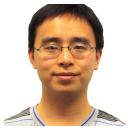
DeepMind

Towards Learning Universal Hyperparameter Optimizers with Transformers

Yutian Chen
Staff Research Scientist

arXiv: 2205.13320, Google AI Blog

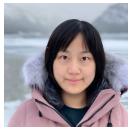
Collaborators



Yutian Chen

Richard Song

Chansoo Lee



Zi Wang

Richard Zhang

David Dohan

Kazuya Kawakami

Greg Kochanski

Sagi Perel

Arnaud Doucet

Marc'aurelio Ranzato

Nando de Freitas

Outlines

- Why learn a hyperparameter optimizer?
- How to learn it from existing data?
- Does it work?
- Summary and future work

Outlines

- Why learn a hyperparameter optimizer?
- How to learn it from existing data?
- Does it work?
- Summary and future work

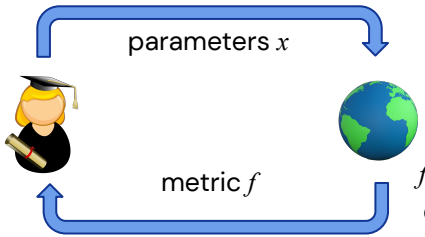
Hyperparameter optimization (HPO)

- Maximizing the ResNet accuracy wrt the learning rate and layer size

	Action (parameters)		Observation (metric)
Trial 1:	learning rate = 1e-3, layer size = 64	→	accuracy = 0.6
Trial 2:	learning rate = 1e-4, layer size = 128	→	accuracy = 0.7
Trial 3:	learning rate = 3e-4, layer size = 256	→	accuracy = 0.65
...			

- Optimize a black-box function in a short episode with a few trials:

  f: hyperparameters → model performance metric



$f(x)$

f is different in every episode

Metadata available:

- Task name
- Parameter names, range
- Metrics name
- ...

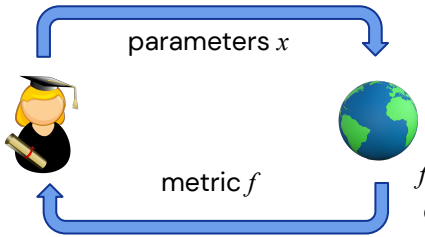
Hyperparameter optimization (HPO)

- Maximizing the ResNet accuracy wrt the learning rate and layer size

	Action (parameters)		Observation (metric)
Trial 1:	learning rate = 1e-3, layer size = 64	→	accuracy = 0.6
Trial 2:	learning rate = 1e-4, layer size = 128	→	accuracy = 0.7
Trial 3:	learning rate = 3e-4, layer size = 256	→	accuracy = 0.65
...			

- Optimize a black-box function in a short episode with a few trials:

  f: hyperparameters → model performance metric



$f(x)$

f is different in every episode

Metadata available:

- Task name
- Parameter names, range
- Metrics name
- ...

~~Gradient descent~~

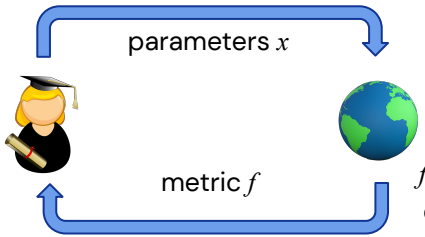
Hyperparameter optimization (HPO)

- Maximizing the ResNet accuracy wrt the learning rate and layer size

	Action (parameters)		Observation (metric)
Trial 1:	learning rate = 1e-3, layer size = 64	→	accuracy = 0.6
Trial 2:	learning rate = 1e-4, layer size = 128	→	accuracy = 0.7
Trial 3:	learning rate = 3e-4, layer size = 256	→	accuracy = 0.65
...			

- Optimize a black-box function in a short episode with a few trials:

  f: hyperparameters → model performance metric



$f(x)$

f is different in every episode

Metadata available:

- Task name
- Parameter names, range
- Metrics name
- ...

Graduate student descent

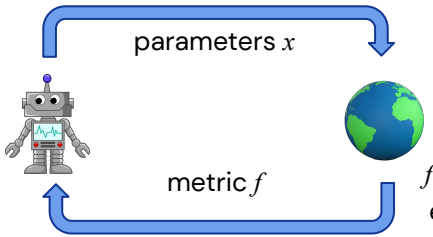
Hyperparameter optimization (HPO)

- Maximizing the ResNet accuracy wrt the learning rate and layer size

	Action (parameters)		Observation (metric)
Trial 1:	learning rate = 1e-3, layer size = 64	→	accuracy = 0.6
Trial 2:	learning rate = 1e-4, layer size = 128	→	accuracy = 0.7
Trial 3:	learning rate = 3e-4, layer size = 256	→	accuracy = 0.65
...			

- Optimize a black-box function in a short episode with a few trials:

  f: hyperparameters → model performance metric



$f(x)$

f is different in every episode

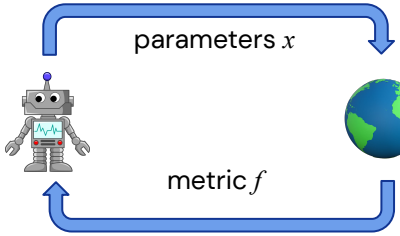
Metadata available:

- Task name
- Parameter names, range
- Metrics name
- ...

Taking the Human Out of the Loop: A Review of Bayesian Optimization. (Shahriari et. al., 2015)

Bayesian optimization: model-based policy

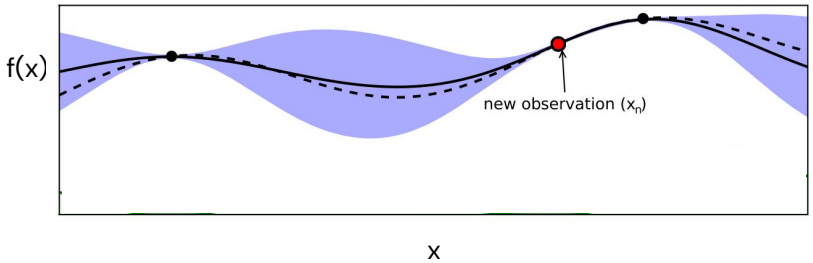
- **Probabilistic model**: e.g. Gaussian Process $p(f|\{(x_t, f_t)\})$
  - General assumption: local smoothness
  - Ignore meta information



f is different in every episode

Metadata available:

- Task name
- Parameter names, range
- Metrics name
- ...



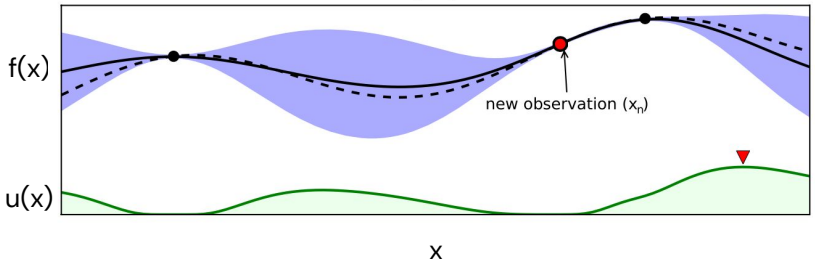
Bayesian optimization: model-based policy

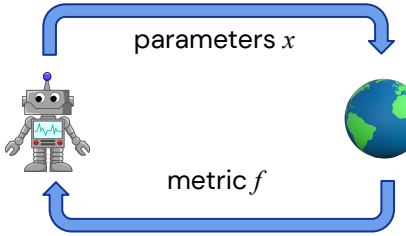
- **Probabilistic model**: e.g. Gaussian Process $p(f \mid \{(x_t, f_t)\})$
  - General assumption: local smoothness
  - Ignore meta information

- **Acquisition function** to suggest the next query (policy)

$$\text{argmax}_x \, u(x \mid p)$$

  - E.g. UCB, expected improvement, Thompson sampling





f is different in every episode

Metadata available:

- Task name
- Parameter names, range
- Metrics name
- ...

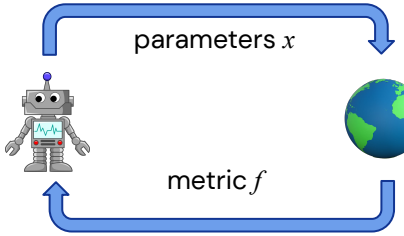
Bayesian optimization: model-based policy

- **Probabilistic model**: e.g. Gaussian Process $p(f \mid \{(x_t, f_t)\})$
  - General assumption: local smoothness
  - Ignore meta information

- **Acquisition function** to suggest the next query (policy)

$$\text{argmax}_x \, u(x \mid p)$$

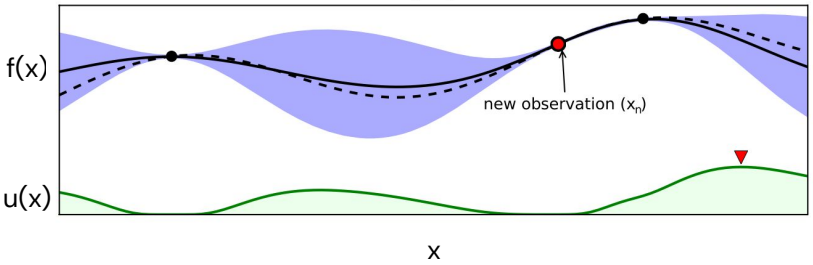
  - E.g. UCB, expected improvement, Thompson sampling

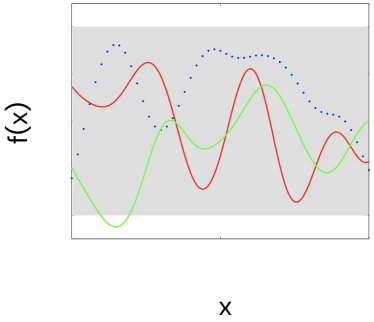


f is different in every episode

Metadata available:

- Task name
- Parameter names, range
- Metrics name
- ...





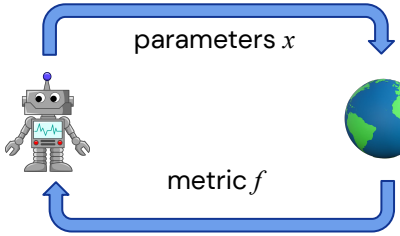
Bayesian optimization: model-based policy

- **Probabilistic model**: e.g. Gaussian Process $p(f \mid \{(x_t, f_t)\})$
  - General assumption: local smoothness
  - Ignore meta information

- **Acquisition function** to suggest the next query (policy)

$$\text{argmax}_x \, u(x \mid p)$$

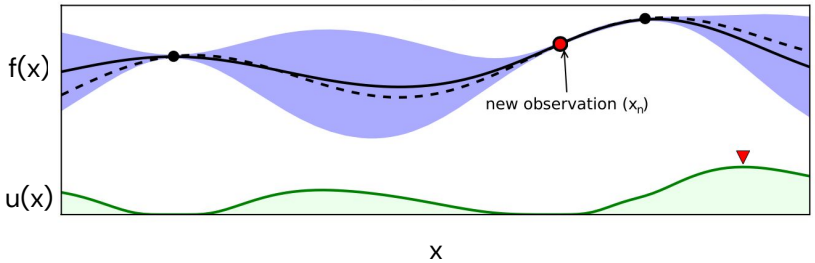
  - E.g. UCB, expected improvement, Thompson sampling

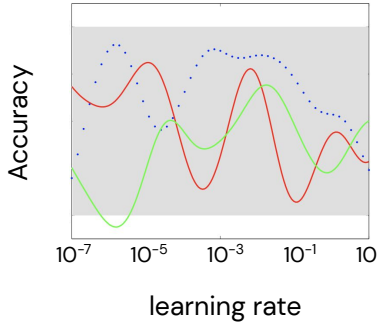


f is different in every episode

Metadata available:

- Task name
- Parameter names, range
- Metrics name
- ...





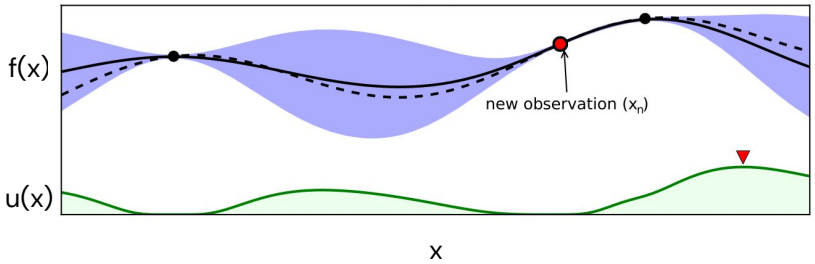
Bayesian optimization: model-based policy

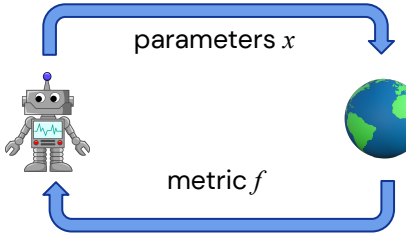
- **Probabilistic model**: e.g. Gaussian Process $p(f \mid \{(x_t, f_t)\})$
  - General assumption: local smoothness
  - Ignore meta information

- **Acquisition function** to suggest the next query (policy)

$$\text{argmax}_x \, u(x \mid p)$$

  - E.g. UCB, expected improvement, Thompson sampling

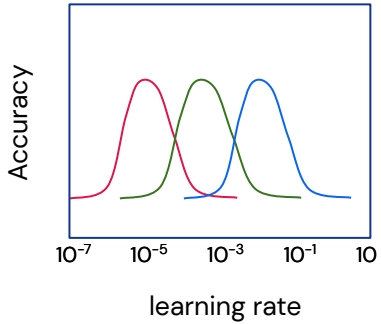




f is different in every episode

Metadata available:

- Task name
- Parameter names, range
- Metrics name
- ...



We have a better prior!

How do we learn it?

Learning the prior + policy from *Data*

- **Google Vizier**: hyperparameter optimization service

- Collects >16 million tuning experiments over years
  - Vision, NLP, Speech, Ads, ...

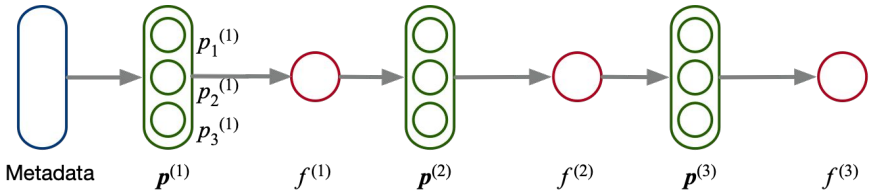
- What if we learn the prior from this dataset, or even directly a HPO policy?
  - **Transfer learning**: transfers knowledge from old experiments to future similar ones
  - **Zero shot guessing**: suggest optimal parameter ranges given new experiment description
  - **Neverending:** grows the knowledge over time with the dataset
  - **Know everything about tuning ML models**:
    how a metric depends on hyper-parameters and architecture

Outlines

- Why learn a hyperparameter optimizer?
- How to learn it from existing data?
- Does it work?
- Summary and future work

Modeling the tuning experiment data in the wild

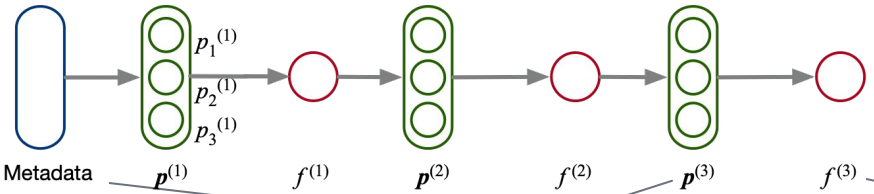
Learning in one task



Modeling the tuning experiment data in the wild

Learning in one task

Problem:
- mixed text and numeric values
- parameter dimension varies




```
"name": "convnet on cifar10",
"metric": "accuracy",
"goal": "MAXIMIZE",
"algorithm": "random_search",
"parameter": {
  "name": "opt_kw.lr",
  "type": "DOUBLE",
  "min_value": 1e-6,
  "max_value": 1e-2,
  "scale_type": "LOG"}
"parameter": {
  "name": "opt_type",
  "type": "CATEGORICAL",
  "categories": ["SGD", "Adam"]}
```

Task info

Param configs

```
"trial" {
"parameter": {
"opt_kw.lr": 0.0021237573,
"opt_type": "SGD"
}
"metric": {
"accuracy": 0.69482429,
}}
```

Param values

Metric values

Modeling the tuning experiment data in the wild

Solution: modeling a tuning history as a text sequence and tokenize everything

Learning in one task

Metadata $p^{(1)}$ $f^{(1)}$ $p^{(2)}$ $f^{(2)}$ $p^{(3)}$ $f^{(3)}$

```
<name>:"convnet on cifar10",
<metric>:"accuracy",
<goal>:<MAXIMIZE>,
<algorithm>:"random_search" &
<name>:"opt_kw.lr",
<type>:<DOUBLE>,
<min_value>:1e-6,
<max_value>:1e-2,
<scale_type>:<LOG>&
<name>:"opt_type",
<type>:<CATEGORICAL>,
<categories>:["SGD", "Adam"]
```

```
<645><1>* <999>|
```

Text sequence after serialization

Modeling the tuning experiment data in the wild

Solution: modeling a tuning history as a text sequence and tokenize everything

Learning in one task

Metadata $p^{(1)}$ $f^{(1)}$ $p^{(2)}$ $f^{(2)}$ $p^{(3)}$ $f^{(3)}$

```
name : " con v net on ci far 10 ",
metric : " acc u racy ",
goal : MAXIMIZE ,
algorithm : " random _ search " &
name : " op t _ kw . lr ",
type : DOUBLE ,
min_value : 1 e -6 ,
max_value : 1 e -2 ,
scale_type : LOG &
name : " op t _ type ",
type : CATEGORICAL ,
categories : [ " SG D ", " A dam " ]
```

```
645 1 * 999 |
```

Single tokens after tokenization

Each parameter and function value is represented by a single token:

An integer index in the vocabulary: [0, 1000)

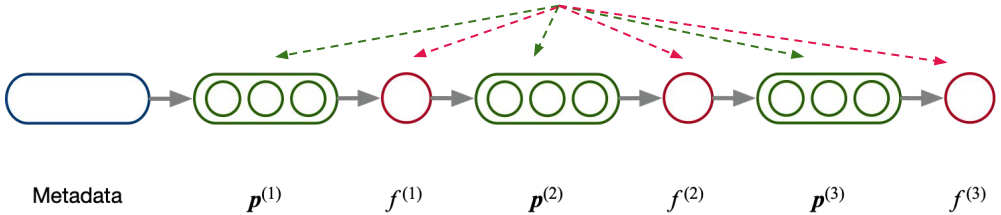
Sequence modeling

Solution: modeling a tuning history as a text sequence and tokenize everything

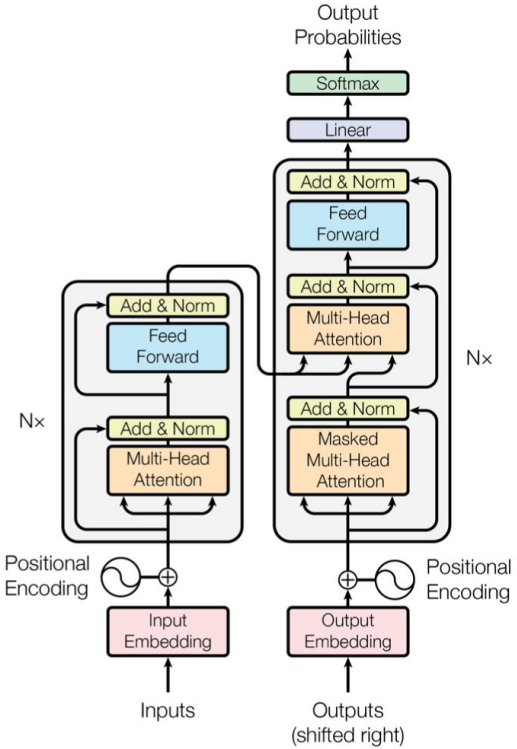
Learning in one task

$P(p_{t+1} \mid m, p_1, f_1, \ldots, p_t, f_t)$: policy

$P(f_t \mid m, p_1, f_1, \ldots, p_t)$: probabilistic model (dynamics and reward)

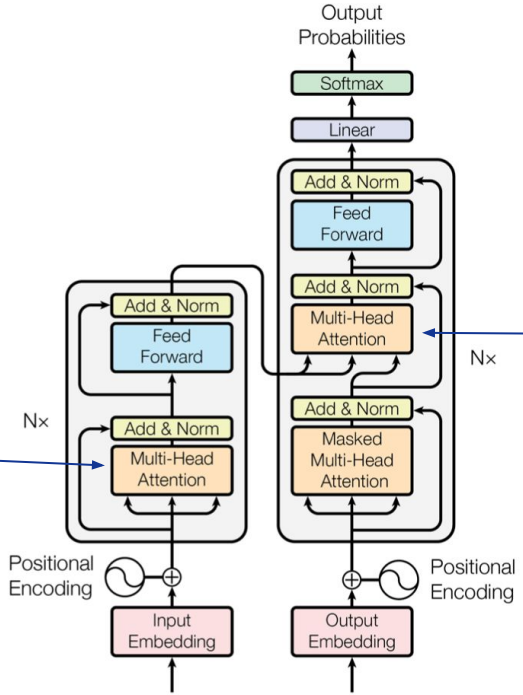


Transformer with an encoder-decoder architecture



The Transformer model architecture.
(Vaswani et al., 2017)

Transformer with an encoder-decoder architecture



Attends to all metadata

Attends to all metadata and trials upto the current step

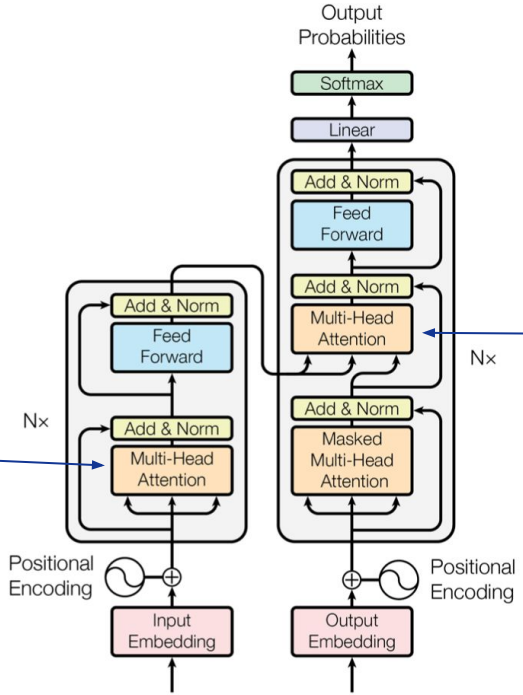
input: metadata (task info, param config, algorithm, ...)

targets: trials (param and metric values)

```
<name>:"convnet on cifar10", <metric>:"accuracy", …
```

```
<831><0>* <0> |<645><1>* <999>| …
```

Transformer with an encoder-decoder architecture



Attends to all metadata

Attends to all metadata and trials upto the current step

input: metadata (task info, param config, algorithm, ...)

targets: trials (param and metric values)

<name>:"convnet on cifar10", <metric>:"accuracy", …

<831><0>* <0> |<645><1>* <999>| …

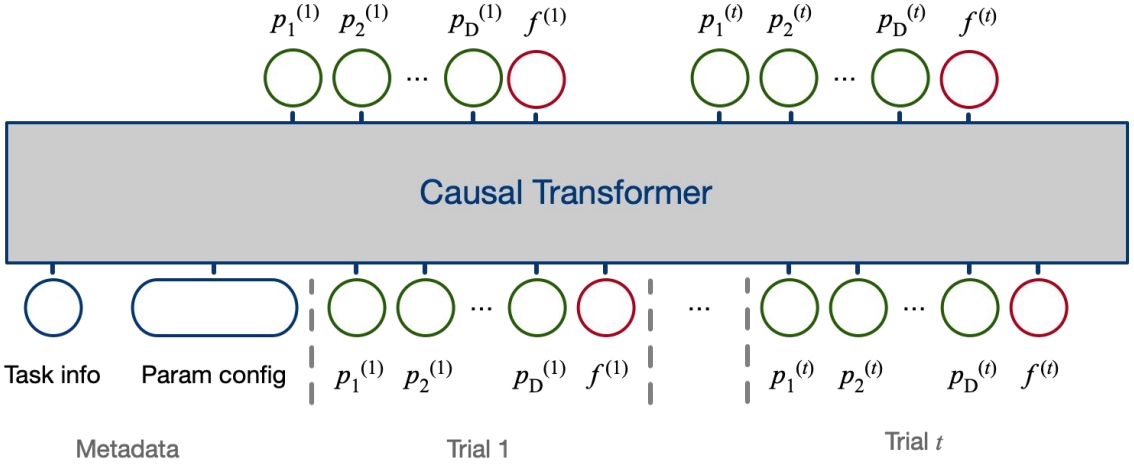
Why Transformer?

- Handles text and numbers in a unified way
- Better long sequence modeling ability than RNNs

OptFormer: transformer for optimization

$P(p_{t+1} \mid m, p_1, f_1, \ldots, p_t, f_t)$: policy

$P(f_t \mid m, p_1, f_1, \ldots, p_t)$: probabilistic model (dynamics and reward)

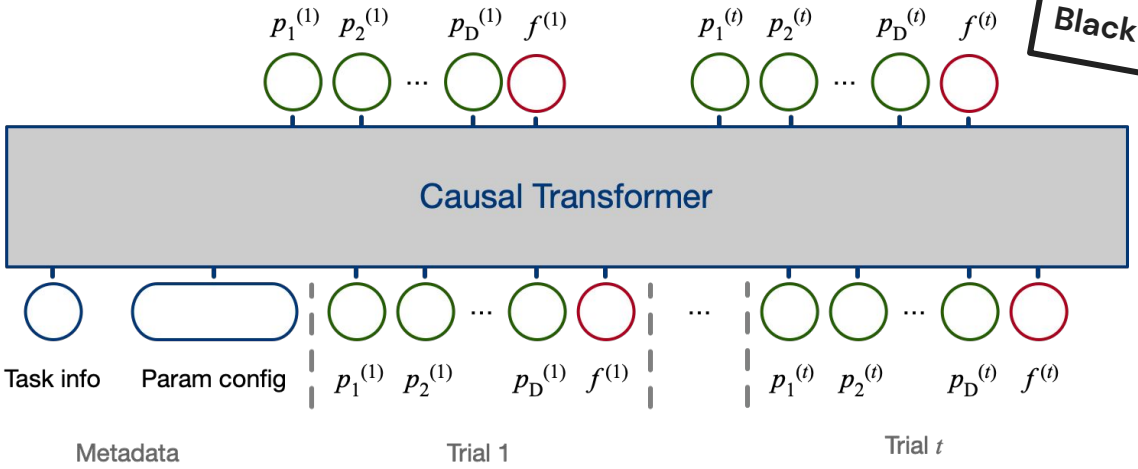


OptFormer: transformer for optimization

$P(p_{t+1} \mid m, p_1, f_1, \ldots, p_t, f_t)$: policy

$P(f_t \mid m, p_1, f_1, \ldots, p_t)$: probabilistic model (dynamics and reward)

Black-box meta-learning approach



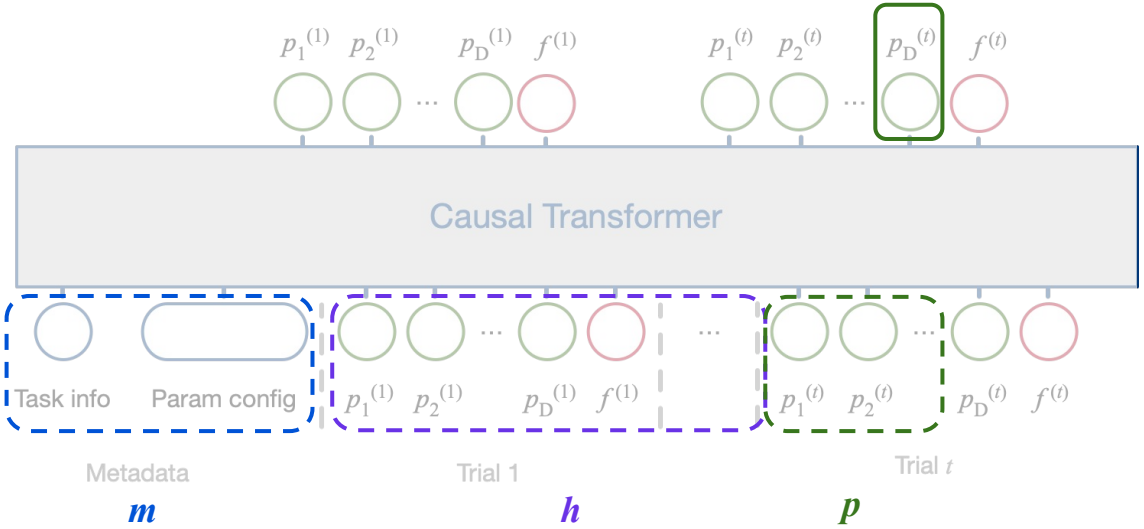
At test time

Training OptFormer: behavioral cloning

Given a sequence of optimization trajectory, learn to clone the behavior policy

- predict the taken actions $p_d^{(t)}$ in the data with *supervised learning*
  given metadata m, history $\boldsymbol{h}$ and partially sampled action dimensions $\boldsymbol{p}_{1:d-1}^{(t)}$

$$P_\theta\left(p_d^{(t)} | m, \mathbf{h}_{t-1}, \mathbf{p}_{1:d-1}^{(t)}\right) = \text{SoftMax}\left(\cdot | \mathbf{logits}_d^{(t)}\right)$$



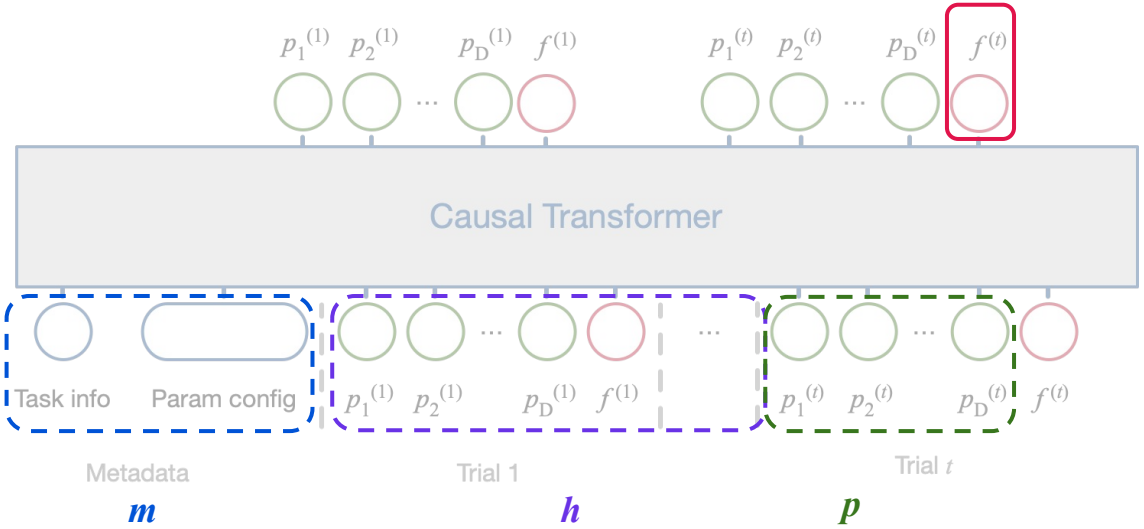
Training OptFormer: behavioral cloning

Given a sequence of optimization trajectory, learn to clone the behavior policy

- predict the taken actions $p_d^{(t)}$ in the data with *supervised learning*
  given metadata m, history $\boldsymbol{h}$ and partially sampled action dimensions $\boldsymbol{p}_{1:d-1}^{(t)}$

  $$P_\theta\left(p_d^{(t)} | m, \mathbf{h}_{t-1}, \mathbf{p}_{1:d-1}^{(t)}\right) = \text{SoftMax}\left(\cdot | \mathbf{logits}_d^{(t)}\right)$$

- predict the observed function f as an auxiliary task



Datasets of trajectories (real and synthetic)

- ***RealWorldData*** *(Vizier)*
  - Trajectories collected from real experiments by Google Vizier, mixed algorithm

- ***HPO-B***: surrogate models fitted to OpenML experiments (hyperparameters → metrics)
- ***BBOB***: synthetic functions from black-box optimization benchmark
  - Run 7 HPO algorithms and generate tuning trajectories
    - Grid search, shuffled grid search
    - Random search
    - Hill-climbing
    - Evolution strategies
      - Reg-evo
      - Eagle strategy
    - Vizier's GP-UCB

Table 3: Offline training datasets considered in this study. More details are given in Appendix C along with examples of studies in Table 5.

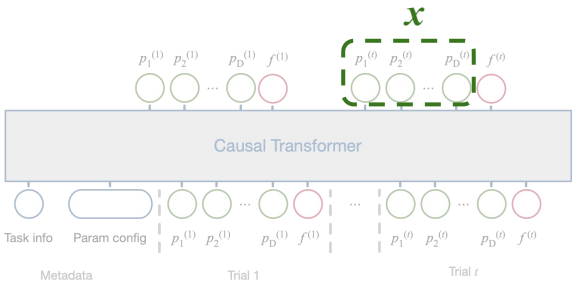
	("R") **RealWorldData**	("H") **HPO-B**	("B") **BBOB**
#Studies	750K	10M	10M
#Trials / study	300 (on average)	120	300
Study source	Google's database	Generated	Generated
π_b	Mixed	Controlled	Controlled
Obj. Functions	HPO tasks	HPO tasks	Synthetic
Search space	Different per task	16 shared search spaces	Randomized

- Data augmentation to mitigate overfitting
  - value scaling, parameter order permutation, metadata dropout

OptFormer policies to optimize hyperparameter

- Prior policy (behavioral cloning)
  - Sample parameters from the output distribution

$$\pi_{\text{prior}}(\boldsymbol{x}_t|m, \boldsymbol{h}_{t-1}) = \prod_{d=1}^{D} p_\theta(x_t^{(d)}|m, \boldsymbol{h}_{t-1}, \boldsymbol{x}_t^{(1:d-1)})$$



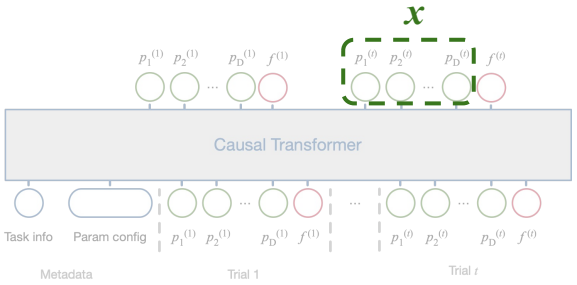
TRAINING MODEL

OptFormer

OptFormer policies to optimize hyperparameter

- Prior policy (behavioral cloning)
  - Sample parameters from the output distribution

$$\pi_{\text{prior}}(\boldsymbol{x}_t|m, \boldsymbol{h}_{t-1}) = \prod_{d=1}^{D} p_\theta(x_t^{(d)}|m, \boldsymbol{h}_{t-1}, \boldsymbol{x}_t^{(1:d-1)})$$

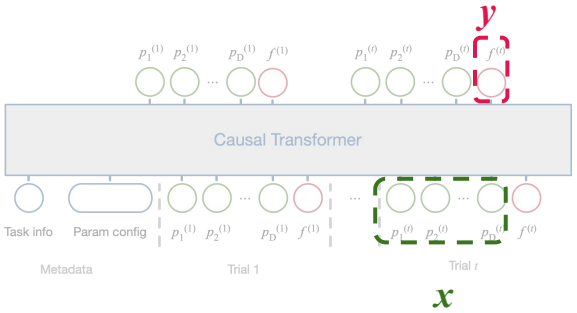


- Augmented policy (go beyond behavioral cloning)
  - **Sample** multiple parameter candidates

$$\boldsymbol{x}^{(i)} \overset{\text{i.i.d.}}{\sim} \pi_{\text{prior}}(\boldsymbol{x}|m, \boldsymbol{h}_{t-1})$$

  - **Rank** candidates using an acquisition function $u(\boldsymbol{x})$ computed from the function prediction head
    - e.g., EI, UCB, PI, ...

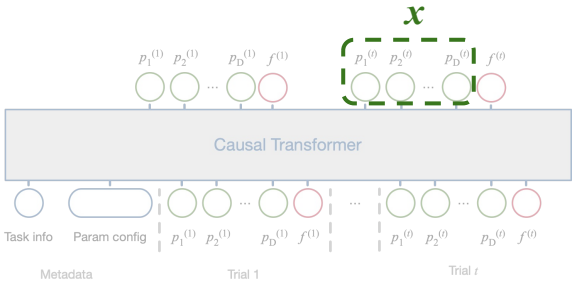
$$\underset{\{\mathbf{x}^{(i)}\}_{i=1}^{M}}{\text{argmax}}\ u(p_\theta(\cdot|\ldots, \mathbf{x}^{(i)}))$$



OptFormer policies to optimize hyperparameter

- Prior policy (behavioral cloning)
  - Sample parameters from the output distribution

$$\pi_{\text{prior}}(\boldsymbol{x}_t|m, \boldsymbol{h}_{t-1}) = \prod_{d=1}^{D} p_\theta(x_t^{(d)}|m, \boldsymbol{h}_{t-1}, \boldsymbol{x}_t^{(1:d-1)})$$

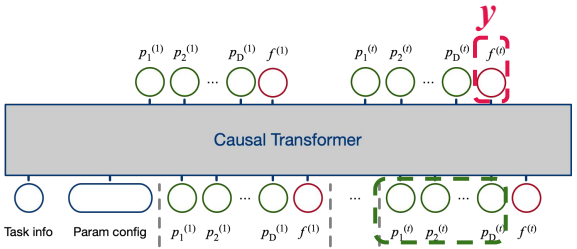


- Augmented policy (go beyond behavioral cloning)
  - **Sample** multiple parameter candidates

$$\boldsymbol{x}^{(i)} \overset{\text{i.i.d.}}{\sim} \pi_{\text{prior}}(\boldsymbol{x}|m, \boldsymbol{h}_{t-1})$$

  - **Rank** candidates using an acquisition function $u(\boldsymbol{x})$ computed from the function prediction head
    - e.g., EI, UCB, PI, ...

$$\underset{\{\mathbf{x}^{(i)}\}_{i=1}^{M}}{\text{argmax}}\ u(p_\theta(\cdot|\ldots, \mathbf{x}^{(i)}))$$



In standard Bayesian optimization

$$\underset{\mathbf{x}\in\mathcal{X}}{\text{argmax}}\ u(p_\theta(\cdot|\ldots, \mathbf{x}))$$

Outlines

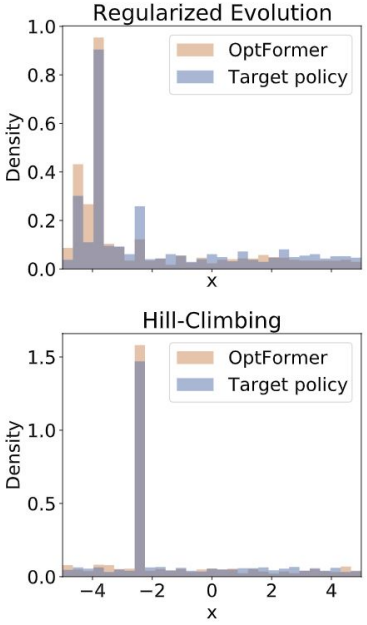
- Why learn a hyperparameter optimizer?
- How to learn it from existing data?
- Does it work?
- Summary and future work

Parameter prediction - imitating HPO algorithms

- Prior policy imitates 7 HPO algorithms
  (same model, different algorithm name prompt)

```
..., algorithm: "Random Search",...
..., algorithm: "Regularized Evolution",...
..., algorithm: "Hill Climbing",...
..., algorithm: "GP-UCB",...
```

- Evaluation: compare imitated policy with behavior policy
  - Predict distribution: $p(x|m, h)$



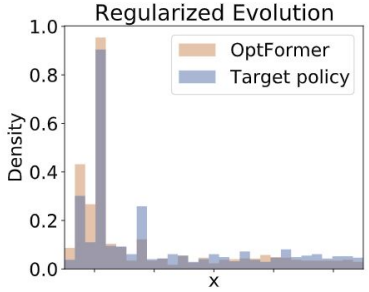
(a) Policy distribution

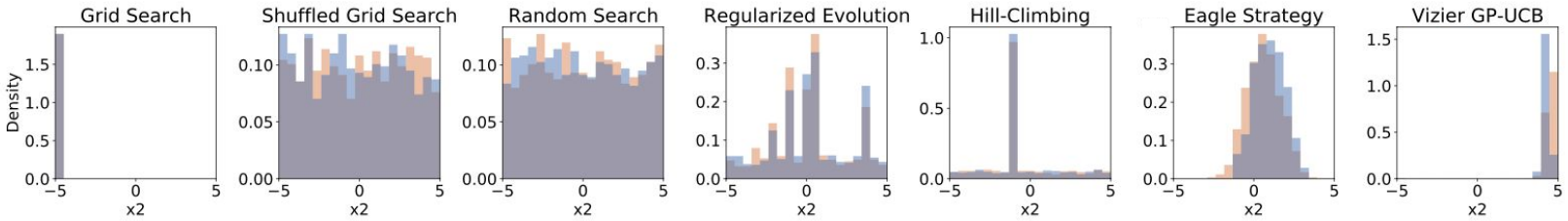
Parameter prediction - imitating HPO algorithms

- Prior policy imitates 7 HPO algorithms
  (same model, different algorithm name prompt)

```
..., algorithm: "Random Search",...
..., algorithm: "Regularized Evolution",...
..., algorithm: "Hill Climbing",...
..., algorithm: "GP-UCB",...
```

- Evaluation: compare imitated policy with behavior policy



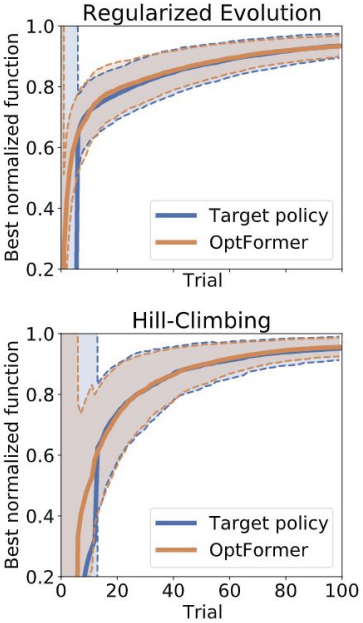


Parameter prediction - imitating HPO algorithms

- Prior policy imitates 7 HPO algorithms
  (same model, different algorithm name prompt)

```
..., algorithm: "Random Search",...
..., algorithm: "Regularized Evolution",...
..., algorithm: "Hill Climbing",...
..., algorithm: "GP-UCB",...
```

- Evaluation: compare imitated policy with behavior policy
  - Predict distribution: $p(x|m,\ h)$
  - Trajectory of function values



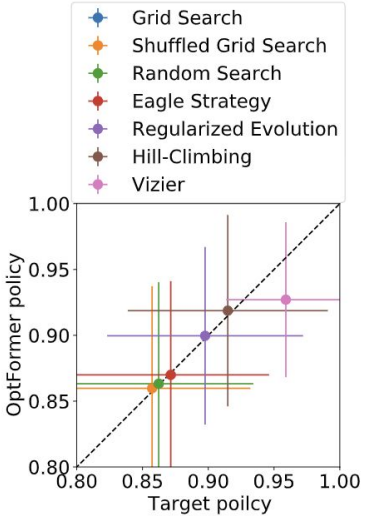
(b) Optimization trajectory

Parameter prediction - imitating HPO algorithms

- Prior policy imitates 7 HPO algorithms
  (same model, different algorithm name prompt)

```
..., algorithm: "Random Search",...
..., algorithm: "Regularized Evolution",...
..., algorithm: "Hill Climbing",...
..., algorithm: "GP-UCB",...
```

- Evaluation: compare imitated policy with behavior policy
  - Predict distribution: $p(x|m,\ h)$
  - Trajectory of function values
  - Policy value for all algorithms



(c) Average best function at trial 100 with standard deviation.

Function prediction

- Predict y_t given a short history of observations

$$p(y_t|m, \mathbf{h}_{t-1}, \mathbf{x}_t)$$

- Evaluation: compare with the GP model
  - Better log-predictive likelihood
  - Better calibration of uncertainty

Table 4: Log-predictive likelihood (with 1-std. standard error, higher is better (↑)) and ECE (percentage of error, lower is better (↓)) on RealWorldData and HPO-B test sets.

	Log-predictive likelihood ↑	
Model	RealWorldData	HPO-B
GP	0.83(0.06)	4.03(0.04)
OPTFORMER	**2.12 (0.05)**	**6.16 (0.04)**
	ECE (percent %) ↓	
Model	RealWorldData	HPO-B
GP	5.34 (0.06)	2.39 (0.05)
OPTFORMER	**1.11 (0.02)**	**1.89 (0.01)**

Function prediction

- Predict y_t given a short history of observations

$$p(y_t|m, \mathbf{h}_{t-1}, \mathbf{x}_t)$$

- Evaluation: compare with the GP model
  - Better log-predictive likelihood
  - Better calibration of uncertainty

Assuming an oracle model of $p(y_t|\mathbf{x}_t, \dots)$

$$CDF(y_t|\mathbf{x}_t, \dots) \sim \text{Uniform}[0, 1]$$

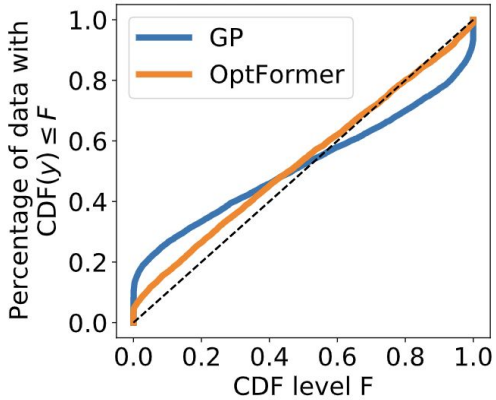
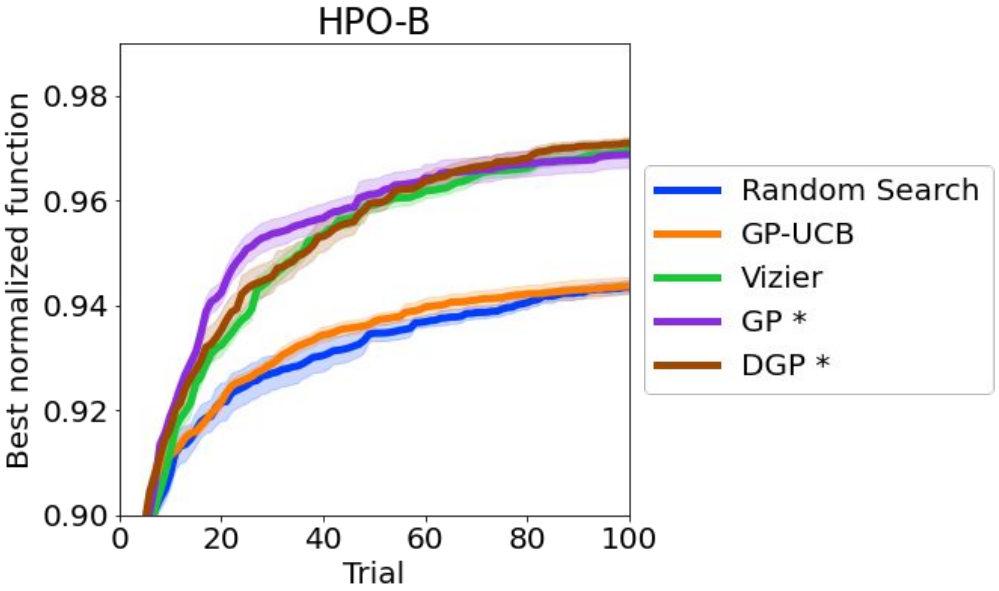


Figure 3: Cumulative histogram of predicted CDF(y) on RealWorldData test set.

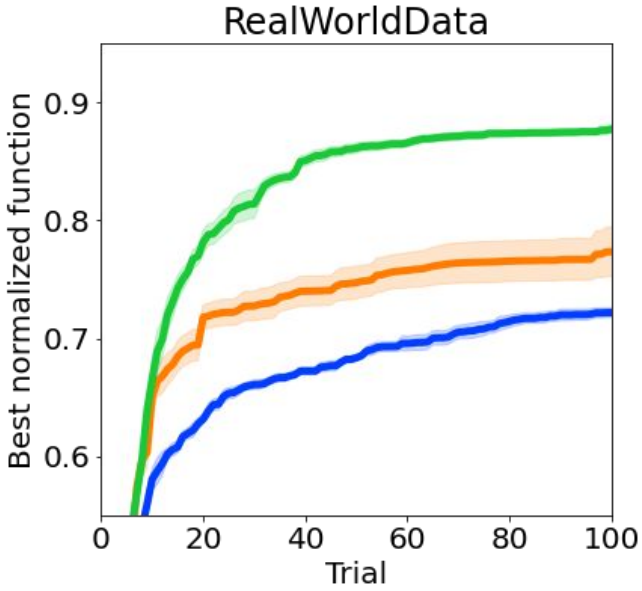
(goodness-of-fit, Rosenblatt (1952))

Augment behavior policy with EI

In-domain transfer

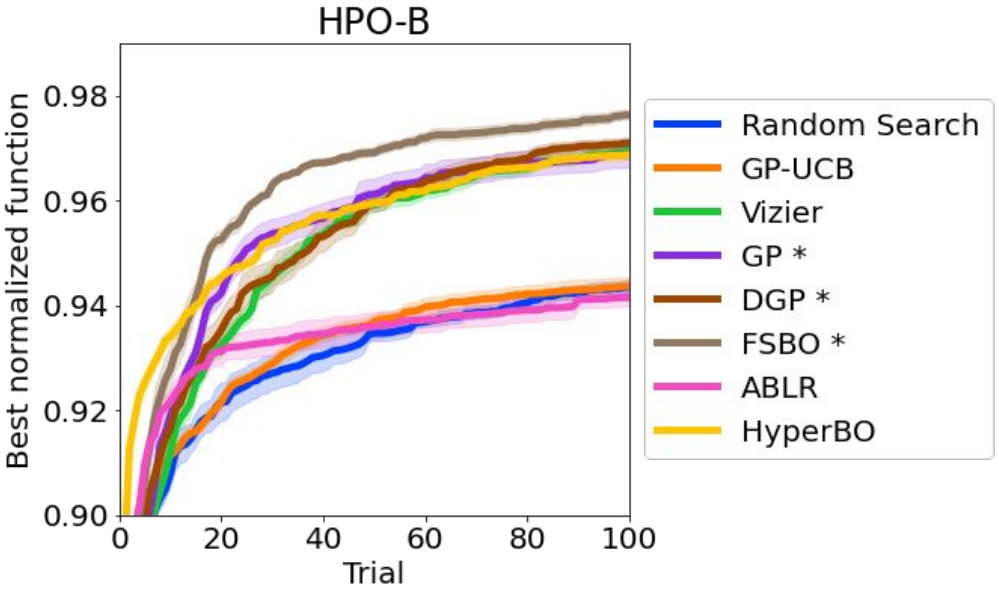


Out-of-domain transfer

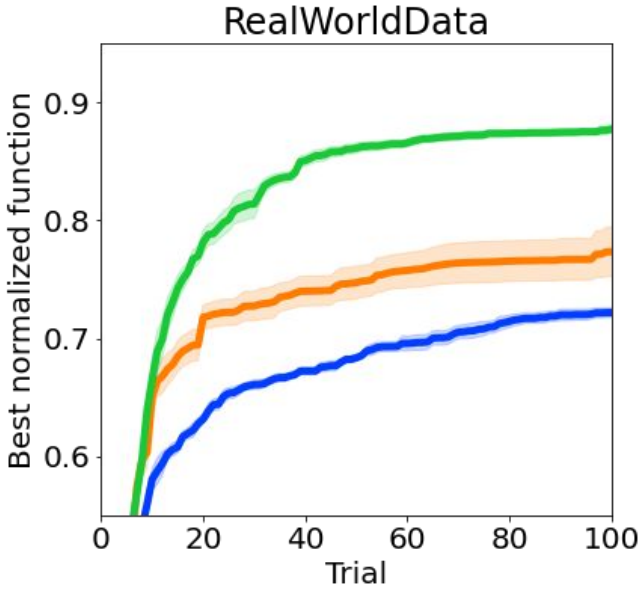


Augment behavior policy with EI

In-domain transfer

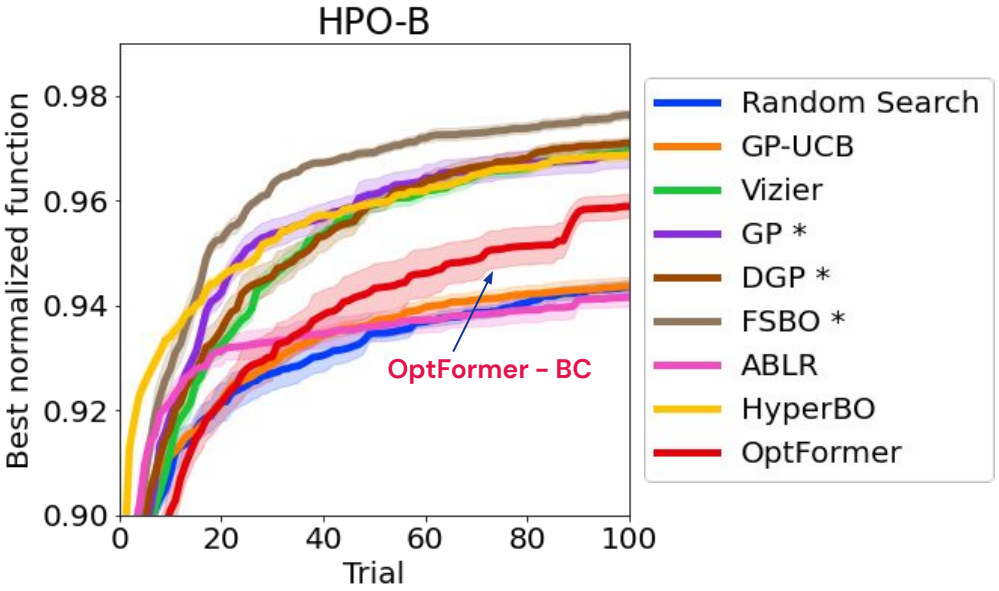


Out-of-domain transfer

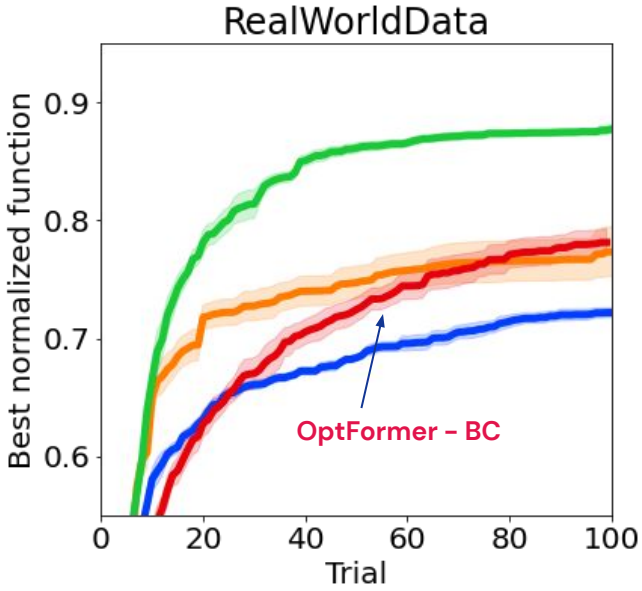


Augment behavior policy with EI

In-domain transfer

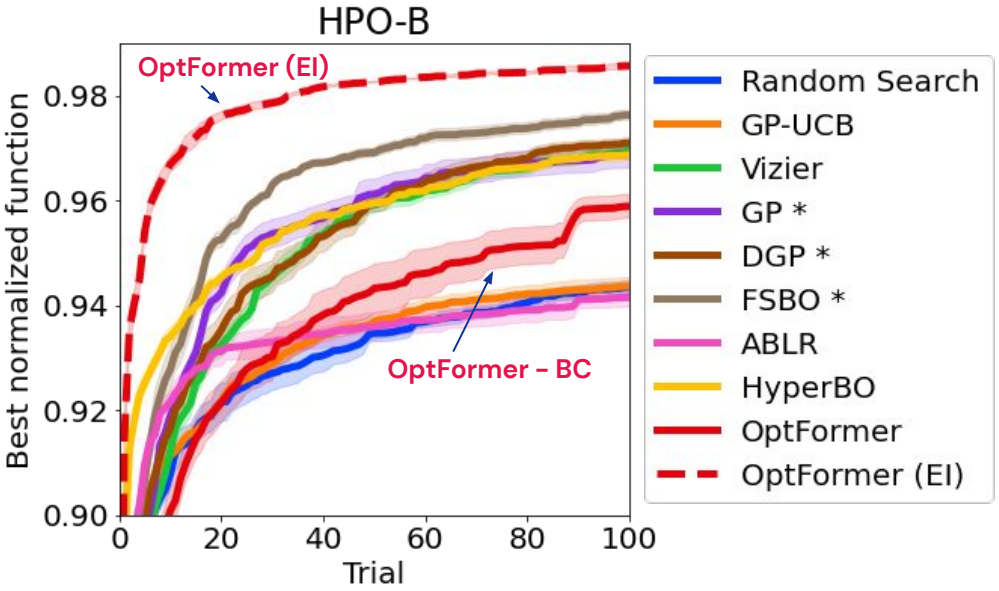


Out-of-domain transfer

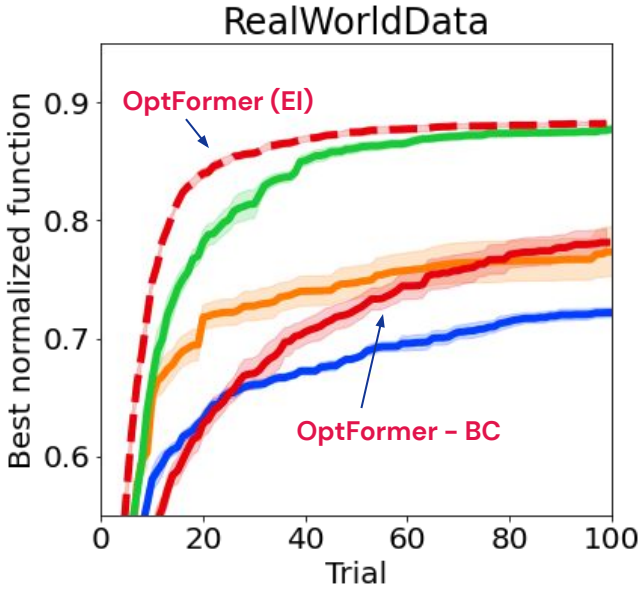


Augment behavior policy with EI

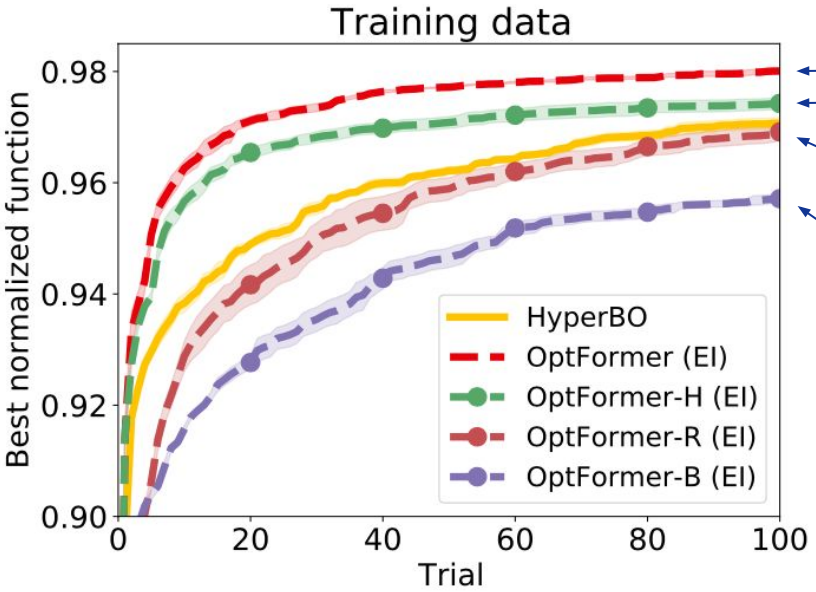
In-domain transfer



Out-of-domain transfer



Ablation on HPO-B: training data

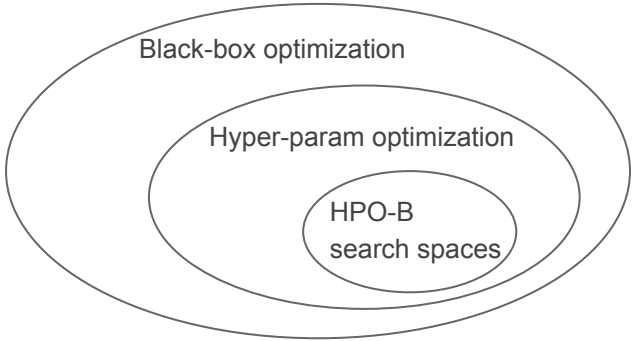


All data

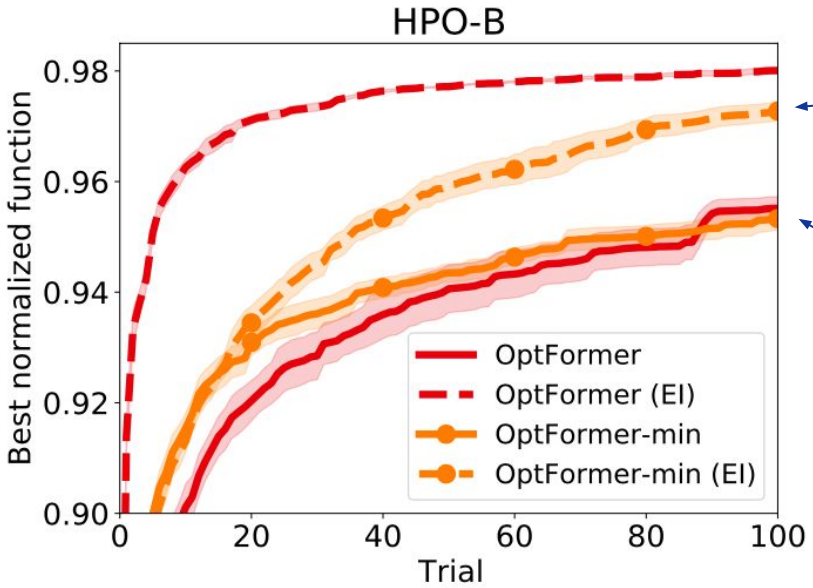
In-domain data

Out-of-domain HPO data (RealWorldData)

Out-of-domain BBO data (BBOB)



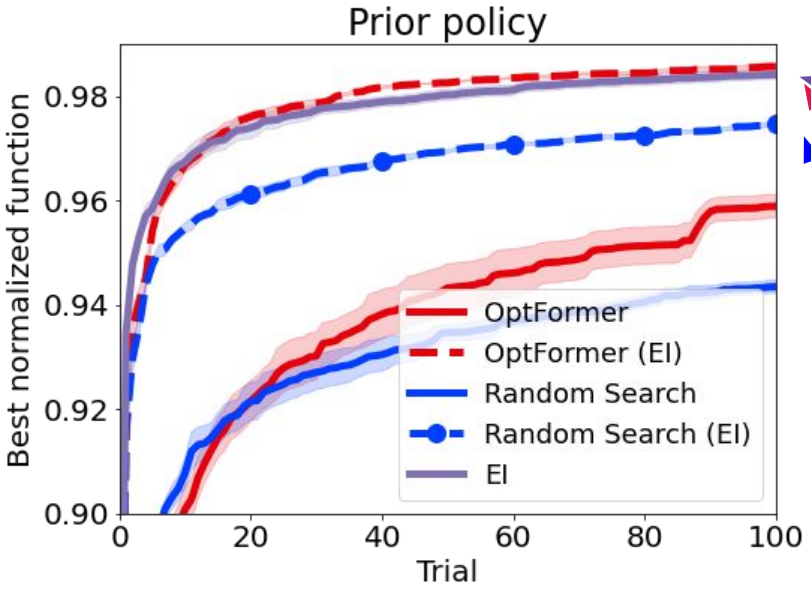
Ablation on HPO-B: metadata



Augmented policy with minimum metadata

Prior policy with minimum metadata

Ablation on HPO-B: prior policy

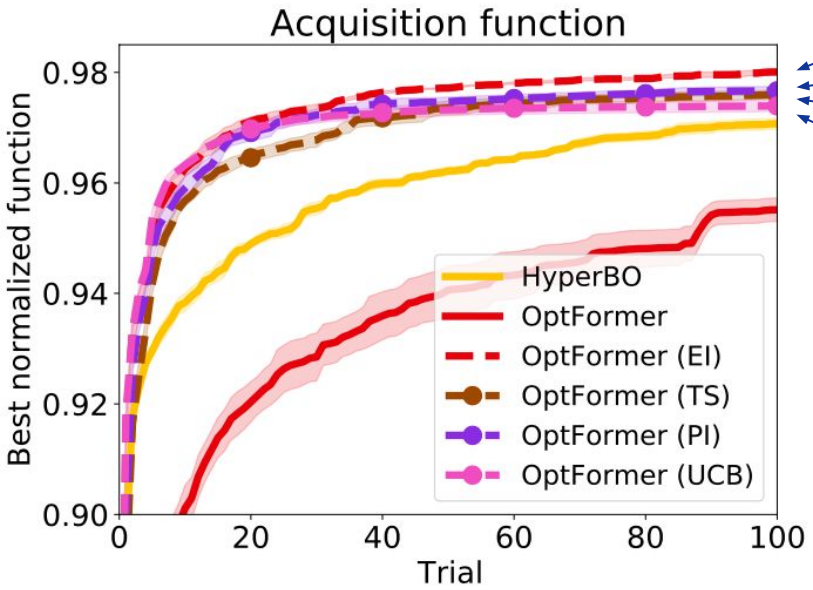


Global search to optimize EI (6400 evals)

100 ***prior samples*** + rank with EI

100 ***random samples*** + rank with EI

Ablation on HPO-B: acquisition function



EI

PI

Thompson sampling*

UCB

Outlines

- Why learn a hyperparameter optimizer?
- How to learn it from existing data?
- Does it work?
- Summary and future work

Summary:

- First step to learning a universal Transformer model for hyperparameter optimization from large scale datasets of tuning experiments.
- A unified interface for tuning experiment data, containing vastly different search spaces and free form task descriptions.
- Imitate 7 fundamentally different HPO policies
- Well calibrated few-shot function predictions
- Competitive optimization performance on unseen test functions comparable with the existing, long-tried GP-based baselines.

Limitations and potential solutions:

- Flat search space
  - Specify parameter dependencies in metadata `..., name: beta1, parent: algo=adam,...`
  - Dynamical parameter list in a trial `..., algo=adam, beta1=0.9, ... | algo=sgd,...`
  - Extension to NAS, AutoML, …

- Sequential optimization (batch size = 1)
  - Random masking function values during training `x1 * y1 | x2 * ∅ | x3 * ∅ | x4 * y4 | ...`
  - Prior policy + batch acquisition function in BO literature

- Go beyond behavioral cloning
  - Offline RL
  - Model-based planning (MPC, MCTS, …)

- Single objective
  - Multi-objective in a trial `x1 * y1(1) y1(2) | x2 * y2(1) y2(2) | ...`

- A lot more ideas for future extensions

Let Transformers learn HPO from all data

TRAINING MODEL

OptFormer