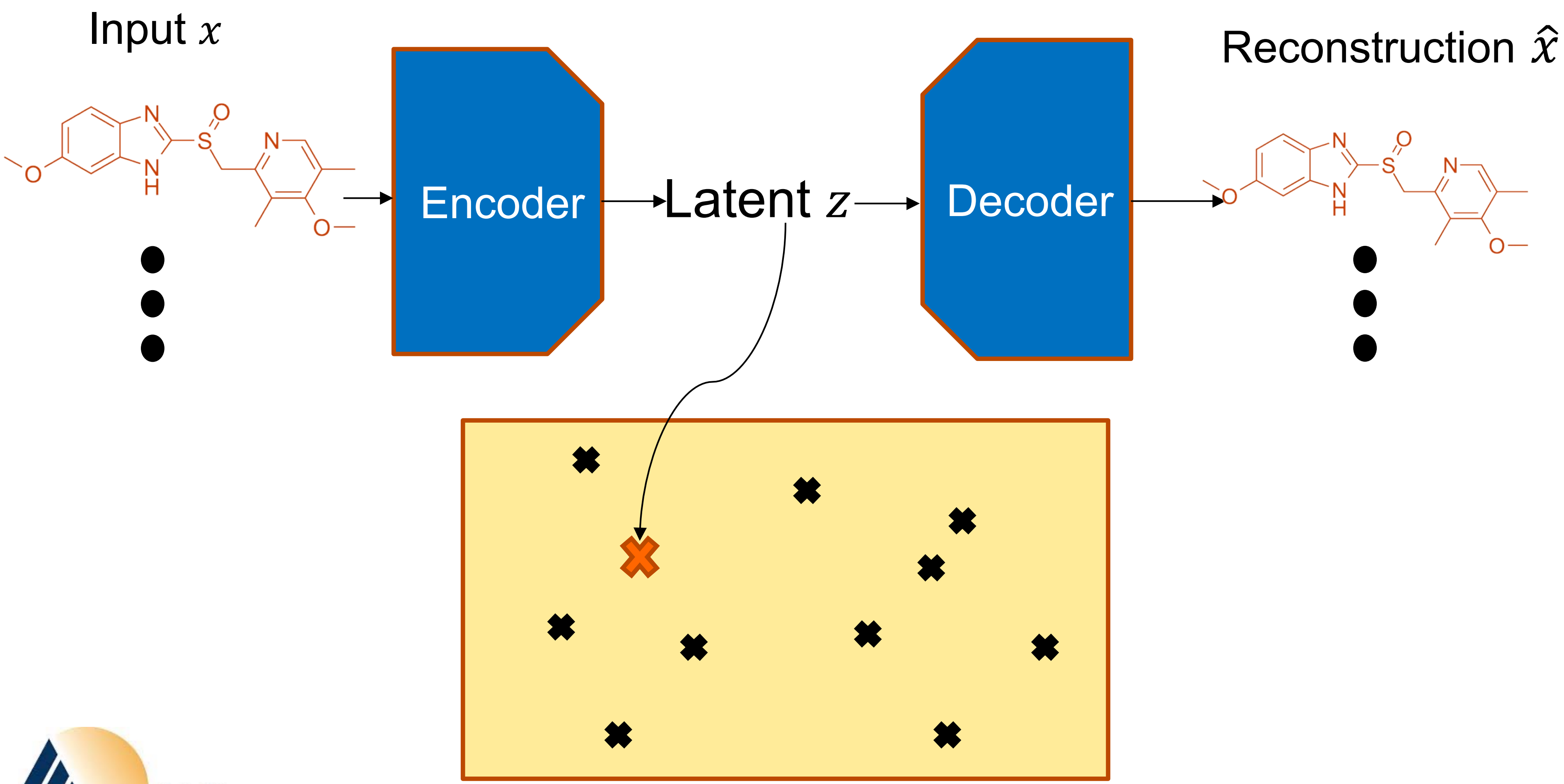
Dimensionality is a Key Challenge

Problem: covering a high dimensional search space takes **exponentially many** evaluations!

Dimensionality is a Key Challenge

Example

(Unsupervised)



Input x

Encoder

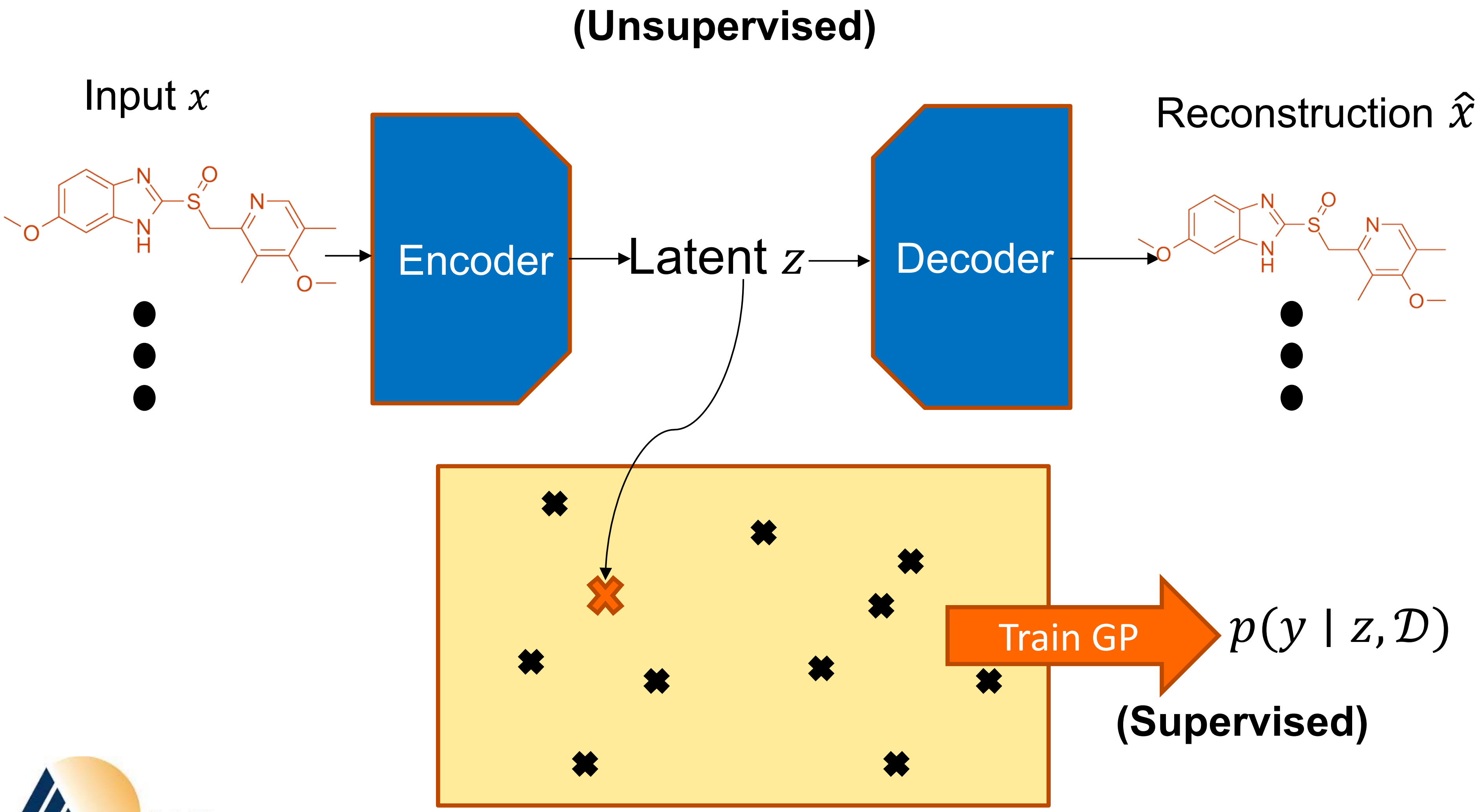
Latent z

Decoder

Reconstruction $\hat{x}$

Dimensionality is a Key Challenge

Example



(Unsupervised)

Input x

Encoder

Latent z

Decoder

Reconstruction $\hat{x}$

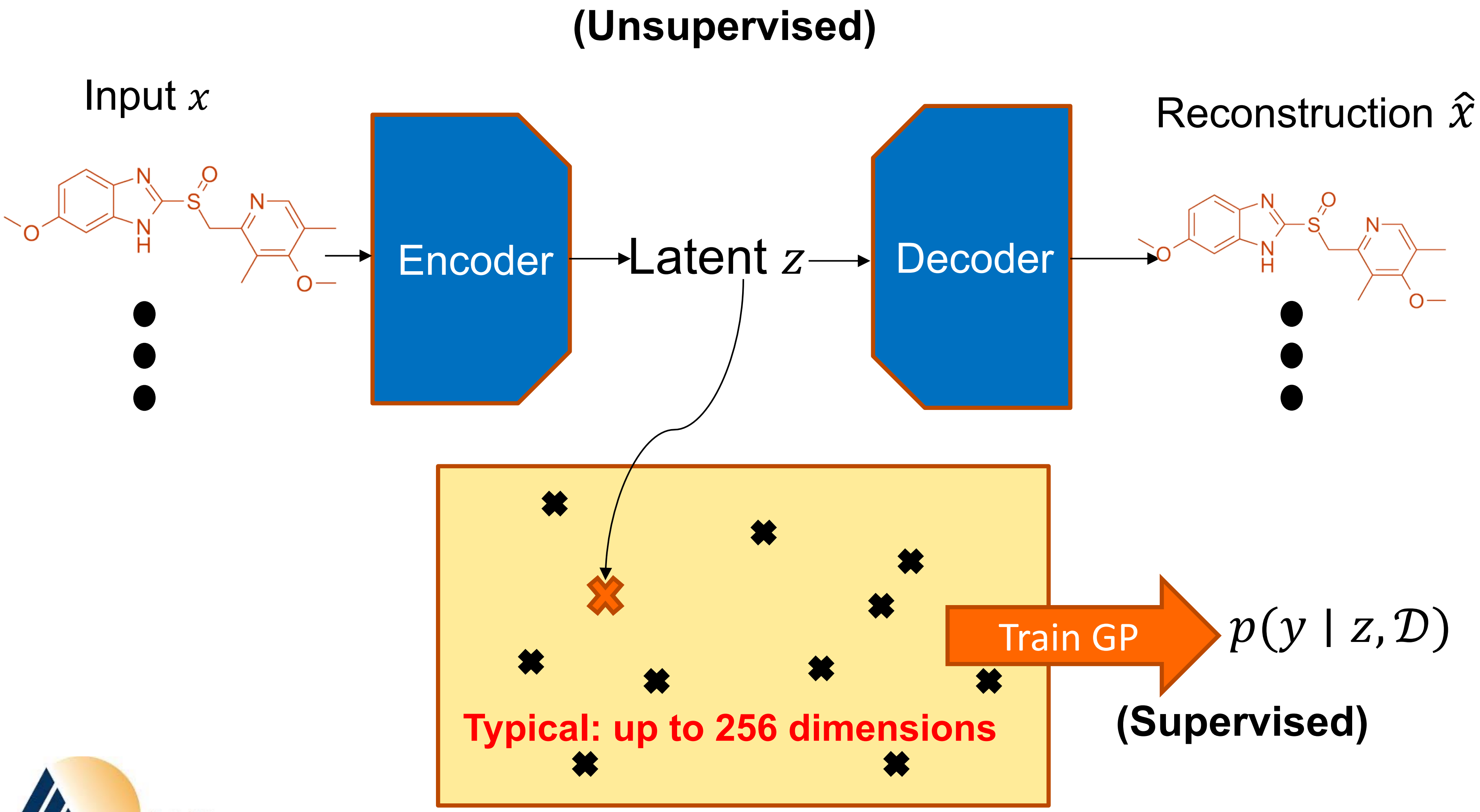
Train GP

$p(y \mid z, \mathcal{D})$

(Supervised)

Dimensionality is a Key Challenge

Example



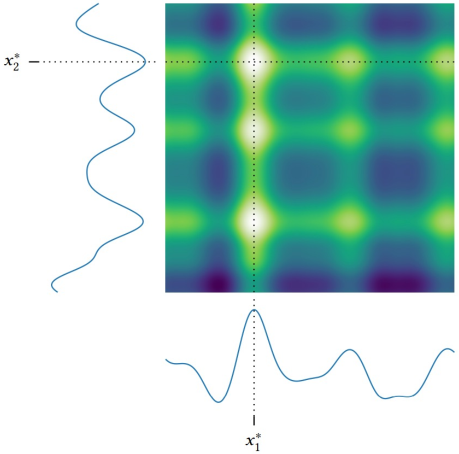
Typical: up to 256 dimensions

Approaches to High Dimensional BO

Approaches to High Dimensional BO

1. Additive Structure

(Kandasamy et al., 2015; Wang et al., 2017; Gardner et al., 2017; Rolland et al., 2018; Mutný et al., 2018)



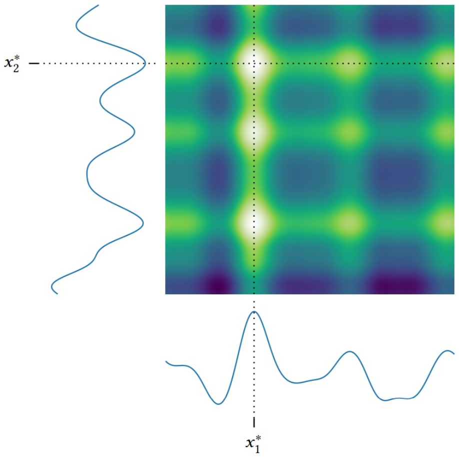
(bayesoptbook.com)

$$f(x) = g_1(x_1) + g_2(x_2)$$

Approaches to High Dimensional BO

1. Additive Structure

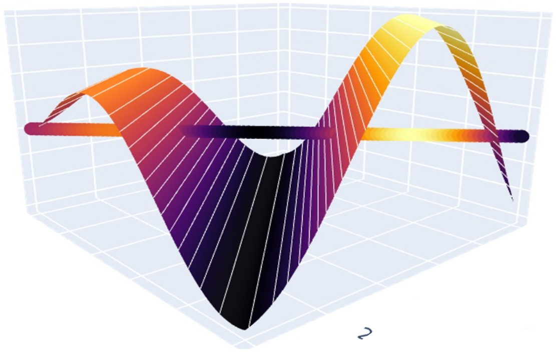
(Kandasamy et al., 2015; Wang et al., 2017; Gardner et al., 2017; Rolland et al., 2018; Mutný et al., 2018)



(bayesoptbook.com)

$$f(x) = g_1(x_1) + g_2(x_2)$$

2. Linear Embeddings

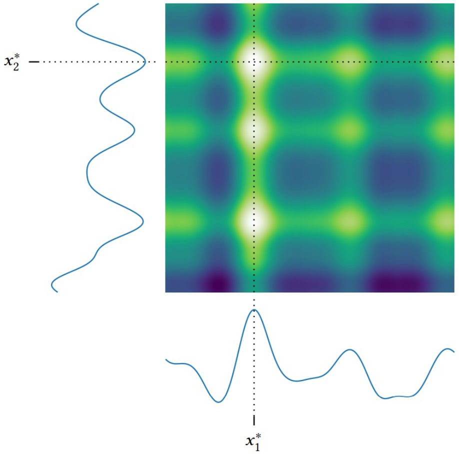


$$f(x) = g(Ax)$$
$$g: \mathbb{R}^d \rightarrow \mathbb{R}$$

Approaches to High Dimensional BO

1. Additive Structure

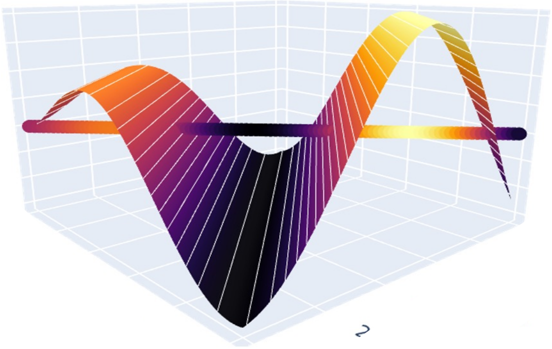
(Kandasamy et al., 2015; Wang et al., 2017; Gardner et al., 2017; Rolland et al., 2018; Mutný et al., 2018)



(bayesoptbook.com)

$$f(x) = g_1(x_1) + g_2(x_2)$$

2. Linear Embeddings



$$f(x) = g(Ax)$$
$$g: \mathbb{R}^d \to \mathbb{R}$$

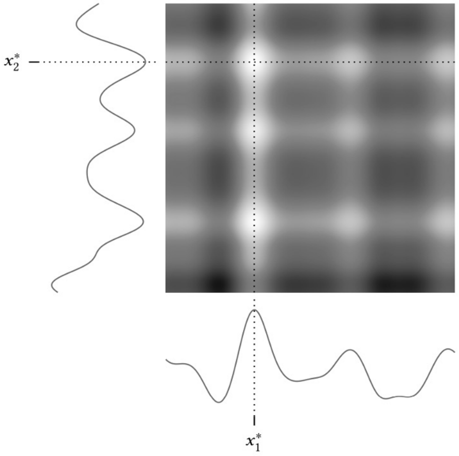
3. Local BayesOpt

“Any optimum will do”

Approaches to High Dimensional BO

1. Additive Structure

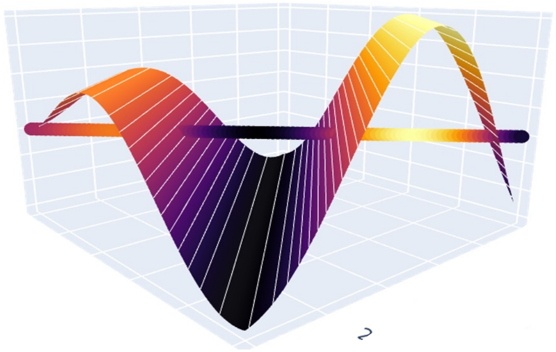
(Kandasamy et al., 2015; Wang et al., 2017; Gardner et al., 2017; Rolland et al., 2018; Mutný et al., 2018)



(bayesoptbook.com)

$$f(x) = g_1(x_1) + g_2(x_2)$$

2. Linear Embeddings



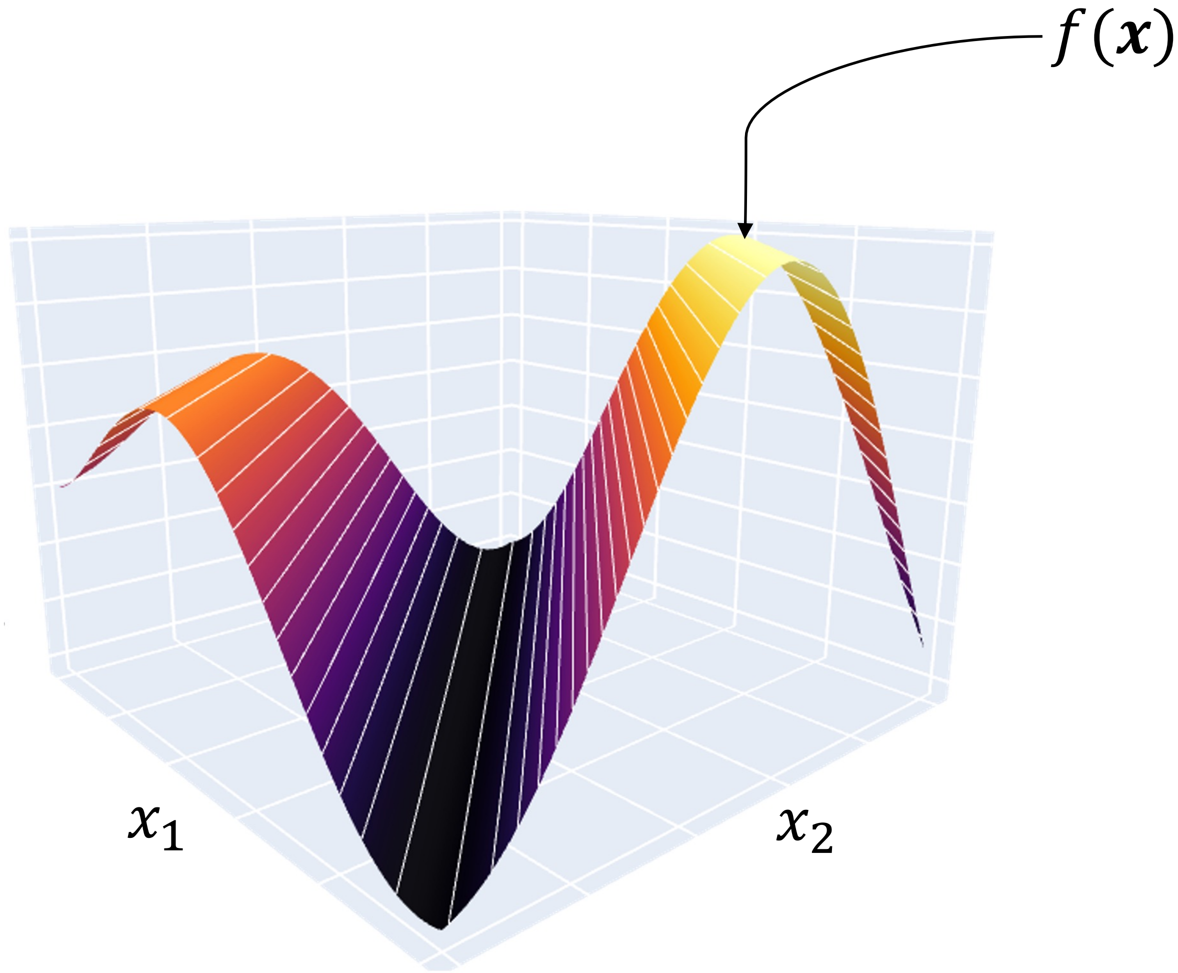
$$f(x) = g(Ax)$$
$$g\colon \mathbb{R}^d \to \mathbb{R}$$

3. Local BayesOpt

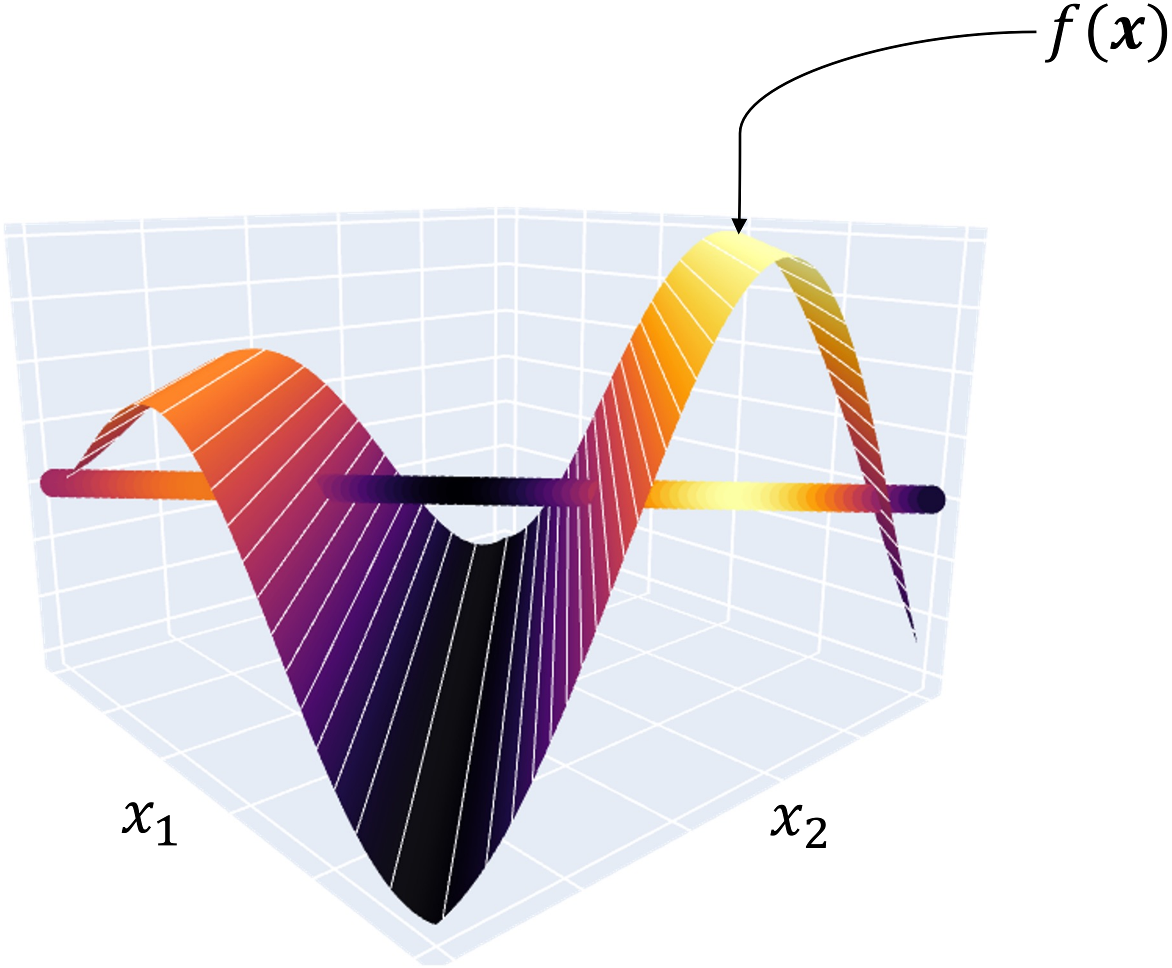
“Any optimum will do”

High Dimensional BO via Linear Embeddings

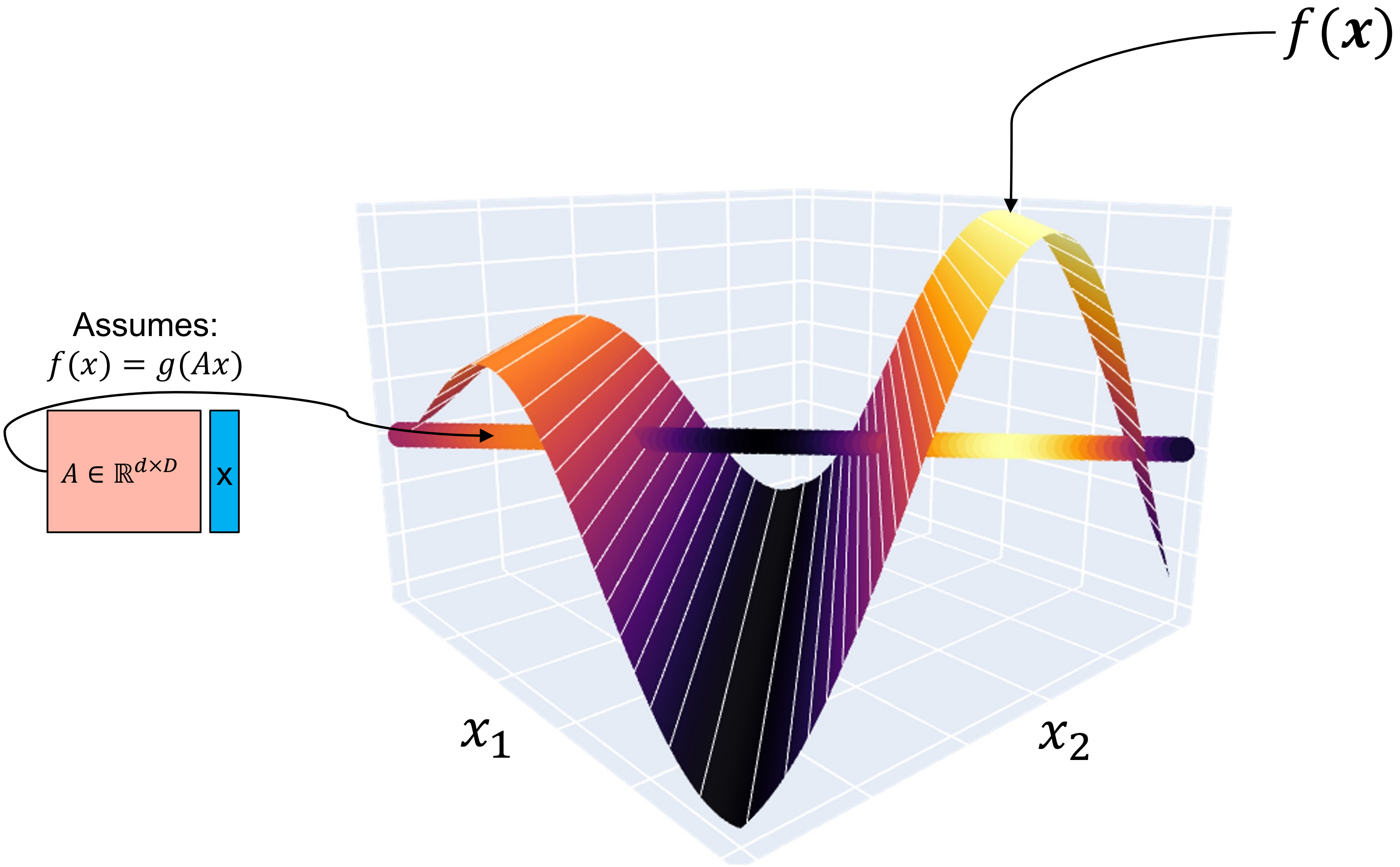
High Dimensional BO: Linear Embeddings



High Dimensional BO: Linear Embeddings



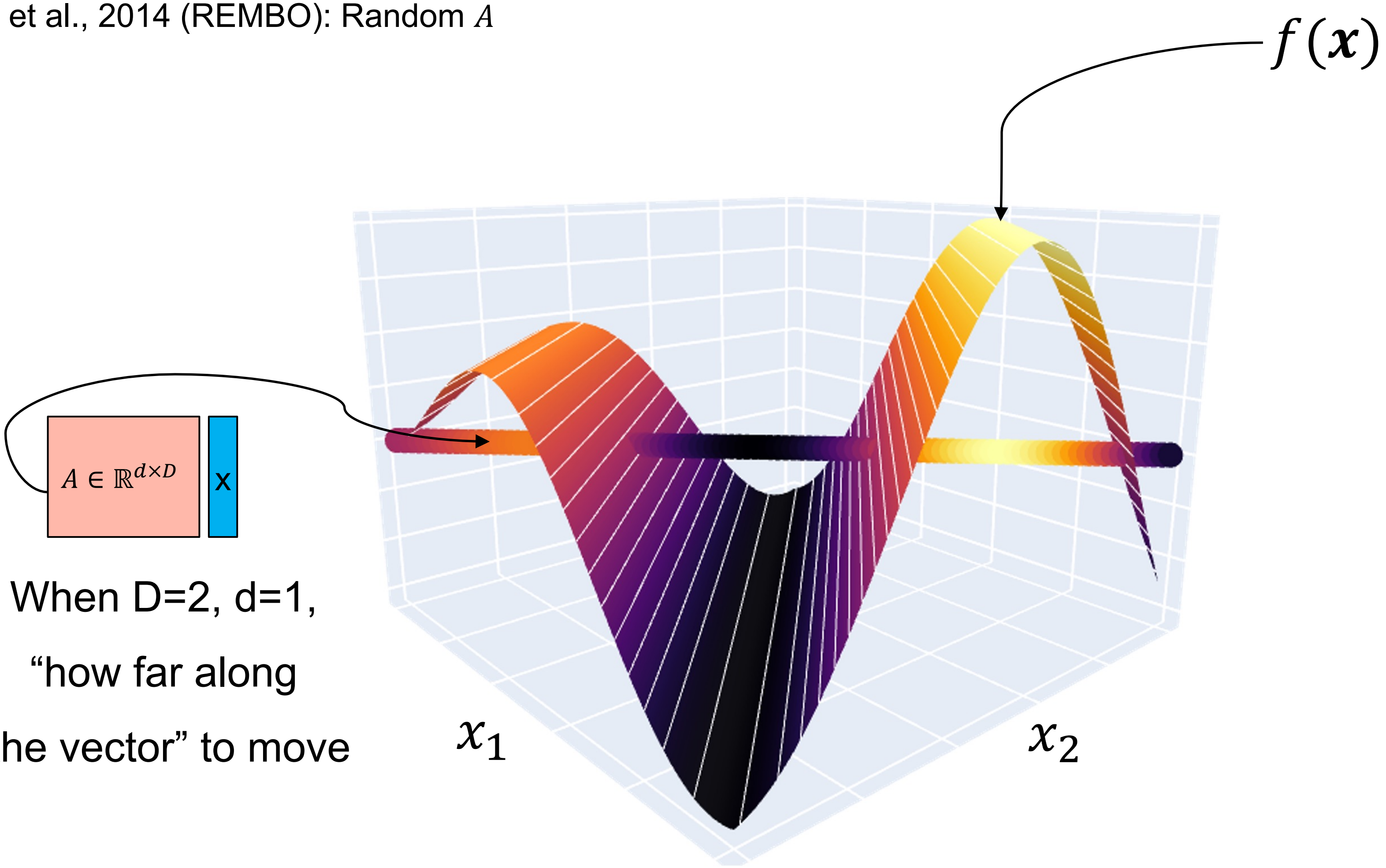
High Dimensional BO: Linear Embeddings



Assumes:
$f(x) = g(Ax)$

High Dimensional BO: Linear Embeddings

Wang et al., 2014 (REMBO): Random A



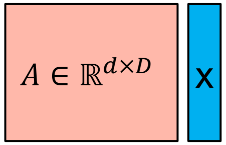
When D=2, d=1,

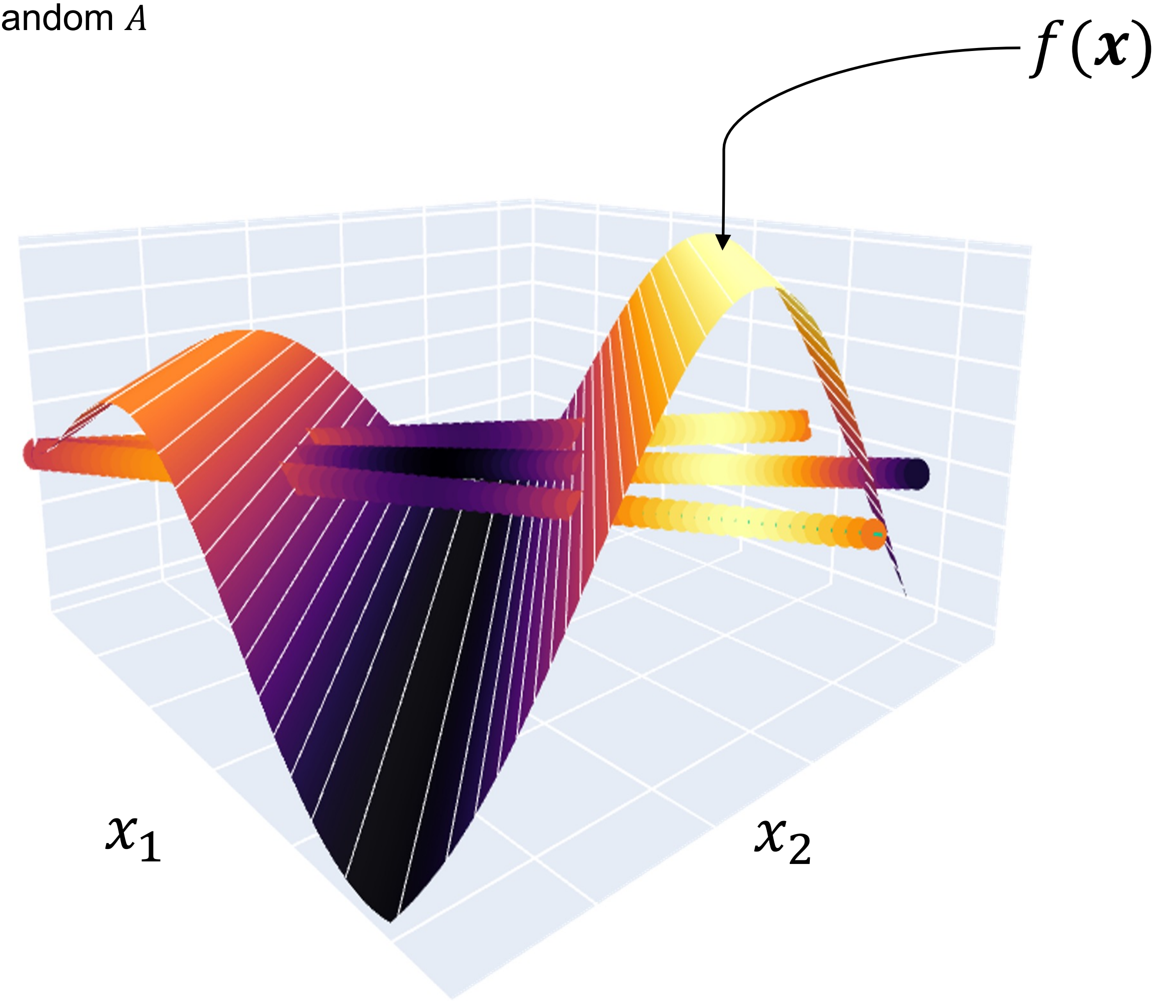
“how far along

the vector” to move

High Dimensional BO: Linear Embeddings

Wang et al., 2014 (REMBO): Random A

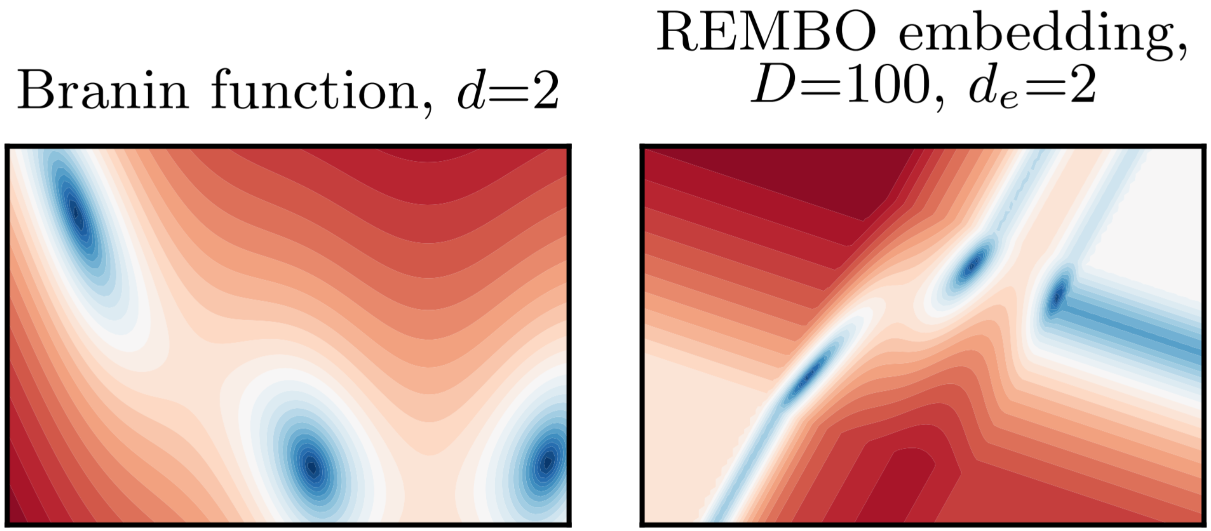




High Dimensional BO: Linear Embeddings

Wang et al., 2014 (REMBO): Random A

Letham et al., 2020 (ALEBO): Use projection pseudo-inverse to avoid bound clipping via constrained acquisition maximization, introduce pseudo inverse into Mahalanobis metric in kernel



(Figure 1, Letham et al., 2020)

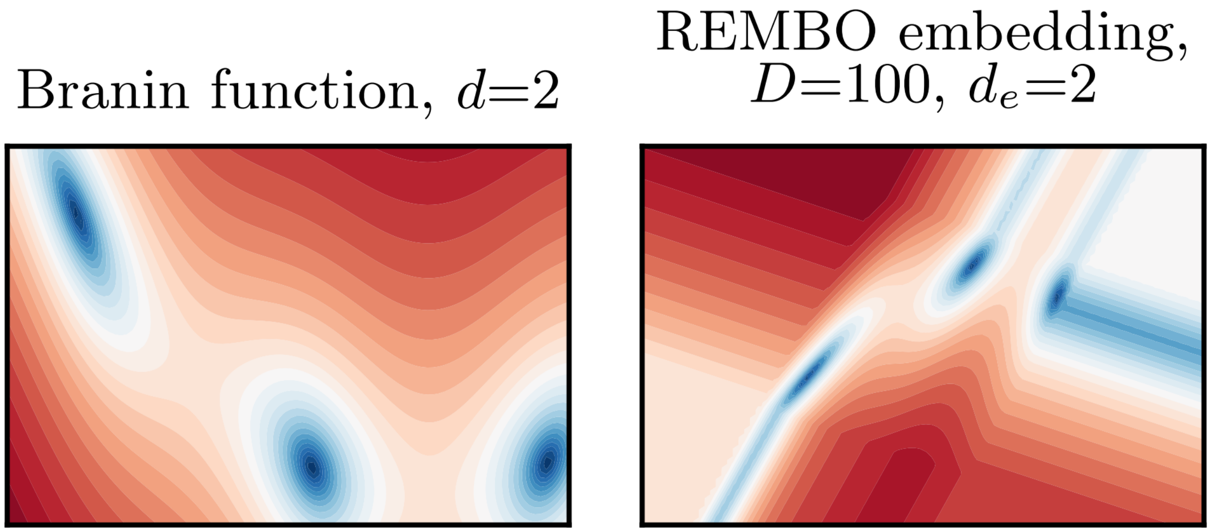
High Dimensional BO: Linear Embeddings

Wang et al., 2014 (REMBO): Random A

Letham et al., 2020 (ALEBO): Use projection pseudo-inverse to avoid bound clipping via constrained acquisition maximization, introduce pseudo inverse into Mahalanobis metric in kernel

Binois et al., 2020: The original space bounds clipping problem is not trivially solved by heuristic bounds in the embedding.

Munteanu et al., 2019 (HeSBO): Avoids bounds clipping via their embedding method.



Branin function, d=2

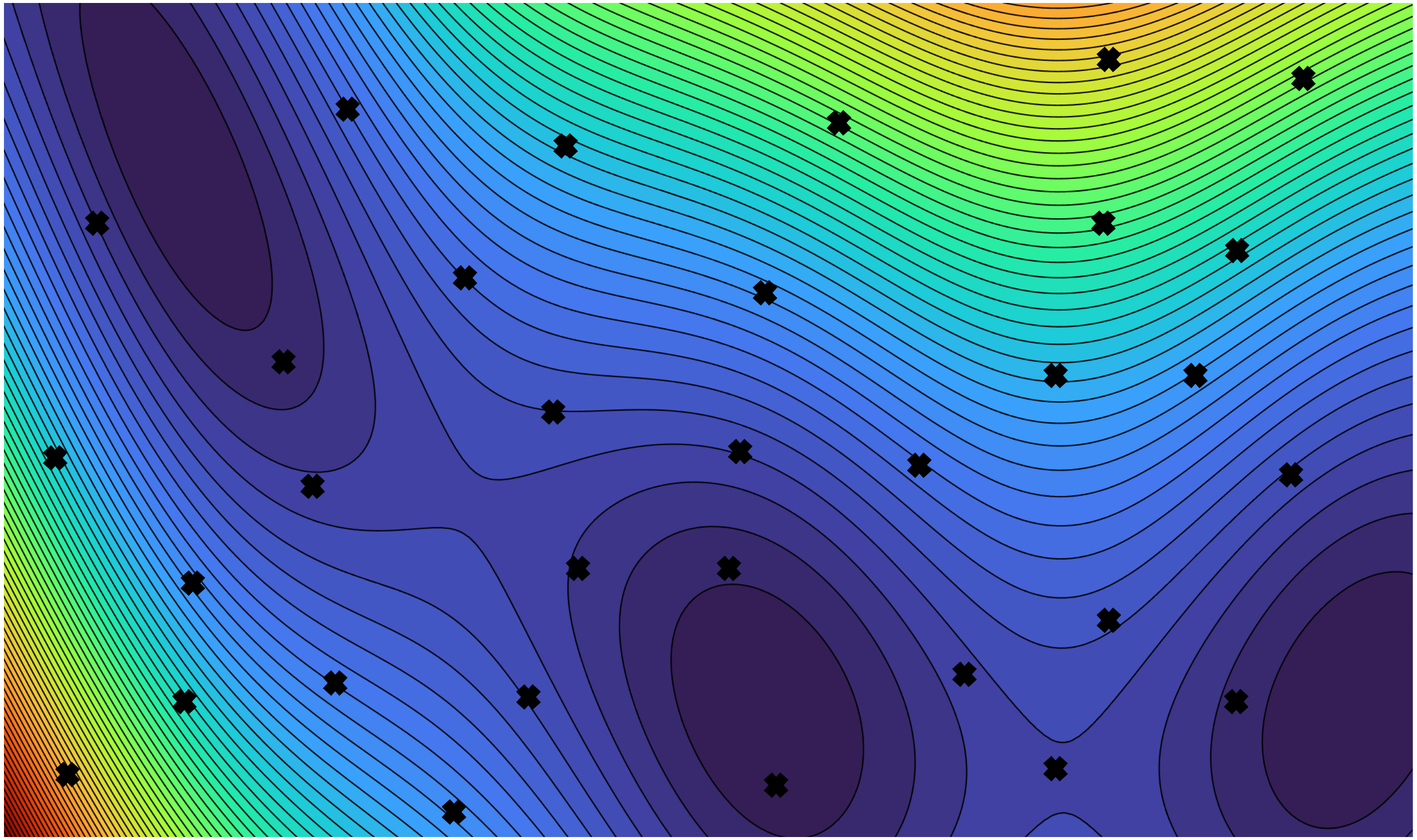
REMBO embedding, D=100, d_e=2

(Figure 1, Letham et al., 2020)

High Dimensional BO via Local BO

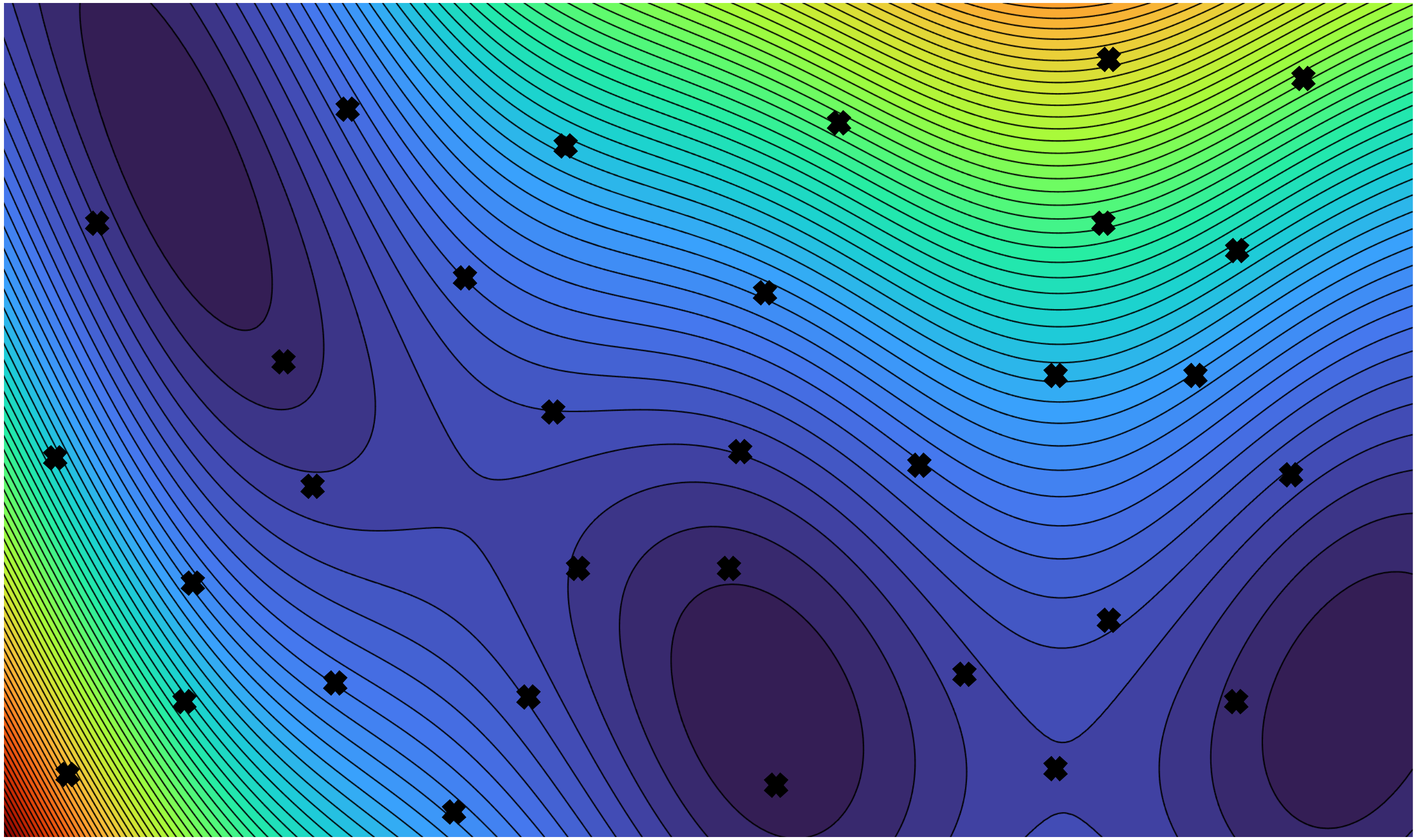
High Dimensional BO: Local BO

(e.g. Eriksson et al., 2019, Wang et al., 2020)



High Dimensional BO: Local BO

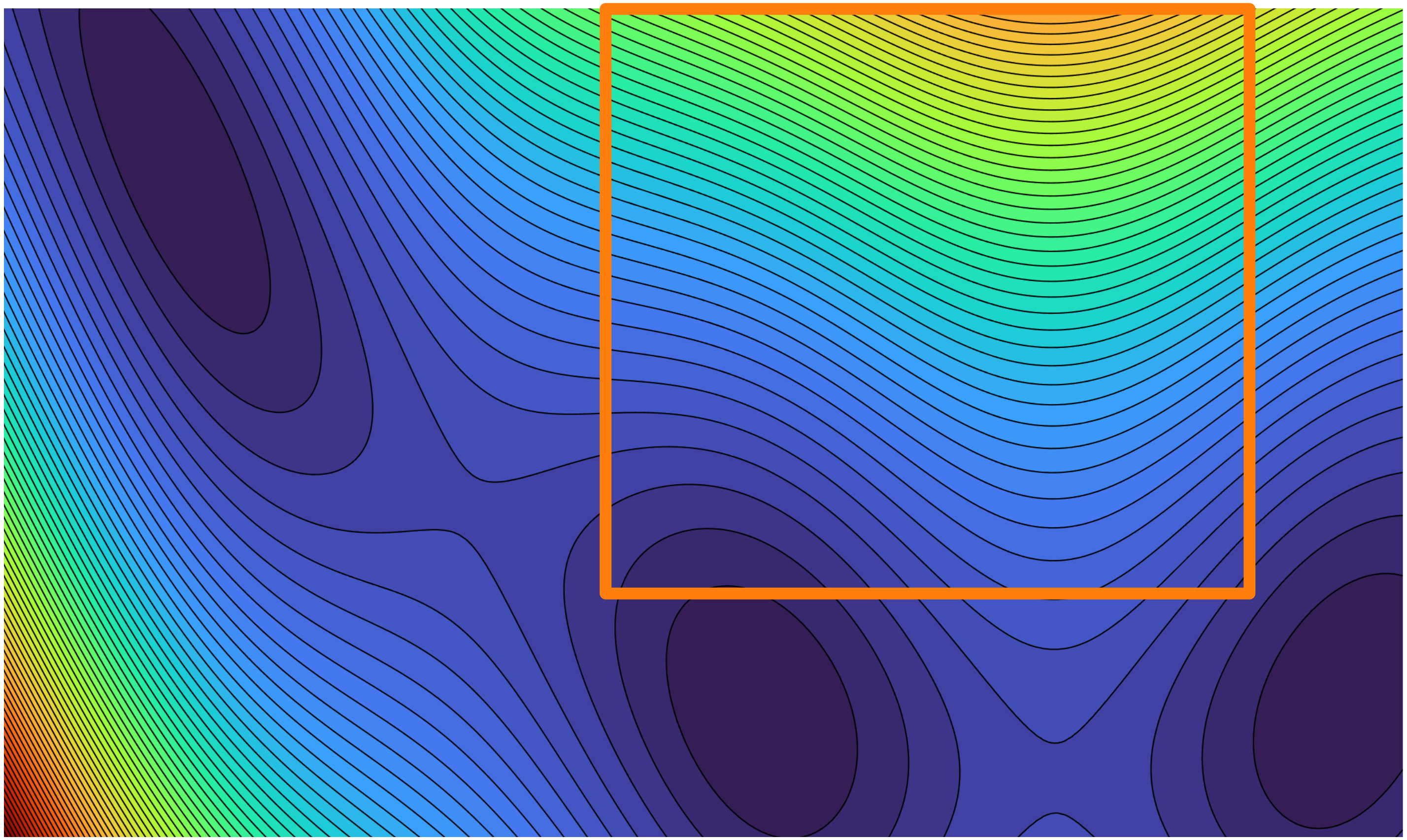
(e.g. Eriksson et al., 2019, Wang et al., 2020)



Idea: Perform BO inside a *trust region* (TuRBO)

High Dimensional BO: Local BO

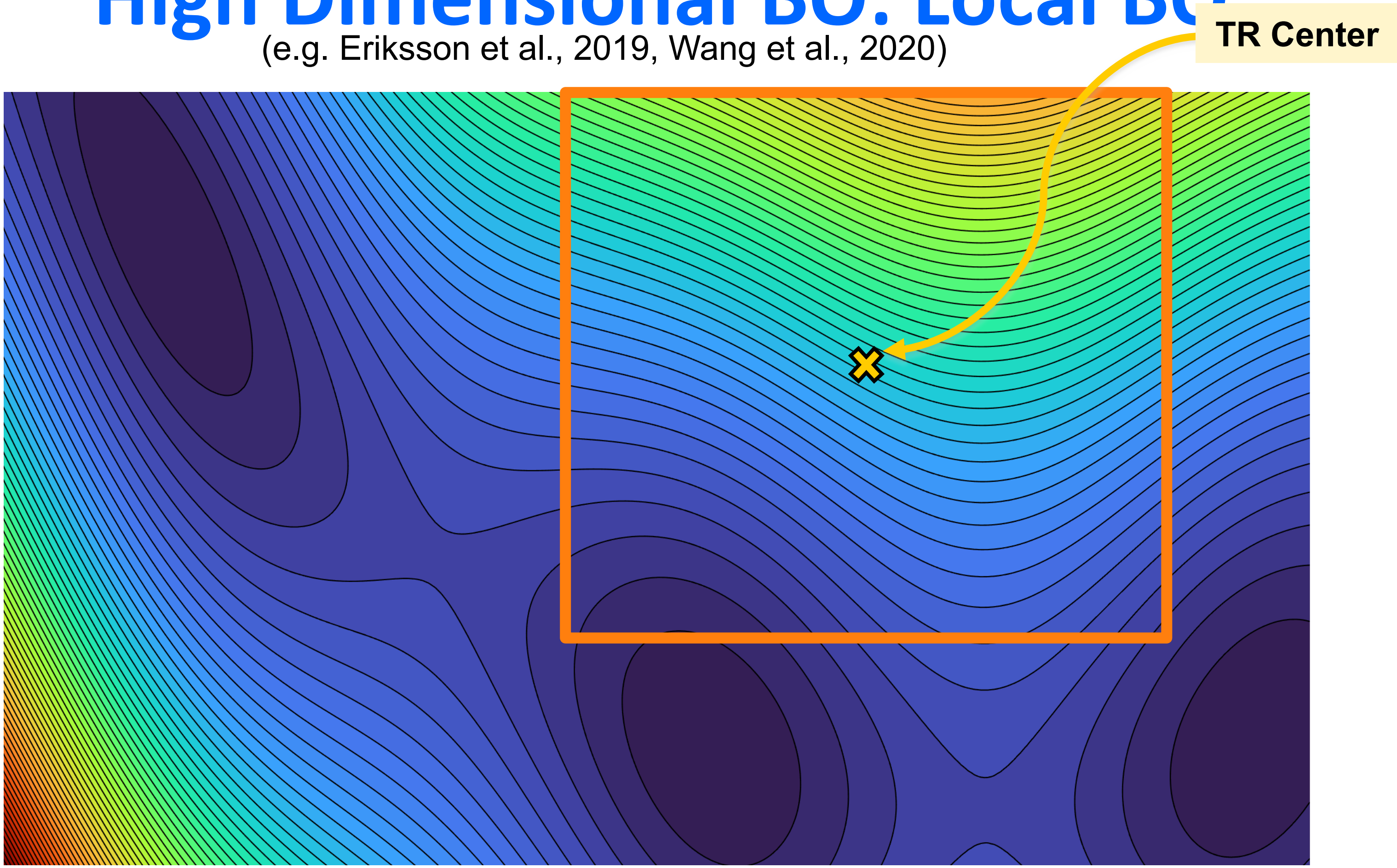
(e.g. Eriksson et al., 2019, Wang et al., 2020)



Idea: Perform BO inside a *trust region* (TuRBO)

High Dimensional BO: Local BO

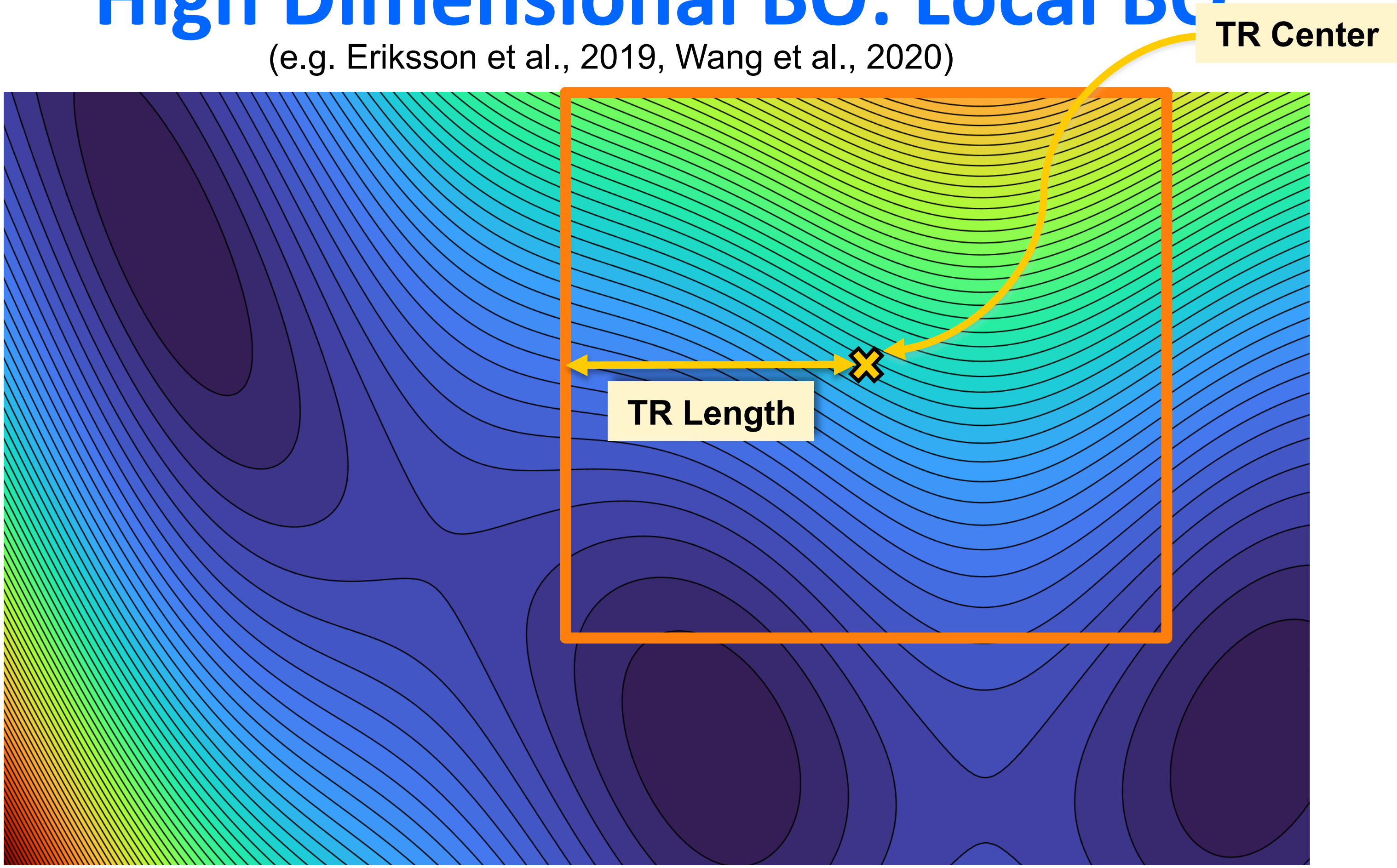
(e.g. Eriksson et al., 2019, Wang et al., 2020)



Idea: Perform BO inside a *trust region* (TuRBO)

High Dimensional BO: Local BO

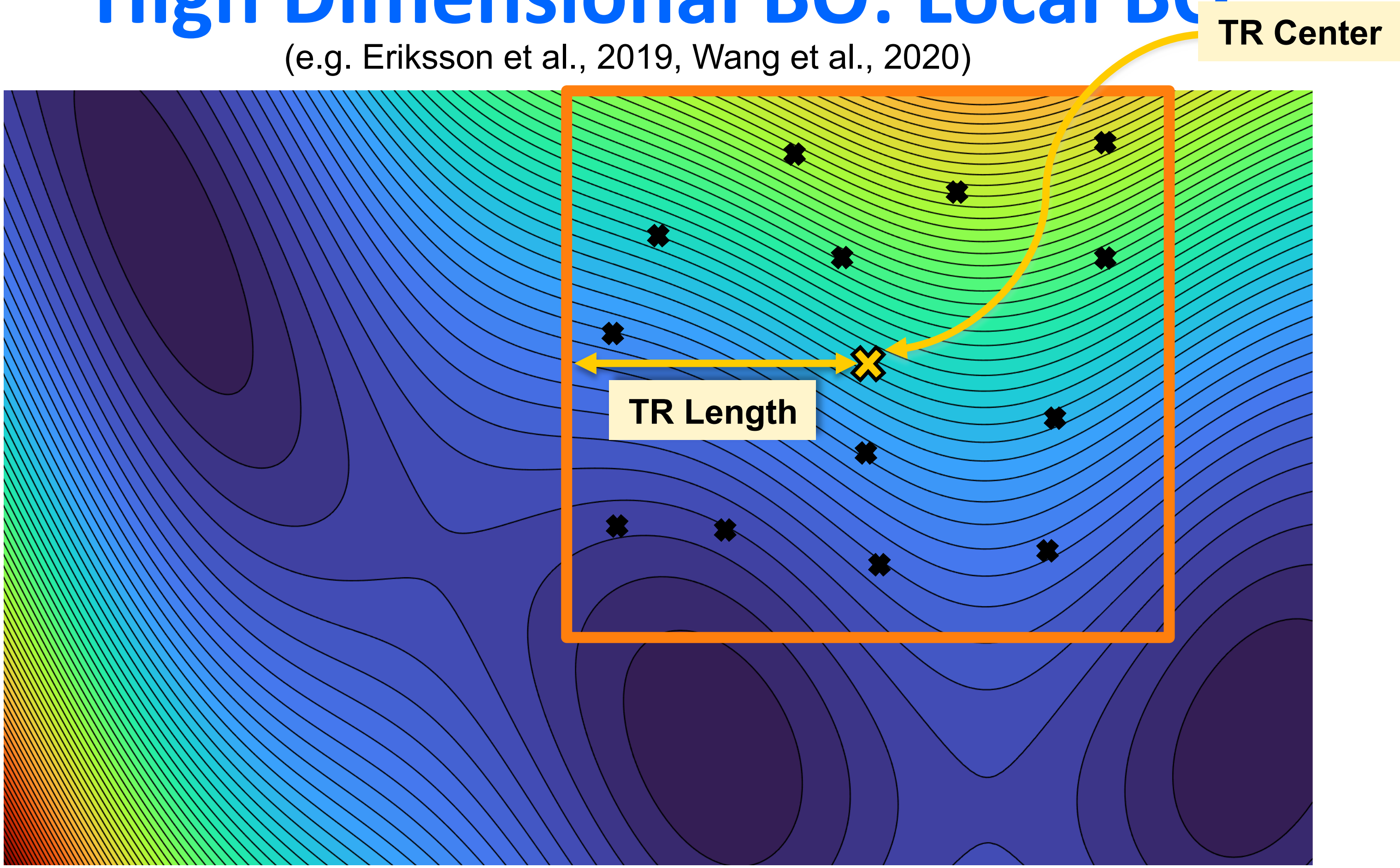
(e.g. Eriksson et al., 2019, Wang et al., 2020)



Idea: Perform BO inside a *trust region* (TuRBO)

High Dimensional BO: Local BO

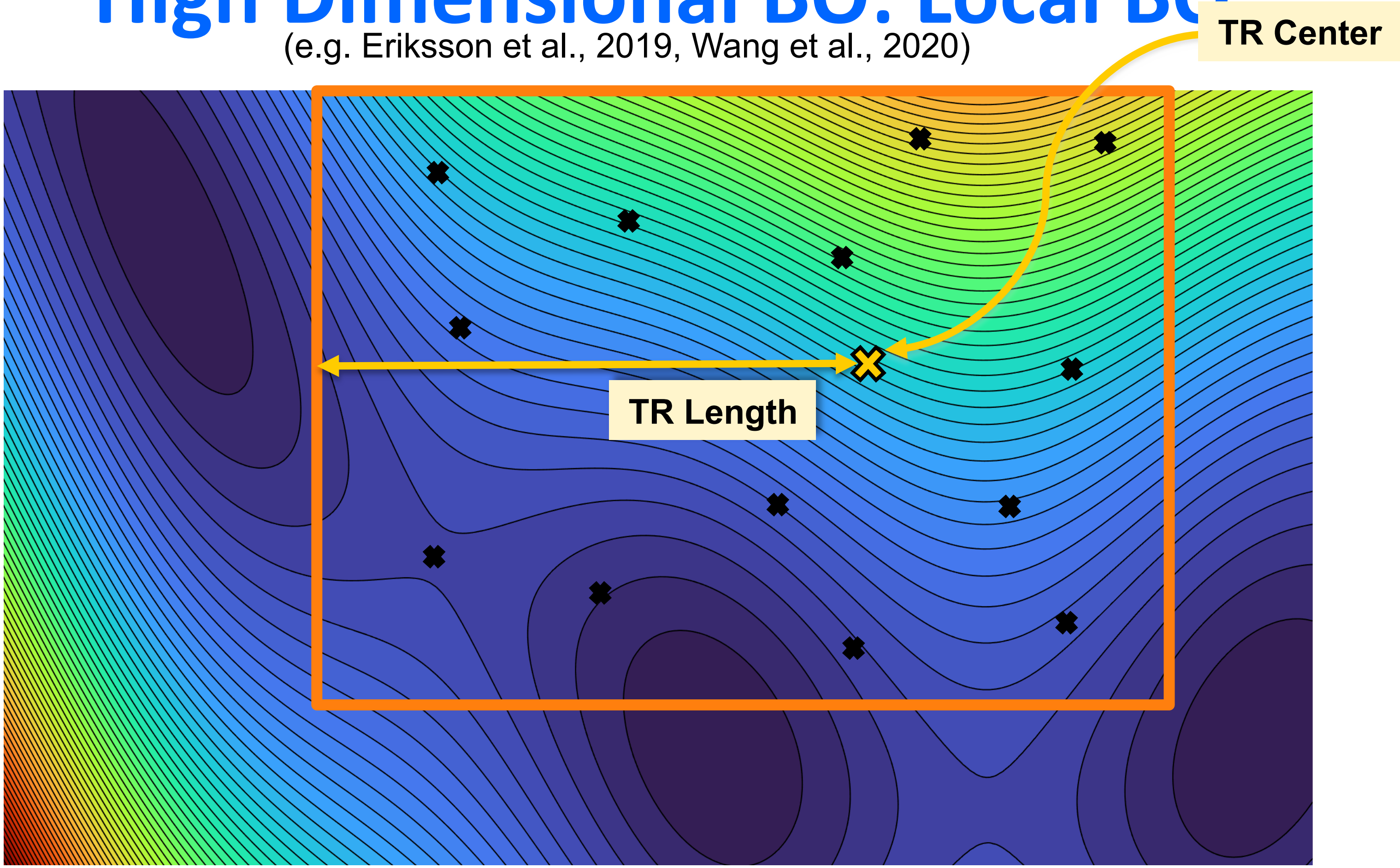
(e.g. Eriksson et al., 2019, Wang et al., 2020)



Idea: Perform BO inside a *trust region* (TuRBO)

High Dimensional BO: Local BO

(e.g. Eriksson et al., 2019, Wang et al., 2020)

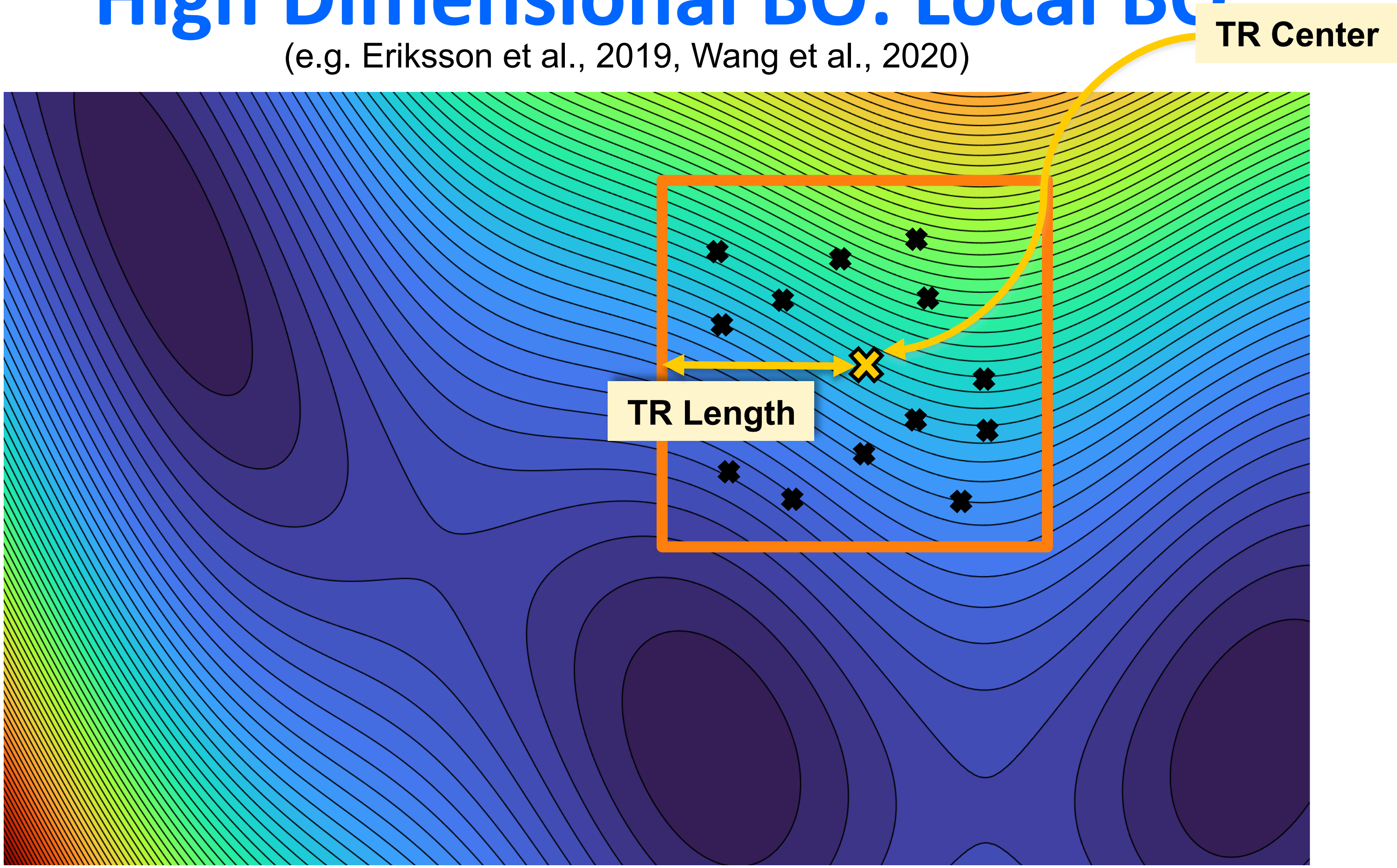


Idea: Perform BO inside a *trust region* (TuRBO)

Op #1: Grow the TR

High Dimensional BO: Local BO

(e.g. Eriksson et al., 2019, Wang et al., 2020)



TR Center

TR Length

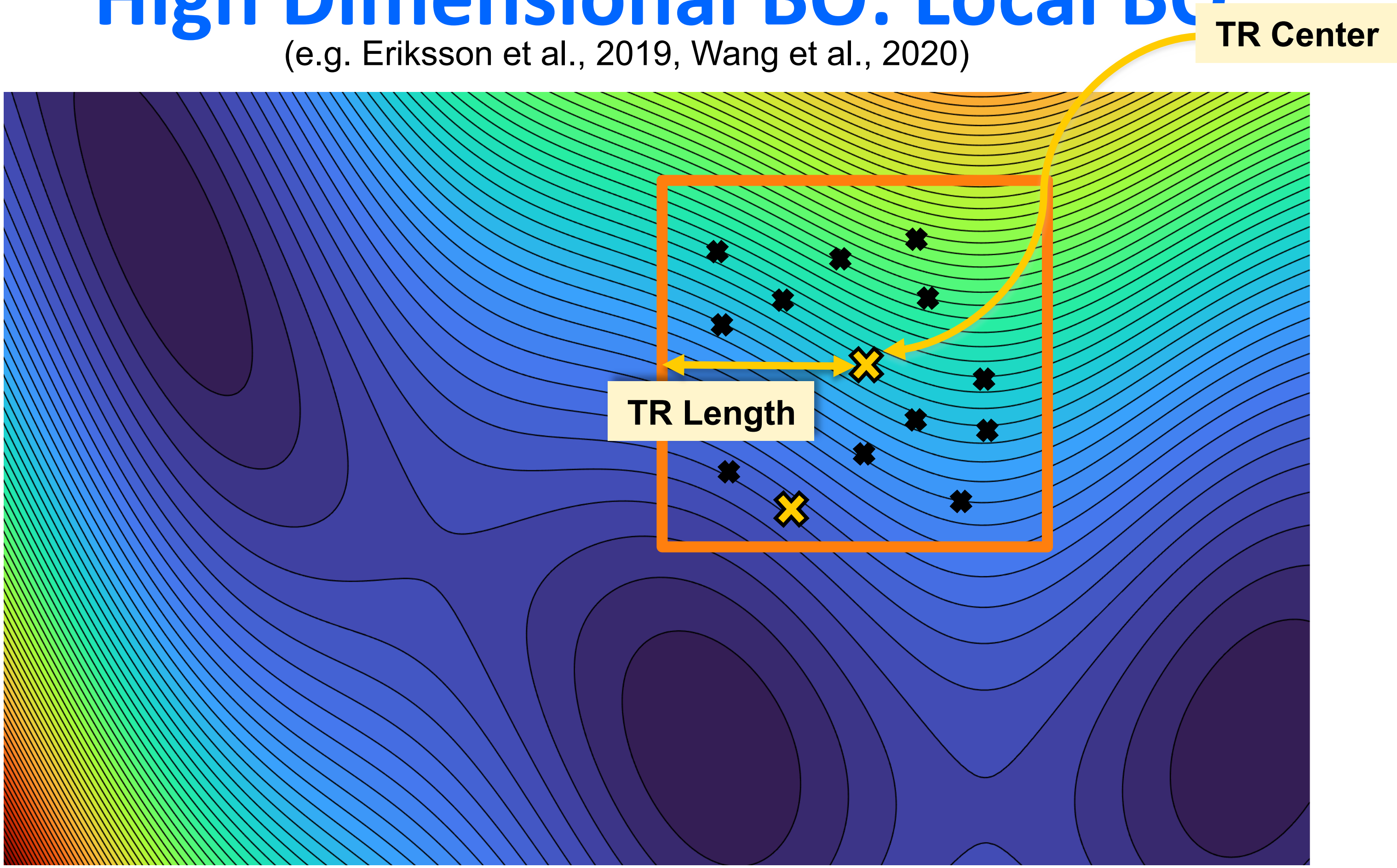
Idea: Perform BO inside a *trust region* (TuRBO)

Op #1: Grow the TR

Op #2: Shrink the TR

High Dimensional BO: Local BO

(e.g. Eriksson et al., 2019, Wang et al., 2020)

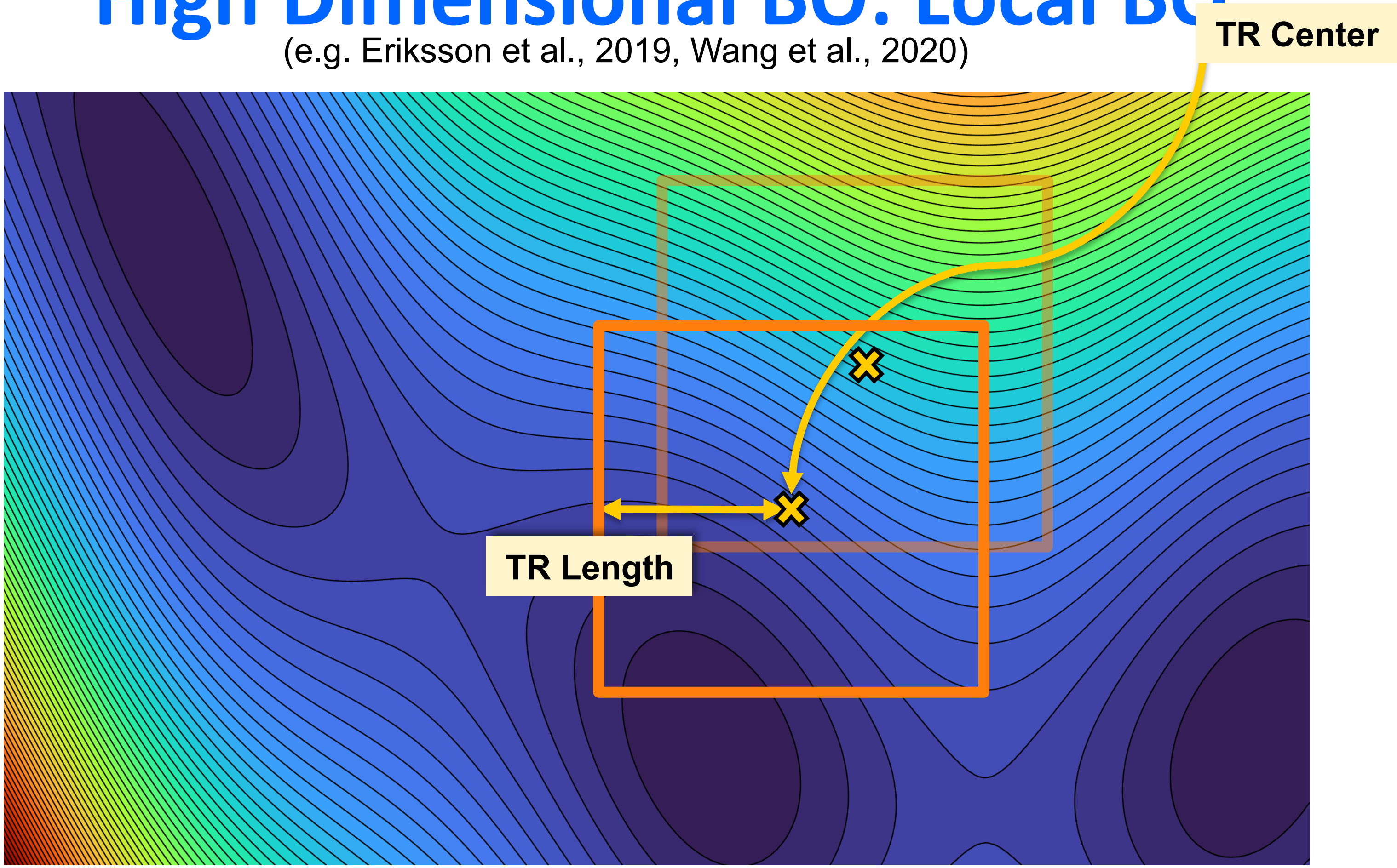


Idea: Perform BO inside a *trust region* (TuRBO)

Op #1: Grow the TR **Op #2: Shrink the TR**

High Dimensional BO: Local BO

(e.g. Eriksson et al., 2019, Wang et al., 2020)

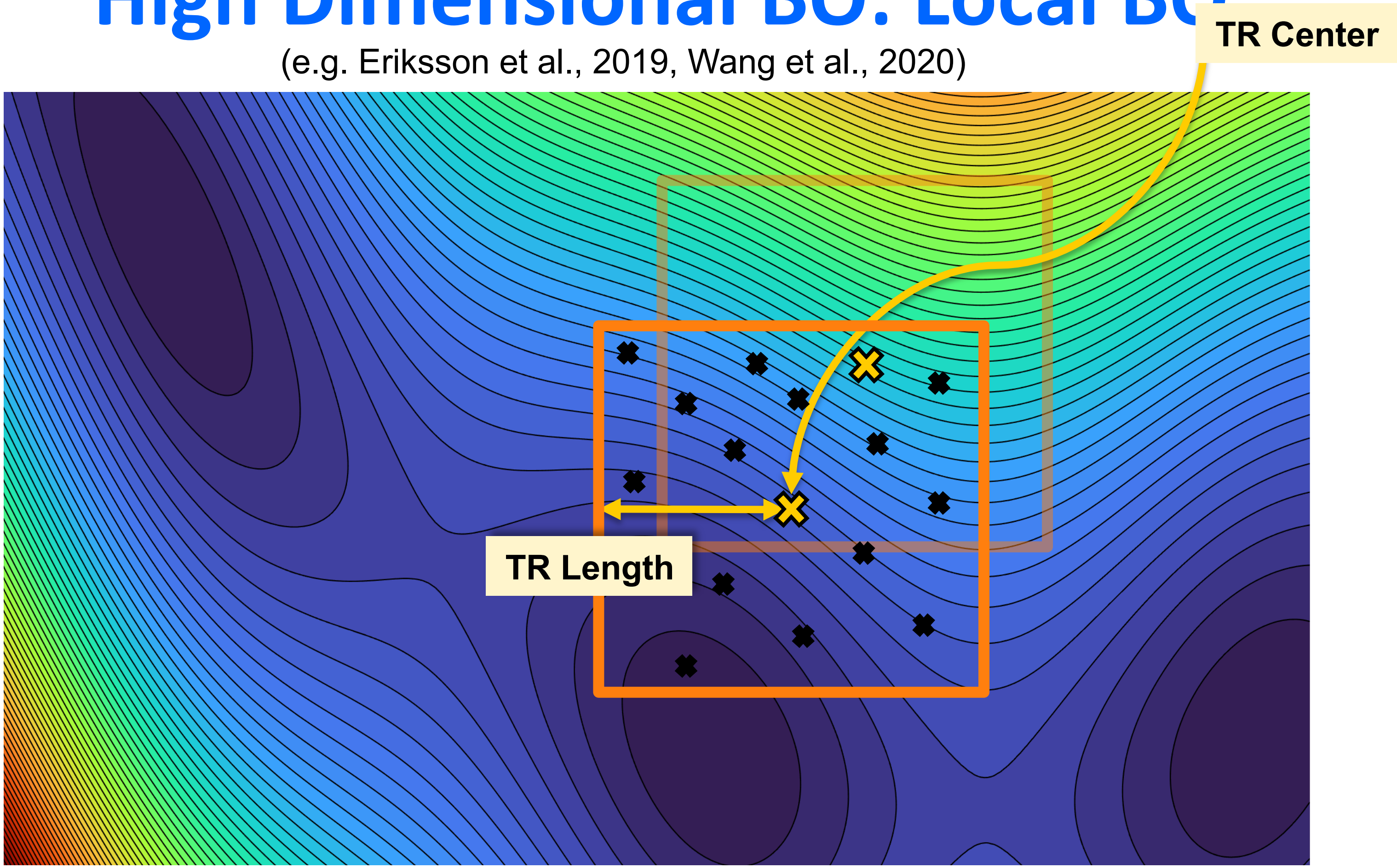


Idea: Perform BO inside a *trust region* (TuRBO)

Op #1: Grow the TR **Op #2: Shrink the TR** **Op #3: Move the TR**

High Dimensional BO: Local BO

(e.g. Eriksson et al., 2019, Wang et al., 2020)

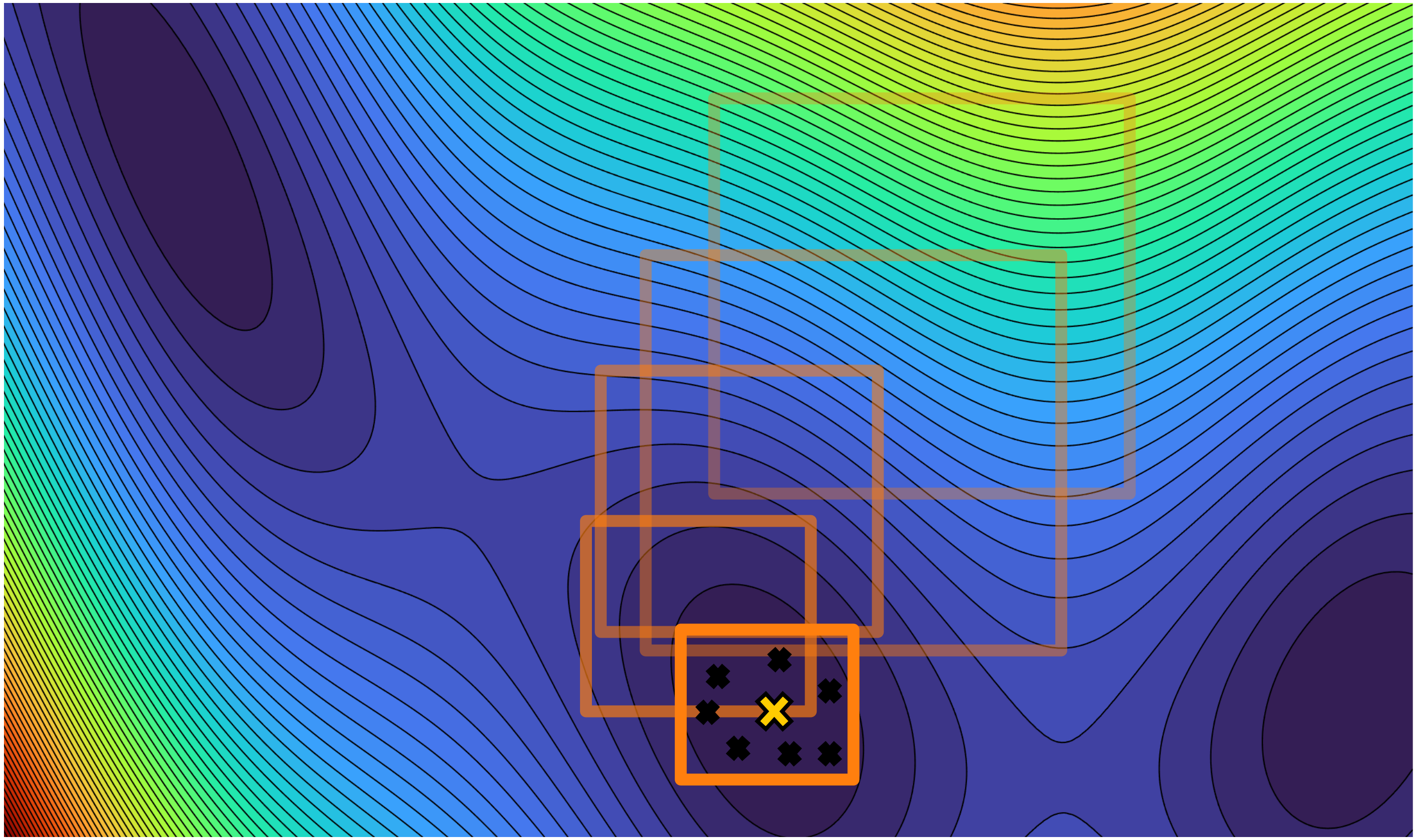


Idea: Perform BO inside a *trust region* (TuRBO)

Op #1: Grow the TR **Op #2: Shrink the TR** **Op #3: Move the TR**

High Dimensional BO: Local BO

(e.g. Eriksson et al., 2019, Wang et al., 2020)



Idea: Perform BO inside a *trust region* (TuRBO)

Op #1: Grow the TR

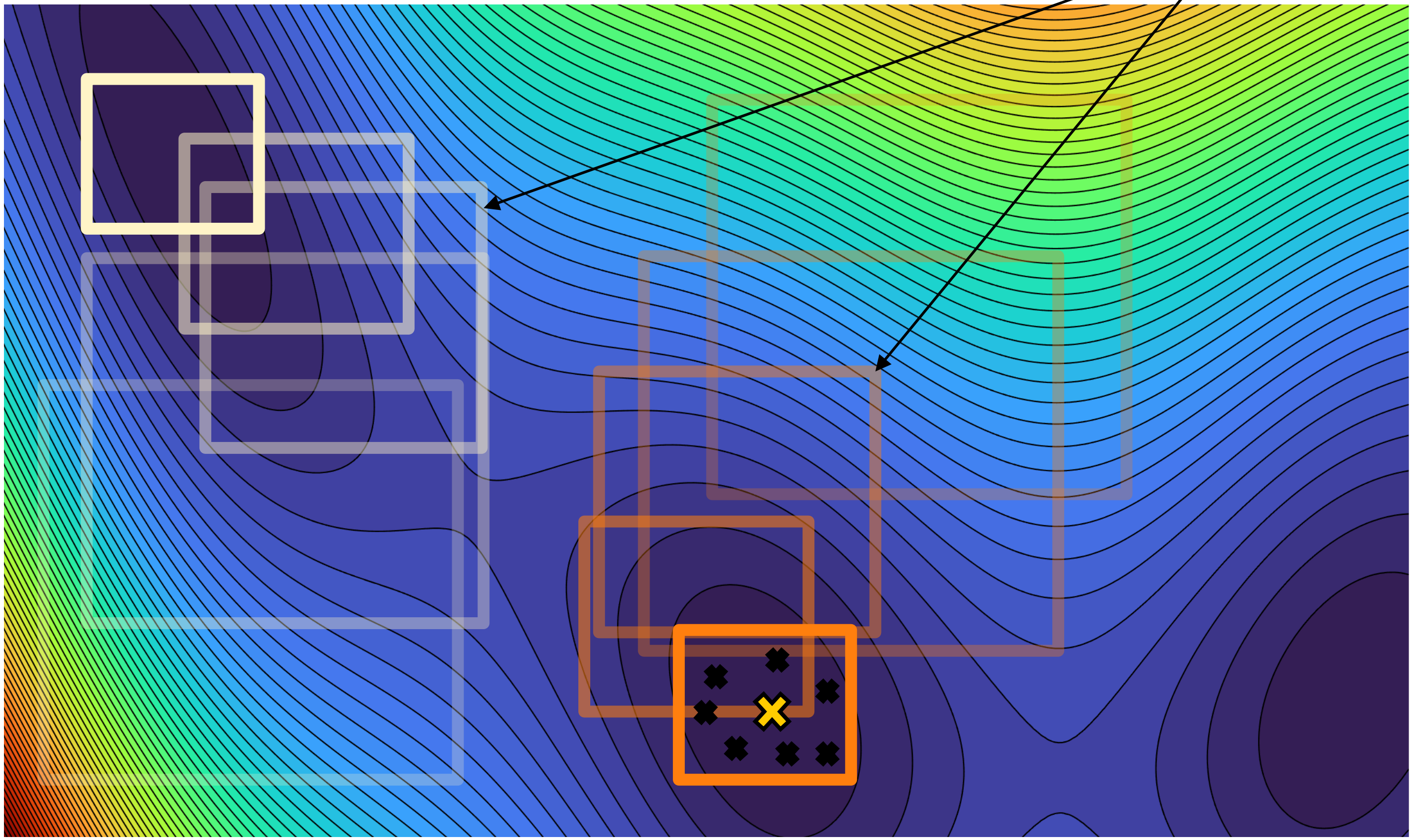
Op #2: Shrink the TR

Op #3: Move the TR

High Dimensional BO: Local BO

(e.g. Eriksson et al., 2019, Wang et al., 2020)

Consider bandit / MCTS over multiple TRs



Idea: Perform BO inside a *trust region* (TuRBO)

Op #1: Grow the TR

Op #2: Shrink the TR

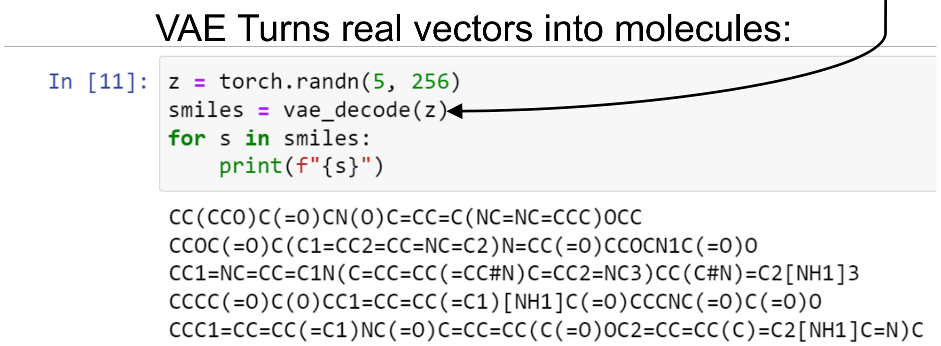
Op #3: Move the TR

GPyTorch + BoTorch Demo: TuRBO + LS-BO

GPyTorch + BoTorch Demo: TuRBO + LS-BO

1. Handling the “LS” bit.

Decode from PyTorch VAE Model



VAE Turns real vectors into molecules:

```
In [11]: z = torch.randn(5, 256)
         smiles = vae_decode(z)
         for s in smiles:
             print(f"{s}")
```

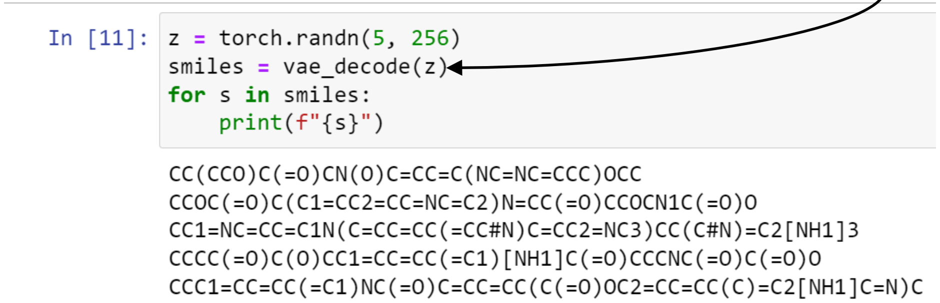
```
CC(CCO)C(=O)CN(O)C=CC=C(NC=NC=CCC)OCC
CCOC(=O)C(C1=CC2=CC=NC=C2)N=CC(=O)CCOCN1C(=O)O
CC1=NC=CC=C1N(C=CC=CC(=CC#N)C=CC2=NC3)CC(C#N)=C2[NH1]3
CCCC(=O)C(O)CC1=CC=CC(=C1)[NH1]C(=O)CCCNC(=O)C(=O)O
CCC1=CC=CC(=C1)NC(=O)C=CC=CC(C(=O)OC2=CC=CC(C)=C2[NH1]C=N)C
```

GPyTorch + BoTorch Demo: TuRBO + LS-BO

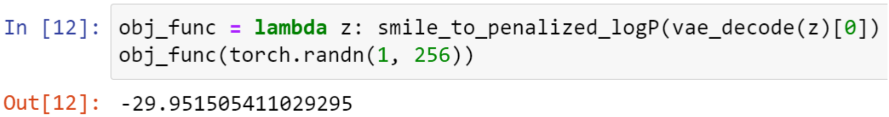
1. Handling the “LS” bit.

Decode from PyTorch VAE Model

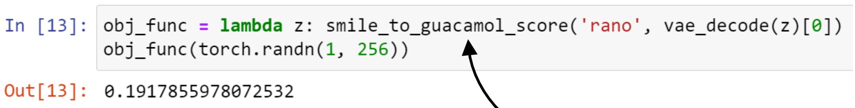
VAE Turns real vectors into molecules:



```
In [11]: z = torch.randn(5, 256)
         smiles = vae_decode(z)
         for s in smiles:
             print(f"{s}")

         CC(CCO)C(=O)CN(O)C=CC=C(NC=NC=CCC)OCC
         CCOC(=O)C(C1=CC2=CC=NC=C2)N=CC(=O)CCOCN1C(=O)O
         CC1=NC=CC=C1N(C=CC=CC(=CC#N)C=CC2=NC3)CC(C#N)=C2[NH1]3
         CCCC(=O)C(O)CC1=CC=CC(=C1)[NH1]C(=O)CCCNC(=O)C(=O)O
         CCC1=CC=CC(=C1)NC(=O)C=CC=CC(C(=O)OC2=CC=CC(C)=C2[NH1]C=N)C
```



```
In [12]: obj_func = lambda z: smile_to_penalized_logP(vae_decode(z)[0])
         obj_func(torch.randn(1, 256))

Out[12]: -29.951505411029295
```



```
In [13]: obj_func = lambda z: smile_to_guacamol_score('rano', vae_decode(z)[0])
         obj_func(torch.randn(1, 256))

Out[13]: 0.1917855978072532
```

Brown et al., 2019

Take away: The “LS” part is pretty much done for you. If that’s what you want.

GPyTorch + BoTorch Demo: TuRBO + LS-BO

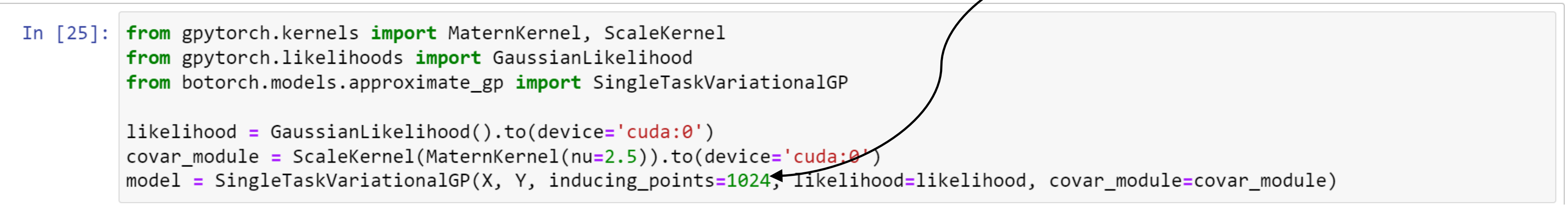
1. Handling the “LS” bit.
2. Train a surrogate model

GPyTorch + BoTorch Demo: TuRBO + LS-BO

1. **Handling the “LS” bit.**
2. **Train a surrogate model**

Pre-baked BoTorch model:

Inducing points initialized using Pivoted Cholesky (Burt et al., 2020)



In [25]:

```
from gpytorch.kernels import MaternKernel, ScaleKernel
from gpytorch.likelihoods import GaussianLikelihood
from botorch.models.approximate_gp import SingleTaskVariationalGP

likelihood = GaussianLikelihood().to(device='cuda:0')
covar_module = ScaleKernel(MaternKernel(nu=2.5)).to(device='cuda:0')
model = SingleTaskVariationalGP(X, Y, inducing_points=1024, likelihood=likelihood, covar_module=covar_module)
```

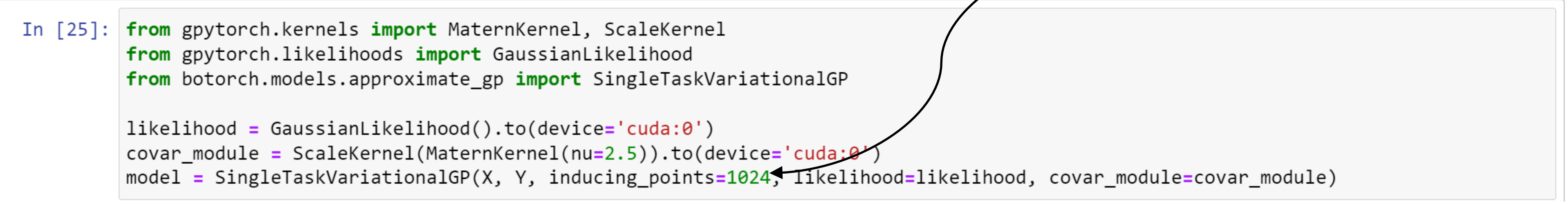
GPyTorch + BoTorch Demo: TuRBO + LS-BO

1. Handling the "LS" bit.

2. Train a surrogate model

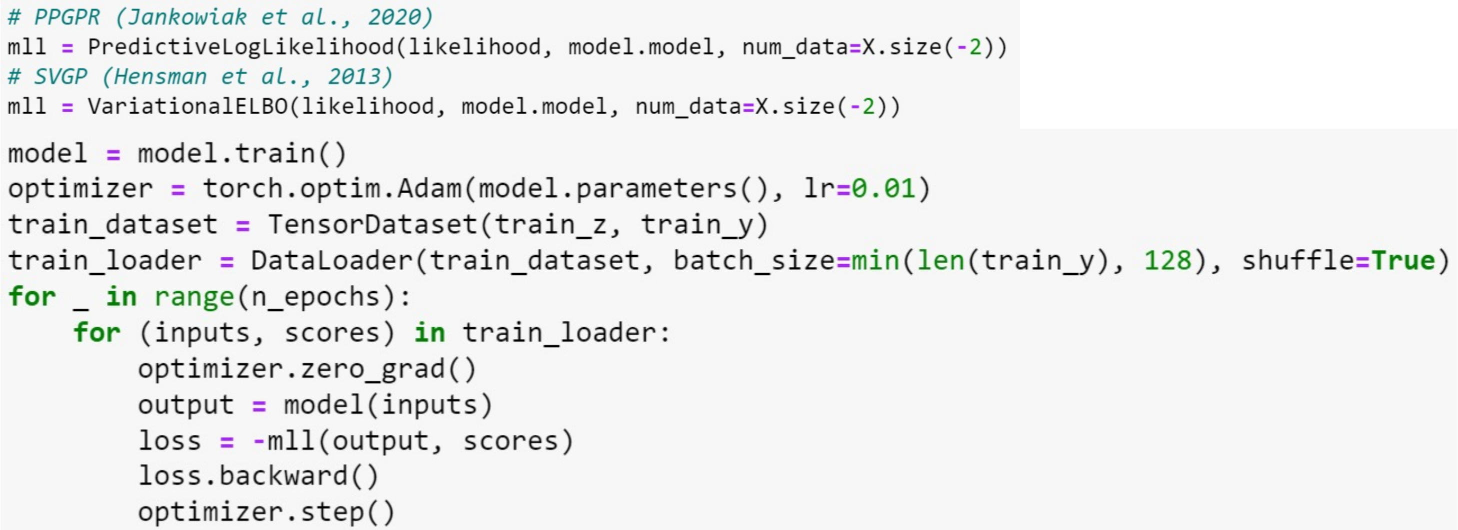
Pre-baked BoTorch model:

Inducing points initialized using Pivoted Cholesky (Burt et al., 2020)



```
In [25]: from gpytorch.kernels import MaternKernel, ScaleKernel
         from gpytorch.likelihoods import GaussianLikelihood
         from botorch.models.approximate_gp import SingleTaskVariationalGP

         likelihood = GaussianLikelihood().to(device='cuda:0')
         covar_module = ScaleKernel(MaternKernel(nu=2.5)).to(device='cuda:0')
         model = SingleTaskVariationalGP(X, Y, inducing_points=1024, likelihood=likelihood, covar_module=covar_module)
```

Training:

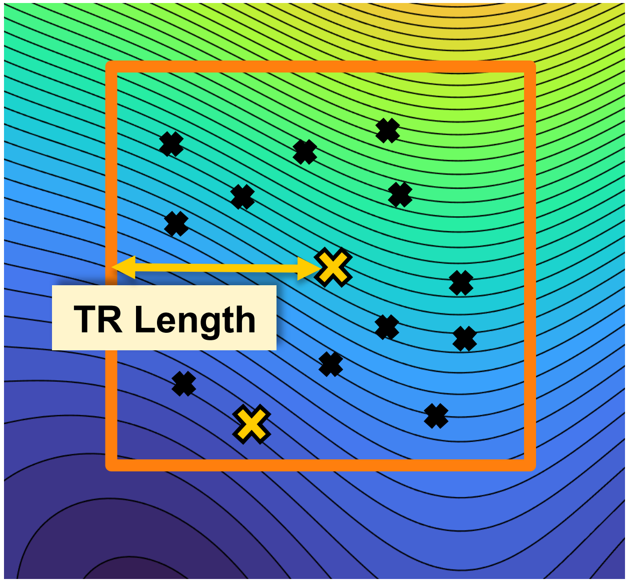


```
# PPGPR (Jankowiak et al., 2020)
mll = PredictiveLogLikelihood(likelihood, model.model, num_data=X.size(-2))
# SVGP (Hensman et al., 2013)
mll = VariationalELBO(likelihood, model.model, num_data=X.size(-2))
model = model.train()
optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
train_dataset = TensorDataset(train_z, train_y)
train_loader = DataLoader(train_dataset, batch_size=min(len(train_y), 128), shuffle=True)
for _ in range(n_epochs):
    for (inputs, scores) in train_loader:
        optimizer.zero_grad()
        output = model(inputs)
        loss = -mll(output, scores)
        loss.backward()
        optimizer.step()
```

(Soon: https://github.com/pytorch/botorch/pull/1439/)

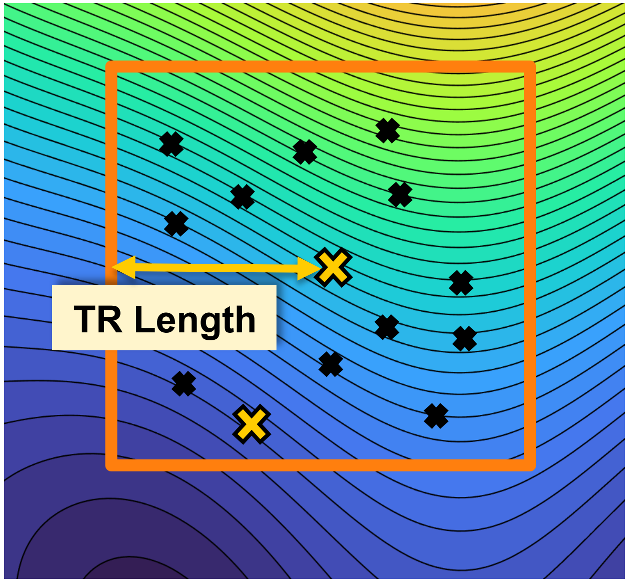
GPyTorch + BoTorch Demo: TuRBO + LS-BO

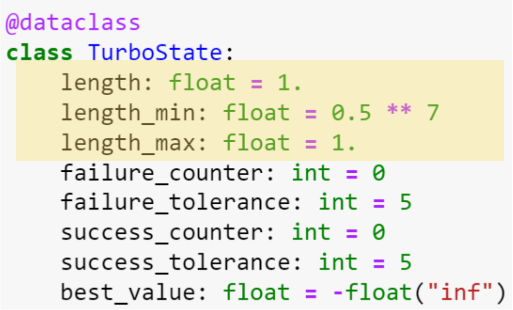
1. Handling the “LS” bit.
2. Train a surrogate model
3. Trust region state



GPyTorch + BoTorch Demo: TuRBO + LS-BO

1. Handling the “LS” bit.
2. Train a surrogate model
3. Trust region state



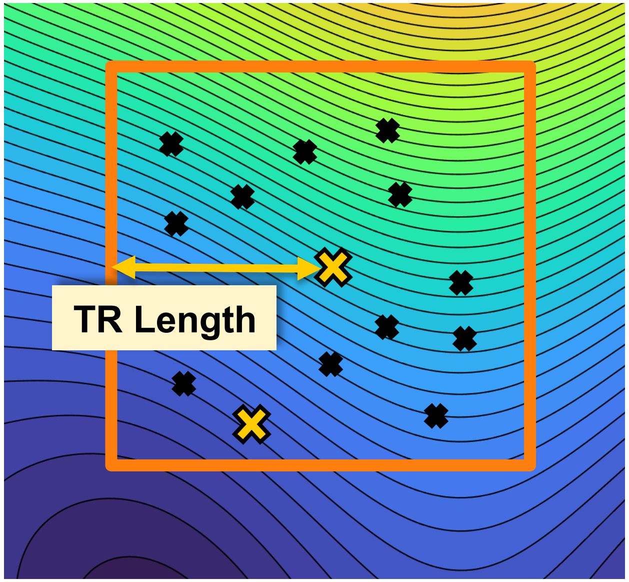


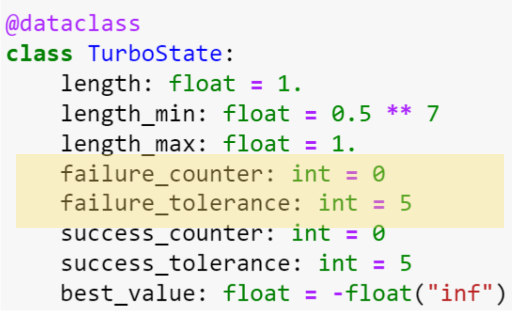
```python
@dataclass
class TurboState:
    length: float = 1.
    length_min: float = 0.5 ** 7
    length_max: float = 1.
    failure_counter: int = 0
    failure_tolerance: int = 5
    success_counter: int = 0
    success_tolerance: int = 5
    best_value: float = -float("inf")
```

TR length $\in [0, 1]$
(× some fixed init. Length)

GPyTorch + BoTorch Demo: TuRBO + LS-BO

1. **Handling the "LS" bit.**
2. **Train a surrogate model**
3. **Trust region state**



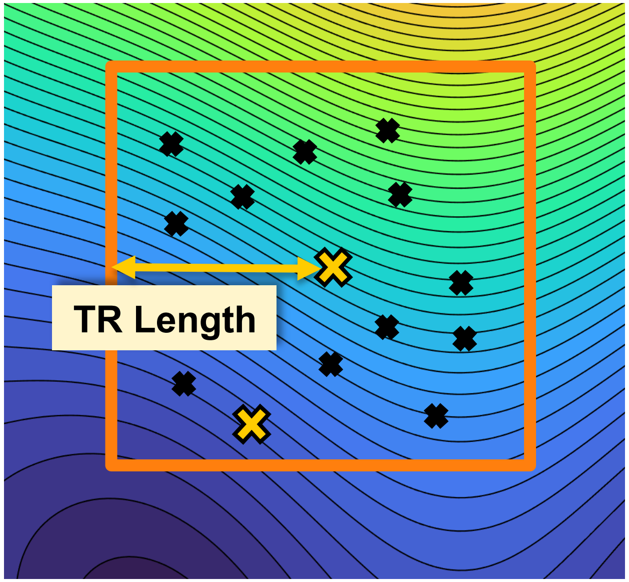


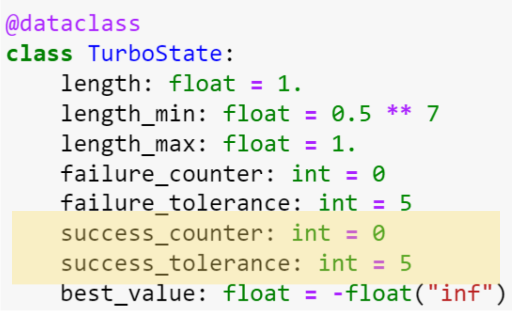
```
@dataclass
class TurboState:
    length: float = 1.
    length_min: float = 0.5 ** 7
    length_max: float = 1.
    failure_counter: int = 0
    failure_tolerance: int = 5
    success_counter: int = 0
    success_tolerance: int = 5
    best_value: float = -float("inf")
```

Shrink TR if we fail to make progress 5 times in a row.

GPyTorch + BoTorch Demo: TuRBO + LS-BO

1. **Handling the “LS” bit.**
2. **Train a surrogate model**
3. **Trust region state**



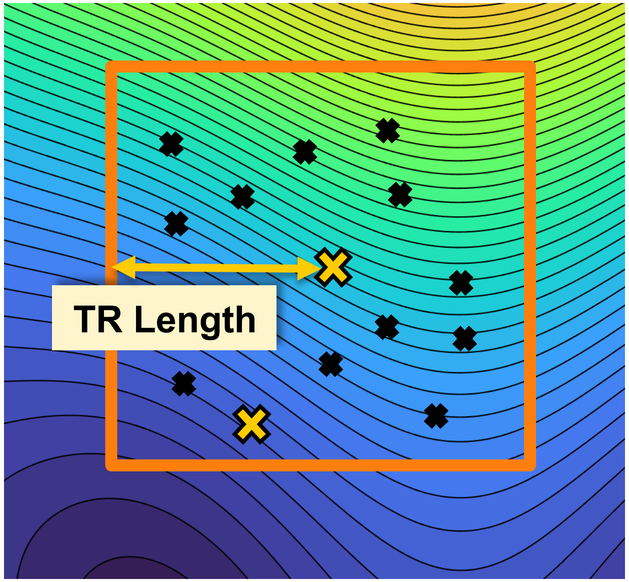


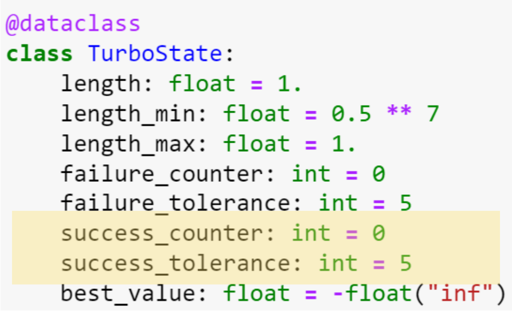
```python
@dataclass
class TurboState:
    length: float = 1.
    length_min: float = 0.5 ** 7
    length_max: float = 1.
    failure_counter: int = 0
    failure_tolerance: int = 5
    success_counter: int = 0
    success_tolerance: int = 5
    best_value: float = -float("inf")
```

Grow TR if we make progress 5 times in a row.

GPyTorch + BoTorch Demo: TuRBO + LS-BO

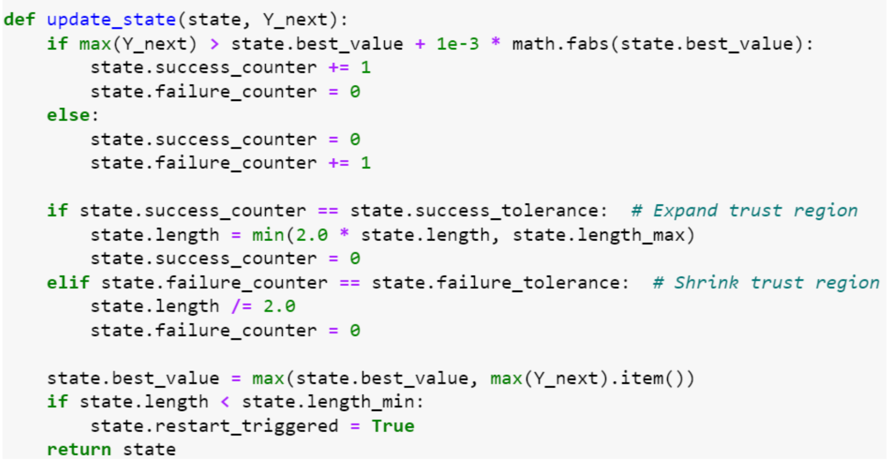
1. Handling the “LS” bit.
2. Train a surrogate model
3. Trust region state





```
@dataclass
class TurboState:
    length: float = 1.
    length_min: float = 0.5 ** 7
    length_max: float = 1.
    failure_counter: int = 0
    failure_tolerance: int = 5
    success_counter: int = 0
    success_tolerance: int = 5
    best_value: float = -float("inf")
```

Grow TR if we make progress 5 times in a row.



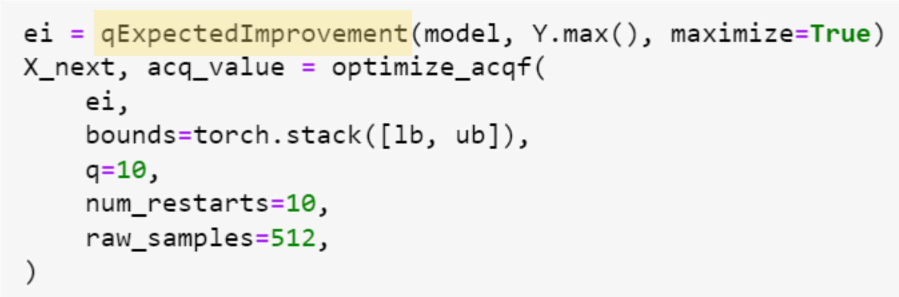
```
def update_state(state, Y_next):
    if max(Y_next) > state.best_value + 1e-3 * math.fabs(state.best_value):
        state.success_counter += 1
        state.failure_counter = 0
    else:
        state.success_counter = 0
        state.failure_counter += 1

    if state.success_counter == state.success_tolerance:  # Expand trust region
        state.length = min(2.0 * state.length, state.length_max)
        state.success_counter = 0
    elif state.failure_counter == state.failure_tolerance:  # Shrink trust region
        state.length /= 2.0
        state.failure_counter = 0

    state.best_value = max(state.best_value, max(Y_next).item())
    if state.length < state.length_min:
        state.restart_triggered = True
    return state
```

GPyTorch + BoTorch Demo: TuRBO + LS-BO

1. Handling the “LS” bit.
2. Train a surrogate model
3. Trust region state
4. Maximize an acquisition function

GPyTorch + BoTorch Demo: TuRBO + LS-BO

1. **Handling the “LS” bit.**
2. **Train a surrogate model**
3. **Trust region state**
4. **Maximize an acquisition function**

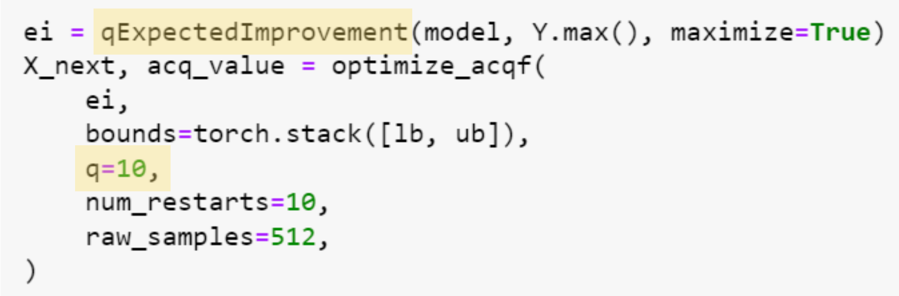


```
ei = qExpectedImprovement(model, Y.max(), maximize=True)
X_next, acq_value = optimize_acqf(
    ei,
    bounds=torch.stack([lb, ub]),
    q=10,
    num_restarts=10,
    raw_samples=512,
)
```

Monte-Carlo Acquisition functions (Wilson et al., 2018)

GPyTorch + BoTorch Demo: TuRBO + LS-BO

1. Handling the “LS” bit.
2. Train a surrogate model
3. Trust region state
4. Maximize an acquisition function



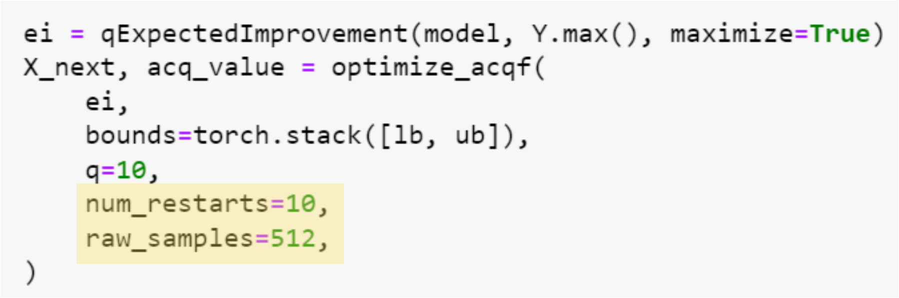
```
ei = qExpectedImprovement(model, Y.max(), maximize=True)
X_next, acq_value = optimize_acqf(
    ei,
    bounds=torch.stack([lb, ub]),
    q=10,
    num_restarts=10,
    raw_samples=512,
)
```

Monte-Carlo Acquisition functions (Wilson et al., 2018)

Batch size = 10

GPyTorch + BoTorch Demo: TuRBO + LS-BO

1. **Handling the “LS” bit.**
2. **Train a surrogate model**
3. **Trust region state**
4. **Maximize an acquisition function**



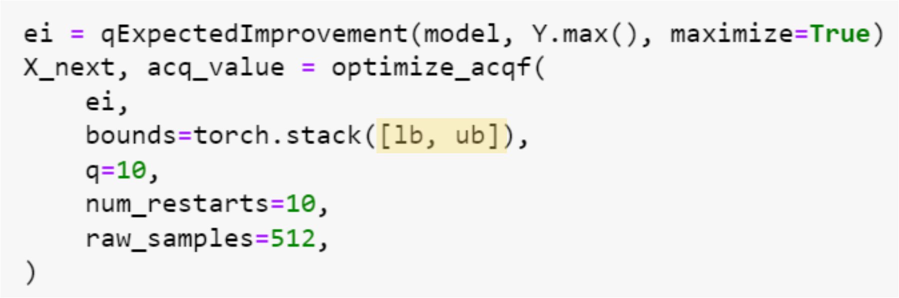
```
ei = qExpectedImprovement(model, Y.max(), maximize=True)
X_next, acq_value = optimize_acqf(
    ei,
    bounds=torch.stack([lb, ub]),
    q=10,
    num_restarts=10,
    raw_samples=512,
)
```

Run local optimization from 10 initial conditions.

ICs chosen from among 512 Sobol samples.

GPyTorch + BoTorch Demo: TuRBO + LS-BO

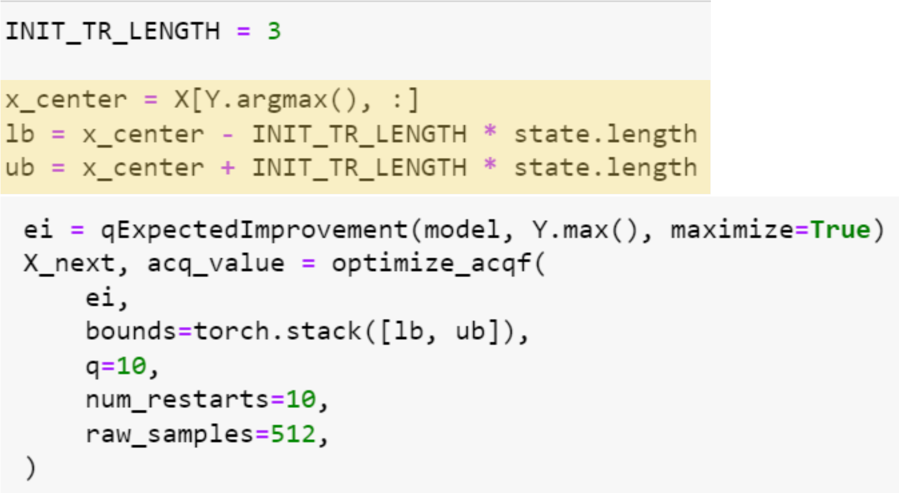
1. Handling the “LS” bit.
2. Train a surrogate model
3. Trust region state
4. Maximize an acquisition function



```
ei = qExpectedImprovement(model, Y.max(), maximize=True)
X_next, acq_value = optimize_acqf(
    ei,
    bounds=torch.stack([lb, ub]),
    q=10,
    num_restarts=10,
    raw_samples=512,
)
```

GPyTorch + BoTorch Demo: TuRBO + LS-BO

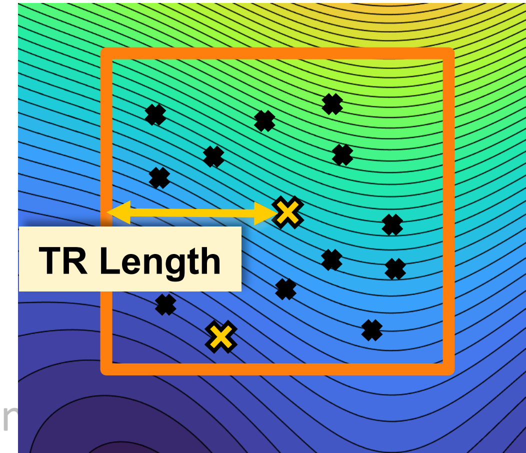
1. Handling the “LS” bit.
2. Train a surrogate model
3. Trust region state
4. Maximize an acquisition function



```
INIT_TR_LENGTH = 3
```

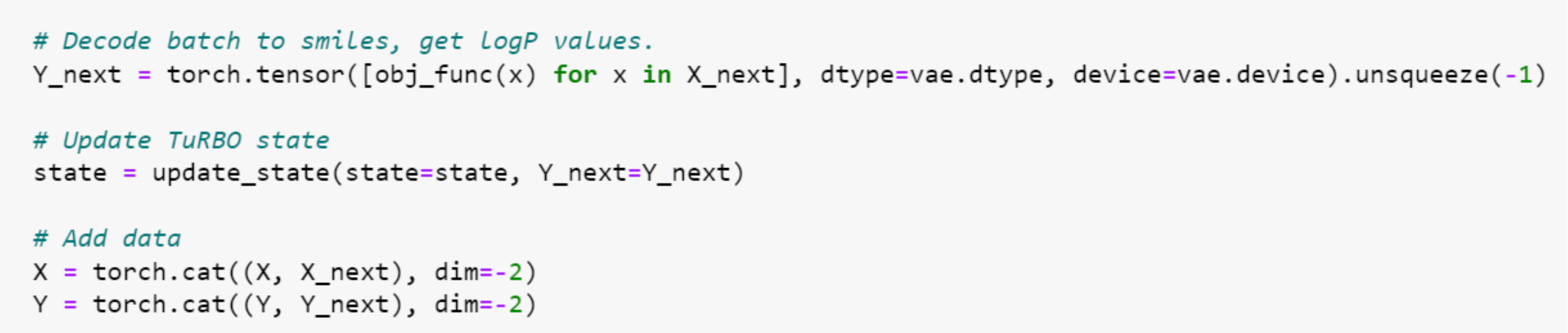
```
x_center = X[Y.argmax(), :]
lb = x_center - INIT_TR_LENGTH * state.length
ub = x_center + INIT_TR_LENGTH * state.length
```

```
ei = qExpectedImprovement(model, Y.max(), maximize=True)
X_next, acq_value = optimize_acqf(
    ei,
    bounds=torch.stack([lb, ub]),
    q=10,
    num_restarts=10,
    raw_samples=512,
)
```



GPyTorch + BoTorch Demo: TuRBO + LS-BO

1. Handling the "LS" bit.
2. Train a surrogate model
3. Trust region state
4. Maximize an acquisition function
4. Update state

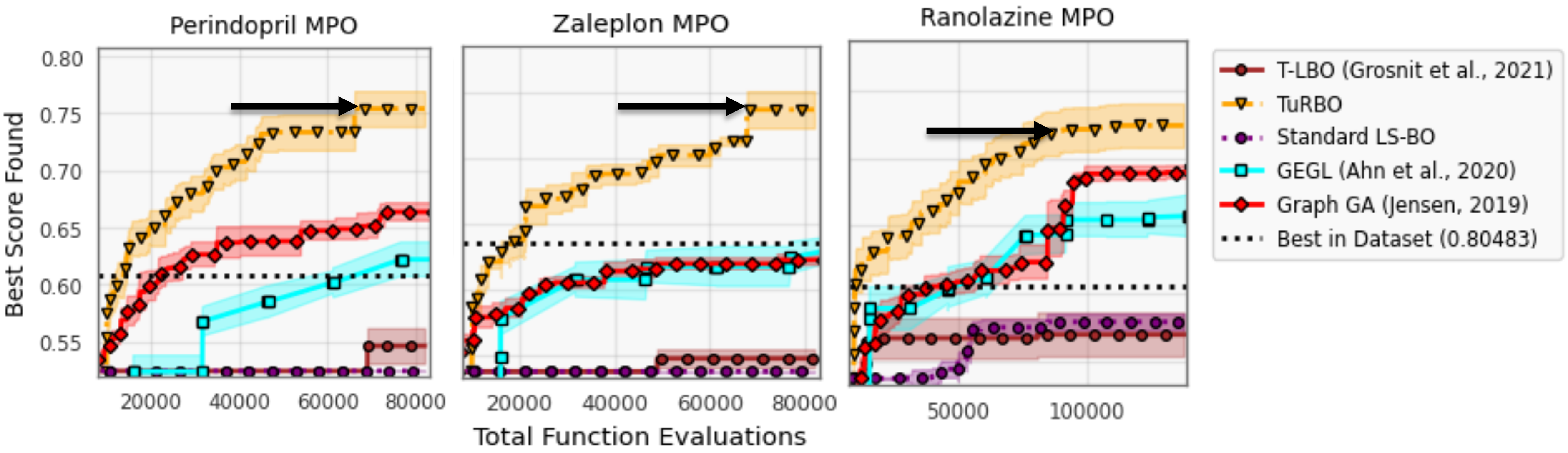


```
# Decode batch to smiles, get logP values.
Y_next = torch.tensor([obj_func(x) for x in X_next], dtype=vae.dtype, device=vae.device).unsqueeze(-1)

# Update TuRBO state
state = update_state(state=state, Y_next=Y_next)

# Add data
X = torch.cat((X, X_next), dim=-2)
Y = torch.cat((Y, Y_next), dim=-2)
```

Results

$\log P$: ~140 in 200-300 steps.

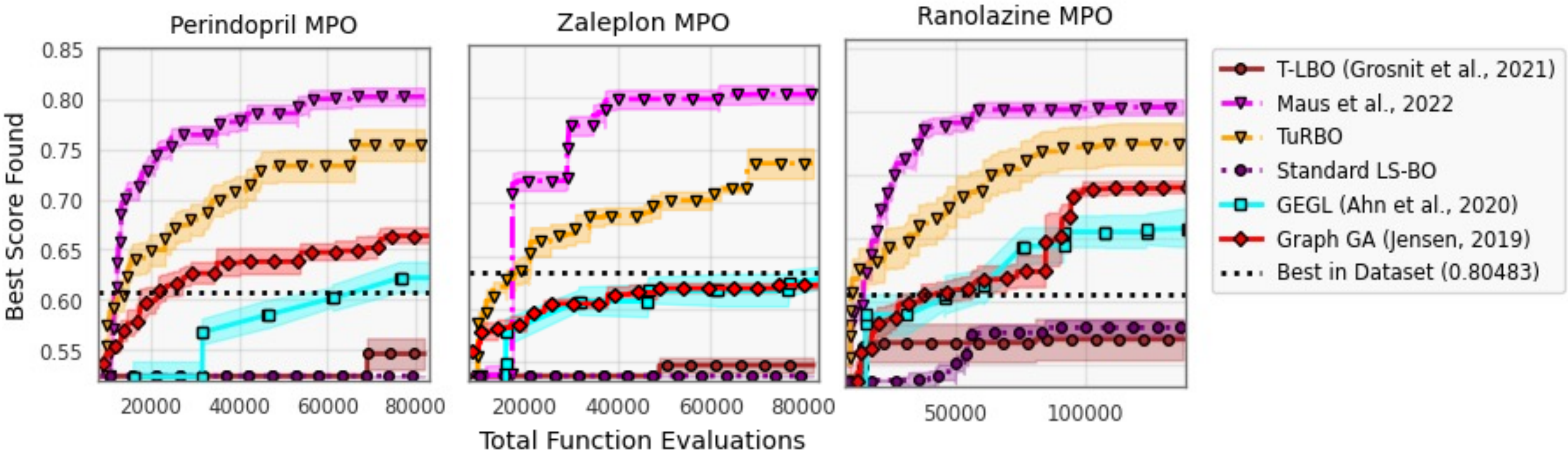
Guacamol:



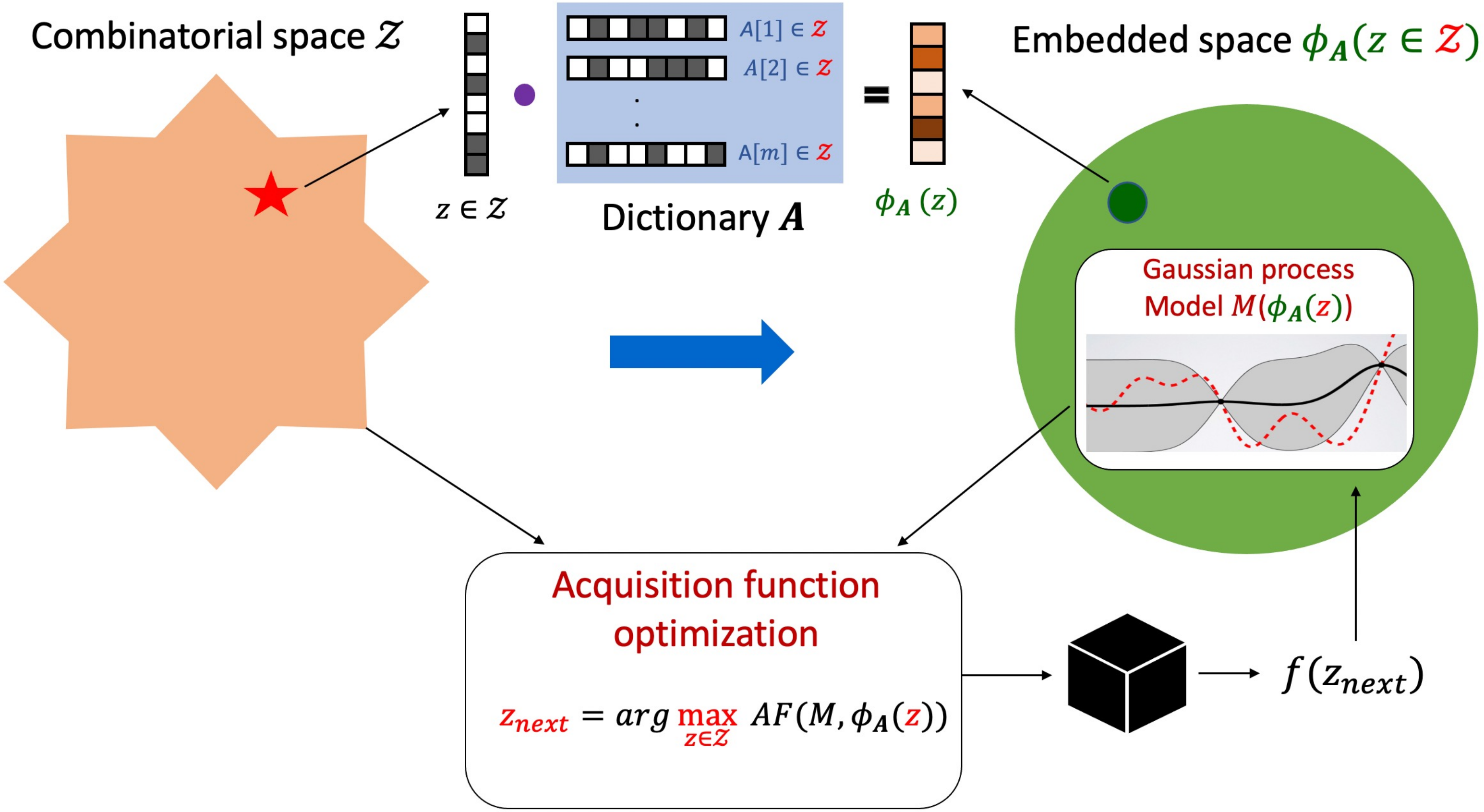
Results

$\log P$: ~140 in 200-300 steps.

Guacamol:



BODi: High-dimensional Combinatorial BO via Dictionary-based Embeddings



Deshwal et al., Bayesian Optimization over High-Dimensional Combinatorial Spaces via Dictionary based Randomized Continuous Embeddings, AISTATS-2023.

BODi: High-dimensional Combinatorial BO via Dictionary-based Embeddings

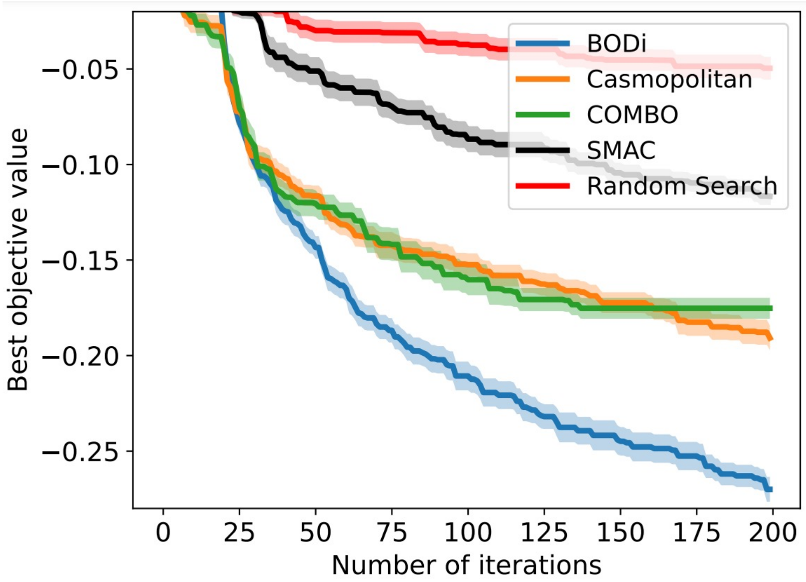
- $\chi \in \{0,1\}^d$ where d is large

- **Key Idea**
  - Embed inputs into low-dimensional embedding using a dictionary of combinatorial structures from χ
  - Allows us to use standard GP surrogate models with continuous kernels in the embedded space

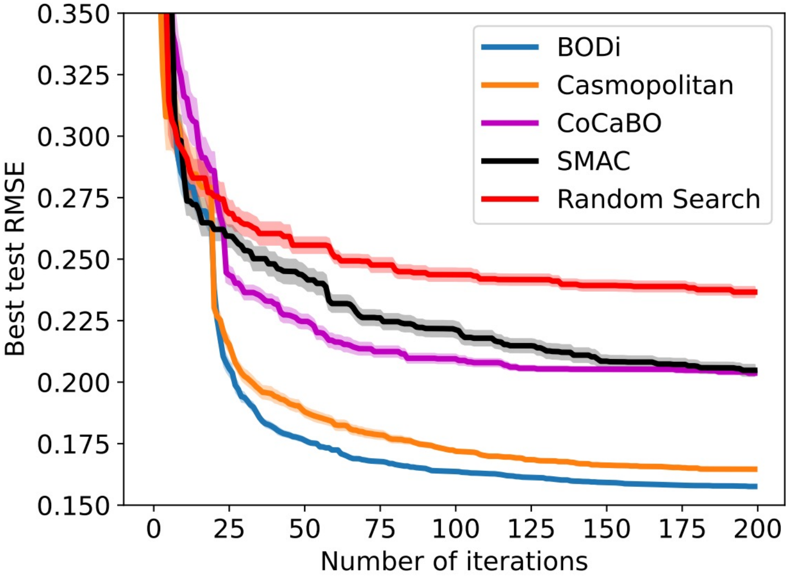
- Dictionary construction by maximizing diversity
  - Wavelet based design for binary spaces
  - Randomized approach for categorical spaces

BODi: Experimental Results

BODi consistently outperforms all state-of-the-art baselines



LABS: 60 dim binary space



Joint feature selection and hyper-parameter optimization for **SVM** training

50 dim binary x 3 dim continuous space

Outline of the Tutorial

- Overview of the BO Framework, GPs, advances in GPs and acquisition functions, and BoTorch demo

- Bayesian Optimization over Discrete/Hybrid Spaces

- Multi-fidelity Bayesian Optimization

30 mins Break

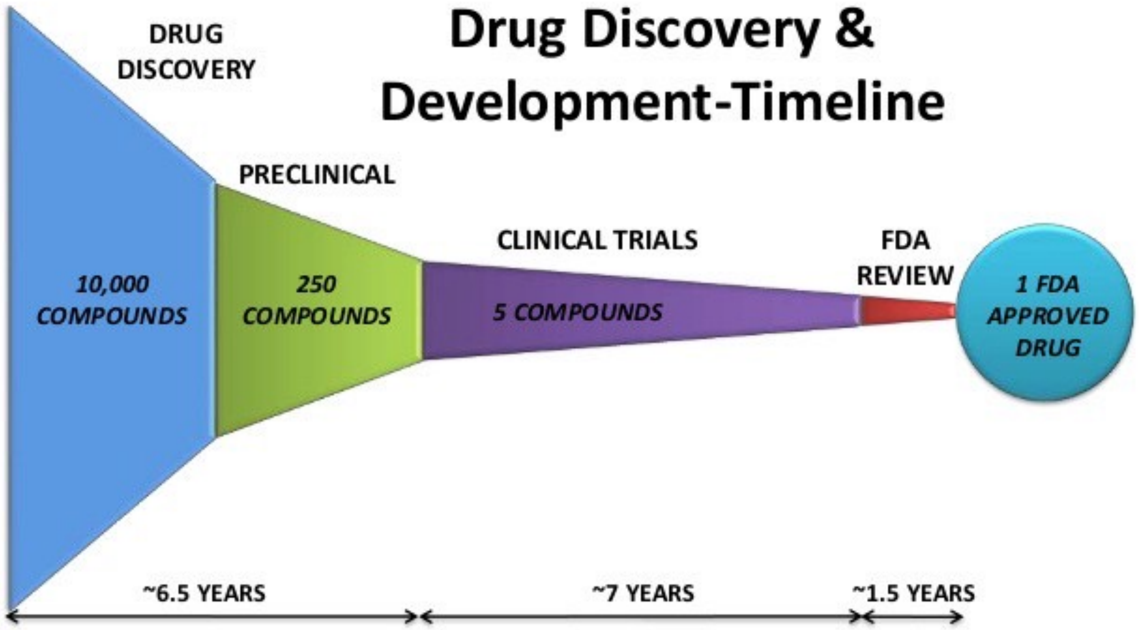
- High-Dimensional BO and BoTorch Hands-on demo

- Multi-Objective BO and BoTorch Hands-on demo

- Summary and Outstanding Challenges in BO

Multi-Objective Bayesian Optimization

Application #1: Drug/Vaccine Design

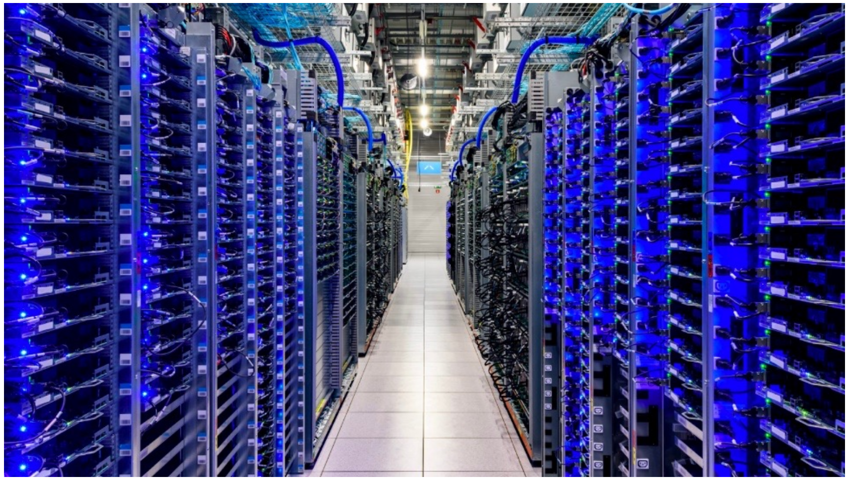


Credit: MIMA healthcare

- Effectiveness ↑
- Safety ↑
- Cost ↓

➤ Accelerate the discovery of promising designs

Application #2: Hardware Design for Datacenters

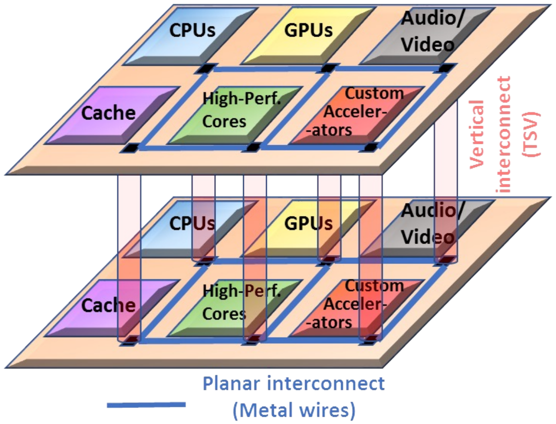


America’s Data Centers Are Wasting Huge Amounts of Energy

By 2020, data centers are projected to consume roughly 140 billion kilowatt-hours annually, costing American businesses $13 billion annually in electricity bills and emitting nearly 150 million metric tons of carbon pollution

Report from Natural Resources Defense Council:.
https://www.nrdc.org/sites/default/files/data-center-efficiency-assessment-IB.pdf

High-performance and Energy-efficient manycore chips

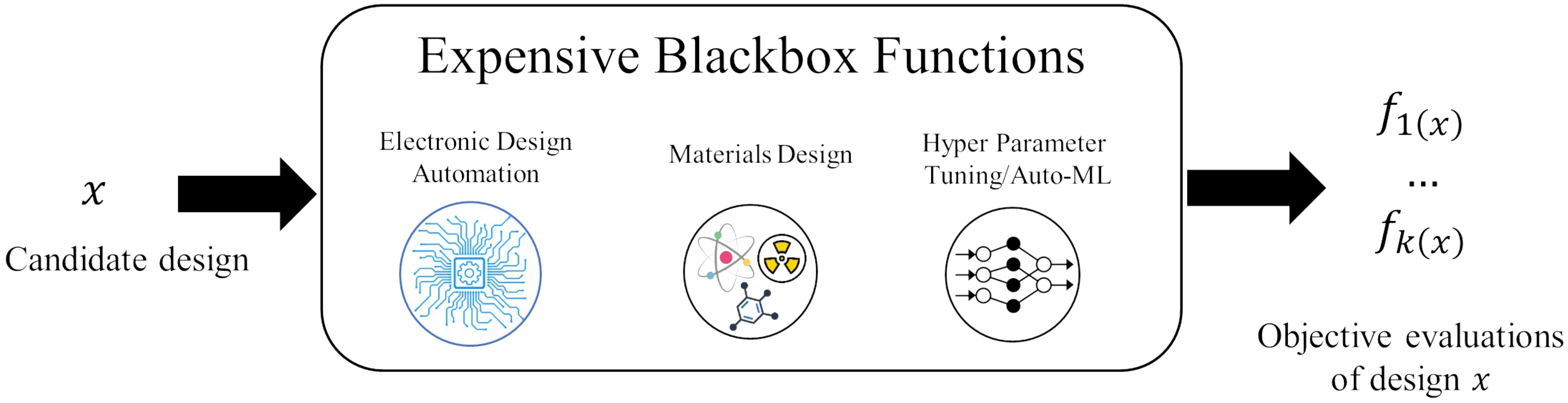


Performance ↑

Reliability ↑

Power ↓

Multi-Objective Optimization: The Problem



x

Candidate design

Expensive Blackbox Functions

Electronic Design Automation

Materials Design

Hyper Parameter Tuning/Auto-ML

$f_{1(x)}$

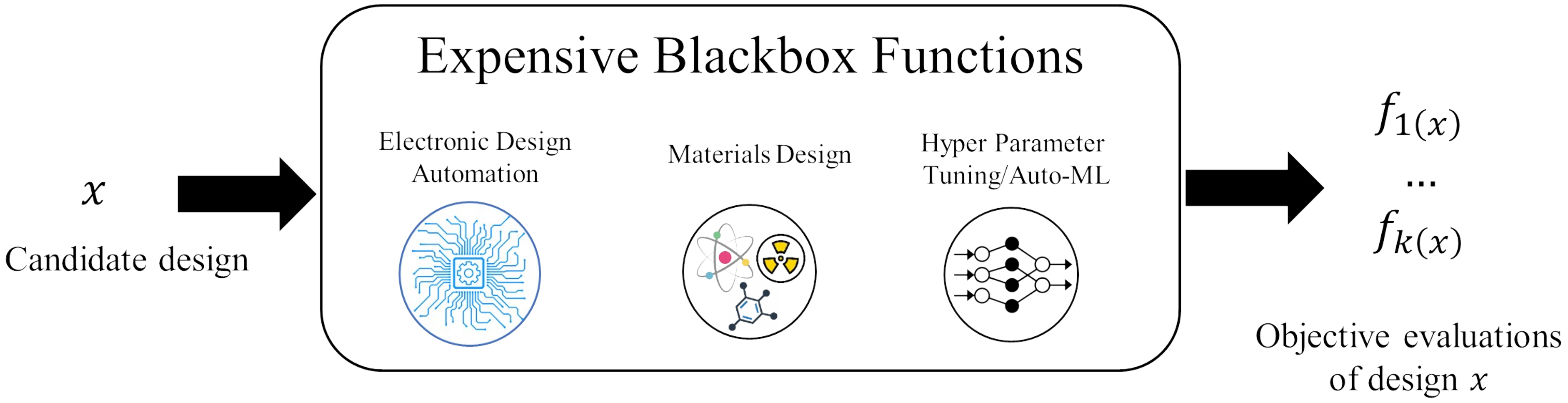
…

$f_{k(x)}$

Objective evaluations of design x

- **Goal**: Find designs with optimal trade-offs by minimizing the total resource cost of experiments

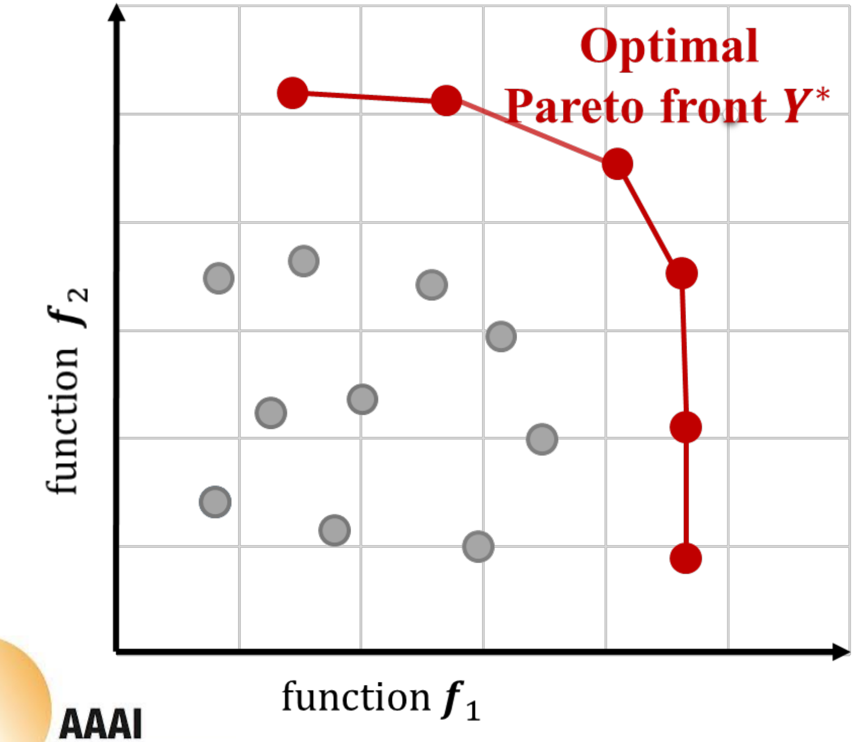
Multi-Objective Optimization: Key Challenge


x
Candidate design
Expensive Blackbox Functions
Electronic Design Automation
Materials Design
Hyper Parameter Tuning/Auto-ML
$f_1(x)$
…
$f_k(x)$
Objective evaluations of design x

- Challenge: Optimize multiple conflicting objective functions

Multi-Objective Optimization: The Solution

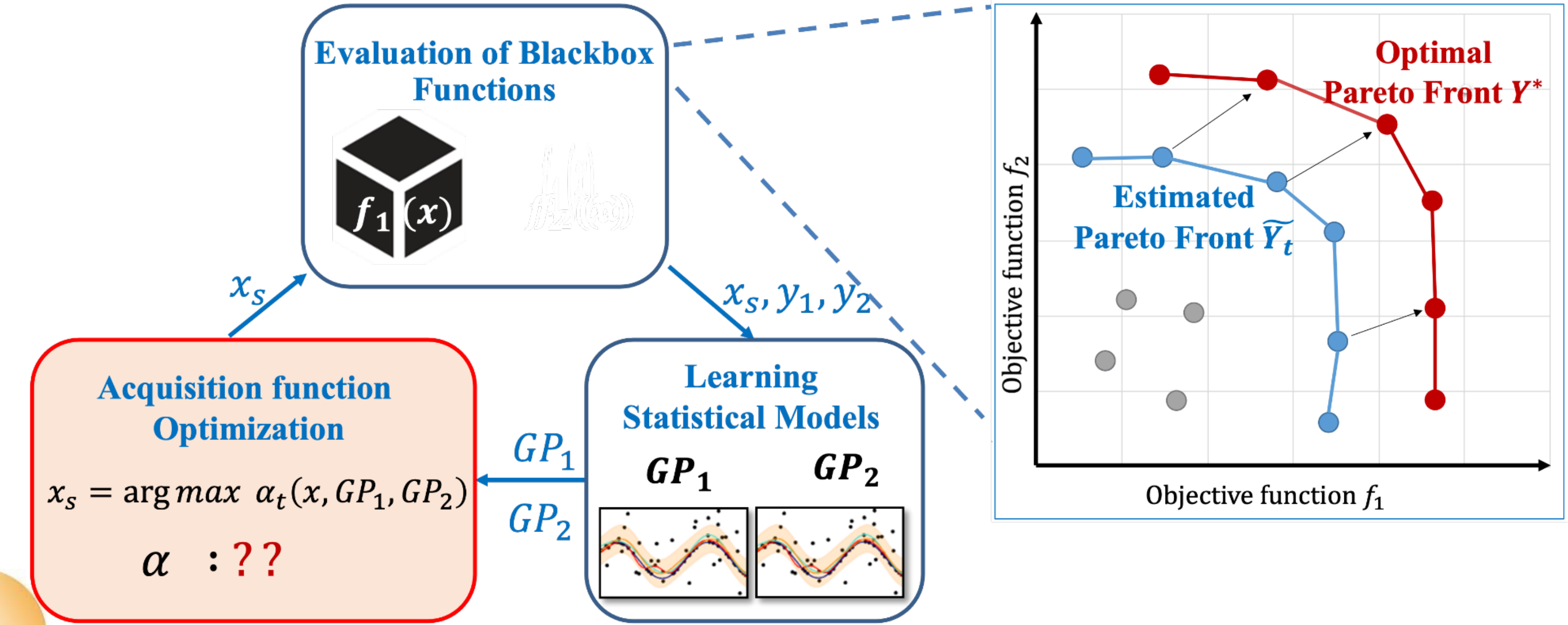
- Set of input designs with optimal trade-offs called the optimal Pareto set χ^*
- Corresponding set of function values called optimal pareto front Pareto front Y^*



- Pareto hypervolume measures the quality of a Pareto front

Single => Multi-Objective BO

- Challenge #1: Statistical modeling
  - Typically, one GP model for each objective function (tractability)
- Challenge #2: Acquisition function design
  - Capture the trade-off between multiple objectives



Evaluation of Blackbox Functions
$f_1(x)$
$f_2(x)$
x_s
x_s, y_1, y_2
Learning Statistical Models
GP_1
GP_2
GP_1
GP_2
Acquisition function Optimization
$x_s = \arg max\ \alpha_t(x, GP_1, GP_2)$
α : ??
Optimal Pareto Front Y^*
Estimated Pareto Front $\widetilde{Y_t}$
Objective function f_2
Objective function f_1

Multi-Objective BO: Summary of Approaches

- Reduction to single-objective via scalarization
  - ParEGO [Knowles et al., 2006] and MOBO-RS [Paria et al., 2019]
- Hypervolume improvement
  - EHI [Emmerich et al., 2008] , SUR [Picheny et al., 2015] , SMSego [Ponweiser et al., 2008] , qEHVI [Daulton et al., 2020], qNEHVI [Daulton et al., 2021]
- Wrapper methods via single-objective acquisition functions
  - USeMO [Belakaria et al., 2020], DGEMO [Lukovic et al. 2020]
- Information-theoretic methods
  - ϵ-PAL [Zuluaga et al., 2013] , PESMO [Hernandez-Lobato et al., 2016] , MESMO [Belakaria et al., 2019] , JES [Tu et al., 2022]

Multi-Objective BO: Summary of Approaches

- Reduction to single-objective via scalarization
  - ParEGO [Knowles et al., 2006] and MOBO-RS [Paria et al., 2019]
- Hypervolume improvement
  - EHI [Emmerich et al., 2008] , SUR [Picheny et al., 2015] , SMSego [Ponweiser et al., 2008] , qEHVI [Daulton et al., 2020], qNEHVI [Daulton et al., 2021]
- Wrapper methods via single-objective acquisition functions
  - USeMO [Belakaria et al., 2020], DGEMO [Lukovic et al. 2020]
- Information-theoretic methods
  - ϵ-PAL [Zuluaga et al., 2013] , PESMO [Hernandez-Lobato et al., 2016] , MESMO [Belakaria et al., 2019] , JES [Tu et al., 2022]

Reduction via Random Scalarization

- Reduce the problem to single objective optimization
- ParEGO [Knowles et al., 2006]
  - BO over scalarized objective function using EI

$$f(x) = \sum_{i=1}^{k} \lambda_i . f_i(x)$$

  - Scalar weights are sampled from a uniform distribution
- MOBO-RS [Paria et al., 2019]
  - Optimize scalarized objective function over a set of scalar weight-vectors using a prior specified by the user

Reduction via Random Scalarization

- Reduce the problem to single objective optimization
- ParEGO [Knowles et al., 2006]
  - BO over scalarized objective function using EI

$$f(x) = \sum_{i=1}^{k} \lambda_i . f_i(x)$$

  - Scalar weights are sampled from a uniform distribution
- MOBO-RS [Paria et al., 2019]
  - Optimize scalarized objective function over a set of scalar weight-vectors using a prior specified by the user

Hard to define the scalars or specify priors over scalars, which can lead to sub-optimal results

Multi-Objective BO: Summary of Approaches

- Reduction to single-objective via scalarization
  - ParEGO [Knowles et al., 2006] and MOBO-RS [Paria et al., 2019]
- Hypervolume improvement
  - EHI [Emmerich et al., 2008] , SUR [Picheny et al., 2015] , SMSego [Ponweiser et al., 2008] , qEHVI [Daulton et al., 2020], qNEHVI [Daulton et al., 2021]
- Wrapper methods via single-objective acquisition functions
  - USeMO [Belakaria et al., 2020], DGEMO [Lukovic et al. 2020]
- Information-theoretic methods
  - ϵ-PAL [Zuluaga et al., 2013] , PESMO [Hernandez-Lobato et al., 2016] , MESMO [Belakaria et al., 2019], JES [Tu et al., 2022]

Hypervolume Improvement Approaches

- EHI: Expected improvement in PHV [Emmerich et al., 2008]

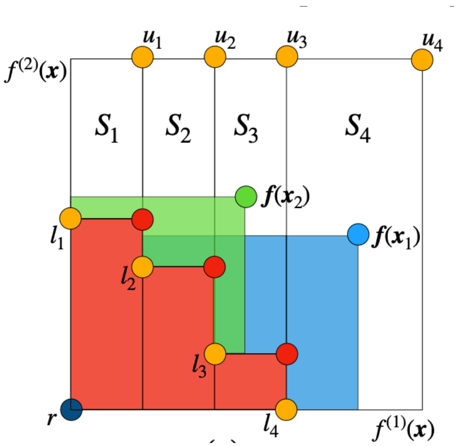
- SUR: Probability of improvement in PHV [Picheny et al., 2015]

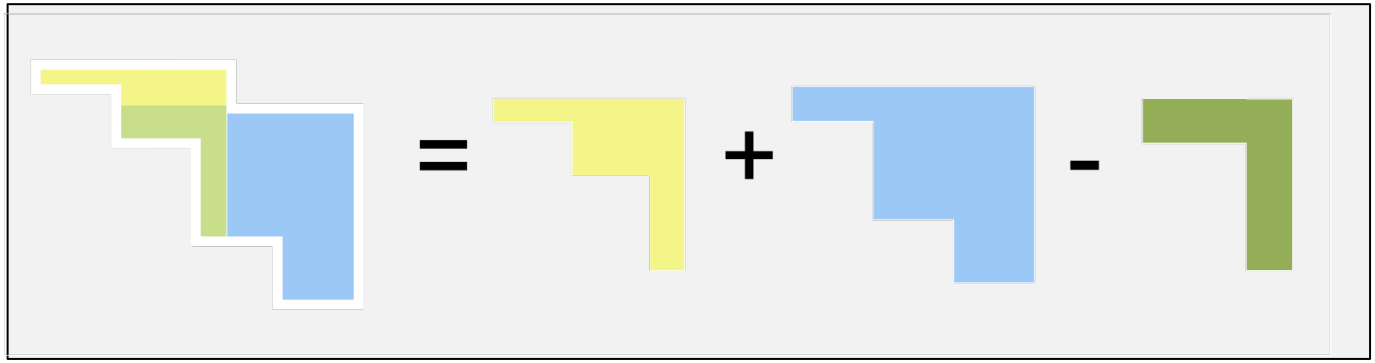
- SMSego [Ponweiser et al., 2008]
  - Improves the scalability of PHV computation by automatically reducing the search space

- qEHVI [Daulton et al., 2020], qNEHVI [Daulton et al., 2021]
  - Differentiable hypervolume improvement

qEHVI Algorithm [Daulton et al., 2020]

- Parallel EHVI via the Inclusion-Exclusion Principle





  - Practical since q is usually small
  - Computation of all intersections be parallelized
  - This formulation simplifies computation of overlapping hypervolumes

qEHVI Algorithm [Daulton et al., 2020]

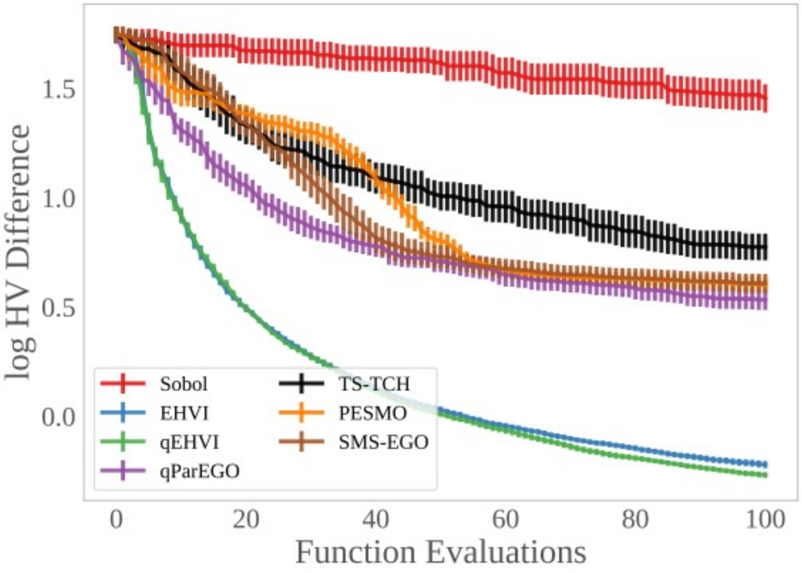
- Differentiable Hypervolume Improvement
  - Sample path gradients via the reparameterization trick
  - Unbiased gradient estimator

$$\mathbb{E}[\nabla_{\boldsymbol{x}} \hat{\alpha}_{qEHVI}(\boldsymbol{x})] = \nabla_{\boldsymbol{x}} \alpha_{qEHVI}(\boldsymbol{x})$$

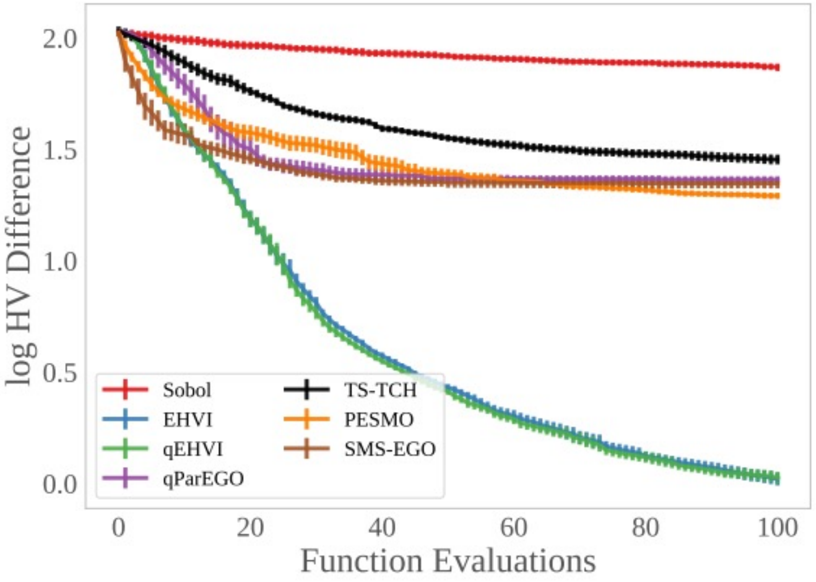
  - q can be selected via joint optimization (dq) or via a sequential greedy approximation
  - Sequential greedy approximation enjoys a regret of no more than $\frac{1}{e}\alpha^{*}{}_{qEHVI}$

Experimental Results for qEHVI [Daulton et al., 2020]

Vehicle Crash Safety

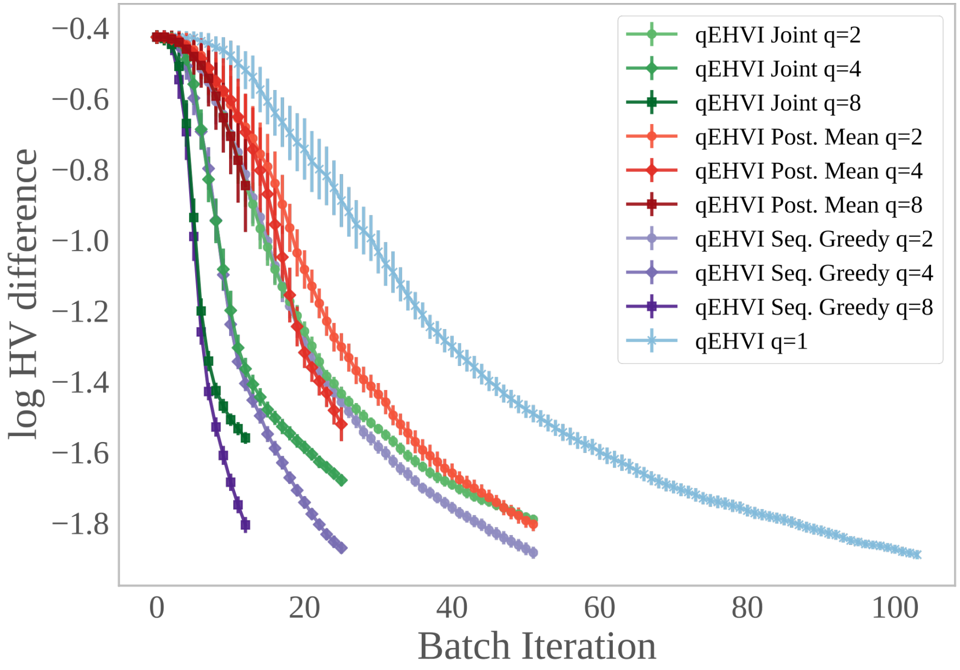


Branin-Currin



qEHVI Algorithm [Daulton et al., 2020]

- Sequential Greedy Optimization



- qNEHVI Algorithm [Daulton et al., 2021]: Extension to the Noisy setting

Hypervolume Improvement Approaches

- EHI: Expected improvement in PHV [Emmerich et al., 2008]
- SUR: Probability of improvement in PHV [Picheny et al., 2015]
- SMSego [Ponweiser et al., 2008]
  - Improves the scalability of PHV computation by automatically reducing the search space

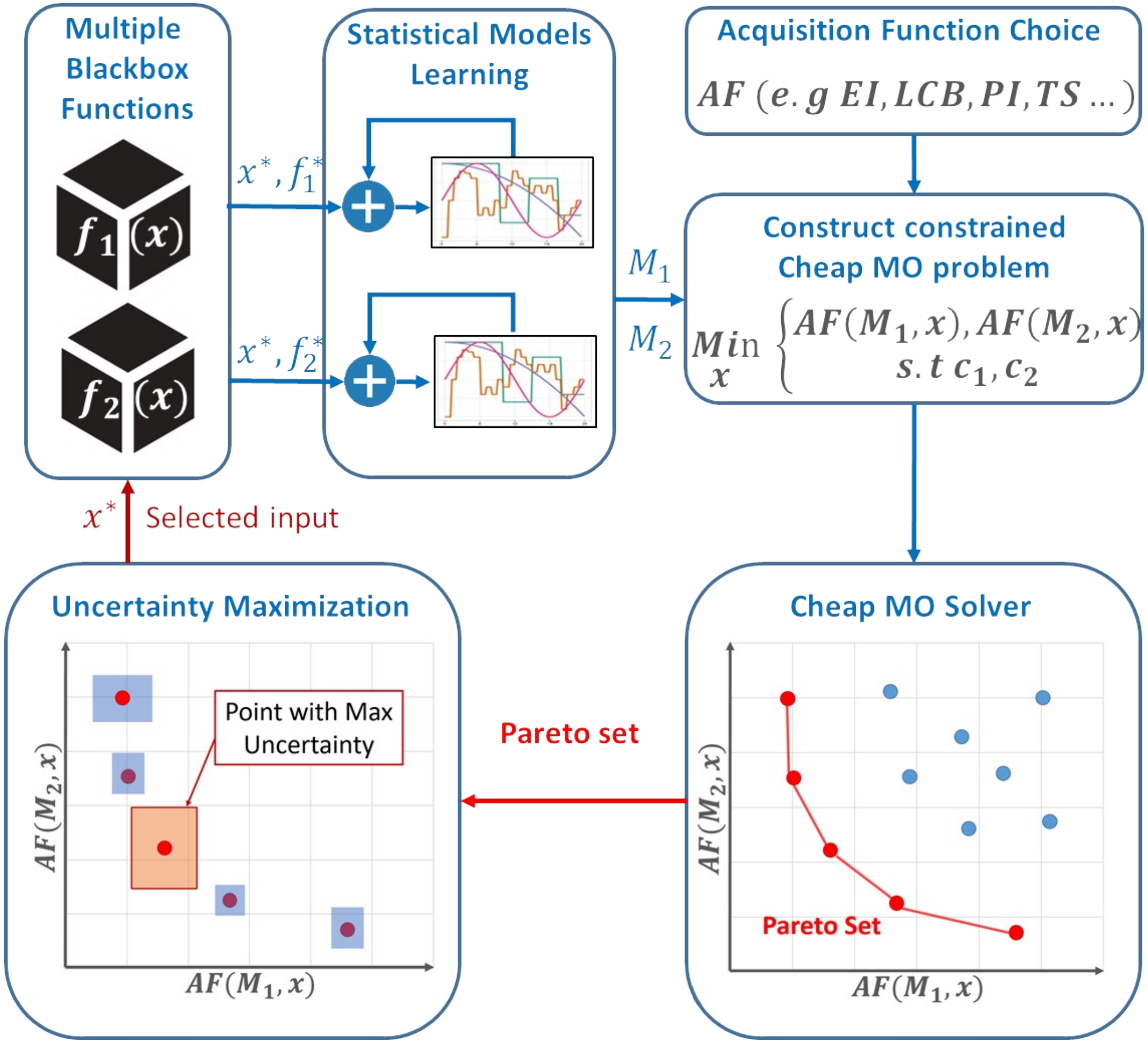
Can potentially lead to more exploitation behavior resulting in sub-optimal solutions

- qEHVI [Daulton et al., 2020], qNEHVI [Daulton et al., 2021]
  - Differentiable hypervolume improvement

Multi-Objective BO: Summary of Approaches

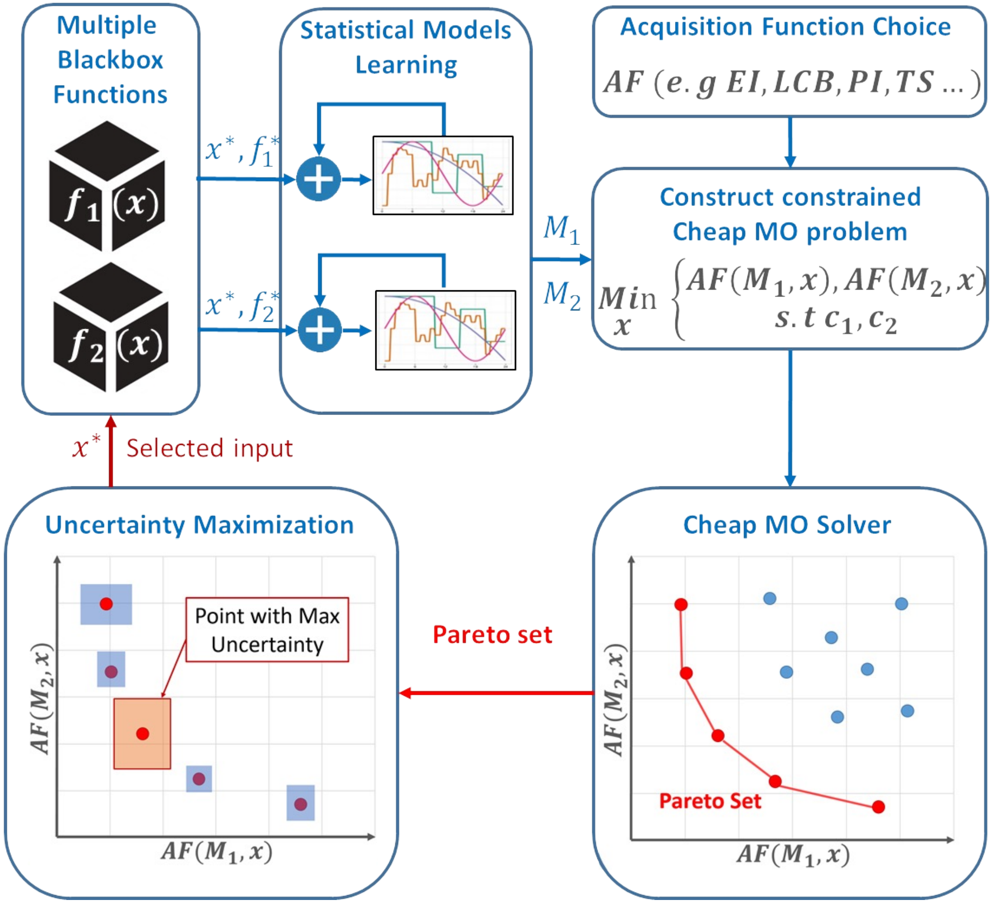
- Reduction to single-objective via scalarization
  - ParEGO [Knowles et al., 2006] and MOBO-RS [Paria et al., 2019]
- Hypervolume improvement
  - EHI [Emmerich et al., 2008] , SUR [Picheny et al., 2015] , SMSego [Ponweiser et al., 2008] , qEHVI [Daulton et al., 2020], qNEHVI [Daulton et al., 2021]
- Wrapper methods via single-objective acquisition functions
  - USeMO [Belakaria et al., 2020], DGEMO [Lukovic et al. 2020]
- Information-theoretic methods
  - ϵ-PAL [Zuluaga et al., 2013] , PESMO [Hernandez-Lobato et al., 2016] , MESMO [Belakaria et al., 2019] , JES [Tu et al., 2022]

USeMO Framework [Belakaria et al., 2020]



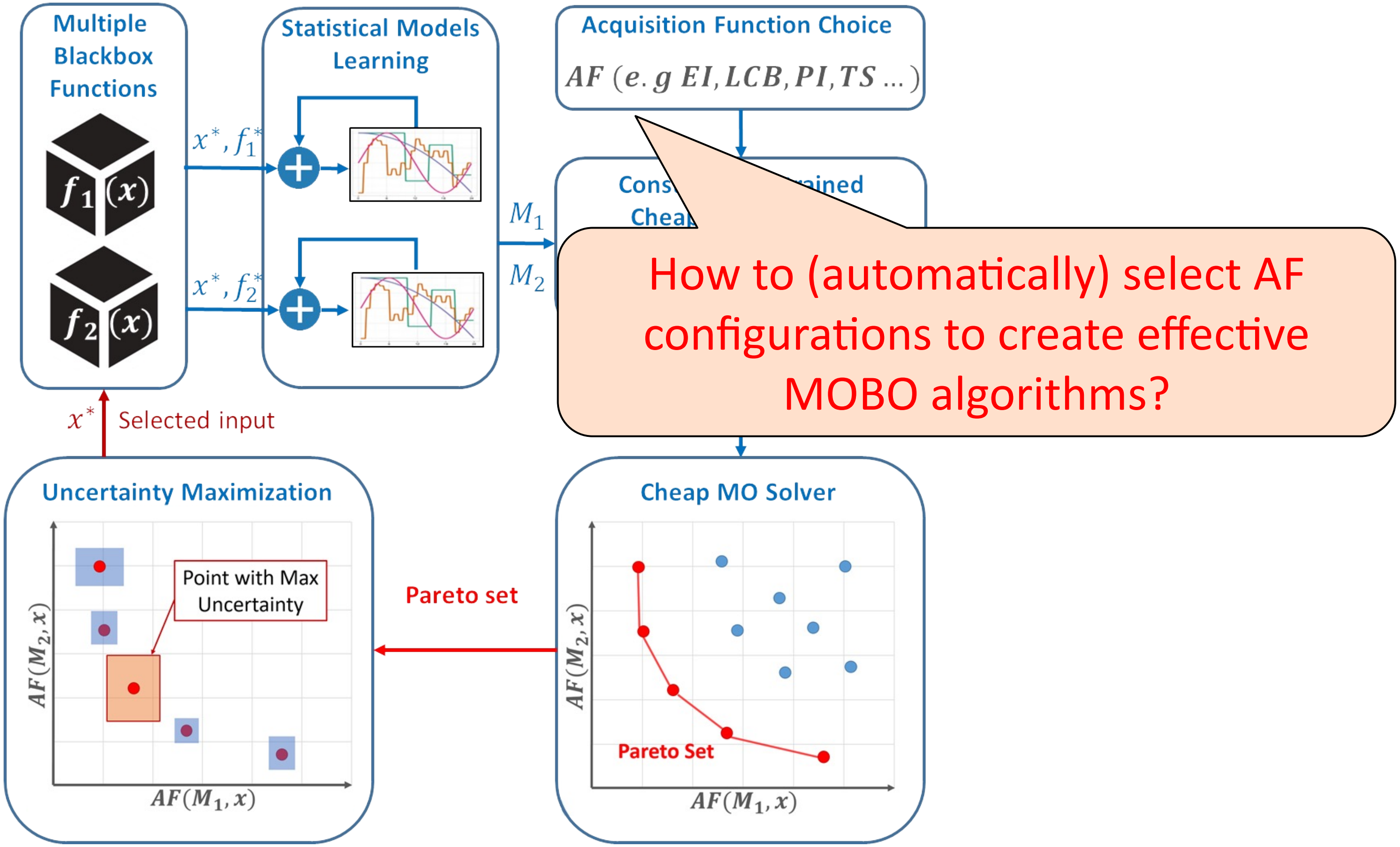
Multiple Blackbox Functions
$f_1(x)$
$f_2(x)$
x^*, f_1^*
x^*, f_2^*
Statistical Models Learning
M_1
M_2
Acquisition Function Choice
$AF\ (e.g\ EI, LCB, PI, TS \dots)$
Construct constrained Cheap MO problem
$\underset{x}{Min} \begin{cases} AF(M_1, x), AF(M_2, x) \\ s.t\ c_1, c_2 \end{cases}$
Cheap MO Solver
$AF(M_2, x)$
$AF(M_1, x)$
Pareto Set
Pareto set
Uncertainty Maximization
Point with Max Uncertainty
$AF(M_2, x)$
$AF(M_1, x)$
x^*
Selected input

USeMO Framework [Belakaria et al., 2020]


Multiple Blackbox Functions
$f_1(x)$
$f_2(x)$
x^*, f_1^*
x^*, f_2^*
Statistical Models Learning
M_1
M_2
Acquisition Function Choice
$AF\ (e.g\ EI, LCB, PI, TS \ldots)$
Construct constrained Cheap MO problem
$\underset{x}{Min} \begin{cases} AF(M_1, x), AF(M_2, x) \\ s.t\ c_1, c_2 \end{cases}$
Cheap MO Solver
$AF(M_2, x)$
$AF(M_1, x)$
Pareto Set
Pareto set
Uncertainty Maximization
Point with Max Uncertainty
$AF(M_2, x)$
$AF(M_1, x)$
x^*
Selected input

- Allows us to leverage acquisition functions from single-objective BO to solve multi-objective BO problems

USeMO Framework [Belakaria et al., 2020]



How to (automatically) select AF configurations to create effective MOBO algorithms?

- Allows us to leverage acquisition functions from single-objective BO to solve multi-objective BO problems

Multi-Objective BO: Summary of Approaches

- Reduction to single-objective via scalarization
  - ParEGO [Knowles et al., 2006] and MOBO-RS [Paria et al., 2019]
- Hypervolume improvement
  - EHI [Emmerich et al., 2008] , SUR [Picheny et al., 2015] , SMSego [Ponweiser et al., 2008] , qEHVI [Daulton et al., 2020], qNEHVI [Daulton et al., 2021]
- Wrapper methods via single-objective acquisition functions
  - USeMO [Belakaria et al., 2020], DGEMO [Lukovic et al. 2020]
- Information-theoretic methods
  - ϵ-PAL [Zuluaga et al., 2013] , PESMO [Hernandez-Lobato et al., 2016] , MESMO [Belakaria et al., 2019], JES [Tu et al., 2022]

ϵ-PAL Algorithm [Zuluaga et al., 2013]

- Classifies candidate inputs into three categories using the learned GP models
  - Pareto-optimal
  - Not Pareto-optimal
  - Uncertain

- In each iteration, selects the candidate input for evaluation to minimize the size of uncertain set

- Accuracy of pruning depends critically on ϵ value

ϵ-PAL Algorithm [Zuluaga et al., 2013]

- Classifies candidate inputs into three categories using the learned GP models
  - Pareto-optimal
  - Not Pareto-optimal
  - Uncertain

Limited applicability as it works only for discrete set of candidate inputs

- In each iteration, selects the candidate input for evaluation to minimize the size of uncertain set

- Accuracy of pruning depends critically on ϵ value

PESMO Algorithm [Hernandez-Lobato et al., 2016]

- Key Idea: select the input that maximizes the information gain about the optimal Pareto set χ^*

- Reminder: Set of input designs with optimal trade-offs is called the optimal Pareto set χ^*

PESMO Algorithm [Hernandez-Lobato et al., 2016]

- Key Idea: select the input that maximizes the information gain about the optimal Pareto set χ^*

$$\begin{aligned}\alpha(x) &= I(\{x,y\},\boxed{\chi^*}|D) \\ &= H(\chi^*|D) \ - \mathbb{E}_y[H(\chi^* \ |D \cup \{x,y\})] \\ &= H(y|D,x) \ - \mathbb{E}_{\chi^*}[H(y \ |D,x,\chi^*)]\end{aligned}$$

PESMO Algorithm [Hernandez-Lobato et al., 2016]

- Key Idea: select the input that maximizes the information gain about the optimal Pareto set χ^*

$$
\begin{aligned}
\alpha(x) &= I(\{x, y\}, \boxed{\chi^*} | D) \\
&= H(\chi^* | D) - \mathbb{E}_y [H(\chi^* | D \cup \{x, y\})] \\
&= H(y | D, x) - \mathbb{E}_{\chi^*} [H(y | D, x, \chi^*)]
\end{aligned}
$$

Equivalent to expected reduction in entropy over the pareto set χ^*

PESMO Algorithm [Hernandez-Lobato et al., 2016]

- Key Idea: select the input that maximizes the information gain about the optimal Pareto set χ^*

$$\begin{aligned}\alpha(x) &= I(\{x,y\},\boxed{\chi^*}|D) \\ &= H(\chi^*|D) \; - \mathbb{E}_y[H(\chi^* \, |D \cup \{x,y\})] \\ &= H(y|D,x) \; - \mathbb{E}_{\chi^*}[H(y\,|D,x,\chi^*)]\end{aligned}$$

Due to symmetric property of information gain

PESMO Algorithm [Hernandez-Lobato et al., 2016]

- Key Idea: select the input that maximizes the information gain about the optimal Pareto set χ^*

$$\alpha(x) = I(\{x, y\}, \boxed{\chi^*}|D)$$
$$= H(\chi^*|D) - \mathbb{E}_y[H(\chi^* \,|D \cup \{x, y\})]$$
$$= H(y|D, x) - \mathbb{E}_{\chi^*}[H(y\,|D, x, \chi^*)]$$

Entropy of factorizable Gaussian distribution

PESMO Algorithm [Hernandez-Lobato et al., 2016]

- Key Idea: select the input that maximizes the information gain about the optimal Pareto set χ^*

input dimension d

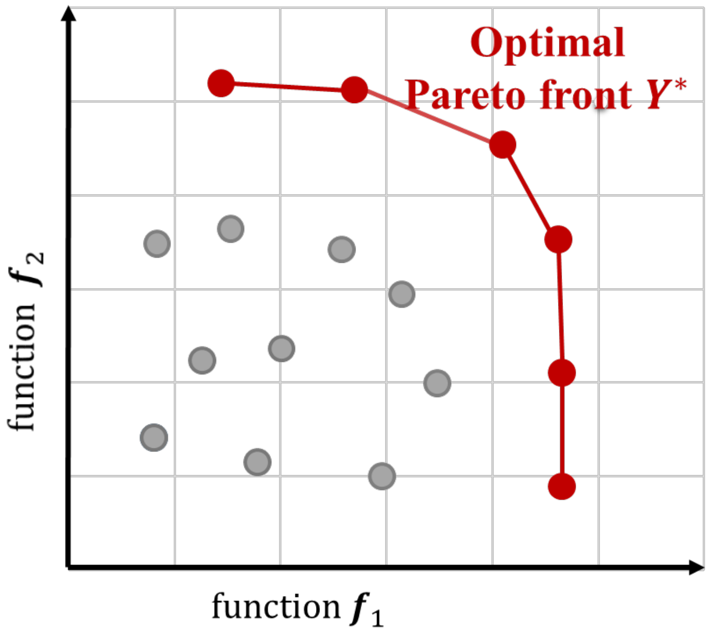
$$
\begin{aligned}
\alpha(x) &= I(\{x, y\}, \boxed{\chi^*} | D) \\
&= H(\chi^* | D) - \mathbb{E}_y[H(\chi^* | D \cup \{x, y\})] \\
&= H(y | D, x) - \boxed{\mathbb{E}_{\chi^*}[H(y | D, x, \chi^*)]}
\end{aligned}
$$

Requires computationally expensive approximation using expectation propagation

MESMO Algorithm [Belakaria et al., 2019]

- Key Idea: select the input that maximizes the information gain about the optimal Pareto front Y^*

- Reminder: Set of function values corresponding to the optimal Pareto set χ^* is called the optimal Pareto front Y^*



MESMO Algorithm [Belakaria et al., 2019]

- Key Idea: select the input that maximizes the information gain about the optimal Pareto front Y^*

$$
\begin{aligned}
\alpha(x) &= I(\{x, y\}, \boxed{Y^*} | D) \\
&= H(Y^* | D) - \mathbb{E}_y [H(Y^* | D \cup \{x, y\})] \\
&= H(y | D, x) - \mathbb{E}_{Y^*} [H(y | D, x, Y^*)]
\end{aligned}
$$

MESMO Algorithm [Belakaria et al., 2019]

- Key Idea: select the input that maximizes the information gain about the optimal Pareto front Y^*

$$\alpha(x) = I(\{x, y\}, \boxed{Y^*} | D)$$
$$= H(Y^* | D) - \mathbb{E}_y[H(Y^* | D \cup \{x, y\})]$$
$$= H(y | D, x) - \mathbb{E}_{Y^*}[H(y | D, x, Y^*)]$$

Equivalent to expected reduction in entropy over the pareto front Y^*

MESMO Algorithm [Belakaria et al., 2019]

- Key Idea: select the input that maximizes the information gain about the optimal Pareto front Y^*

$$\begin{aligned}\alpha(x) &= I(\{x,y\},\boxed{Y^*}|D)\\ &= H(Y^*|D) - \mathbb{E}_y[H(Y^* \,|D \cup \{x,y\})]\\ &= H(y|D,x) - \mathbb{E}_{Y^*}[H(y\,|D,x,Y^*)]\end{aligned}$$

Due to symmetric property of information gain

MESMO Algorithm [Belakaria et al., 2019]

- Key Idea: select the input that maximizes the information gain about the optimal Pareto front Y^*

$$\begin{aligned}\alpha(x) &= I(\{x,y\},\boxed{Y^*}|D) \\ &= H(Y^*|D) - \mathbb{E}_y[H(Y^*|D \cup \{x,y\})] \\ &= H(y|D,x) - \mathbb{E}_{Y^*}[H(y|D,x,Y^*)]\end{aligned}$$

Entropy of factorizable Gaussian distribution

MESMO Algorithm [Belakaria et al., 2019]

- Key Idea: select the input that maximizes the information gain about the optimal Pareto front Y^*

Output dimension $k \ll d$

$$\alpha(x) = I(\{x,y\}, \boxed{Y^*} | D)$$

$$= H(Y^*|D) - \mathbb{E}_y[H(Y^* | D \cup \{x,y\})]$$

$$= H(y|D,x) - \boxed{\mathbb{E}_{Y^*}[H(y | D, x, Y^*)]}$$

Closed form using properties of entropy and truncated Gaussian distribution

MESMO Algorithm [Belakaria et al., 2019]

$$\alpha(x) = H(y|D,x) - \mathbb{E}_{Y^*}[H(y\,|D,x,Y^*)]$$

- The first term is the entropy of a factorizable k-dimensional Gaussian distribution $P(y\,|D,x)$

$$H(y|D,x) = \frac{K(1+\ln(2\pi))}{2} + \sum_{j=1}^{k} \ln(\sigma_j(x))$$

MESMO Algorithm [Belakaria et al., 2019]

$$\alpha(x) = H(y|D,x) - \mathbb{E}_{Y^*}[H(y\,|D,x,Y^*)]$$

- We can approximately compute the second term via Monte-Carlo sampling (S is the number of samples)

$$\mathbb{E}_{Y^*}[H(y\,|D,x,Y^*)] \approx \frac{1}{S}\sum_{s=1}^{S} H(y\,|D,x,Y_s^*)$$

MESMO Algorithm [Belakaria et al., 2019]

- Approximate computation via Monte-Carlo sampling

$$\mathbb{E}_{Y^*}[H(y\,|D,x,Y^*)] \approx \frac{1}{S}\sum_{s=1}^{S} H(y\,|D,x,Y_s^*)$$

- **Two key steps**
  - How to compute Pareto front samples Y_s^* ?
  - How to compute the entropy with respect to a given Pareto front sample Y_s^*?

MESMO Algorithm [Belakaria et al., 2019]

- Approximate computation via Monte-Carlo sampling

$$\mathbb{E}_{Y^*}[H(y\,|D,x,Y^*)] \approx \frac{1}{S}\sum_{s=1}^{S}\ H(y\,|D,x,Y_s^*)$$

- How to compute Pareto front samples Y_s^* ?
  - Sample functions from posterior GPs via random Fourier features
  - Solve a cheap MO problem over the sampled functions $\tilde{f}_1 \ldots \tilde{f}_k$ to compute sample Pareto front

MESMO Algorithm [Belakaria et al., 2019]

- How to compute the entropy with respect to a given Pareto front sample Y_s^*?

$$Y_s^* = \{\boldsymbol{v^1}, \ldots, \boldsymbol{v^l}\}\ with\ \boldsymbol{v^i} = \{v_1^i, \ldots, v_K^i\},$$
$$y_j \leq y_{j_s}^* = \max\{v_1^1, \ldots, v_j^l\}\ \forall j \in \{1, \ldots, K\}$$

- Decompose the entropy of a set of independent variables into a sum of entropies of individual variables
- Model each component y_j as a truncated Gaussian distribution

$$H(y\,|D, x, Y_s^*) \approx \sum_{j=1}^{K} H\left(y_j \,|\, D, x, y_{j_s}^*\right)$$

MESMO Algorithm [Belakaria et al., 2019]

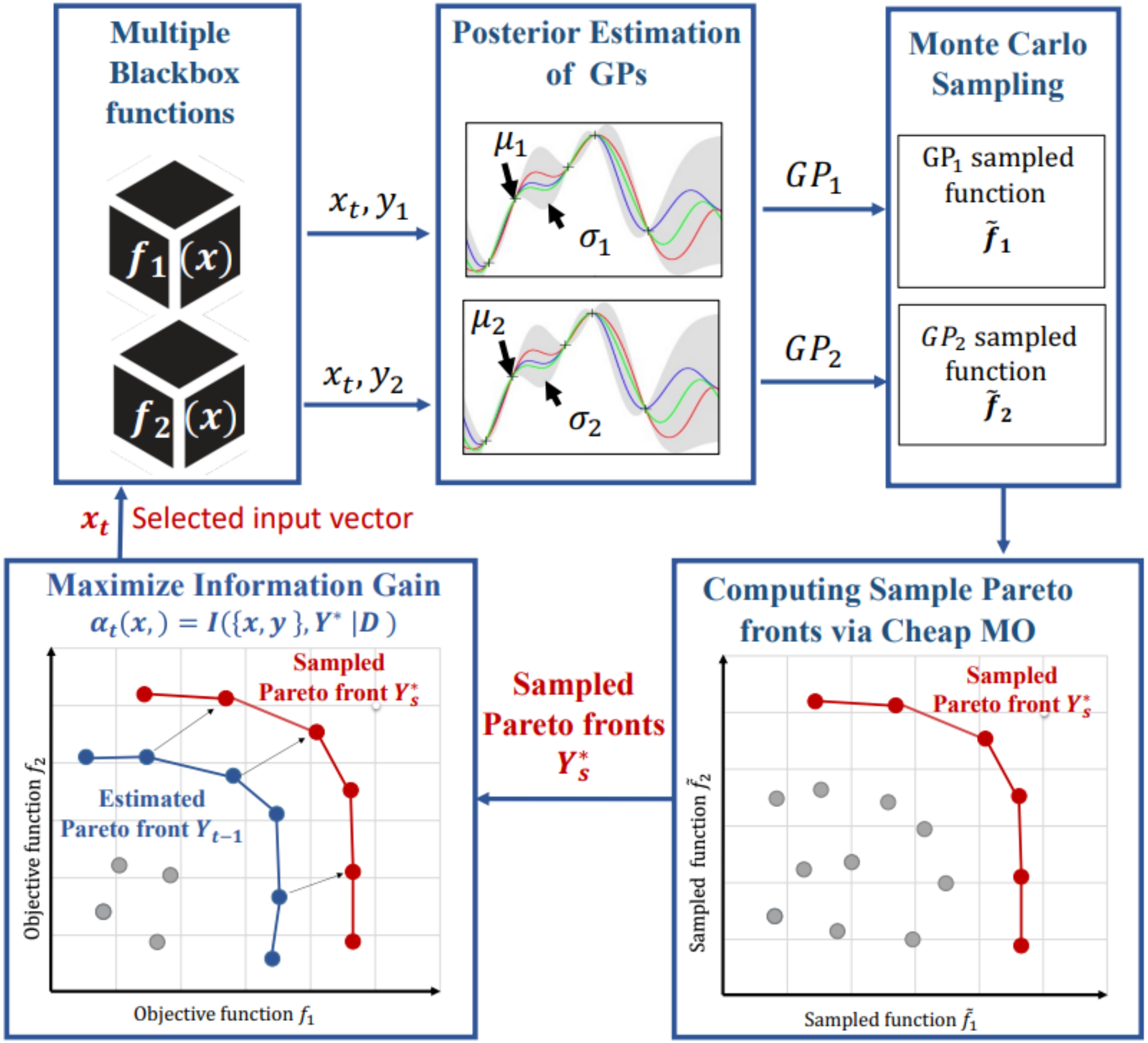
- Final acquisition function

$$\alpha(x) \approx \frac{1}{S} \sum_{s=1}^{S} \sum_{j=1}^{K} \left[\frac{\gamma_s^j(x)\phi\left(\gamma_s^j(x)\right)}{2\Phi\left(\gamma_s^j(x)\right)} - \ln\Phi\left(\gamma_s^j(x)\right)\right]$$

Closed form

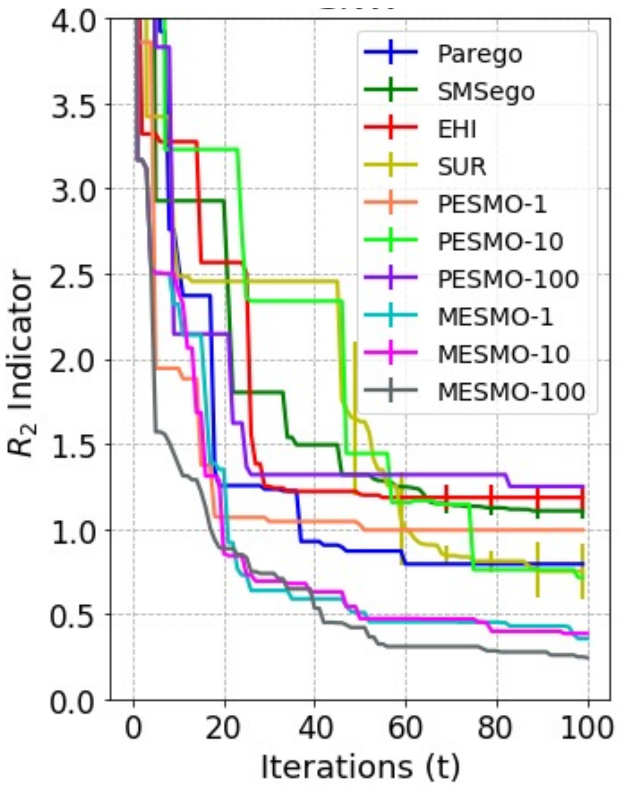
where $\gamma_s^j(x) = \frac{y_{js}^* - \mu_j(x)}{\sigma_j(x)}$, ϕ and Φ are the p.d.f and c.d.f of a standard normal distribution

MESMO Algorithm [Belakaria et al., 2019]

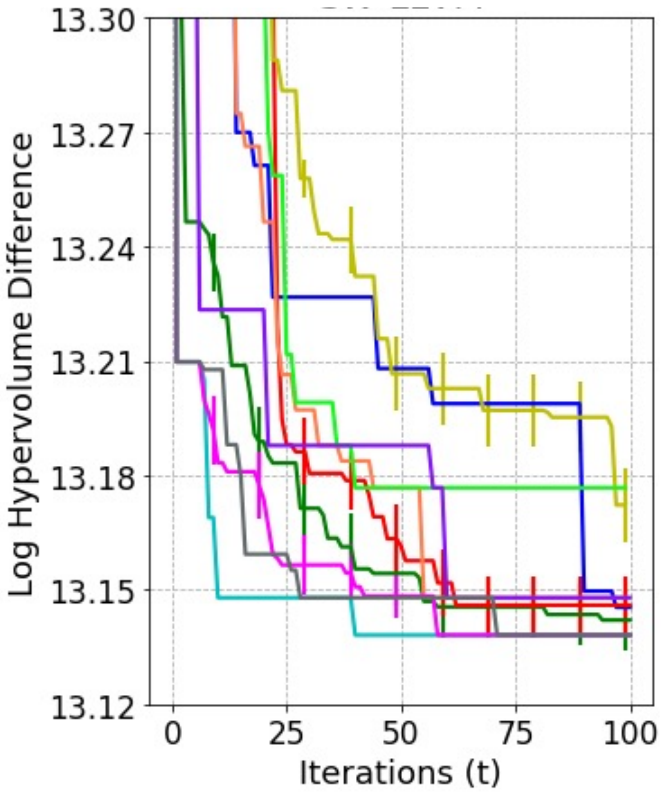

Multiple Blackbox functions
$f_1(x)$
$f_2(x)$
x_t, y_1
x_t, y_2
Posterior Estimation of GPs
μ_1
σ_1
μ_2
σ_2
GP_1
GP_2
Monte Carlo Sampling
GP_1 sampled function $\tilde{f}_1$
GP_2 sampled function $\tilde{f}_2$
x_t Selected input vector
Maximize Information Gain
$\alpha_t(x,) = I(\{x, y\}, Y^* | D)$
Sampled Pareto front Y_s^*
Estimated Pareto front Y_{t-1}
Objective function f_2
Objective function f_1
Sampled Pareto fronts Y_s^*
Computing Sample Pareto fronts via Cheap MO
Sampled Pareto front Y_s^*
Sampled function $\tilde{f}_2$
Sampled function $\tilde{f}_1$

MOBO Experiments and Results #1

Network on Chip Design

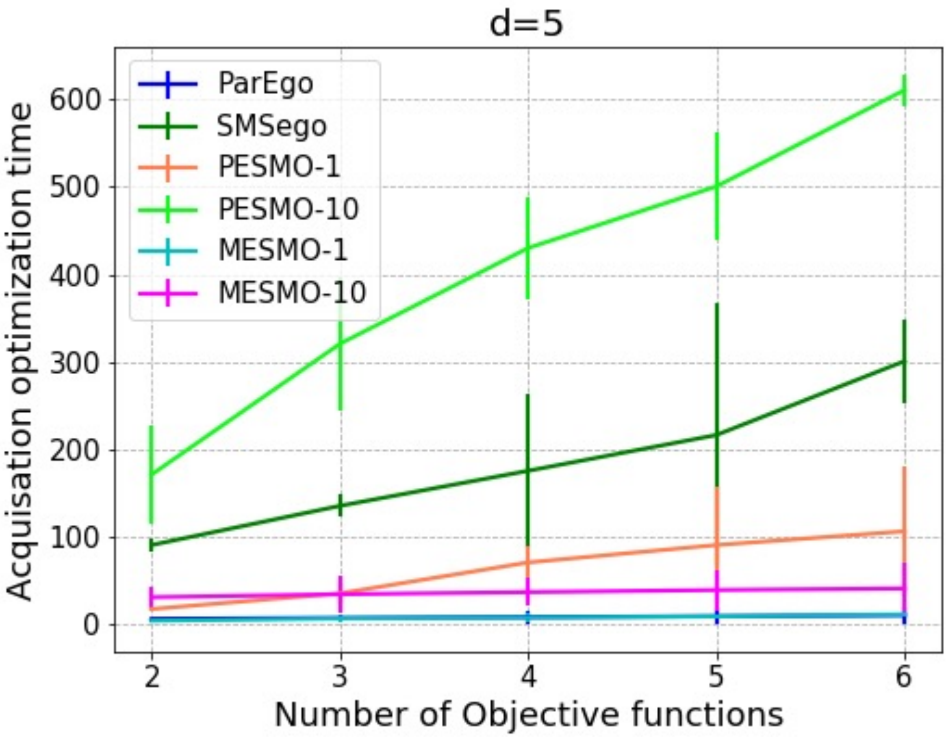


Compiler Settings Optimization



- MESMO is better than PESMO
- MESMO converges faster
- MESMO is robust to the number of samples (even a single sample!)

MOBO Experiments and Results #2



- MESMO is highly scalable when compared to PESMO
- MESMO with one sample is comparable to ParEGO
- Time for PESMO and SMSego increases significantly with the number of objectives

JES Algorithm [Tu et al., 2022]

- Key Idea: select the input that maximizes the information gain about the optimal Pareto Set χ^* and Pareto front Y^*

$$\alpha(x) = I(\{x, y\}, \boxed{\{\chi^*, Y^*\}} | D)$$

$$= H(y|x, D) - \mathbb{E}_{\{\chi^*, Y^*\}}[H(y\, | D, x, \{\chi^*, Y^*\})]$$

Entropy of factorizable Gaussian distribution

JES Algorithm [Tu et al., 2022]

- Key Idea: select the input that maximizes the information gain about the optimal Pareto Set χ^* and Pareto front Y^*

$$\alpha(x) = I(\{x, y\}, \boxed{\{\chi^*, Y^*\}} | D)$$

$$= H(y|x, D) - \mathbb{E}_{\{\chi^*, Y^*\}}[H(y\,|D, x, \{\chi^*, Y^*\})]$$

Entropy of factorizable Gaussian distribution

Approximated by a tractable lower-bound expression

JES Algorithm [Tu et al., 2022]

$$\alpha(x) = H(y|D,x) - \mathbb{E}_{\{\chi^*,Y^*\}}[H(y\,|D,x,\{\chi^*,Y^*\})]$$

- We can approximately compute the second term via Monte-Carlo sampling (S is the number of samples)

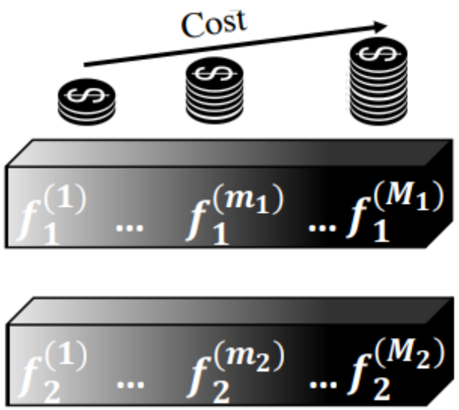
$$\mathbb{E}_{\{\chi^*,Y^*\}}[H(y\,|D,x,\{\chi^*,Y^*\})] \approx \frac{1}{S}\sum_{s=1}^{S} H(y\,|D,x,\{\chi_s^*,Y_s^*\})$$

$$\alpha(x) \approx \frac{1}{2}\sum_{j=1}^{K} \log(\sigma_j(x) - \frac{1}{2S}\sum_{s=1}^{S}\sum_{j=1}^{K} \log(\det(var(W))$$

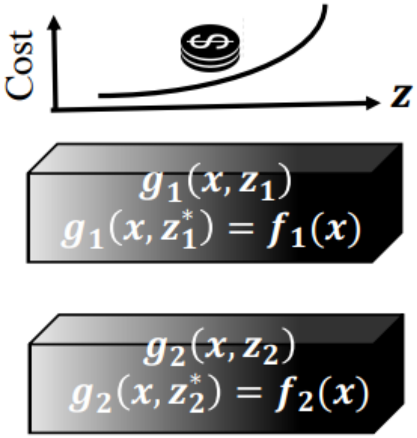
$$W \sim y|x,D,Y_s^*$$

Multi-Objective Bayesian Optimization via Multi-Fidelity Function Evaluations

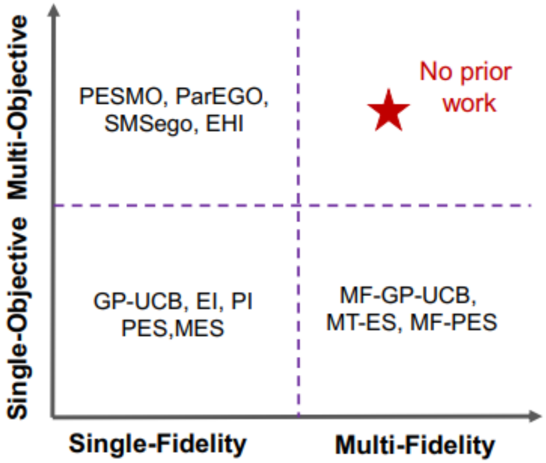
Multi-Fidelity Multi-Objective BO: The Problem


Cost
$f_1^{(1)}$ ... $f_1^{(m_1)}$ ... $f_1^{(M_1)}$
$f_2^{(1)}$ ... $f_2^{(m_2)}$ ... $f_2^{(M_2)}$

Discrete fidelity



Continuous fidelity



PESMO, ParEGO, SMSego, EHI

No prior work ★

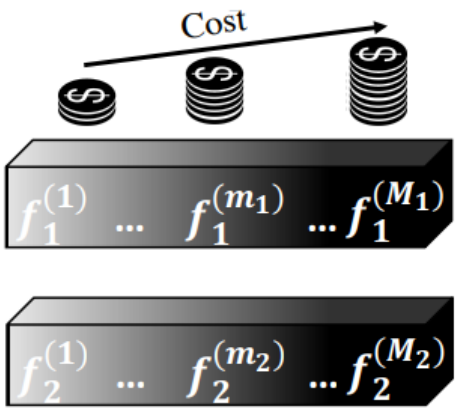
GP-UCB, EI, PI PES,MES

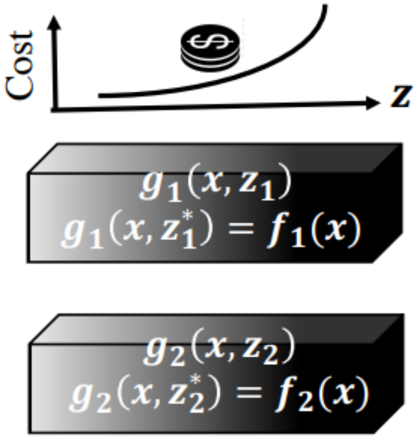
MF-GP-UCB, MT-ES, MF-PES

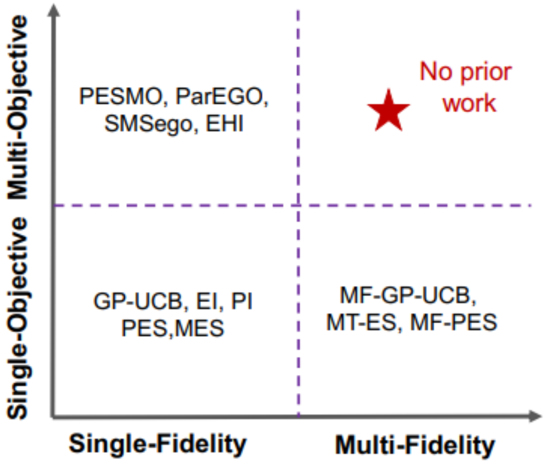
- Continuous-fidelity is the most general case
  - Discrete-fidelity is a special case

- Goal: find the approximate (optimal) Pareto set by minimizing the total resource cost of experiments

Multi-Fidelity Multi-Objective BO: Key Challenges


Cost
$f_1^{(1)}$ ... $f_1^{(m_1)}$ ... $f_1^{(M_1)}$
$f_2^{(1)}$ ... $f_2^{(m_2)}$ ... $f_2^{(M_2)}$





Single-Objective Multi-Objective

PESMO, ParEGO, SMSego, EHI

No prior work

GP-UCB, EI, PI PES,MES

MF-GP-UCB, MT-ES, MF-PES

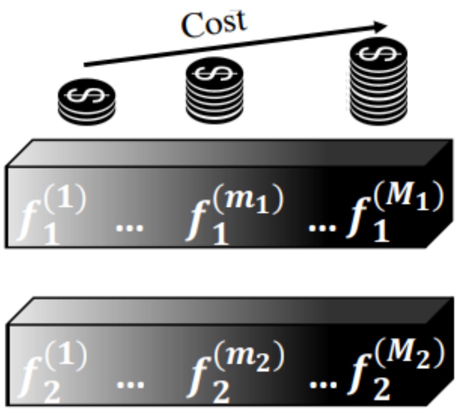
Single-Fidelity

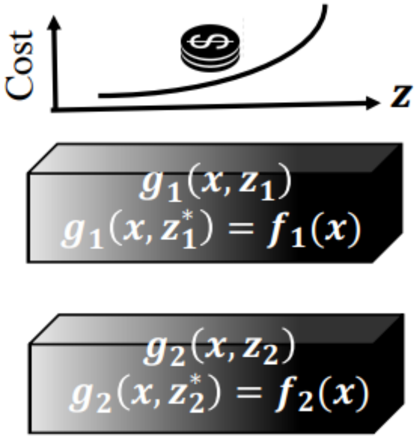
Multi-Fidelity

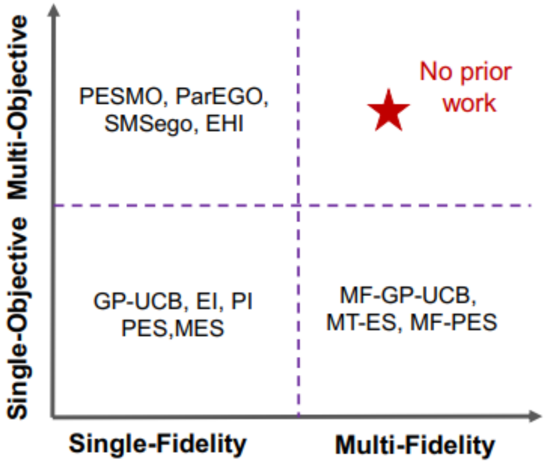
Challenges:

- How to model functions with multiple fidelities?
- How to jointly select the input design and fidelity-vector pair in each BO iteration?
- How to progressively select higher fidelity experiments?

Multi-Fidelity Multi-Objective BO: Key Challenges


Cost
$f_1^{(1)}$ ... $f_1^{(m_1)}$ ... $f_1^{(M_1)}$
$f_2^{(1)}$ ... $f_2^{(m_2)}$ ... $f_2^{(M_2)}$





Challenges:

- How to model functions with multiple fidelities?

Already covered

- How to join n and fidelity-vector pair in each BO iteration?
- How to progressively select higher fidelity experiments?

iMOCA Algorithm [Belakaria et al., 2021]

- Key Idea: Select the input and fidelity-vector that maximizes information gain per unit resource cost about the optimal Pareto front Y^*

$$\alpha(\boldsymbol{x},\boldsymbol{z}) = I(\{\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}\}, Y^* | D)/C(\boldsymbol{x},\boldsymbol{z})$$

$$= (H(\boldsymbol{y}|D,\boldsymbol{x},\boldsymbol{z}) - \mathbb{E}_{Y^*}[H(\boldsymbol{y}\,|D,\boldsymbol{x},\boldsymbol{z},Y^*)])/C(\boldsymbol{x},\boldsymbol{z})$$

$$= (\textstyle\sum_{j=1}^{K} \ln\left(\sqrt{2\pi e}\,\sigma_{g_j}(\boldsymbol{x},z_j)\right) - \frac{1}{S}\sum_{s=1}^{S}\ \sum_{j=1}^{K} H\big(y_j|\,D,\boldsymbol{x},z_j,f_s^{j*}\,)) / \,C(\boldsymbol{x},\boldsymbol{z})$$

where $C(\boldsymbol{x},\boldsymbol{z}) = \sum_{j=1}^{K} \frac{C(\boldsymbol{x},z_j)}{C(\boldsymbol{x},z_j^*)}$ is the normalized cost over different functions

iMOCA Algorithm [Belakaria et al., 2021]

- **Assumption:** Values at lower fidelities are smaller than maximum value of the highest fidelity $y_j \leq f_s^{j*} \; \forall j \in \{1, \dots, K\}$

- Truncated Gaussian approximation (Closed-form)

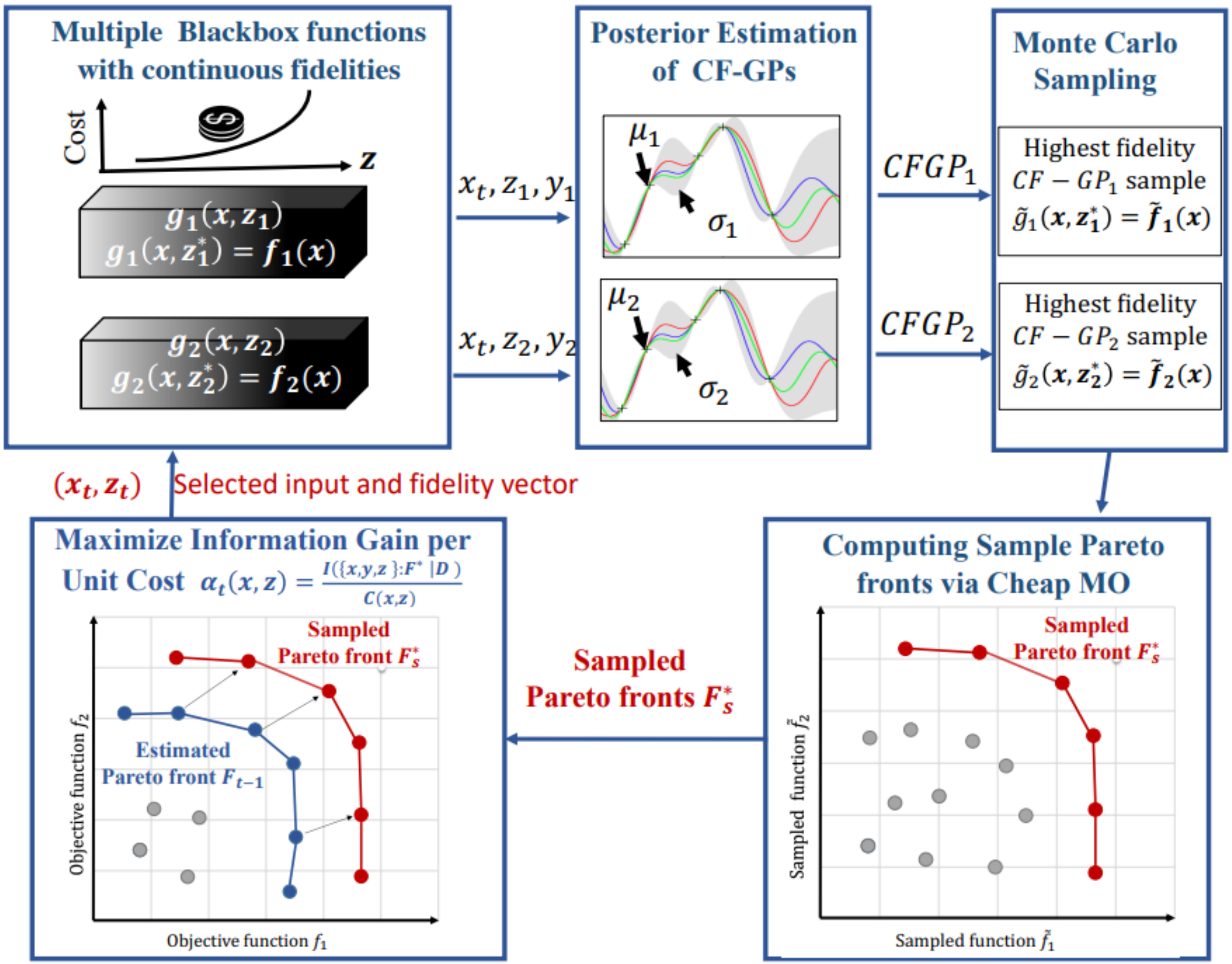
$$\alpha(\boldsymbol{x}, \boldsymbol{z}) \approx \frac{1}{C(\boldsymbol{x},\boldsymbol{z})S} \sum_{s=1}^{S} \sum_{j=1}^{K} \left[\frac{\gamma_s^{(g_j)} \phi\left(\gamma_s^{(g_j)}\right)}{2\Phi\left(\gamma_s^{(g_j)}\right)} - \ln\Phi\left(\gamma_s^{(g_j)}\right)\right]$$

Where $\gamma_s^{(g_j)} = \frac{f_s^{j*} - \mu_{g_j}}{\sigma_{g_j}}$, $\boldsymbol{\phi}$ and $\boldsymbol{\Phi}$ are the p.d.f and c.d.f of a standard normal distribution

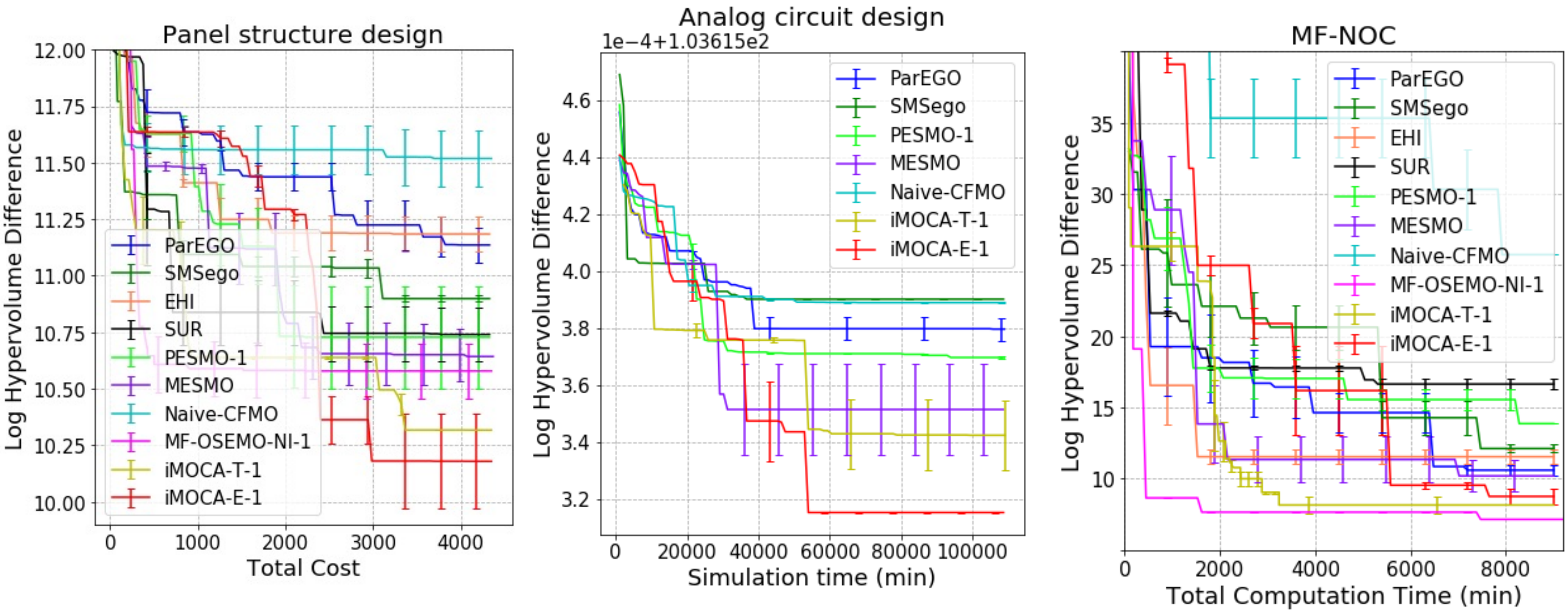
iMOCA Algorithm [Belakaria et al., 2021]

- Challenges of large (potentially infinite) fidelity space
  - Select costly fidelity with less accuracy
  - Tendency to select lower fidelities due to normalization by cost
- iMOCA reduces the fidelity search space using a scheme similar to the BOCA algorithm [Kandasamy et al., 2017]

iMOCA Algorithm [Belakaria et al., 2021]


Multiple Blackbox functions with continuous fidelities
Cost
z
$g_1(x, z_1)$
$g_1(x, z_1^*) = f_1(x)$
$g_2(x, z_2)$
$g_2(x, z_2^*) = f_2(x)$
x_t, z_1, y_1
x_t, z_2, y_2
Posterior Estimation of CF-GPs
μ_1
σ_1
μ_2
σ_2
$CFGP_1$
$CFGP_2$
Monte Carlo Sampling
Highest fidelity $CF-GP_1$ sample
$\tilde{g}_1(x, z_1^*) = \tilde{f}_1(x)$
Highest fidelity $CF-GP_2$ sample
$\tilde{g}_2(x, z_2^*) = \tilde{f}_2(x)$
(x_t, z_t)
Selected input and fidelity vector
Maximize Information Gain per Unit Cost $\alpha_t(x,z) = \frac{I(\{x,y,z\}; F^* \mid D)}{C(x,z)}$
Sampled Pareto front F_s^*
Estimated Pareto front F_{t-1}
Objective function f_2
Objective function f_1
Sampled Pareto fronts F_s^*
Computing Sample Pareto fronts via Cheap MO
Sampled Pareto front F_s^*
Sampled function $\tilde{f}_2$
Sampled function $\tilde{f}_1$

iMOCA Experiments and Results



- iMOCA performs better than all baselines
- Both variants of iMOCA converge at a much lower cost
- Robust to the number of samples

iMOCA Experiments and Results

- **Cost reduction factor**
  - Although the metric gives advantage to baselines, the results in the table show a consistently high gain ranging from 52% to 85%

Name	BC	ARS	Circuit	Rocket
$\mathcal{C}_B$	200	300	115000	9500
$\mathcal{C}$	30	100	55000	2000
$\mathcal{G}$	85%	66.6%	52.1%	78.9%

Table: *Best* convergence cost from all baselines $\mathcal{C}_B$, *Worst* convergence cost for iMOCA $\mathcal{C}$, and cost reduction factor $\mathcal{G}$.

Software and Code

- PESMO: github.com/HIPS/Spearmint/tree/PESM
- MESMO: github.com/belakaria/MESMO
- USeMO: github.com/belakaria/USeMO
- BoTorch
  - botorch.org/tutorials/multi_objective_bo
  - https://botorch.org/tutorials/information_theoretic_acquisition_functions
- DGEMO: github.com/yunshengtian/DGEMO
- MF-OSEMO: github.com/belakaria/MF-OSEMO
- iMOCA: github.com/belakaria/iMOCA

Questions ?

Outline of the Tutorial

- Overview of the BO Framework, GPs, advances in GPs and acquisition functions, and BoTorch demo
- Bayesian Optimization over Discrete/Hybrid Spaces

- Multi-fidelity Bayesian Optimization

30 mins Break

- High-Dimensional BO and BoTorch Hands-on demo

- Multi-Objective BO and BoTorch Hands-on demo

- Summary and Outstanding Challenges in BO

Open Challenges in BO

- **High-dimensional BO**
  - Need more effective approaches for high-dimensional spaces

- **BO over Combinatorial Structures**
  - How to combine domain knowledge, kernels, and (geometric) deep learning to build effective surrogate models?
  - Effective methods to select large and diverse batches?

- **BO over Hybrid Spaces**
  - Methods to sample functions from GP posterior?
  - Effective latent space BO methods?

Open Challenges in BO

- **Constrained BO**
  - Need more effective approaches for input spaces, where no. of invalid inputs >> no. of valid inputs
- **BO over Nested Function Pipelines**
  - Relatively less explored problem
- **BO with Resource Constraints**
  - Real-world experiments need resources and setup time
  - Critical for BO deployment in science and engineering labs
- **Democratize causal BO for diverse real-world applications**

Acknowledgements: Collaborators

- Nano-porous materials

- Hardware design

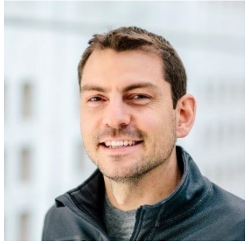
- Microbial fuel cells

- Electric transportation systems

- Catalysis

- Agriculture

BoTorch



Acknowledgements: Funding

NSF

Primary source

SCIENCE TO SHAPE THE FUTURE OF THE ARMY

ARMY RESEARCH OFFICE

1951

SRC®

Semiconductor
Research
Corporation

intel®

Questions ?