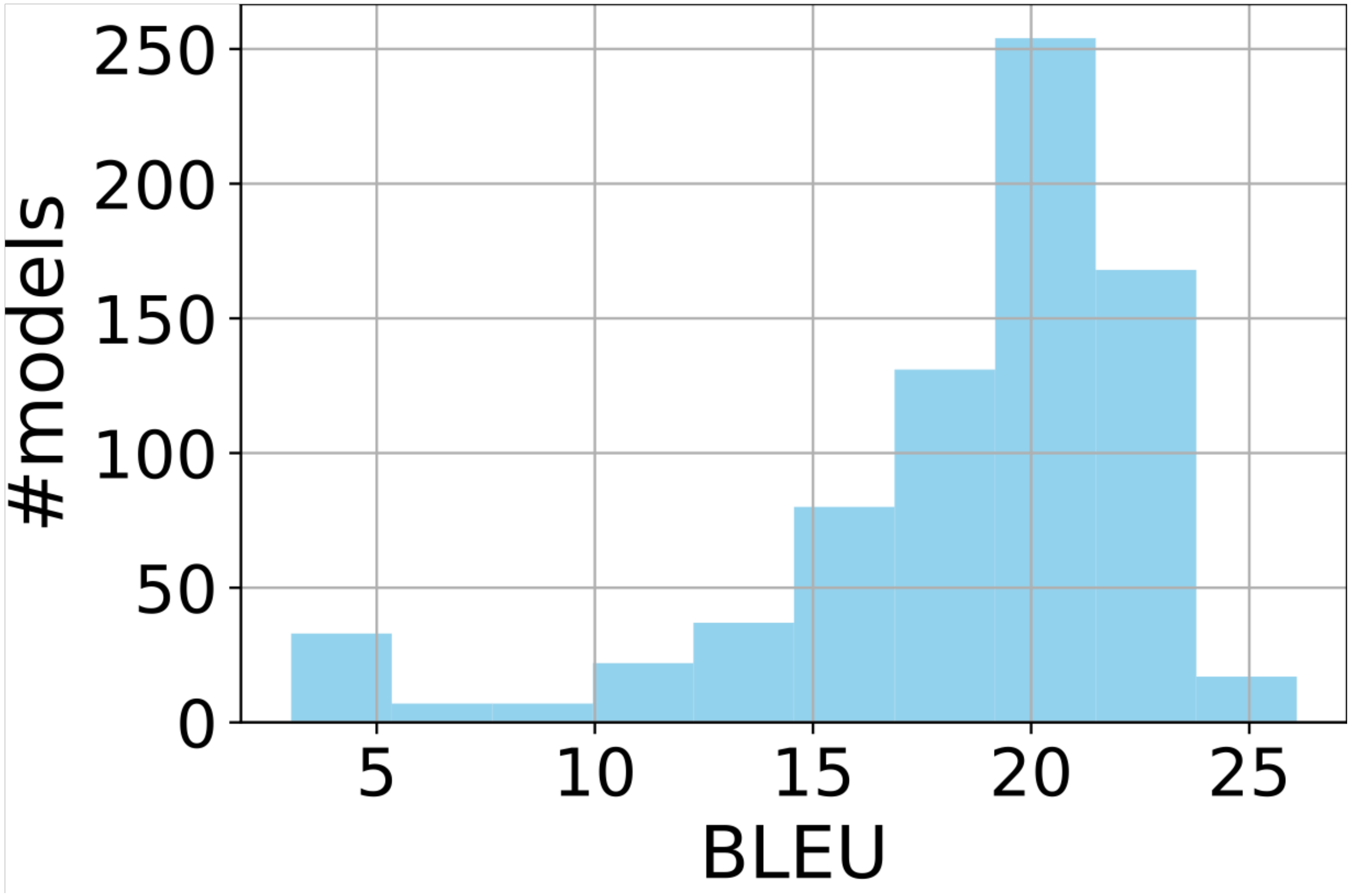
AutoML for Natural Language Processing

Kevin Duh and Xuan Zhang

Johns Hopkins University

It's important to tune hyperparameters!

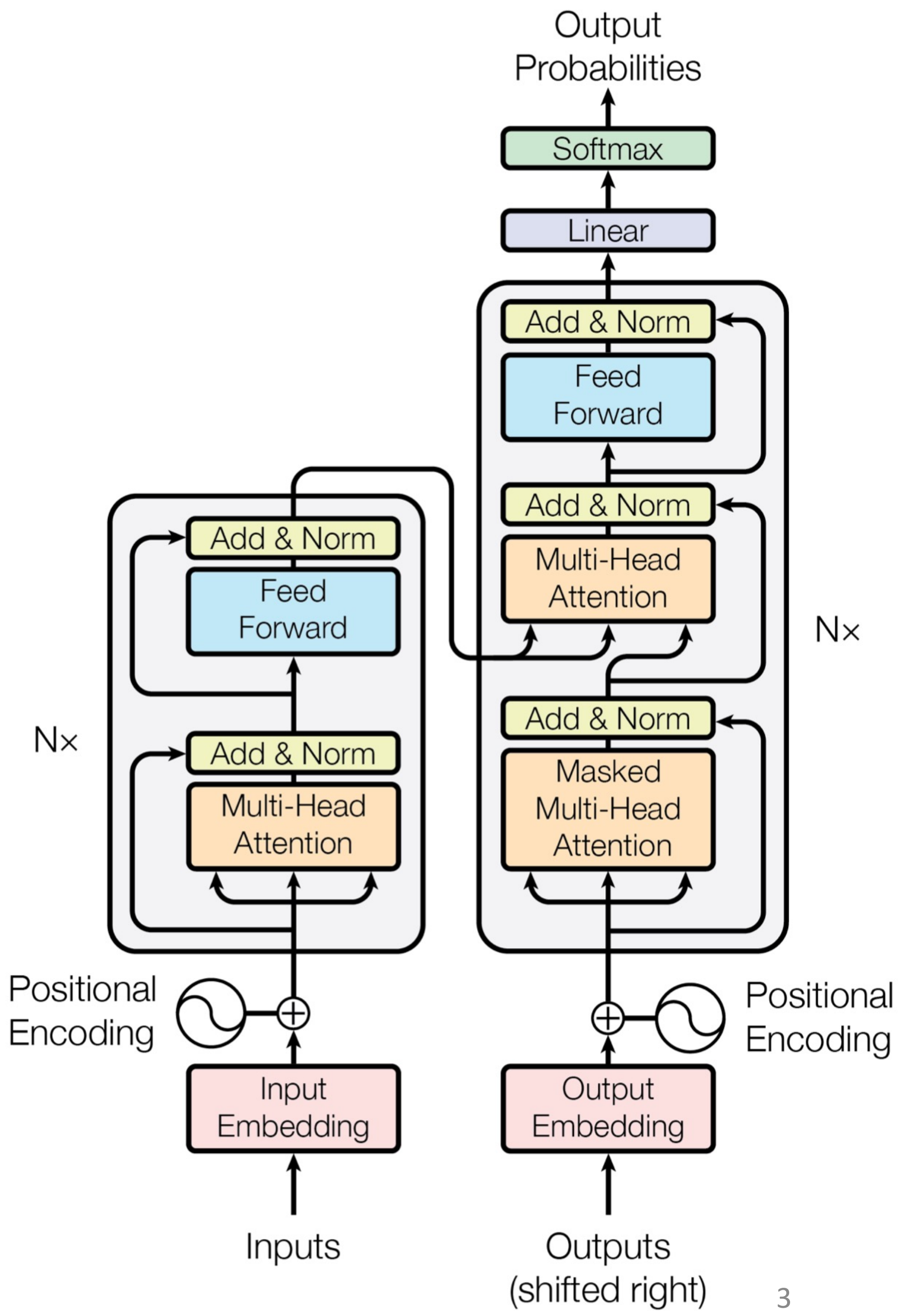


Histogram of BLEU scores for 700+ Swahili-English Neural Machine Translation (NMT) models

Note the large variance!

Hyperparameters

- Hyperparameters = Configurations of a model that are not updated in training
- Architectural hyperparameters:
  - # of layers
  - # of hidden units in feed-forward layer
  - # attention heads
  - Word embedding dimension
- Training pipeline hyperparameters:
  - # of subword units
- Optimizer hyperparameters:
  - Initial learning rate for ADAM, etc.



AutoML: Automated Machine Learning – what it might mean to different people

- For consumers: Democratization of ML
  - Upload own data, get ML model that can be plugged in application
- For developers: Reduce effort
  - Automate part of model building pipeline, more time for other priorities
  - Especially useful for optimizing models with speed-accuracy tradeoff
- For NMT researchers: Obtain state-of-the-art results
  - Fair comparison of methods
- For (some) ML researchers: Discover the next "Transformer"

AutoML: Automated Machine Learning – what it might mean to different people

- For consumers: Democratization of ML
  - Upload own data, get ML model that can be plugged in application

Focus of this talk

- For developers: Reduce effort
  - Automate part of model building pipeline, more time for other priorities
  - Especially useful for optimizing models with speed-accuracy tradeoff
- For NMT researchers: Obtain state-of-the-art results
  - Fair comparison of methods
- For (some) ML researchers: Discover the next "Transformer"

AutoML as an umbrella term

- Topics that might appear at an AutoML conference
  - Hyperparameter Optimization (HPO)
  - Neural Architecture Search (NAS)
  - Meta-Learning
  - Automated Reinforcement Learning (AutoRL)
  - Algorithm Selection
  - Systems for Machine Learning (SysML)

Goal of this tutorial

- Motivate the importance of proper hyperparameter tuning or architecture search
- Explain a few popular methods in HPO and NAS (focus in-depth on a few illustrative methods, then describe general categorizations)
- Case study: describe our experiences in applying AutoML, hope it serves as a reference for you
- We hope AutoML will someday be a useful part of your toolbox!

Roadmap

1. Motivation for AutoML
2. Hyperparameter Optimization (HPO)
3. Neural Architecture Search (NAS)
4. Extension to Multiple Objectives
5. Evaluation
6. Toolkits
7. Survey of HPO/NAS in NLP

Roadmap

1. Motivation for AutoML
2. Hyperparameter Optimization (HPO)
    - Problem Formulation
    - Representative methods:
        - Bayesian Optimization
        - Grid/Random Search
        - Evolutionary strategies
        - Population-Based Training (PBT)
        - Hyperband
    - Generalizations
3. Neural Architecture Search (NAS)
4. Extension to Multiple Objectives
5. Evaluation
6. Toolkits
7. Survey of HPO/NAS in NLP

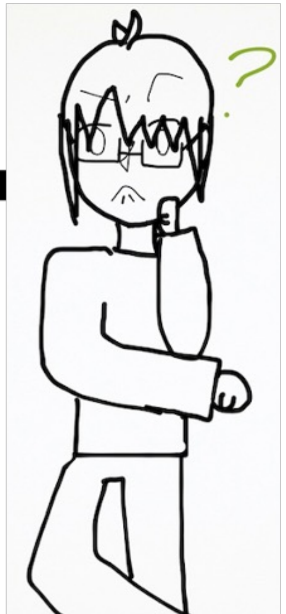
Problem Definition:
Hyperparameter Optimization (HPO)

Hyperparameter setting encoded as vector in R^d

$$\begin{pmatrix} 3 \\ 200 \\ 1 \\ 0.2 \end{pmatrix} \begin{array}{l} \rightarrow \text{\# layers} \\ \rightarrow \text{\# units/layer} \\ \rightarrow \text{optimizer type} \\ \rightarrow \text{learning rate} \end{array}$$

x → **Train Model on dataset, then run diagnostics** → **f(x)**

Accuracy (e.g. BLEU)



Find $x^* = \text{argmax}_x f(x)$ with few function evaluations

*Clipart by Lea Duh

Problem Definition: Hyperparameter Optimization (HPO)

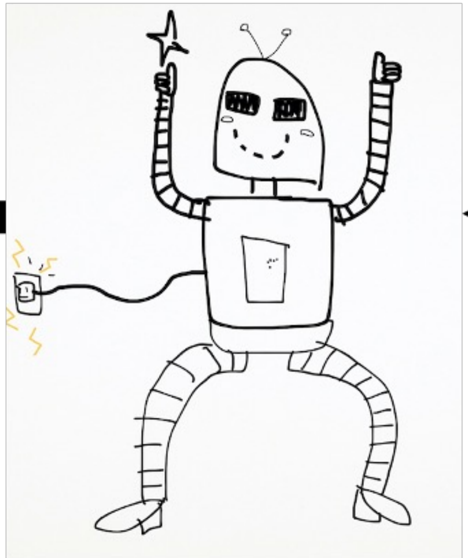
Hyperparameter setting encoded as vector in R^d

$$\begin{pmatrix} 3 \\ 200 \\ 1 \\ 0.2 \end{pmatrix} \begin{matrix} \rightarrow \text{\# layers} \\ \rightarrow \text{\# units/layer} \\ \rightarrow \text{optimizer type} \\ \rightarrow \text{learning rate} \end{matrix}$$

f(.)

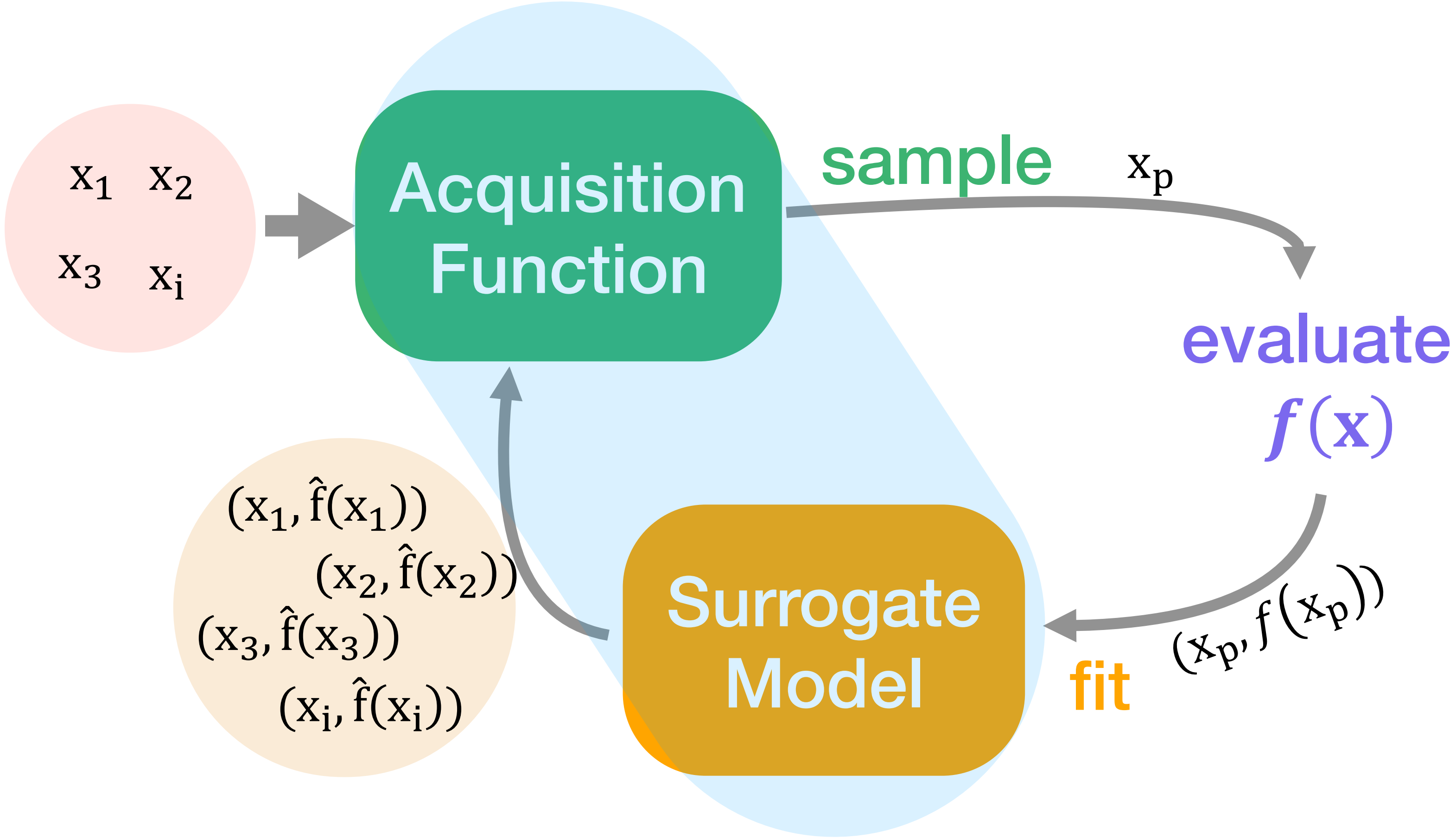
x → Train Model on dataset, then run diagnostics → **y=f(x)**

Accuracy (e.g. BLEU)

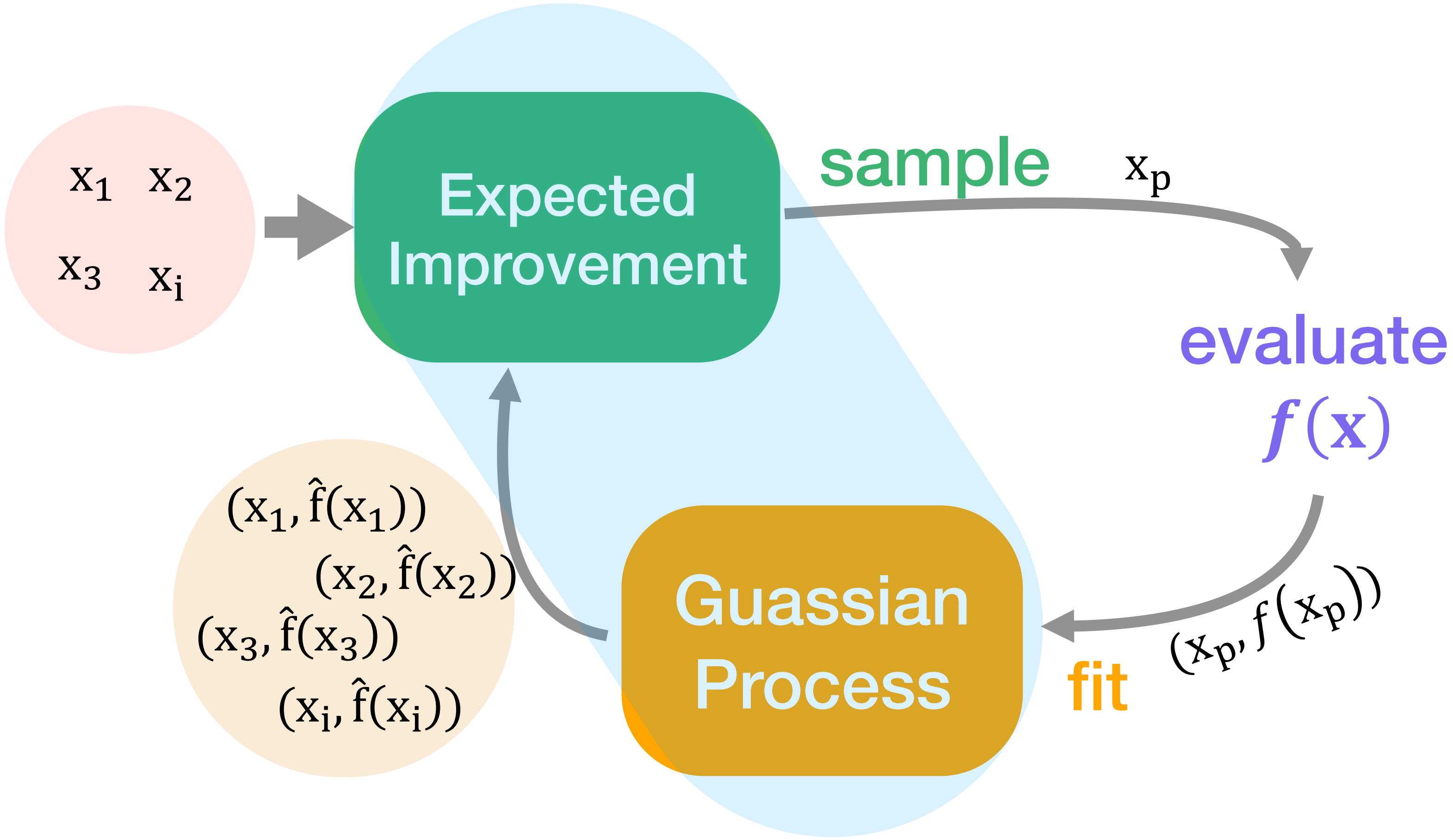


Find $x^* = \text{argmax}_x f(x)$ with few function evaluations

Sequential Model-Based Optimization (SMBO)

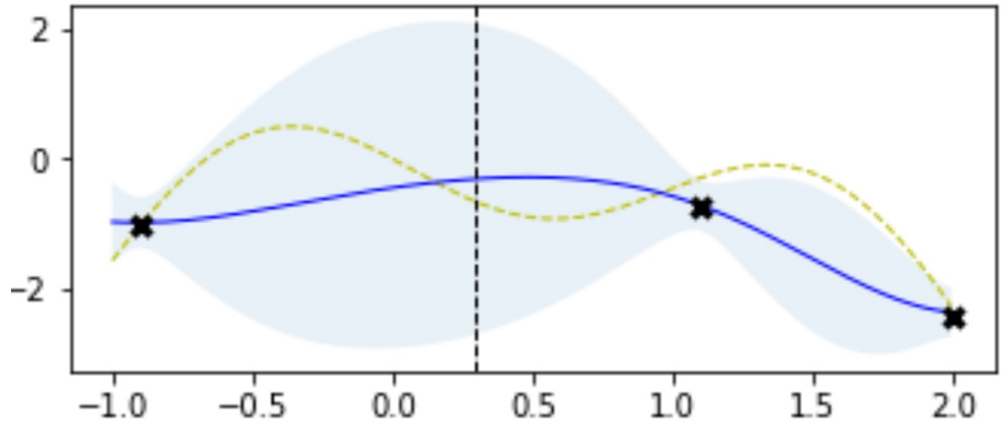


Bayesian Optimization

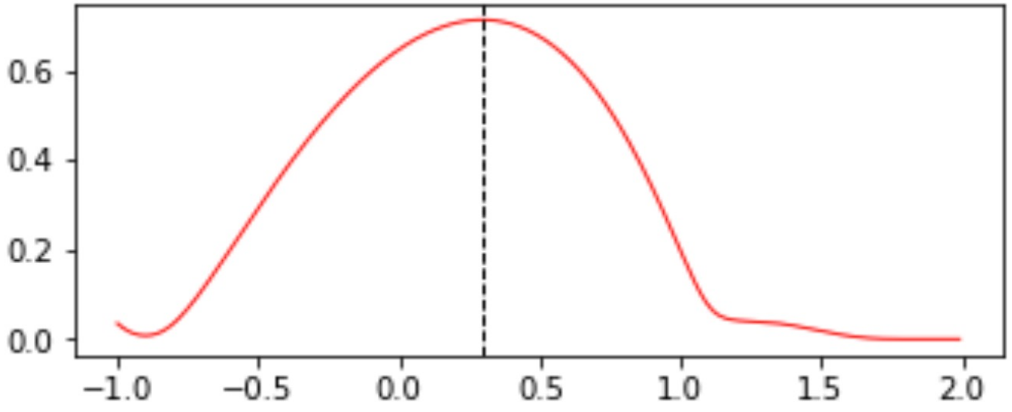


Bayesian Optimization

f(x) objective function
Gaussian Process Prediction
Samples
Expected Improvement
Next Sampling Location



Gaussian Process:
fit with uncertainty

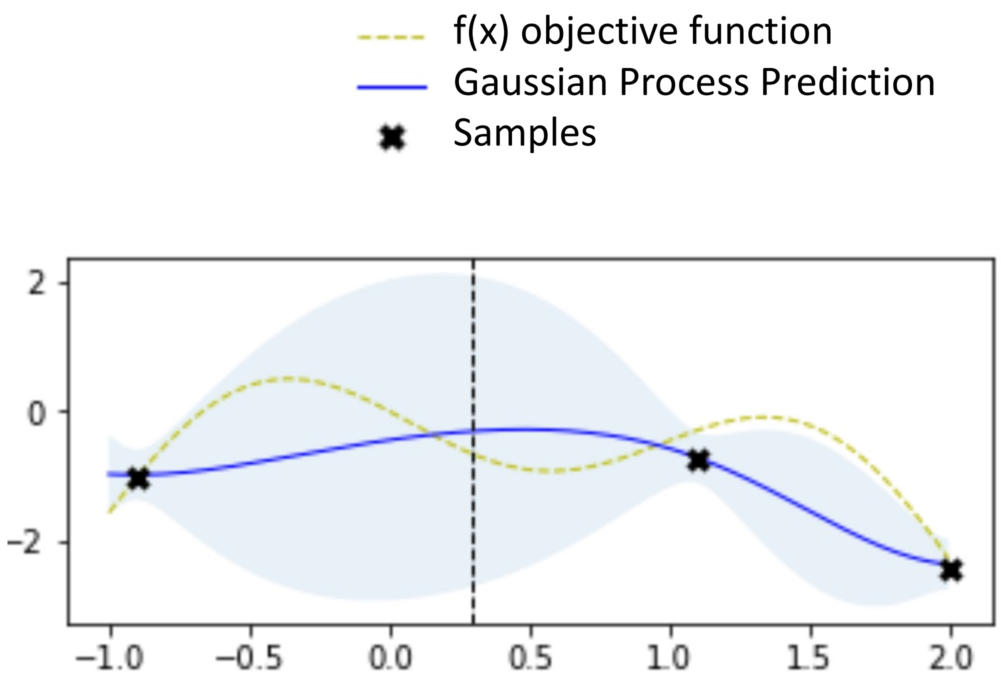


Expected Improvement:
exploitation vs. exploration

From: Martin Krasser. http://krasserm.github.io/2018/03/21/bayesian-optimization/

Gaussian Process Regression

- Nonparametric / kernel methods
- $f_{GP}(x_{1:n})$ is jointly Gaussian; i.e. GP fits each $f_{GP}(x)$ w/ a Gaussian distribution.
- To predict x_{new}, GP compares how "similar" it is to $x_{1:n}$, which is measured by kernel.
- $\mu(x_{new})$ depends on the prior $\mu_0(x_{new})$ & $f(x_{1:n})$

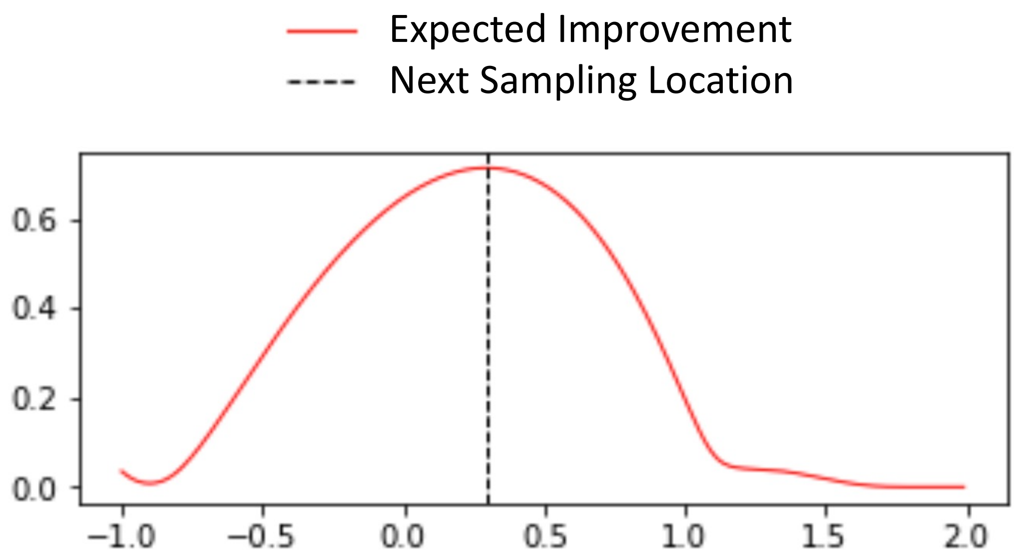


From: Martin Krasser. http://krasserm.github.io/2018/03/21/bayesian-optimization/

Expected Improvement

Definition:

$$\mathrm{EI}_n(x) := E_n\left[[f(x) - f_n^*]^+\right]$$



From: Martin Krasser. http://krasserm.github.io/2018/03/21/bayesian-optimization/

Expected Improvement

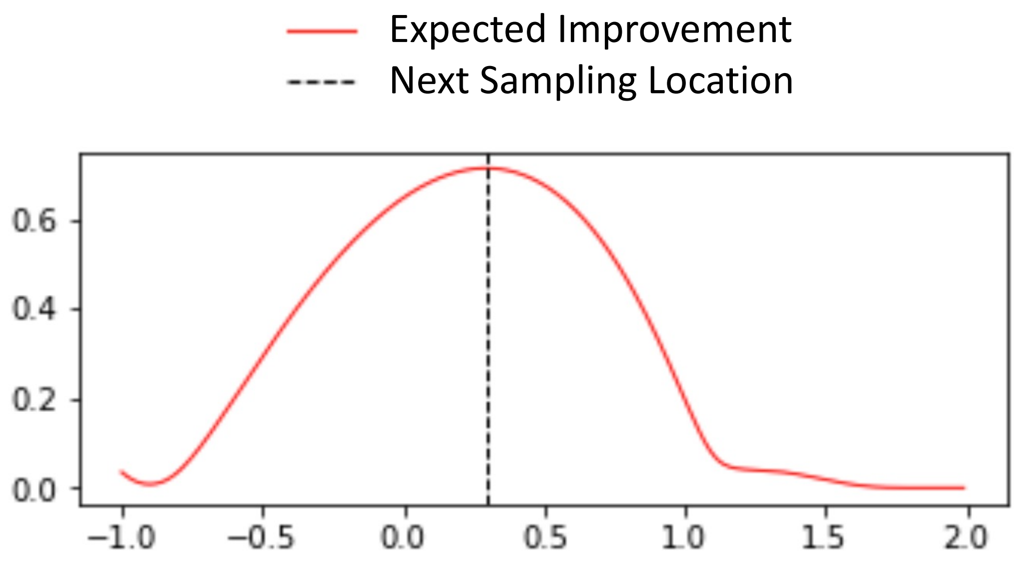
Definition:

$$\mathrm{EI}_n(x) := E_n\left[[f(x) - f_n^*]^+\right]$$

Expected quality Expected uncertainty

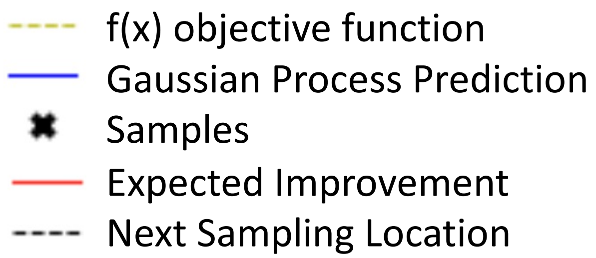
$$EI(x) = \boxed{(f^* - \mu)}\Phi(\frac{f^* - \mu}{\sigma}) + \boxed{\sigma}\phi(\frac{f^* - \mu}{\sigma})$$

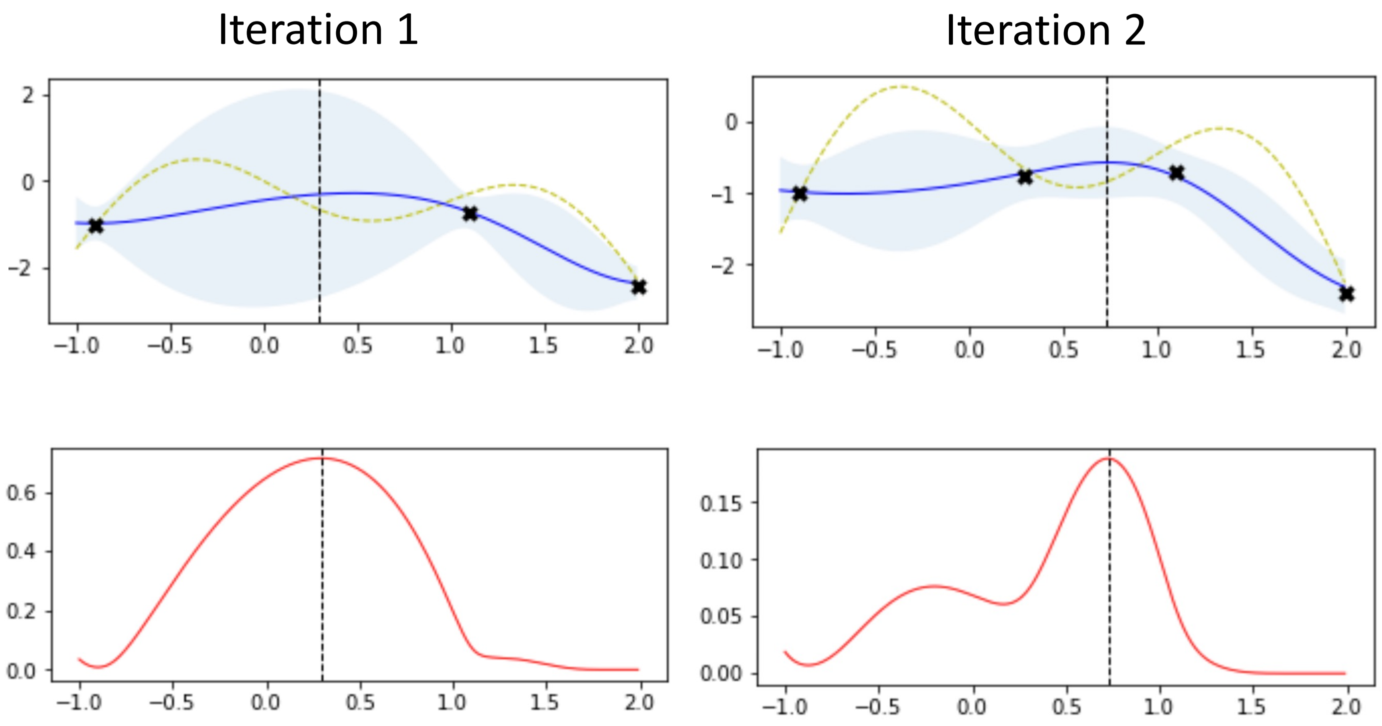
where ϕ, Φ are the PDF, CDF of standard normal distribution.



From: Martin Krasser. http://krasserm.github.io/2018/03/21/bayesian-optimization/

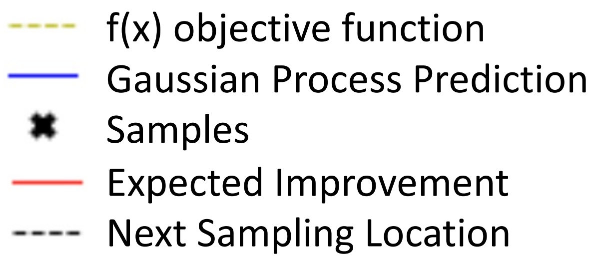
Bayesian Optimization

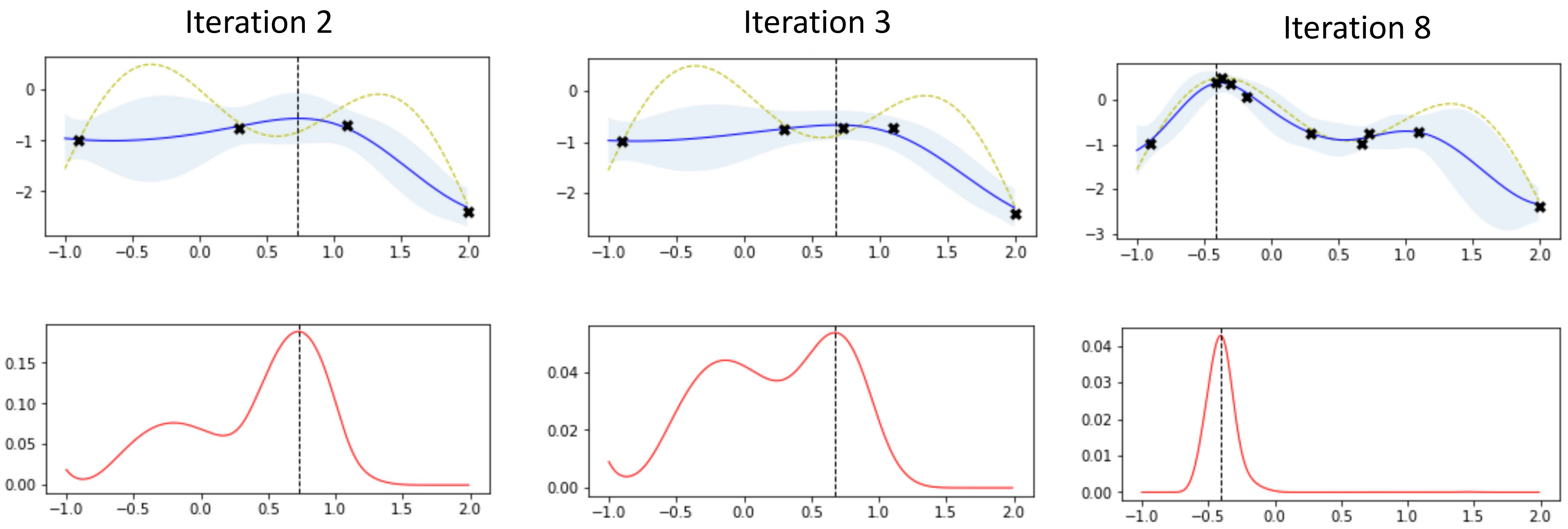




From: Martin Krasser. http://krasserm.github.io/2018/03/21/bayesian-optimization/

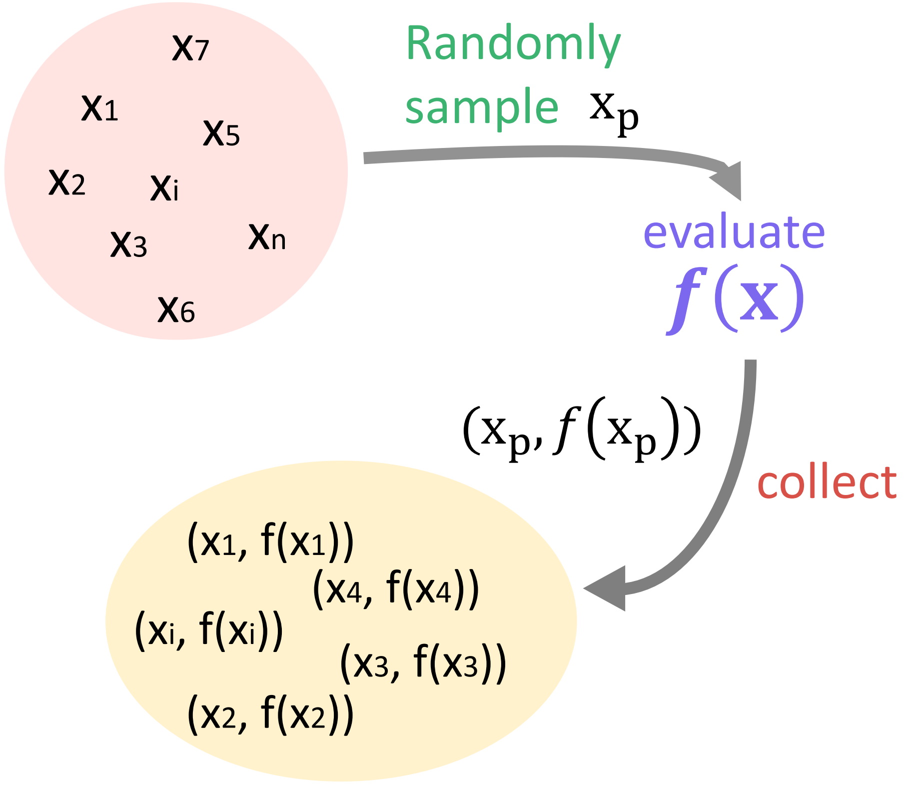
Bayesian Optimization





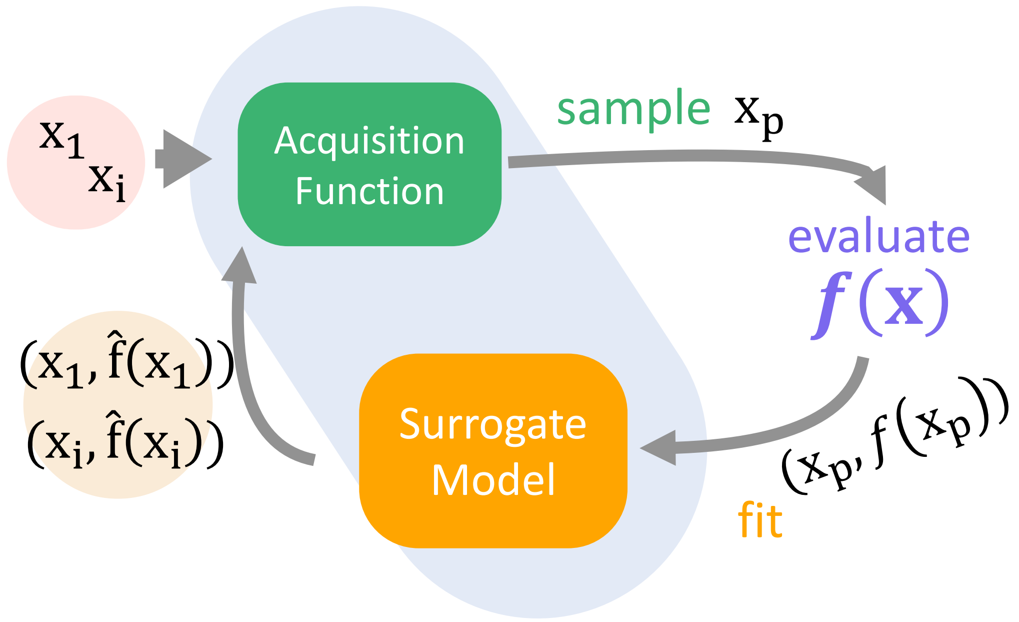
From: Martin Krasser. http://krasserm.github.io/2018/03/21/bayesian-optimization/

Random / Grid Search



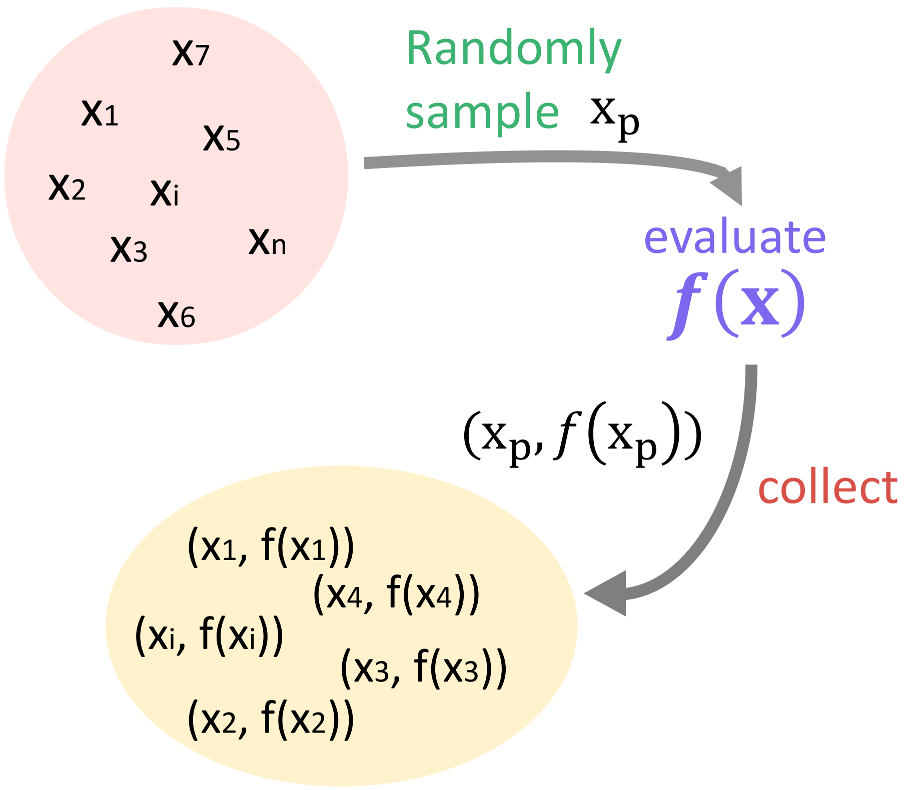
VS.

SMBO



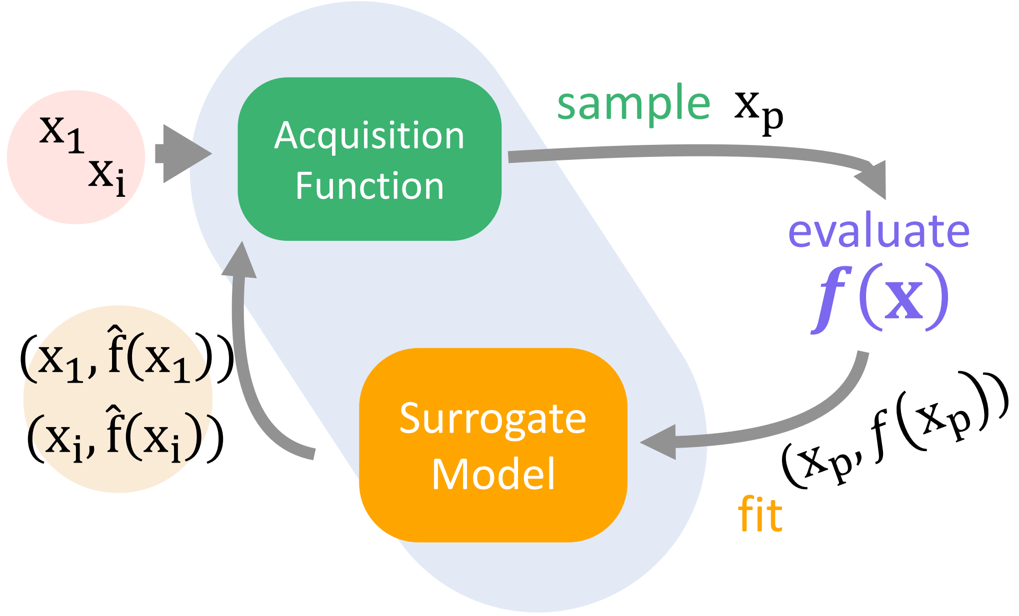
Random / Grid Search

easy to get parallelized

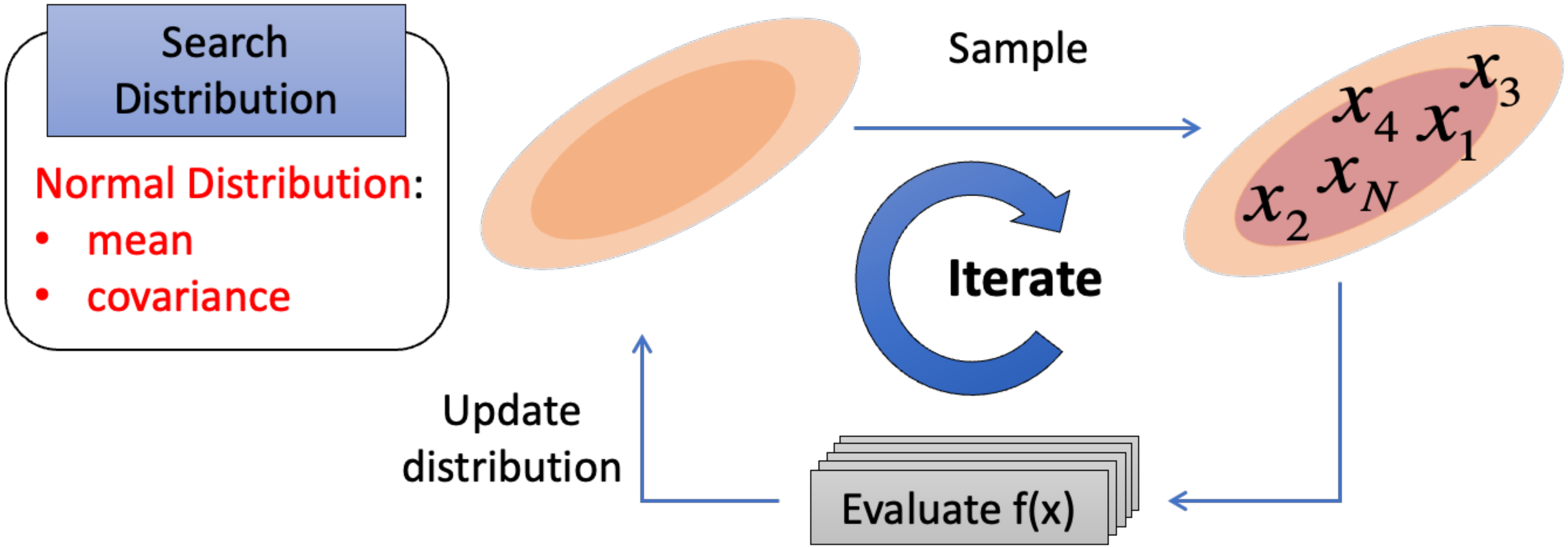


VS.

SMBO

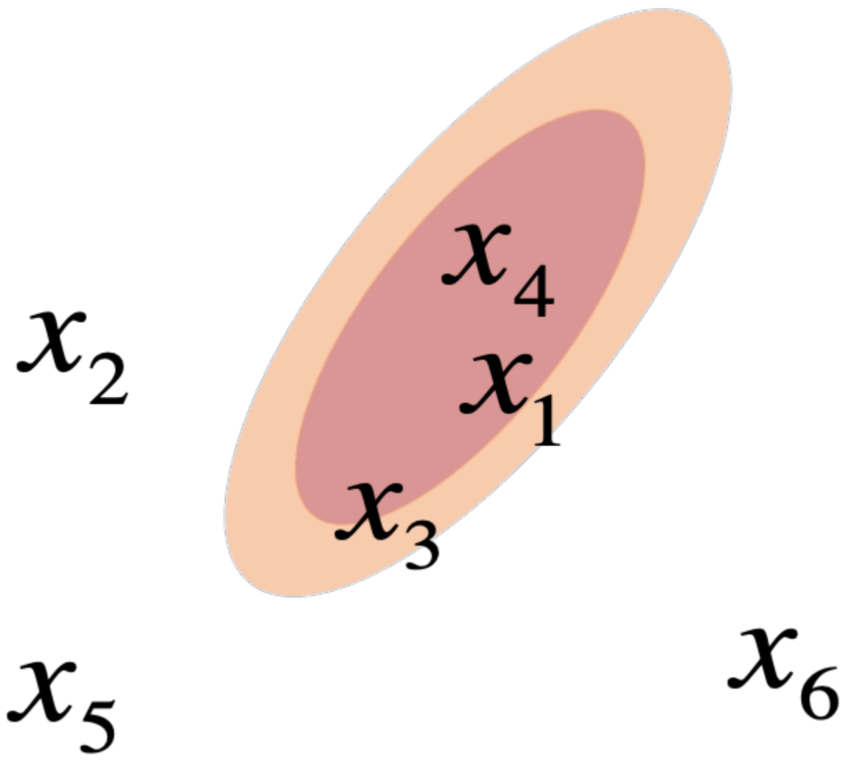


Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES)



N. Hansen, S. D. Muller, and P. Koumoutsakos, “Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES),” Evolutionary Computation, vol. 11, no. 1, pp. 1–18, 2003.

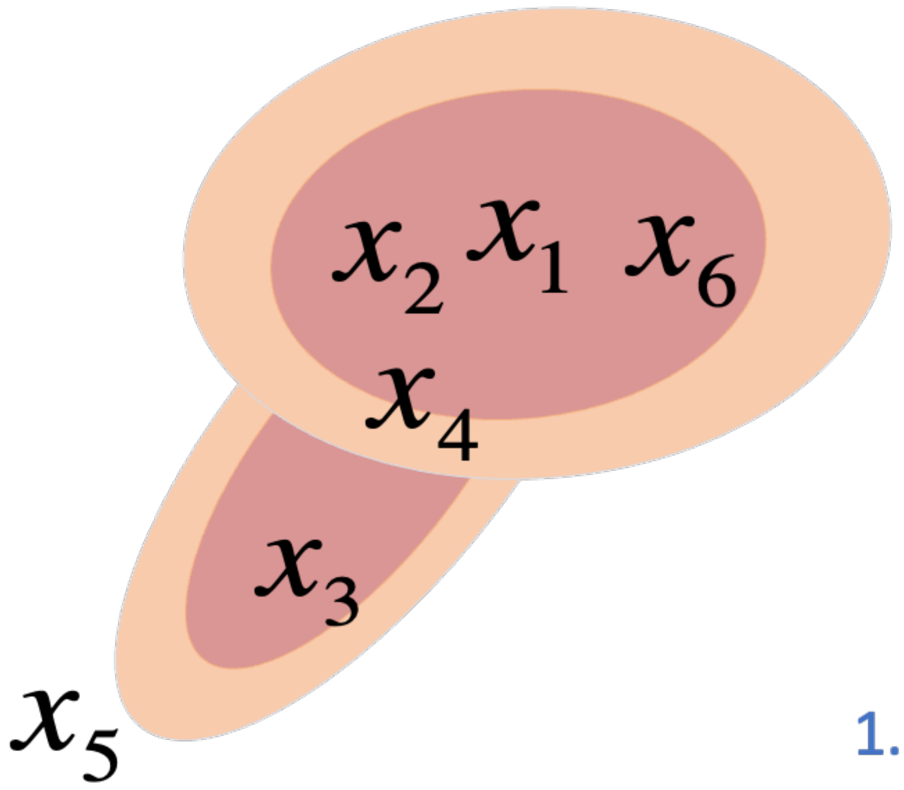
Evolutionary Strategy for HPO



Generation 0

1. Start with a population of "individuals", each representing a hyperparameter setting
2. The "fittest" ones (high **f(x)**) survive and produce offspring

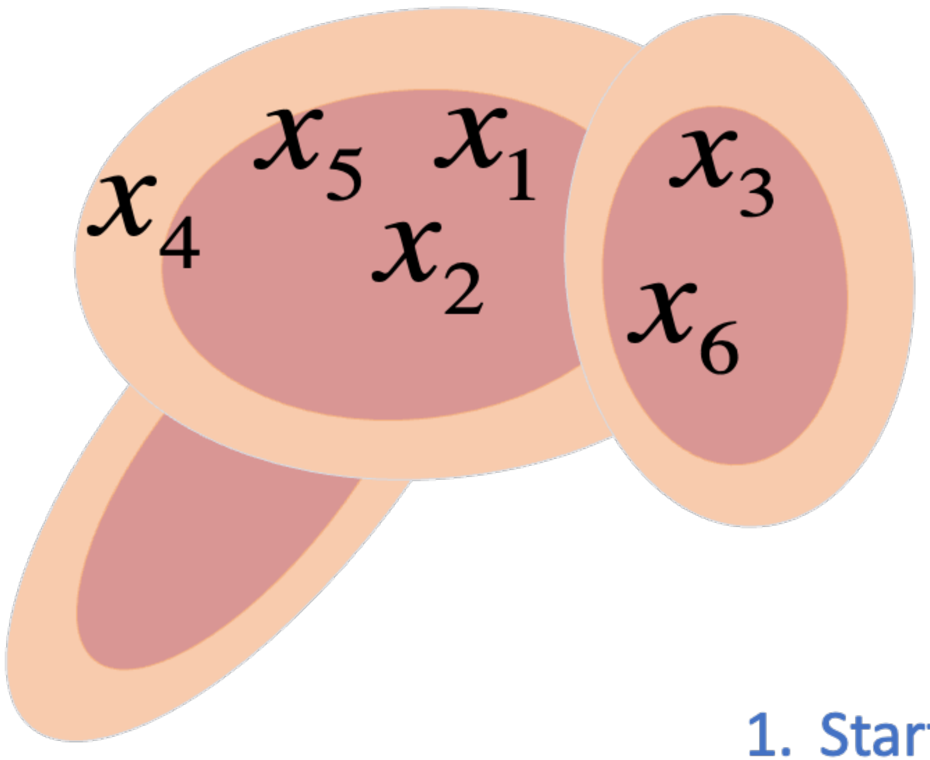
Evolutionary Strategy for HPO



Generation 1

1. Start with a population of "individuals", each representing a hyperparameter setting
2. The "fittest" ones (high **f(x)**) survive and produce offspring

Evolutionary Strategy for HPO

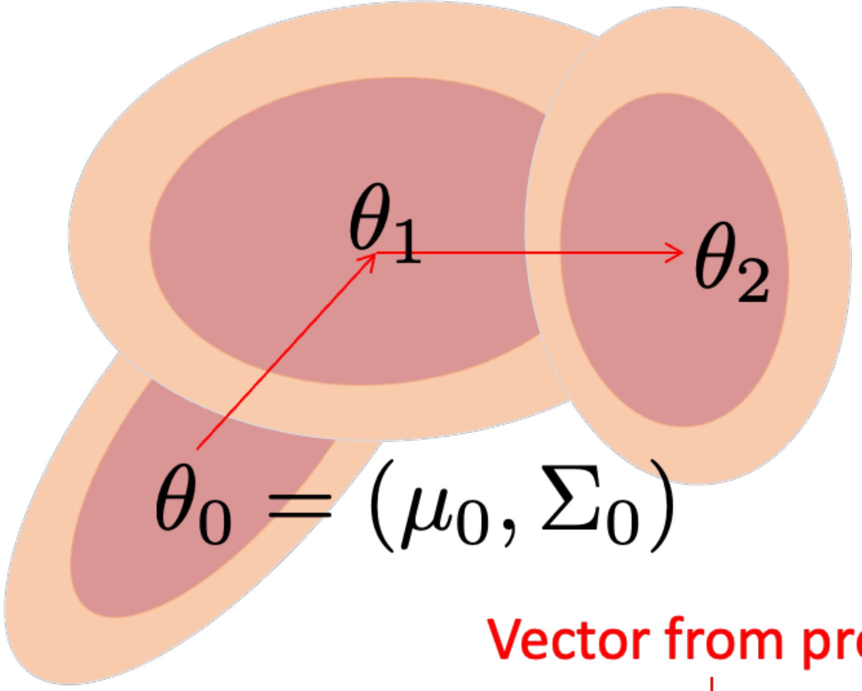


Generation 2

1. Start with a population of "individuals", each representing a hyperparameter setting
2. The "fittest" ones (high **f(x)**) survive and produce offspring

Estimating the search distribution

$$\hat{\theta} = \arg\max_{\theta} \underbrace{\int f(x)\mathcal{N}(x|\theta)dx}_{\triangleq \mathbb{E}[f(x)|\theta]}$$



Vector from previous mean to x_k

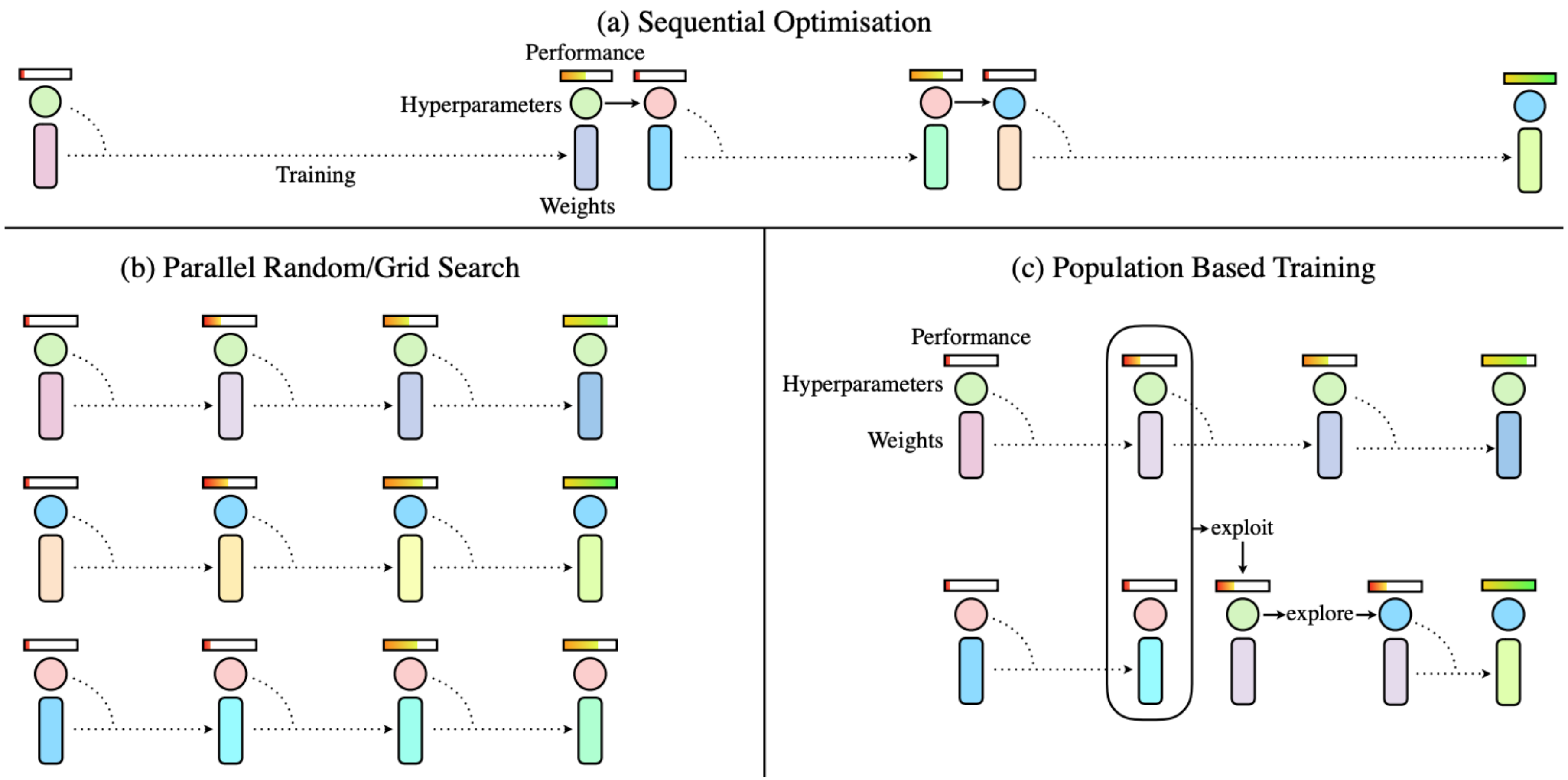
Mean update:

$$\hat{\mu}_n = \hat{\mu}_{n-1} + \epsilon_\mu \sum_{k=1}^{K} w(y_k)(x_k - \hat{\mu}_{n-1})$$

Mean of previous generation

Weight for individual (x_k, $y_k=f(x_k)$),
Better accuracy → higher weight

Population Based Training (PBT)



From : Population Based Training of Neural Networks, Jaderberg et al. 2017

Population Based Training (PBT)

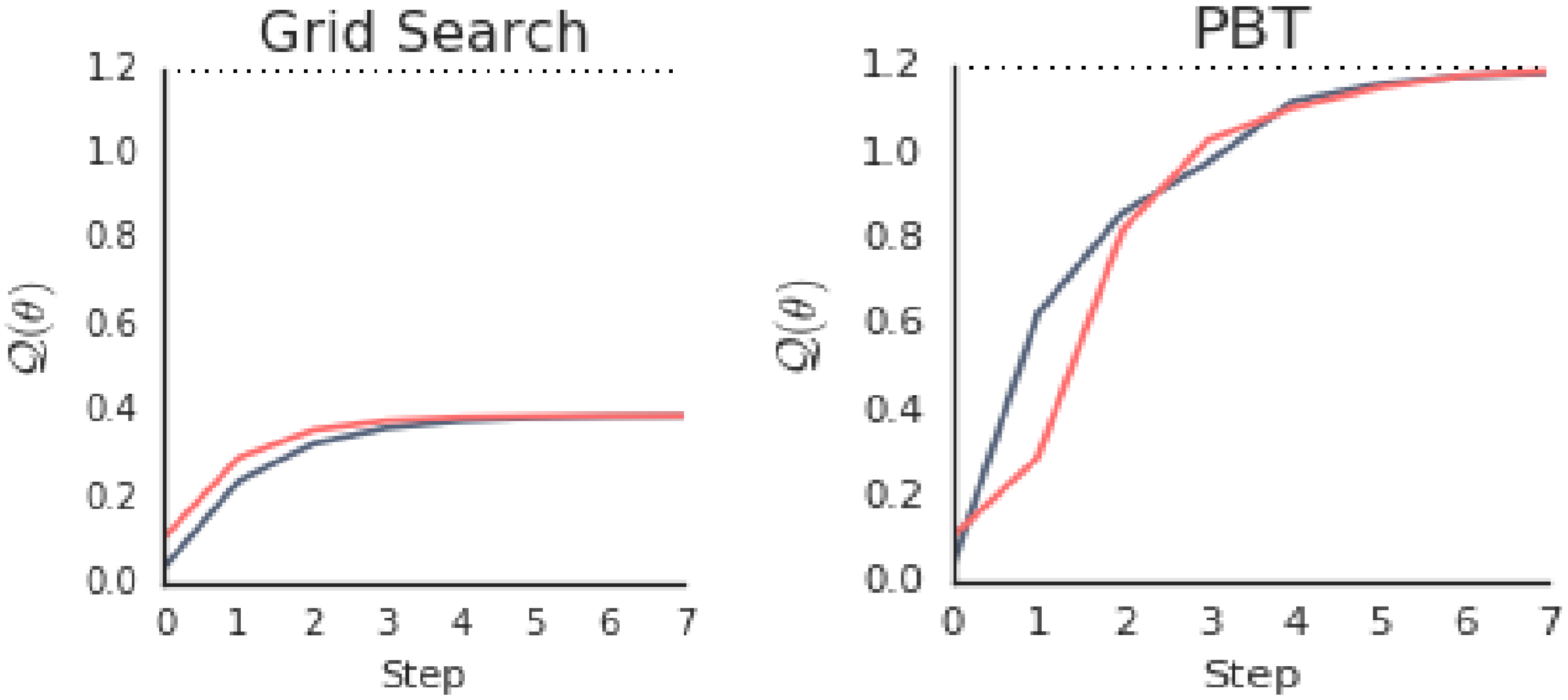


Figure. The objective function value of each worker over time.

From : Population Based Training of Neural Networks, Jaderberg et al. 2017

Go Beyond Blackbox HPO

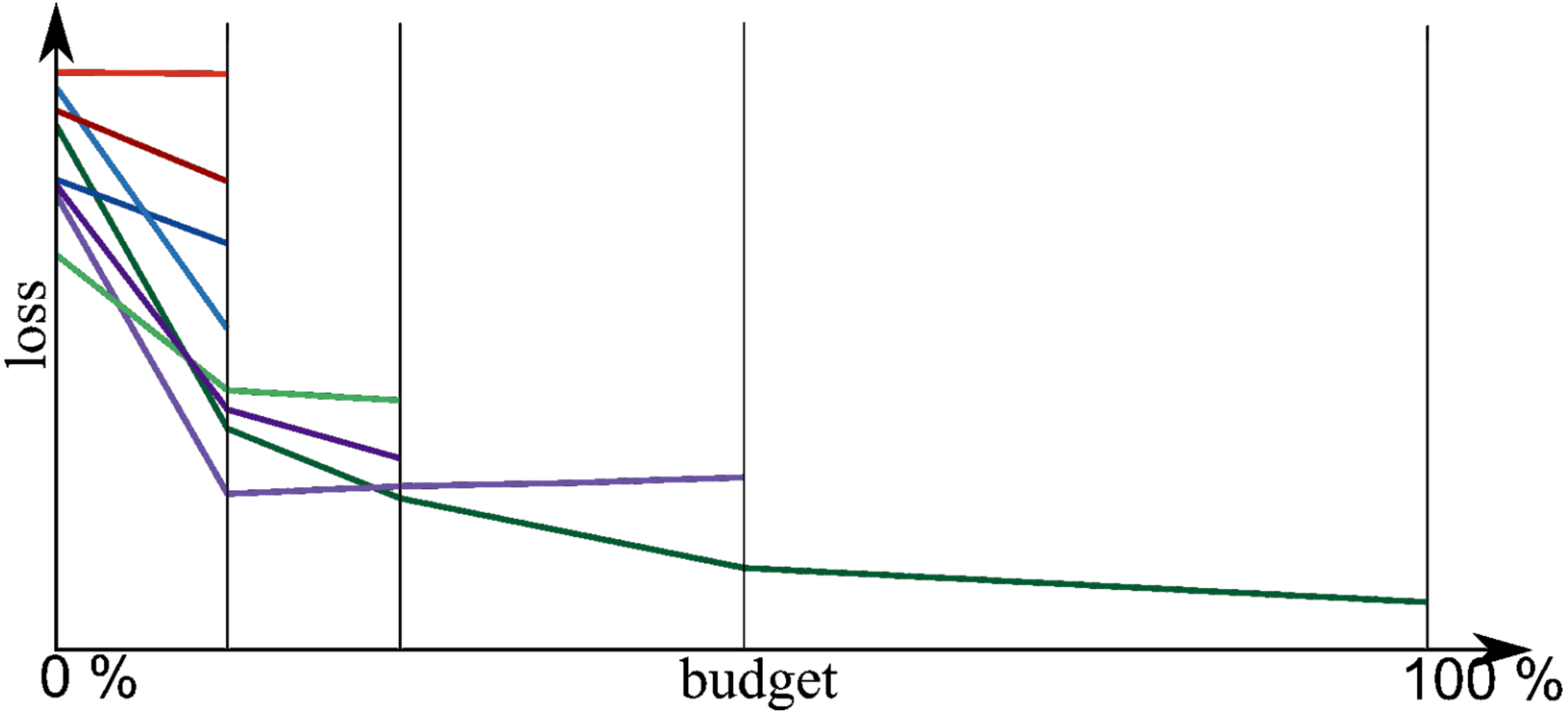
- No need to train to completion every time.
- Performance early in training is highly correlated with performance late in training. (Dodge, et al. 2020.)
- Multi-fidelity Optimization:

  Use cheap approximations of the blackbox.

  e.g. fewer training steps.

Successive Halving (SHA)

-- multi-armed bandit algorithm to perform early stopping



From : automl.org

Successive Halving (SHA)

Two inputs:
Budget B, #configs N

B/n: resources allocated on average across the configurations

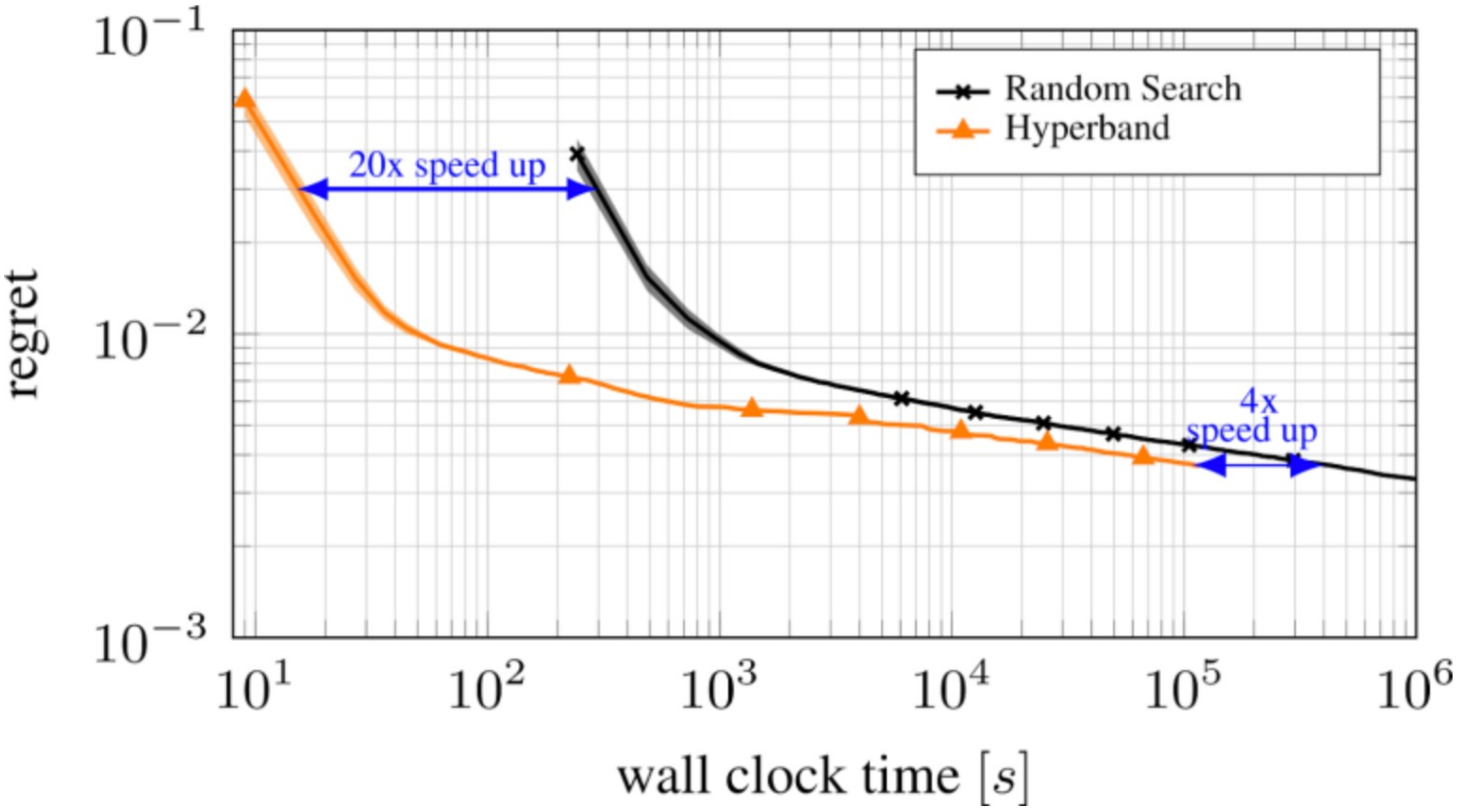
- Large N: small B/N, not enough training time
- Small N: large B/N, not enough configurations are evaluated

HyperBand

-- addresses the "n vs. B/n" problem by calling SHA multiple times with different n

	N=81		N=27		N=9		N=6		N=5	
rung	n	r	n	r	n	r	n	r	n	r
0	81	1	27	3	9	9	6	27	5	81
1	27	3	9	9	3	27	2	81		
2	9	9	3	27	1	81				
3	3	27	1	81						
4	1	81								

HyperBand



From : automl.org

Generalizations

- There are many HPO methods, but they can be categorized along various aspects
  - Parallel vs Sequential
  - Search Algorithm vs Scheduler
  - Blackbox, Graybox, multi-fidelity

Generalization: Parallel vs Sequential

- Parallel vs Sequential:
  - Parallel: Evolutionary strategies, Population-based training
  - Sequential: Bayesian Optimization
  - What's best may depend on your compute setup & requirements

- All methods are iterative
  - All methods are about building on past experience in a HPO run
  - New research area: Meta-learning or transfer learning for HPO
    - Building on past experience from HPO runs on other problems

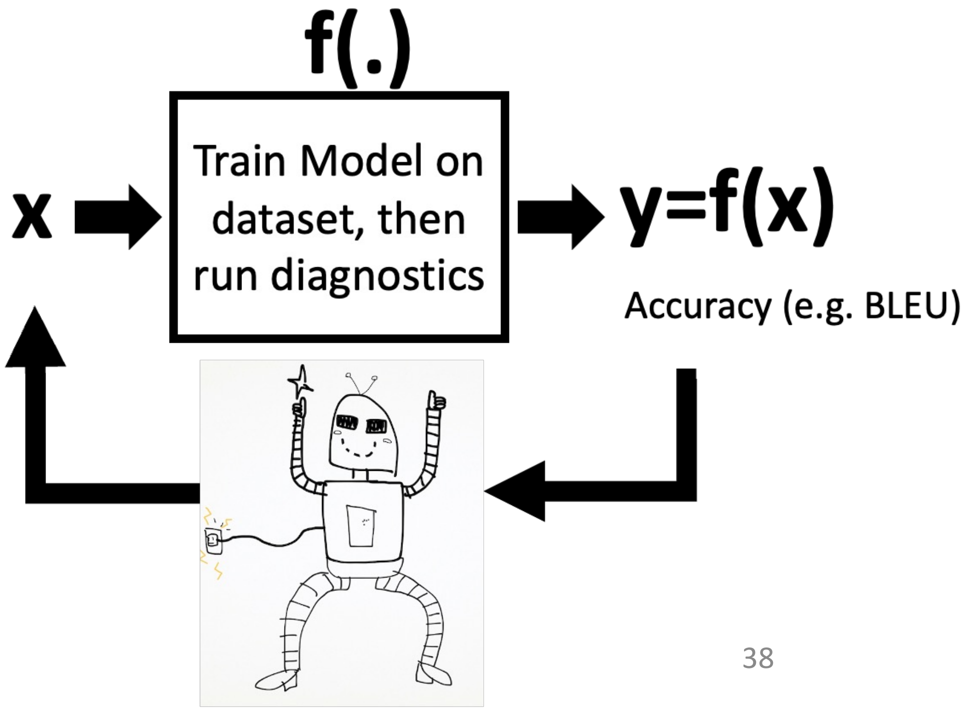
Generalization: Search Algorithm vs Scheduler

- Search algorithm: what to sample next (e.g. Bayes Opt vs CMA-ES)
- Scheduler: when to train a model, when to stop training (Hyperband)
- So these can be mixed and match!
  - HyberBand = Early stopping scheduler + Random Search
  - BOHB = Early stopping scheduler + Bayes Optimization

Generalization: Blackbox, Graybox, Multi-fidelity

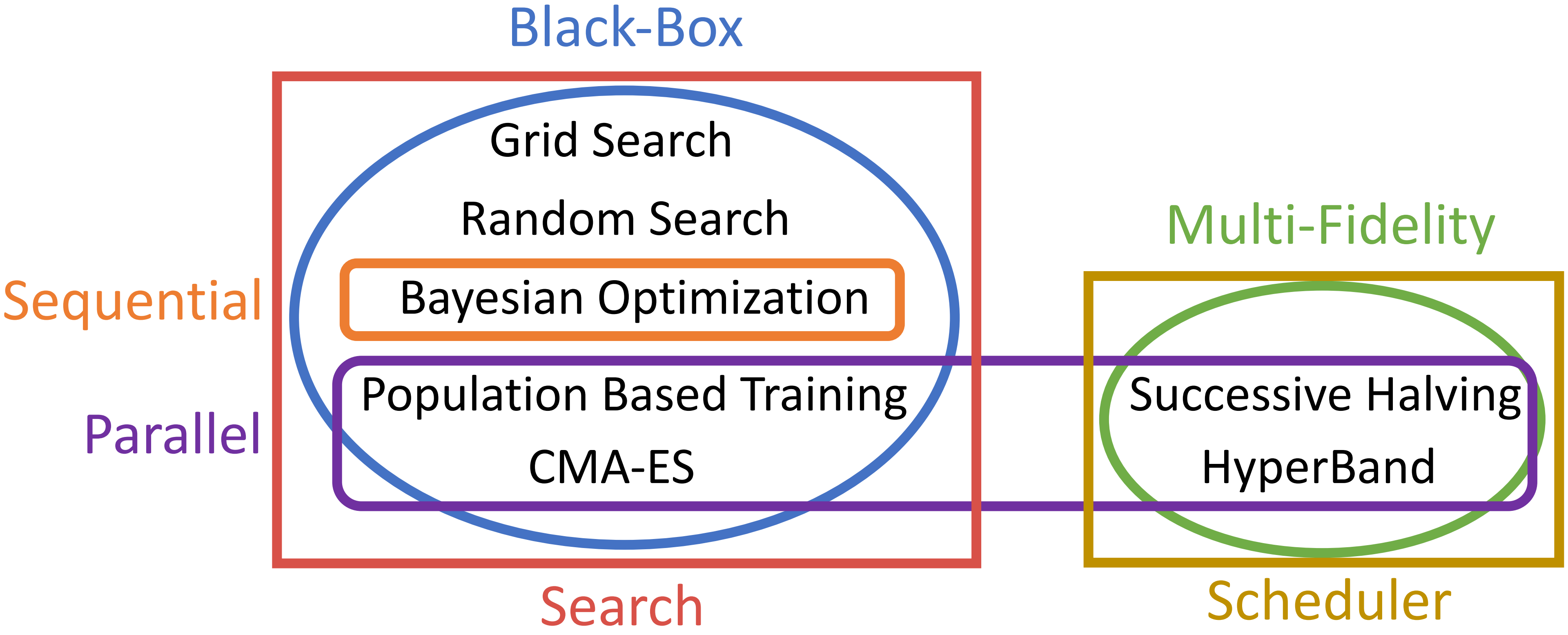
- Blackbox methods don't look inside the model training process
- Graybox methods like Hyperband can improve HPO runtime
- Generally, multi-fidelity methods exploit approximations
  - Limit training time (analogous to Hyperband)
  - Training blackbox on smaller subset of data
  - Noisy measurements

  --> assume precise accuracy isn't needed



Section Summary

- Problem Formulation of HPO
- Representative methods:



Roadmap

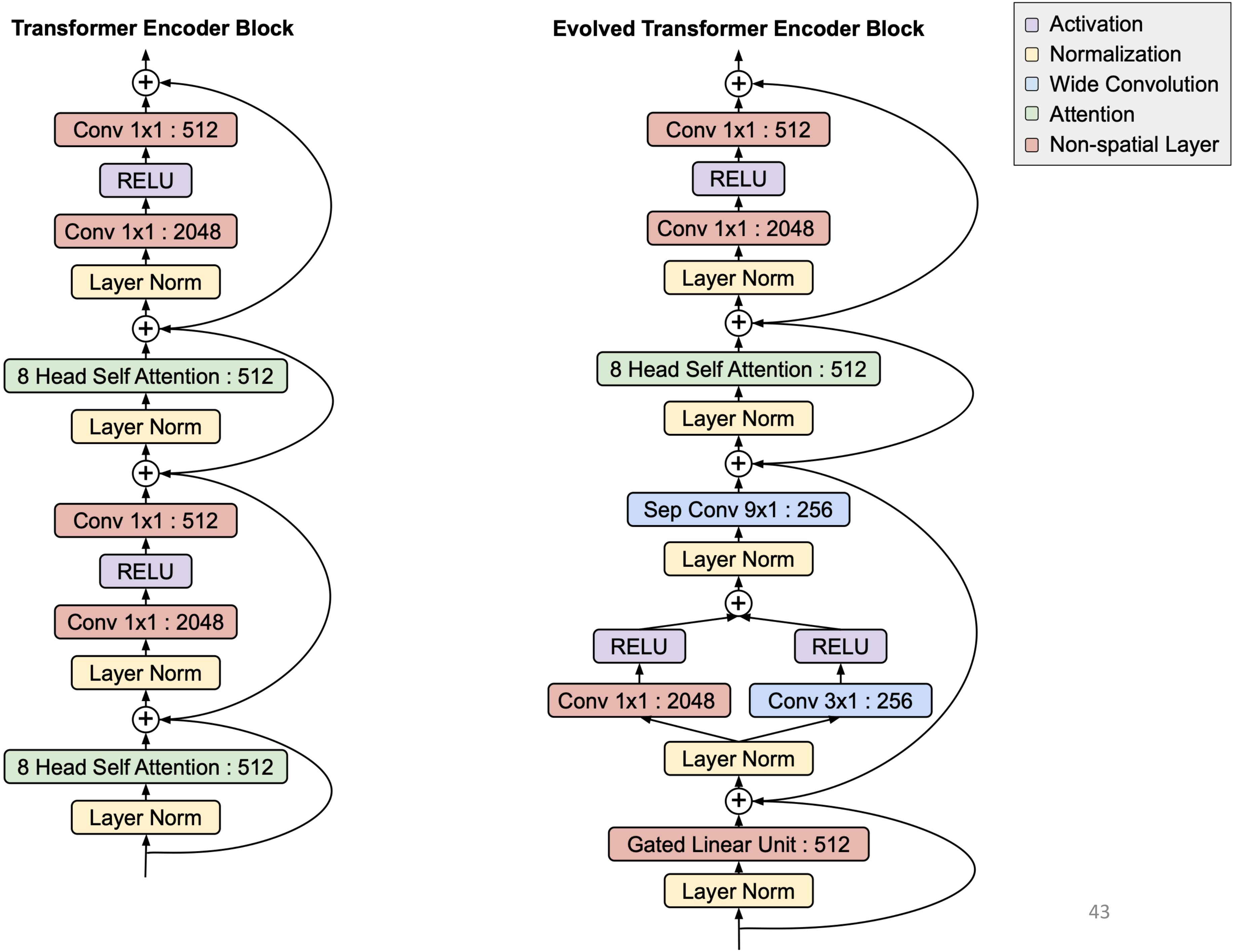
1. Motivation for AutoML
2. Hyperparameter Optimization (HPO)
3. Neural Architecture Search (NAS)
   - NAS vs HPO
   - Designing the NAS Search Space
   - NAS Search Strategy + Performance Estimation
     - Methods similar to HPO
     - One-shot NAS methods
4. Extension to Multiple Objectives
5. Evaluation
6. Toolkits
7. Survey of HPO/NAS in NLP

Hyperparameter Optimization (HPO) vs Neural Architecture Search (NAS)

	Hyperparameter Optimization (HPO)	Neural Architecture Search (NAS)
Machine learning model	Neural Network, Random Forests, Support Vector Machines, etc.	Neural Network
Hyperparameters	Architectural: - #layer for neural net - tree depth for random forests - kernel for support vector machine Training Pipeline: - Preprocessing, Data selection Optimization: - ADAM vs AdaGrad, Learning rate	Architectural - #layer, #dim - "Novel" non-standard architectures
Example of a discovered model	4-layer encoder, 3-layer decoder, each with FFN of 512 dimensions	4-layer encoder: layer 1 has 512 dim, layer 2 has 1024 dim, layer 3 uses 12 heads rather than 8, etc.
Summary	General technique, course-grained but diverse hyperparameters	Focused technique on neural nets, fine-grained architectural space

Example of model discovered by NAS

from:
D. So, C. Liang, Q. Le. The Evolved Transformer (2019)



Three components to an NAS method

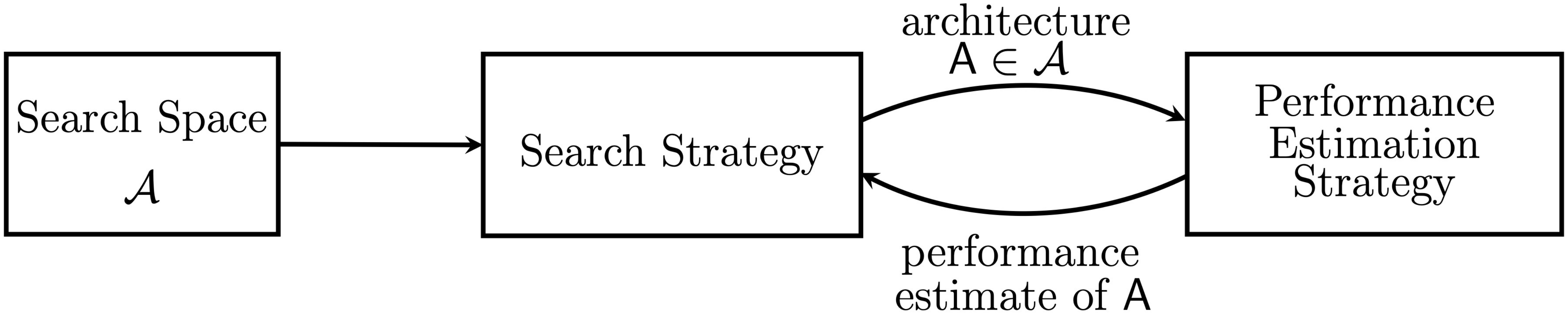


Figure 1: Abstract illustration of Neural Architecture Search methods. A search strategy selects an architecture A from a predefined search space $\mathcal{A}$. The architecture is passed to a performance estimation strategy, which returns the estimated performance of A to the search strategy.

From: Elsken, Metzen, Hutter. Neural Architecture Search, A Survey, JMLR 2019

Three components to an NAS method

We'll discuss:

Sequential vs. Cell-based

Methods similar to HPO
+ Gradient-based

Full train from scratch vs
Weight share, One-shot, etc.

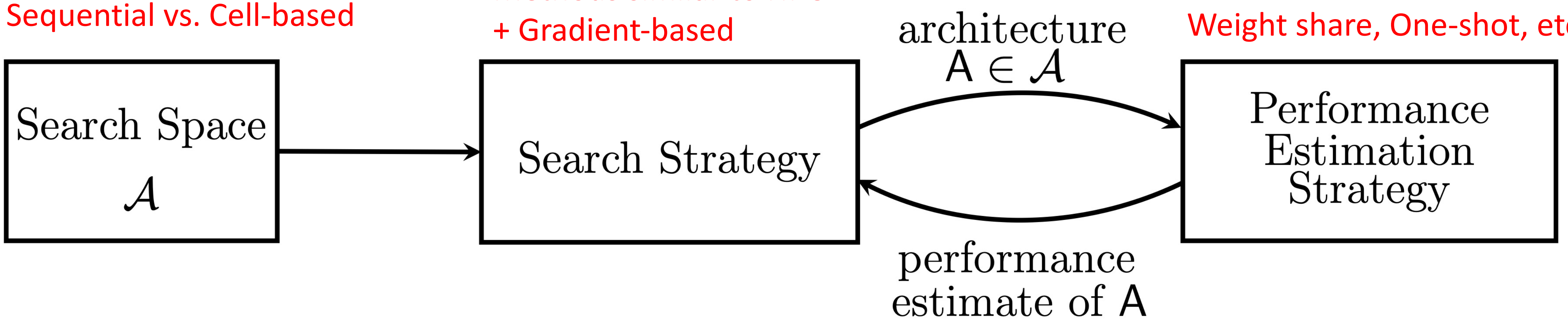
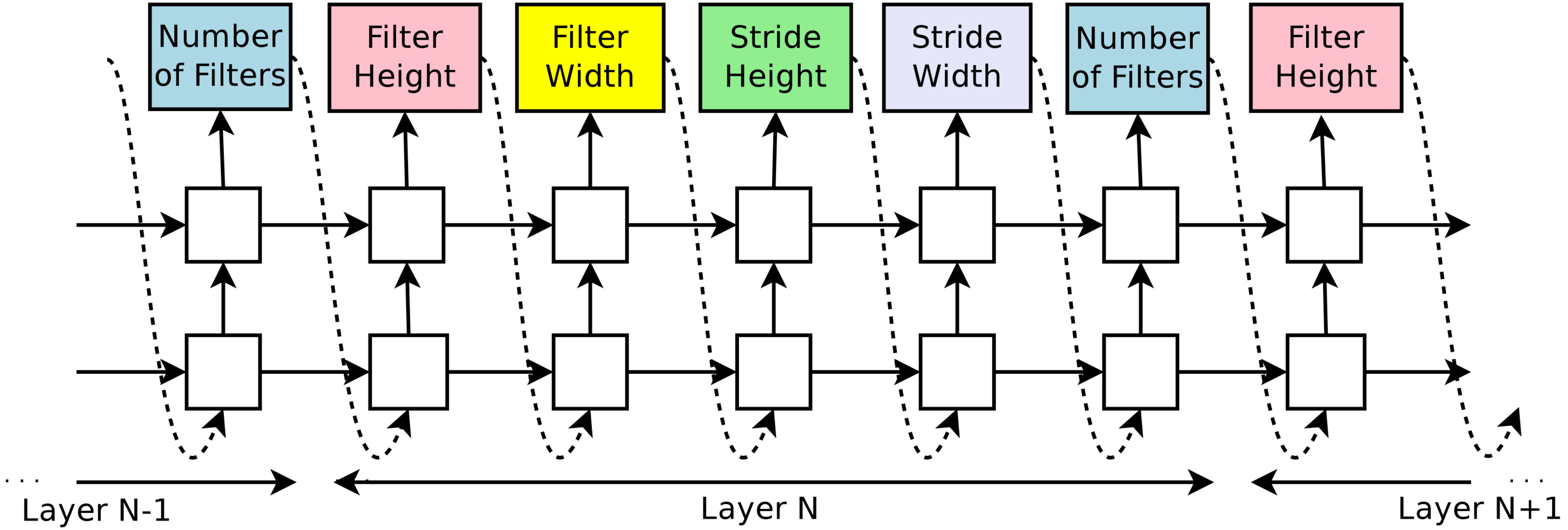


Figure 1: Abstract illustration of Neural Architecture Search methods. A search strategy selects an architecture A from a predefined search space $\mathcal{A}$. The architecture is passed to a performance estimation strategy, which returns the estimated performance of A to the search strategy.

From: Elsken, Metzen, Hutter. Neural Architecture Search, A Survey, JMLR 2019

Search Space defined by sequential decisions

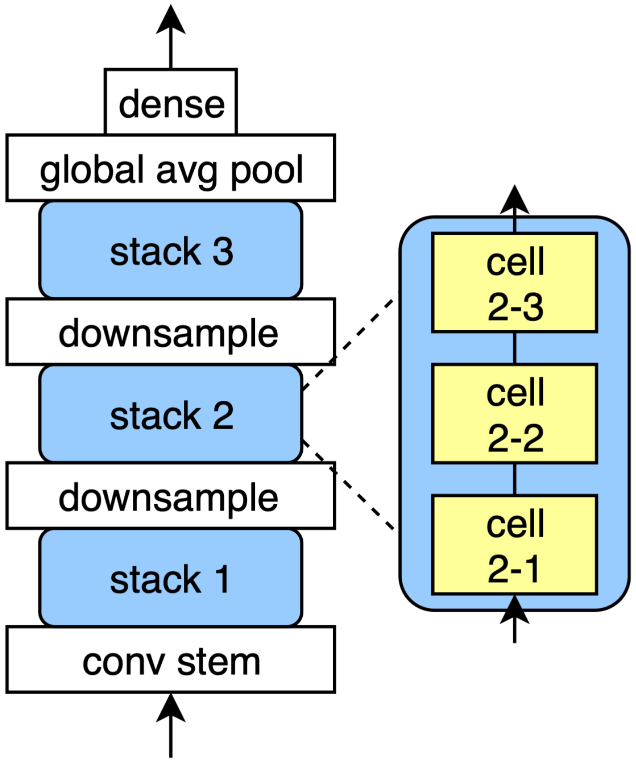
- Suppose we want feed-forward network with convolution layers
- Use a "controller" to predict hyperparameters in sequence

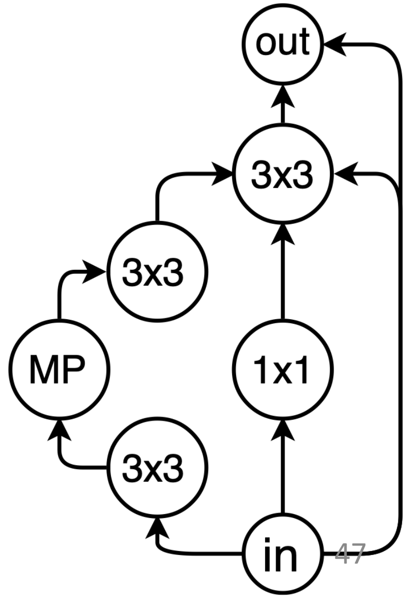


From: Zoph & Le. Neural Architecture Search with Reinforcement Learning, ICLR2017

Cell-based Search Space

- Focus search on smaller cells, which are stacked
- Example:
  - V nodes per cell (e.g. Max |V| = 7)
  - Each node takes one of L operations: 3x3 convolution, 1x1 convolution, 3x3 max-pool
  - Edges connect nodes, form Directed Acyclic Graph (DAG) starting from "in" to "out" node. (e.g. 21 edges max)
  - Encoding: 7x7 upper-triangular matrix + list of 5 operations. 2^21 x 3^5 = 510M unique cells





From: Ying et. Al. NAS-Bench-101: Toward Reproducible NAS

Cell-based Search Space (exercise)

```
# Adjacency matrix of the module
matrix=[[0, 1, 1, 1, 0, 1, 0],    # input layer
        [0, 0, 0, 0, 0, 0, 1],    # 1x1 conv
        [0, 0, 0, 0, 0, 0, 1],    # 3x3 conv
        [0, 0, 0, 0, 1, 0, 0],    # 5x5 conv (replaced by two 3x3's)
        [0, 0, 0, 0, 0, 0, 1],    # 5x5 conv (replaced by two 3x3's)
        [0, 0, 0, 0, 0, 0, 1],    # 3x3 max-pool
        [0, 0, 0, 0, 0, 0, 0]],   # output layer
# Operations at the vertices of the module, matches order of matrix
ops=[INPUT, CONV1X1, CONV3X3, CONV3X3, CONV3X3, MAXPOOL3X3, OUTPUT])
```

From: Ying et. Al. NAS-Bench-101: Toward Reproducible NAS

Three components to an NAS method

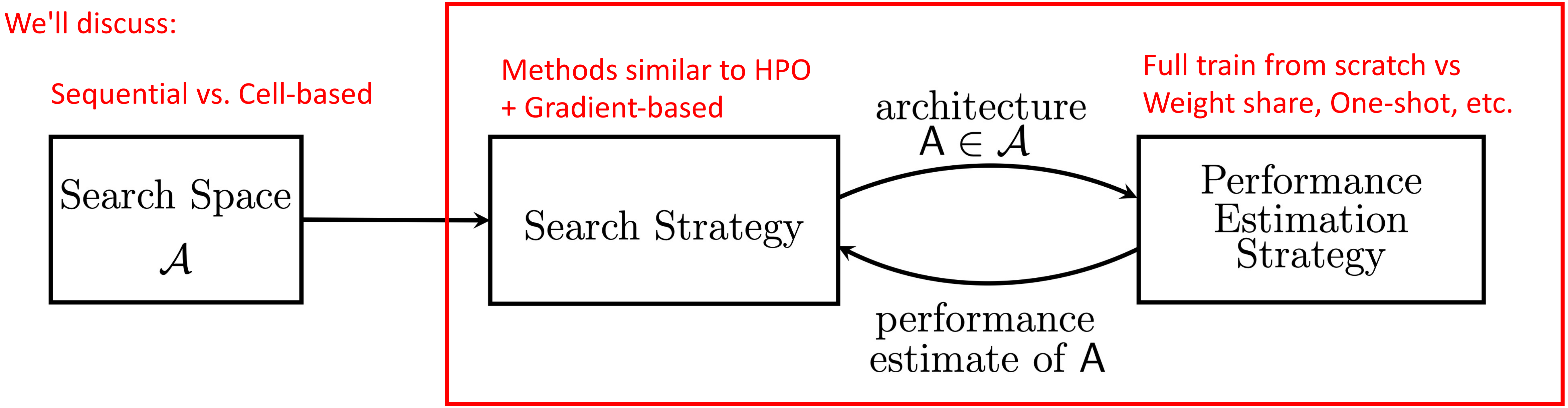
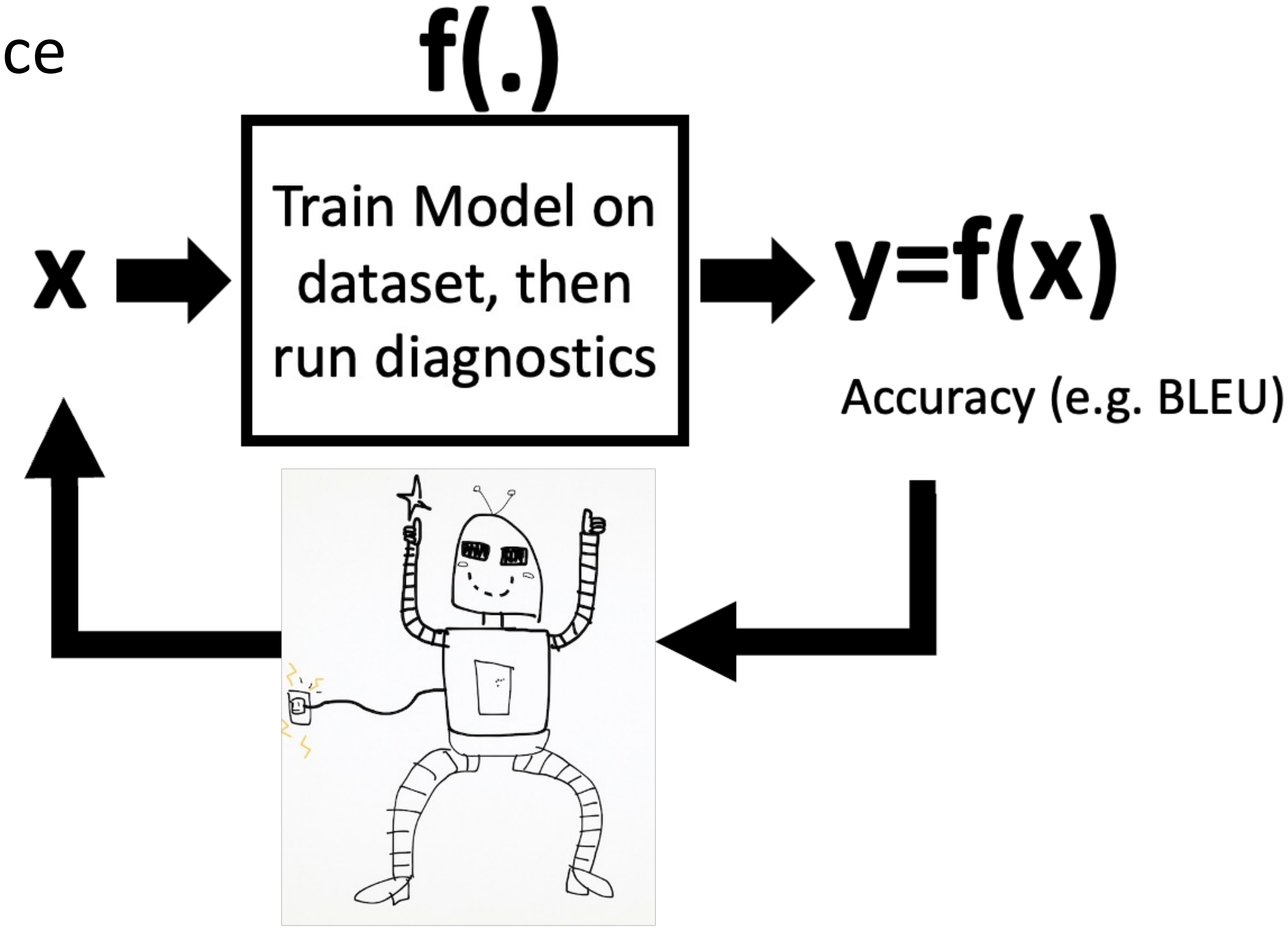


Figure 1: Abstract illustration of Neural Architecture Search methods. A search strategy selects an architecture A from a predefined search space $\mathcal{A}$. The architecture is passed to a performance estimation strategy, which returns the estimated performance of A to the search strategy.

From: Elsken, Metzen, Hutter. Neural Architecture Search, A Survey, JMLR 2019

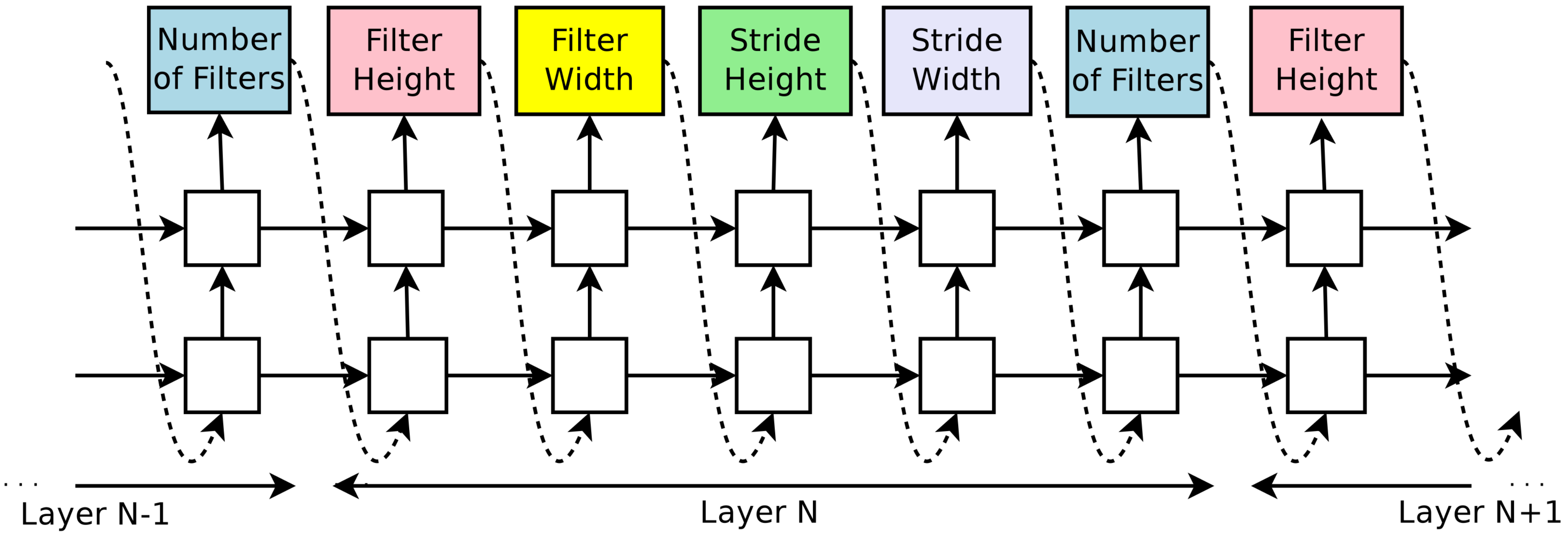
Search Strategy Options: HPO methods

- Sample x from NAS search space
- The rest we can use any HPO method:
  - Random search
  - Bayes Optimization
  - Evolutionary Strategy
  - Population-based Training
  - Hyperband
- Again we treat problem as a black box optimization



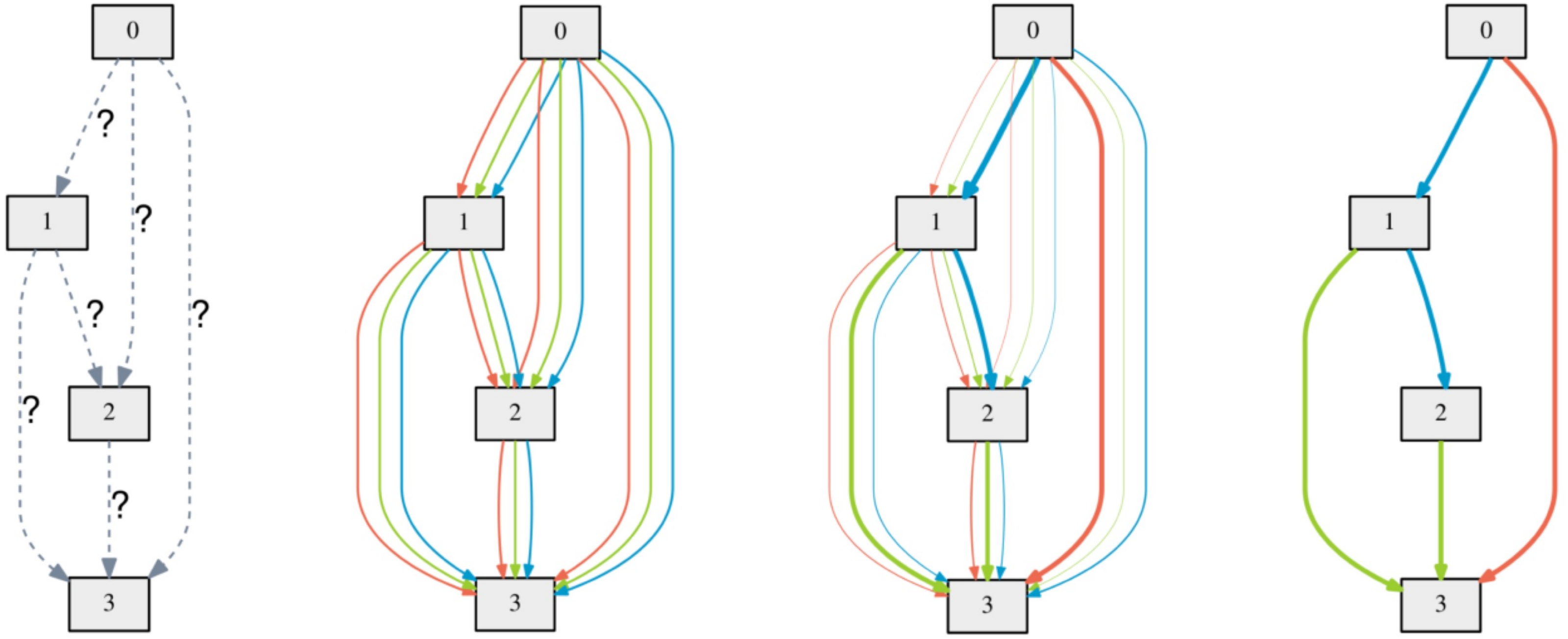
Search Strategy Options: Reinforcement Learning

- View exploration/exploitation in search space as a sequence of decisions



Search Strategy Options: Gradient-based

- DARTS: Differentiable Architecture Search (Liu, Simonyan, Yang; 2019)
  - addresses scalability issue in search + performance estimation by relaxing search space to be continuous



(a) (b) (c) (d)

DARTS

- Let O be set of candidate operations (e.g. convolution, max-pool, zero)
- For each edge (i,j), we have a distribution $\bar{o}^{(i,j)}(x) = \sum_{o \in \mathcal{O}} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in \mathcal{O}} \exp(\alpha_{o'}^{(i,j)})} o(x)$

Algorithm 1: DARTS – Differentiable Architecture Search

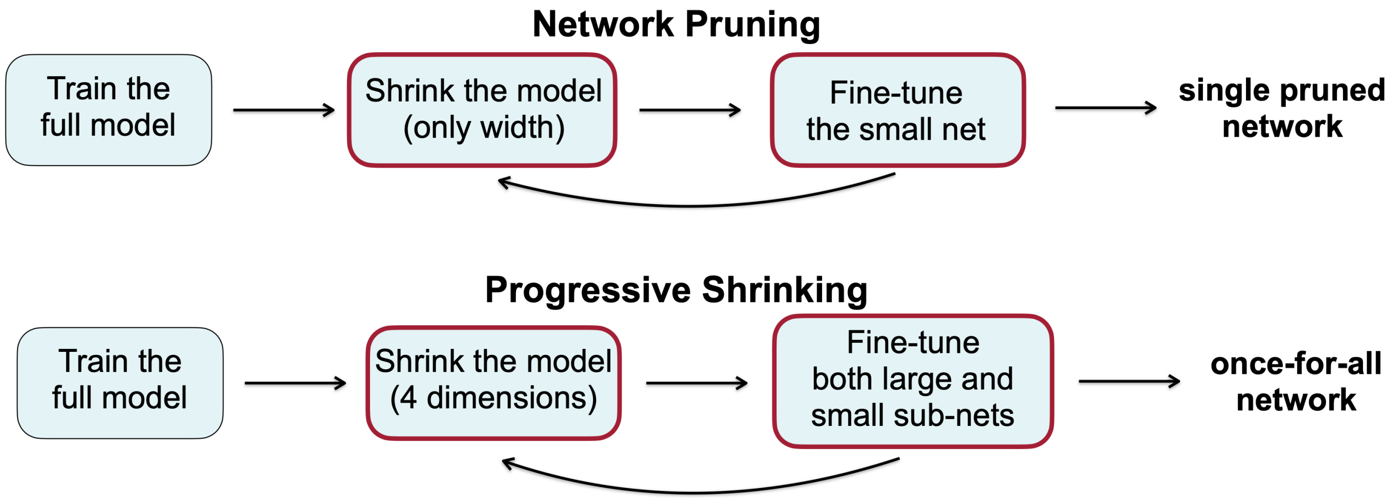
Create a mixed operation $\bar{o}^{(i,j)}$ parametrized by $\alpha^{(i,j)}$ for each edge (i,j)
while *not converged* **do**
1. Update architecture α by descending $\nabla_\alpha \mathcal{L}_{val}(w - \xi \nabla_w \mathcal{L}_{train}(w,\alpha), \alpha)$
   ($\xi = 0$ if using first-order approximation) # learn alpha on validation set
2. Update weights w by descending $\nabla_w \mathcal{L}_{train}(w, \alpha)$ # fix alpha, standard training of parameters

Derive the final architecture based on the learned α. # pick argmax edges, retrain final model

---

Another one-shot NAS method: Once-for-All

- A single "supernet" is trained once
- Subnets x are sampled from supernet, and f(x) is measured without retraining x from scratch
- Progressive shrinking technique:
  - Potentially more representative subnets in supernet



From: Cai et. al. Once-for-all: Train one network and specialize it for efficient deployment. ICLR2020

Pros & Cons of One-Shot NAS

- Pros:
  - Much faster than black-box search + performance estimation
  - Explore much larger achitectural space
- Cons:
  - Difficult to know if the assumption of weight sharing is valid
  - Empirical results are mixed and unstable (some researchers may disagree)
  - Supernet needs to fit in memory
- NAS (one-shot & in general) is a very active research area – stay tuned!

Section Summary

We discussed:

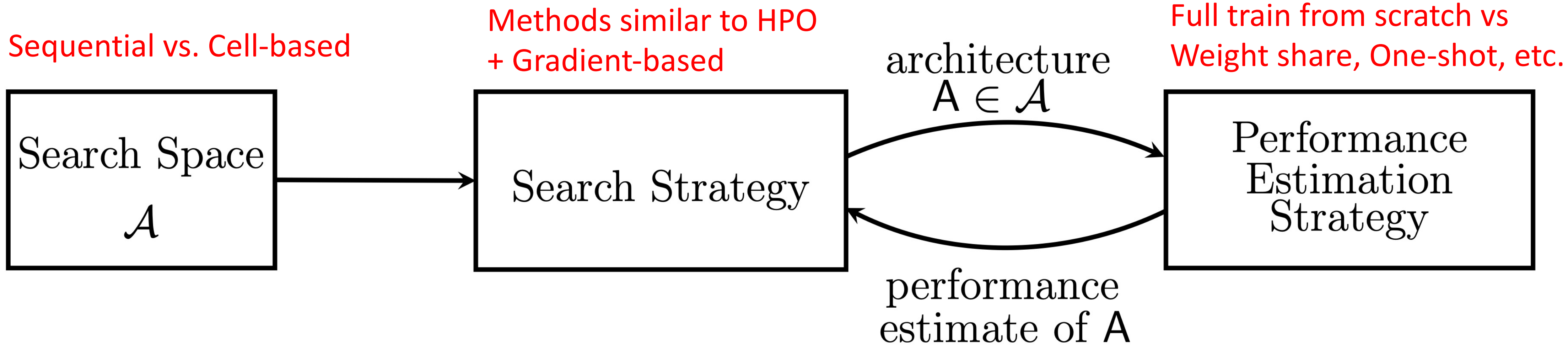


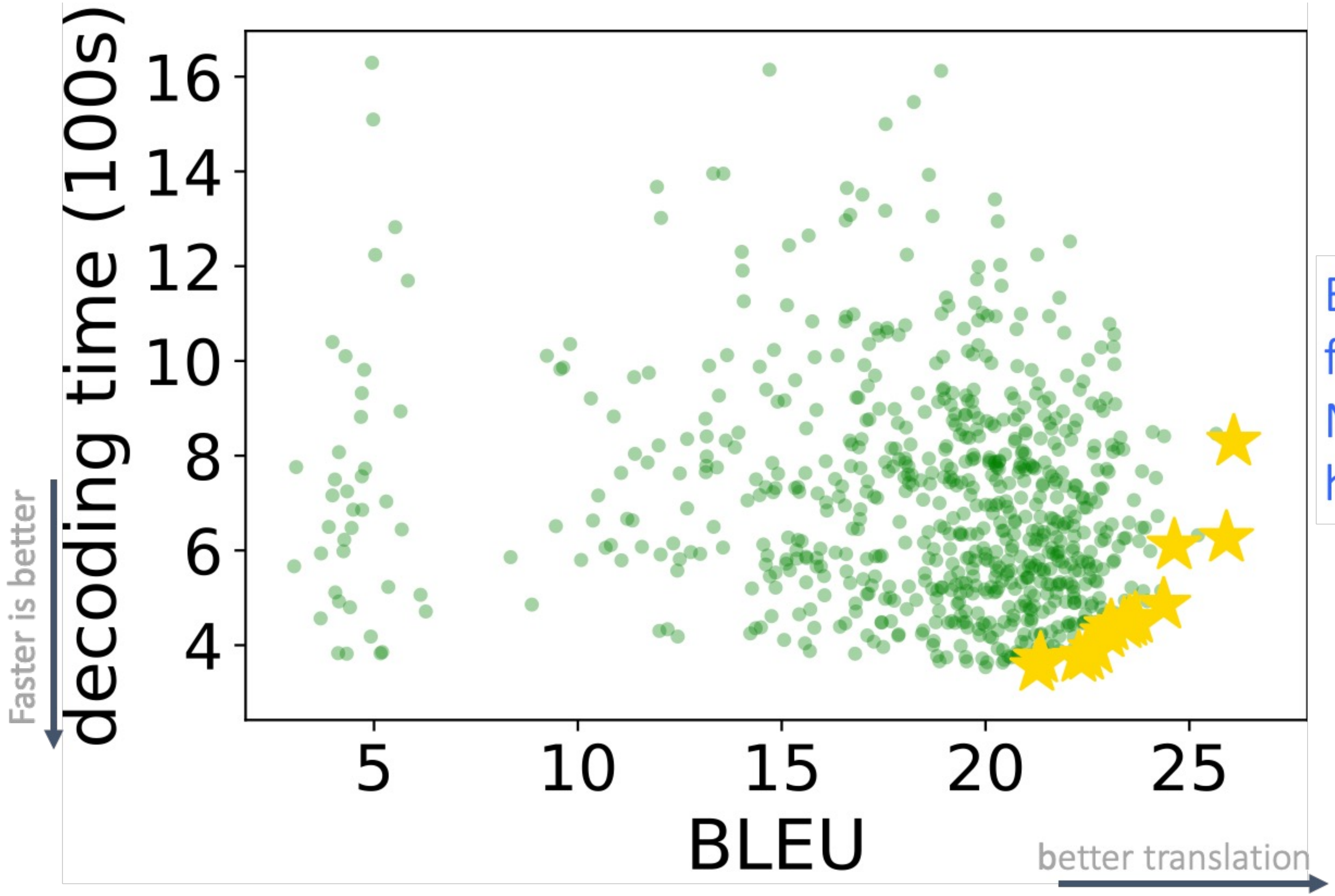
Figure 1: Abstract illustration of Neural Architecture Search methods. A search strategy selects an architecture A from a predefined search space $\mathcal{A}$. The architecture is passed to a performance estimation strategy, which returns the estimated performance of A to the search strategy.

From: Elsken, Metzen, Hutter. Neural Architecture Search, A Survey, JMLR 2019

Roadmap

1. Motivation for AutoML
2. Hyperparameter Optimization (HPO)
3. Neural Architecture Search (NAS)
4. Extension to Multiple Objectives
    - Why it's important
    - Pareto optimality
    - Example Multi-objective HPO/NAS methods
5. Evaluation
6. Toolkits
7. Survey of HPO/NAS in NLP

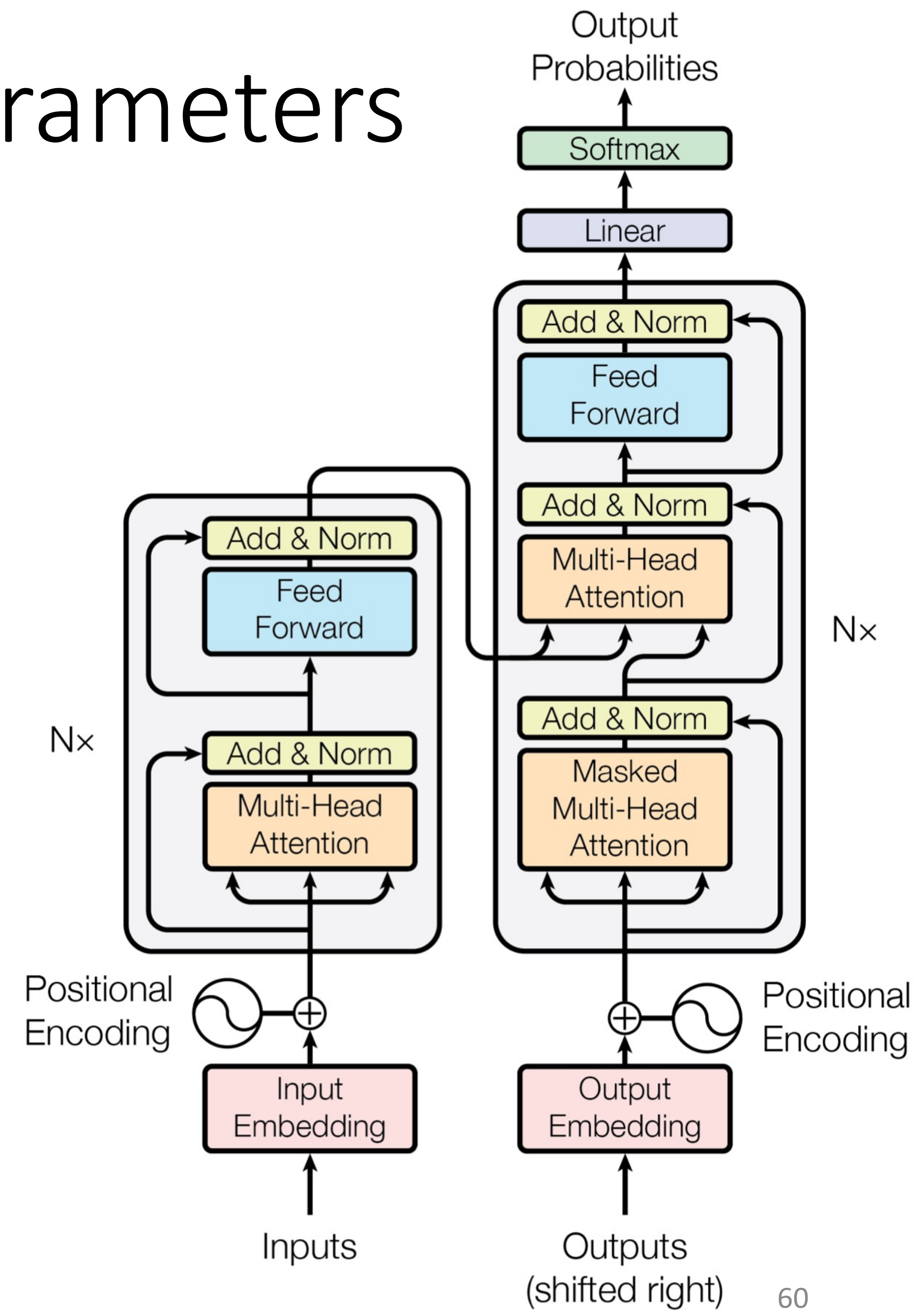
When deploying models, we care about multiple objectives. But it's complex.



BLEU vs Time Scatterplot for **700+** Swahili-English NMT models: unclear how to get best tradeoff

Quiz: How do these hyperparameters impact accuracy and speed?

- Architectural hyperparameters:
  - # of layers
  - # of hidden units in feed-forward layer
  - # attention heads
  - Word embedding dimension
- Training pipeline hyperparameters:
  - # of subword units
- Optimizer hyperparameters:
  - Initial learning rate for ADAM, etc.



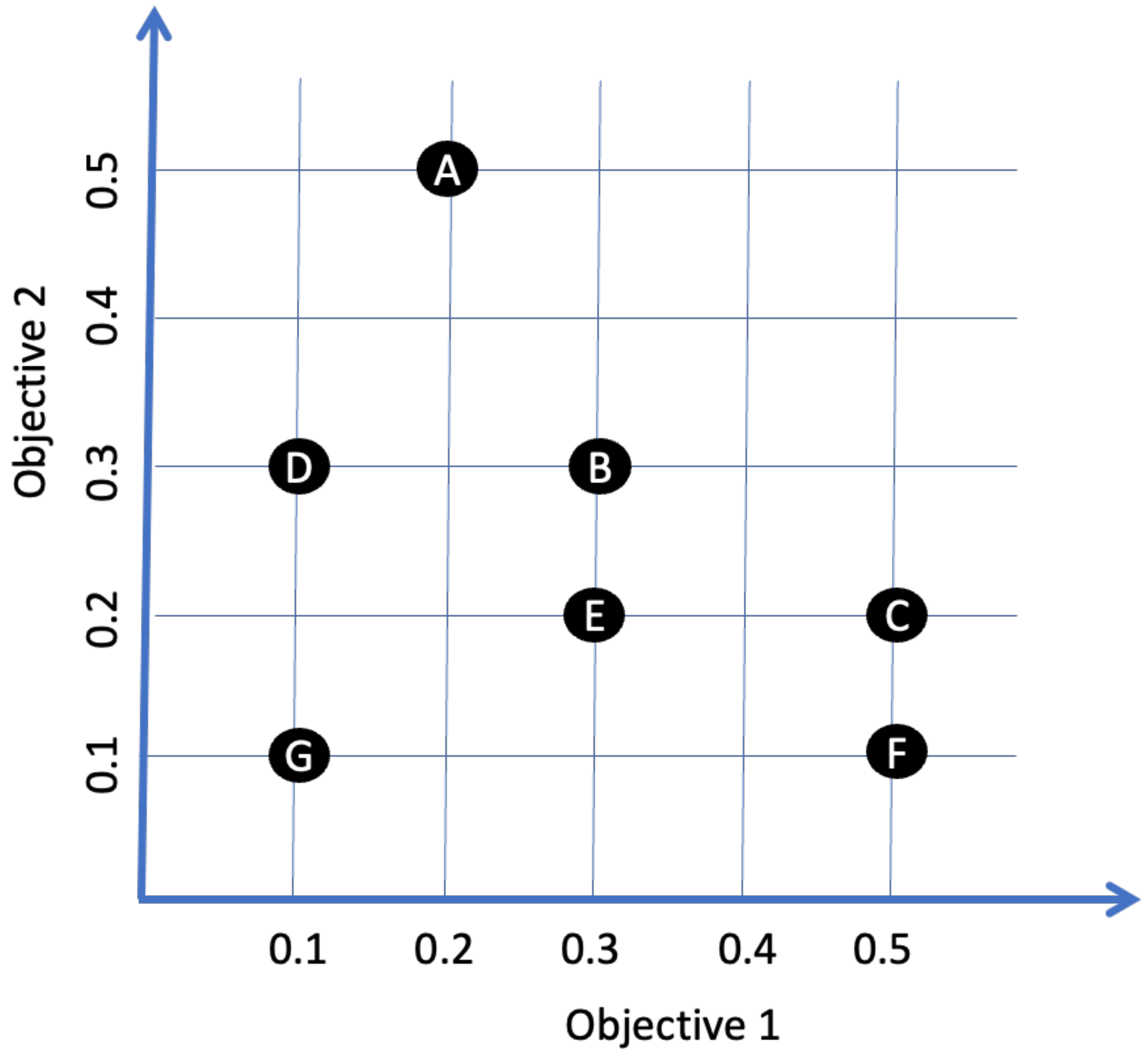
Objectives one may care about

- Accuracy
  - BLEU, COMET, Human evaluation
- Inference speed
  - On GPU, on CPU, in batch or not
  - Throughput vs Latency
- Deployment resource consumption
  - Memory, disk, energy
- Training resource consumption

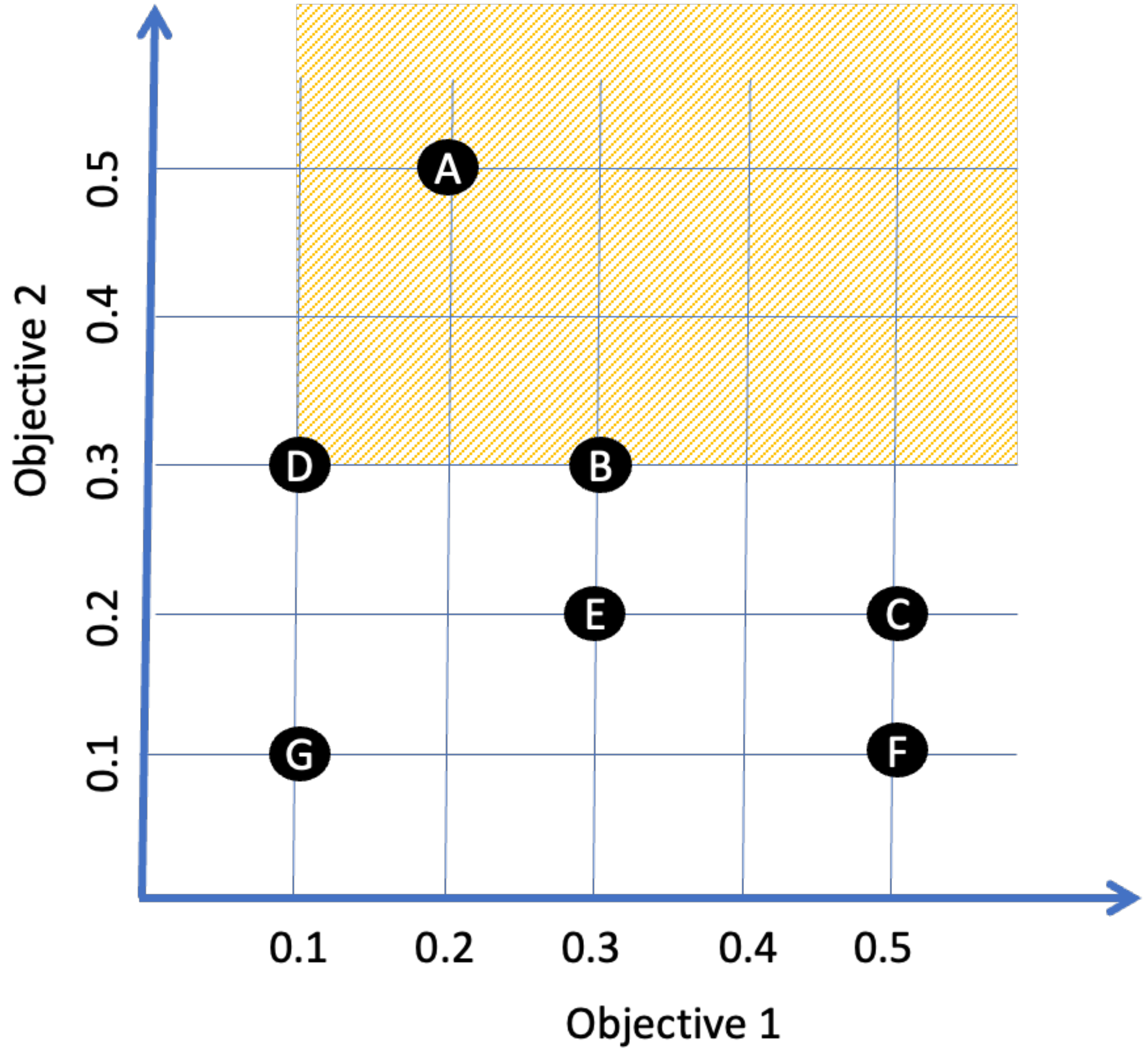
Motivation for Multiple Objectives

- IMHO, this is the strongest motivation for AutoML in deployment
  - While an engineer/researcher may develop good heuristics for tuning hyperparameters for accuracy alone, it is very difficult to reason through multiple interacting objectives
- Ideal future, where AutoML is part of everyone's toolkit
  - import AutoMLtool
  - A=search_space()
  - O=[accuracy(), speed(), memory()]
  - models = multi_objective_NAS(A, O)

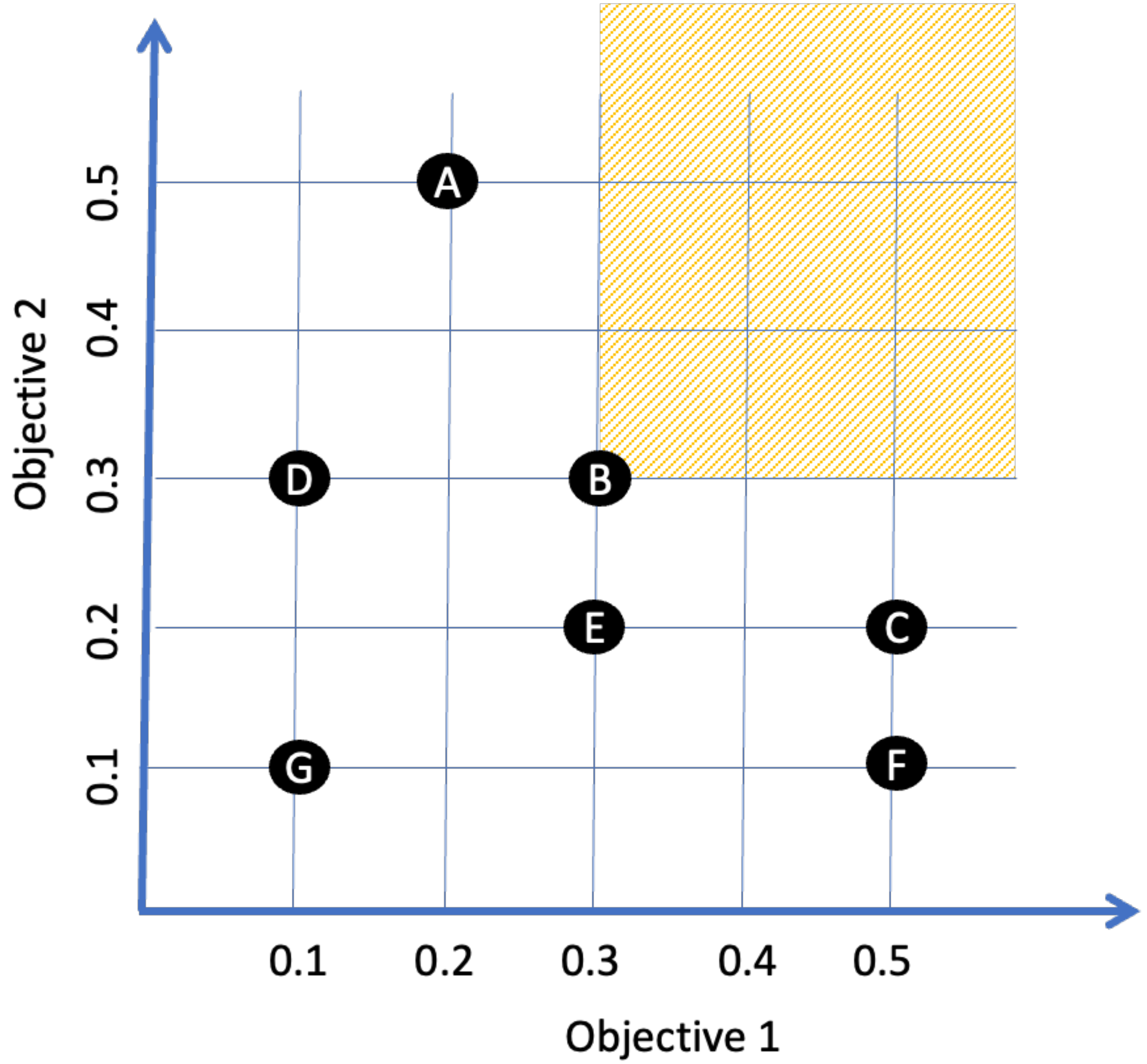
How to define optimality for multi-objective?



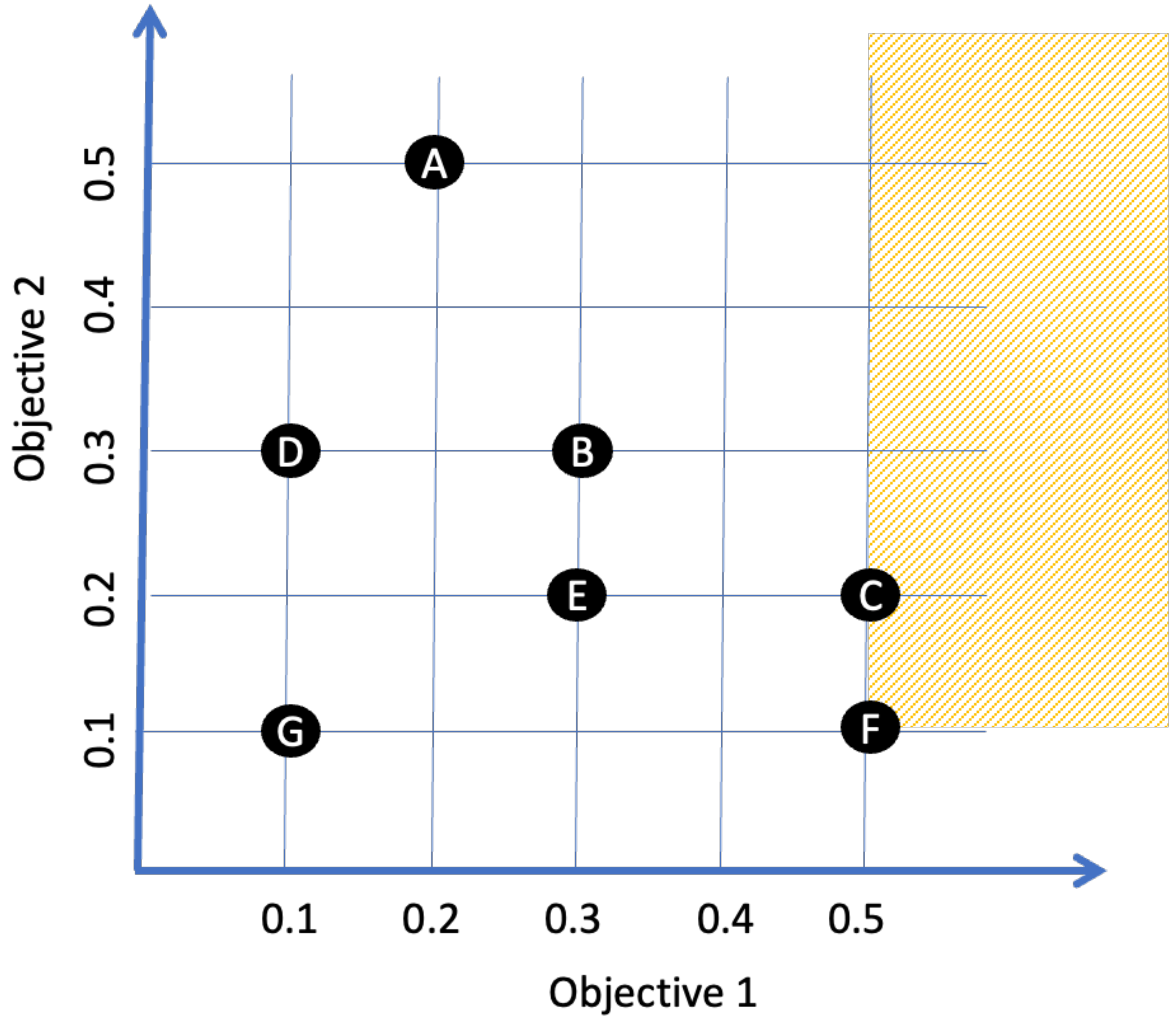
Definition: A point p is **weakly pareto-optimal** iff there does not exist another point q such that $F_k(q) > F_k(p)$ for all k



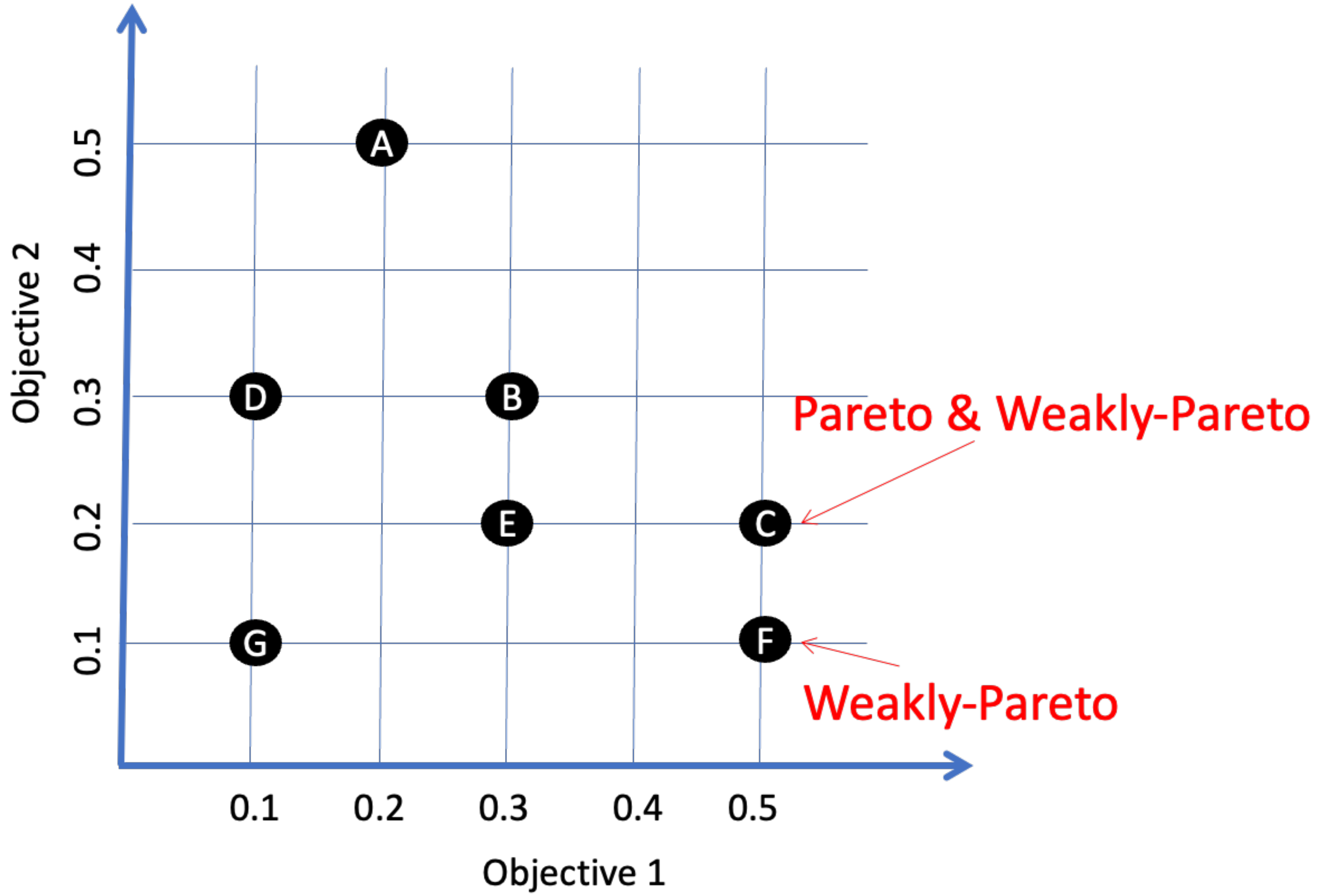
Definition: A point p is **weakly pareto-optimal** iff there does not exist another point q such that $F_k(q) > F_k(p)$ for all k



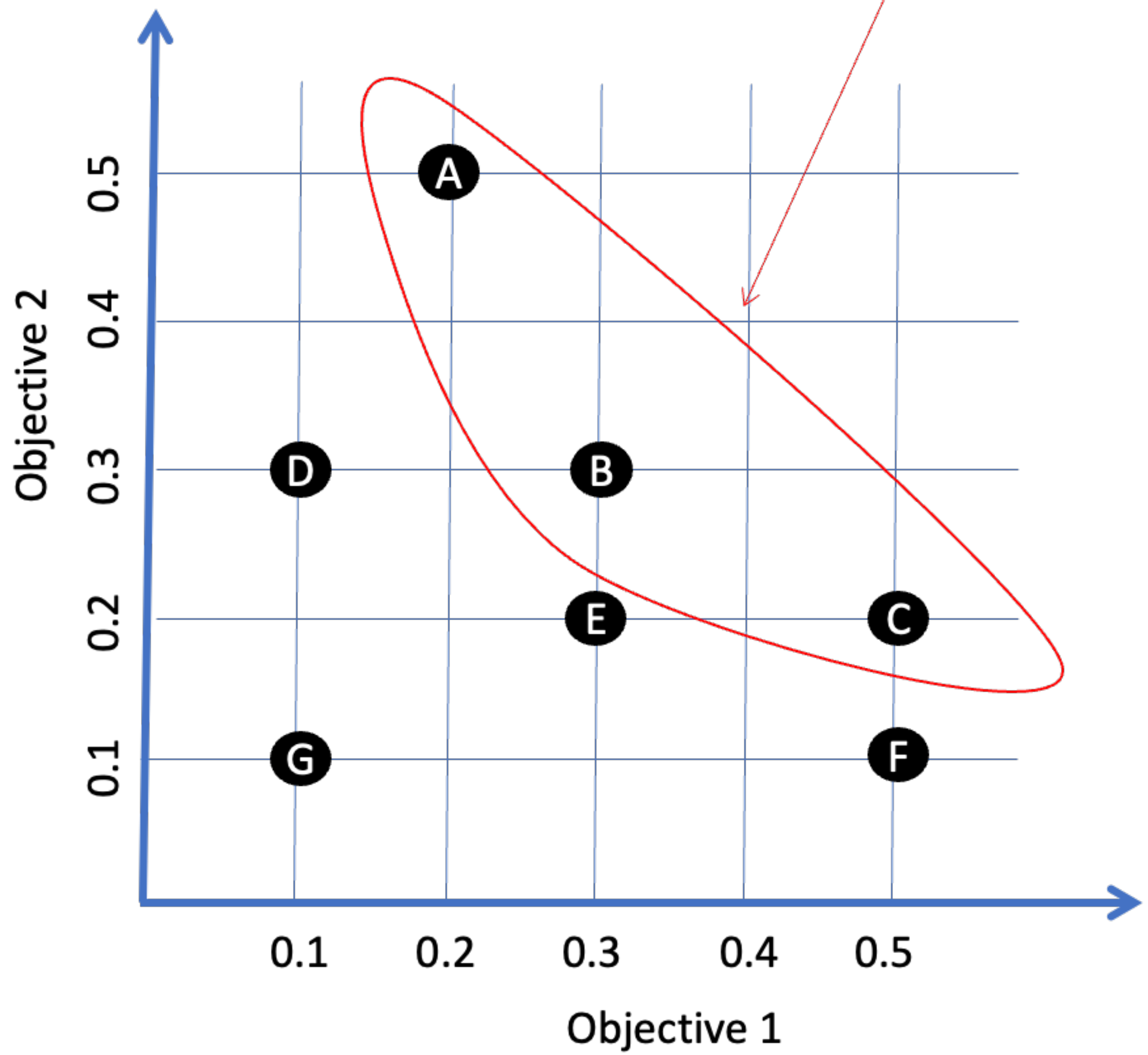
Definition: A point p is **weakly pareto-optimal** iff there does not exist another point q such that $F_k(q) > F_k(p)$ for all k



Definition: A point p is **pareto-optimal** iff there does not exist a q such that $F_k(q) >= F_k(p)$ for all k and $F_k(q) > F_k(p)$ for at least one k



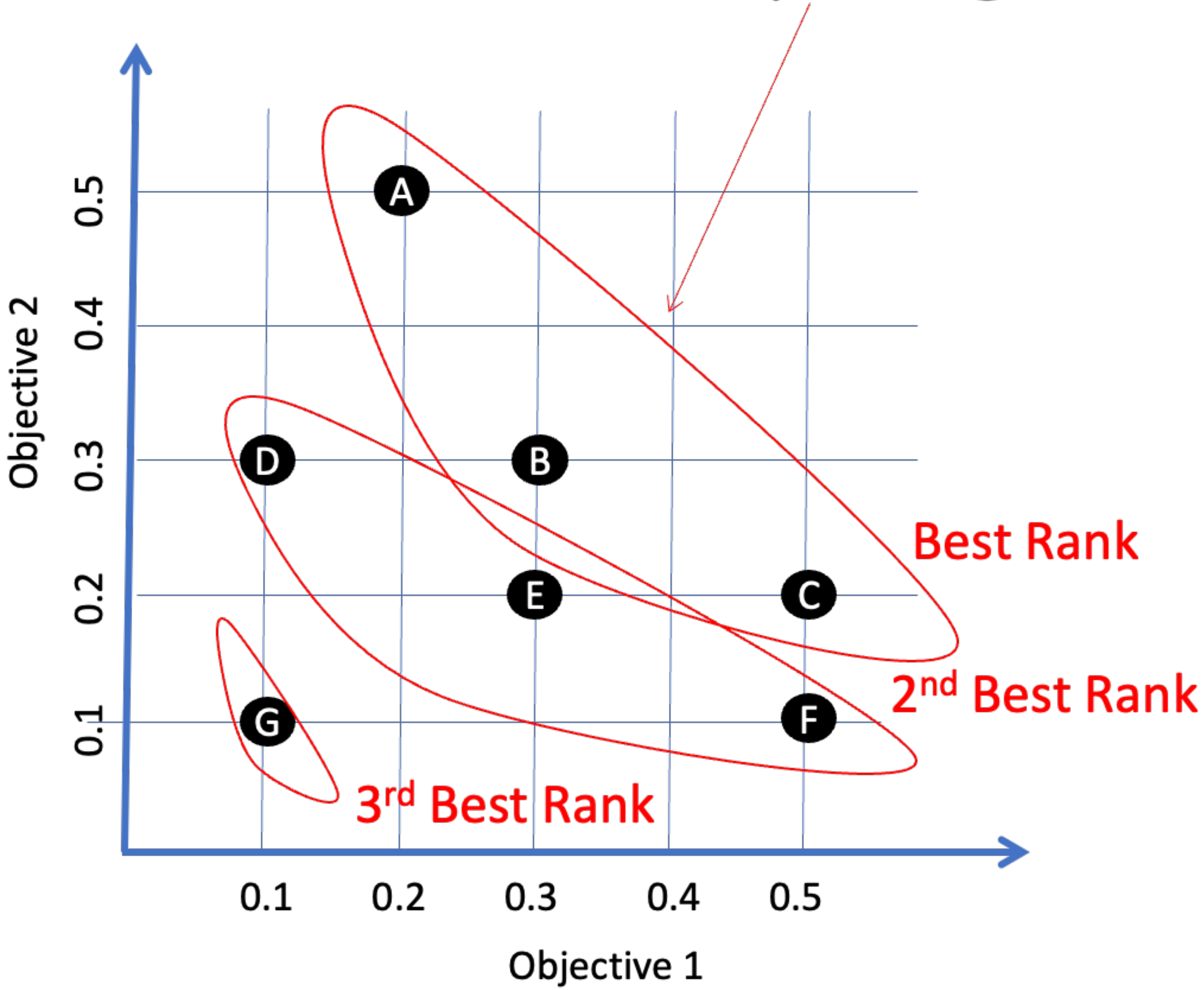
Given a set of points, the subset of pareto-optimal points form the Pareto Frontier



Computing Pareto

- Pseudo-code:
  - Set N=[]
  - For p in ListOfSamples:
    - Set d = 0
    - For q in ListOfSamples:
      - For k in ListOfObjective, see if $F_k(q) > F_k(p)$. If yes, d+=1
    - If d=0, add p to N
  - Return N
- Basic implementation is O(KN^2)
  - K = #objectives, N= #samples
  - O(K NlogN) is possible in two-objective case
- Generally, #pareto increases with K

Points can be ranked by successively peeling off the Pareto Frontier and recomputing



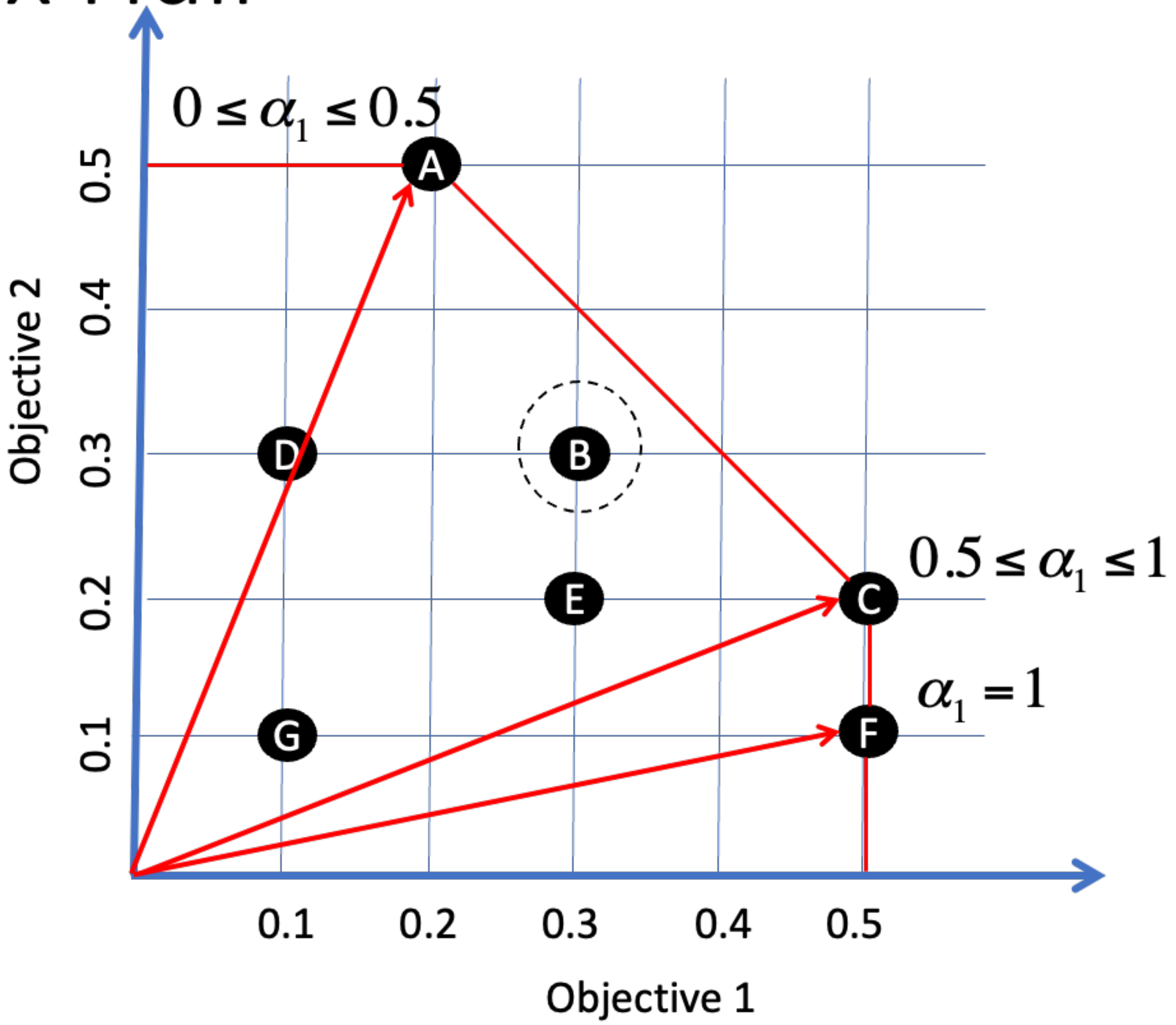
Aside: Alternative to Pareto Optimality

- Combine multiple objectives into one

$$\max_x [f_1(x), f_2(x), \dots, f_M(x)]$$

$$\text{Scalarization: } \max_x [\sum_m \alpha_m f_m(x)] \qquad \alpha_m \geq 0, \sum_{m=1}^{M} \alpha_m = 1$$

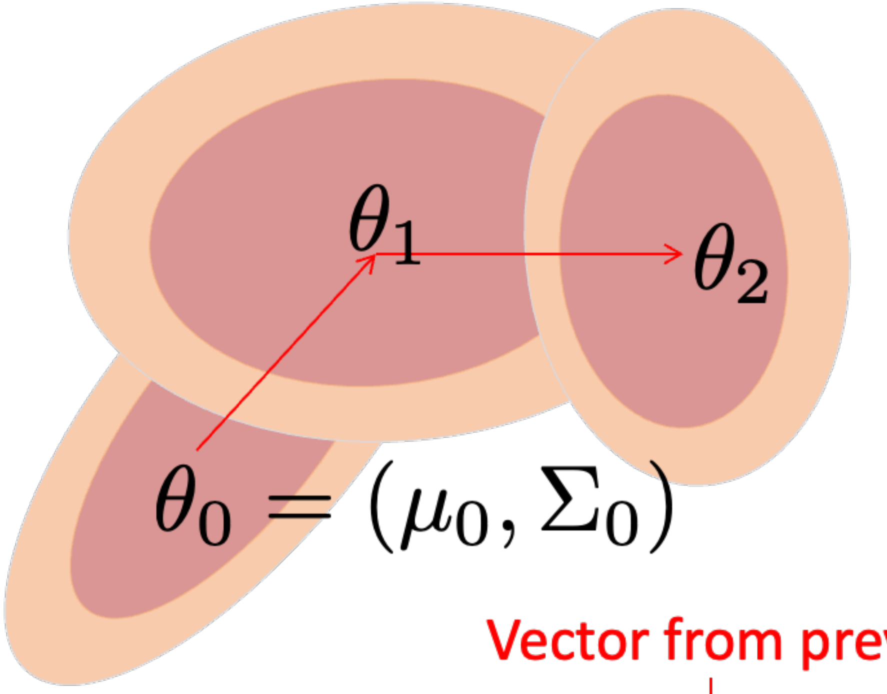
Scalarization misses Pareto points that are not on Convex Hull



For more info, see tutorial: https://www.cs.jhu.edu/~kevinduh/notes/duh11multiobj-handout.pdf

Incorporating Pareto into CMA-ES

$$\hat{\theta} = \arg\max_{\theta} \underbrace{\int f(x)\mathcal{N}(x|\theta)dx}_{\triangleq \mathbb{E}[f(x)|\theta]}$$



Vector from previous mean to x_k

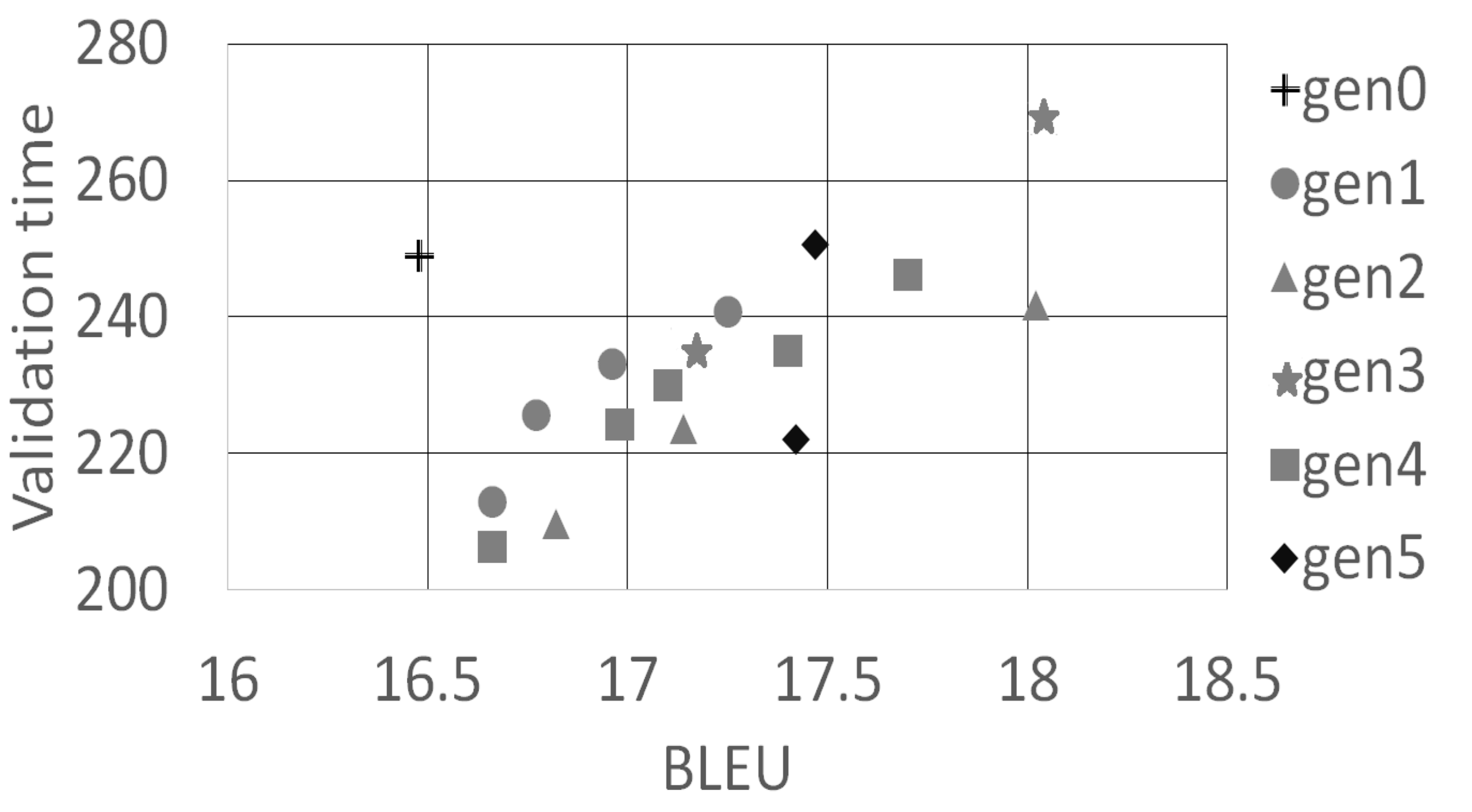
Mean update:

$$\hat{\mu}_n = \hat{\mu}_{n-1} + \epsilon_\mu \sum_{k=1}^{K} w(y_k)(x_k - \hat{\mu}_{n-1})$$

Mean of previous generation

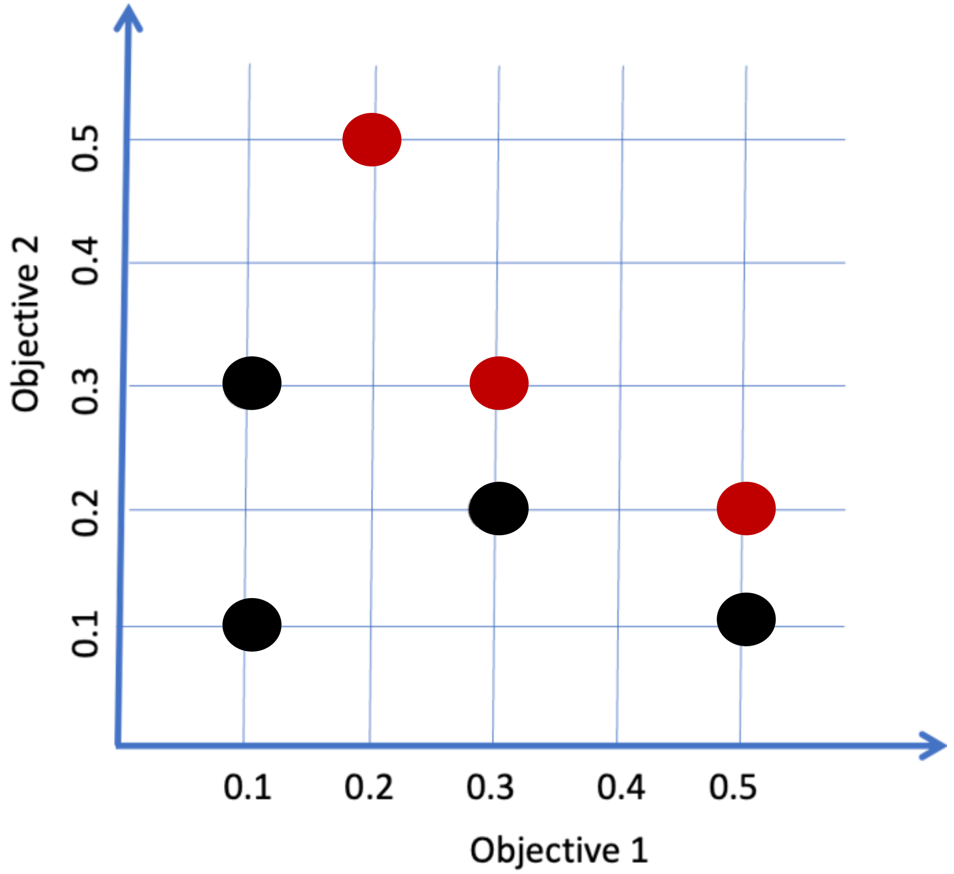
Weight for individual (x_k, y_k=f(x_k)), Better Pareto rank → higher weight

Example MT results from CMA-ES



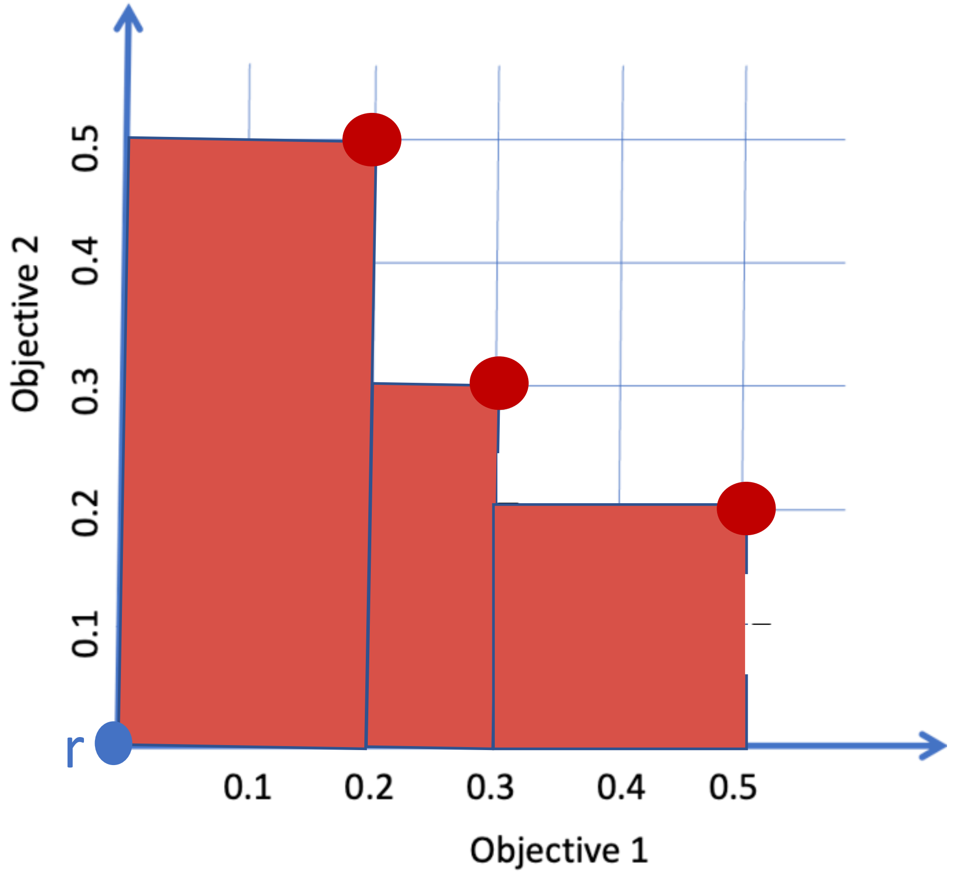
From: Qin, Shinozaki, Duh. Evolutionary strategy based automatic tuning of NMT systems, IWSLT 2017

Multi-Objective Bayesian Optimization with Expected Hypervolume Improvement



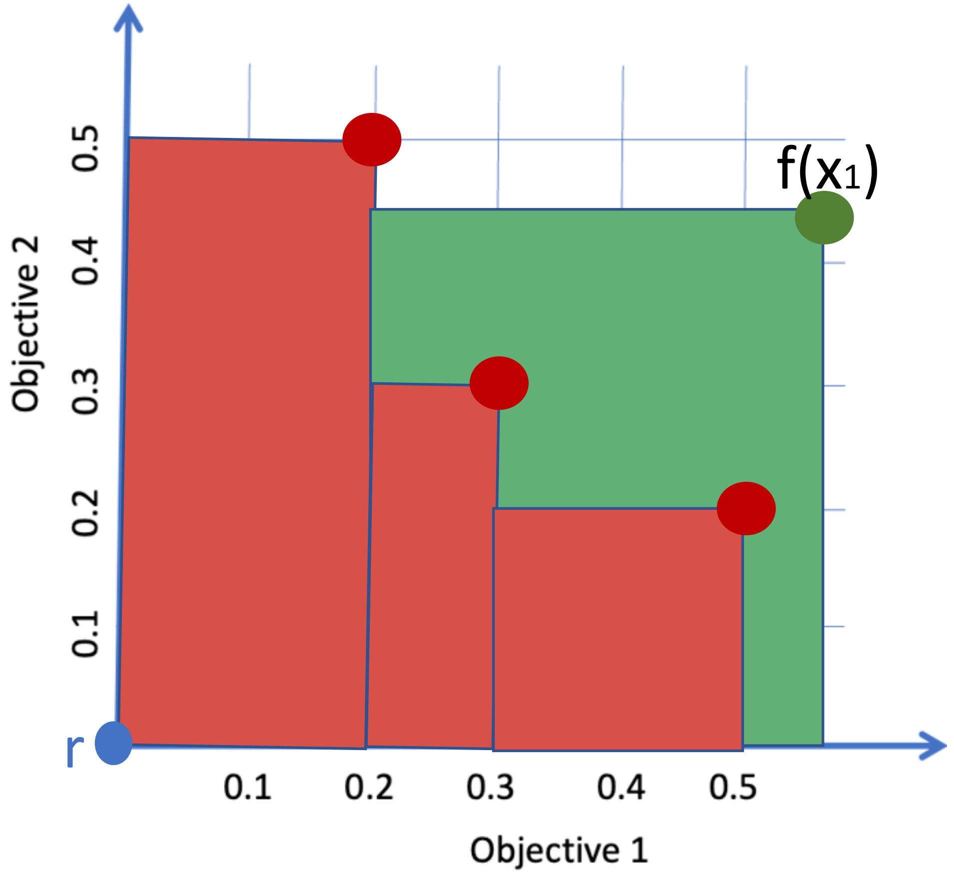
Pareto Frontier

Multi-Objective Bayesian Optimization with Expected Hypervolume Improvement



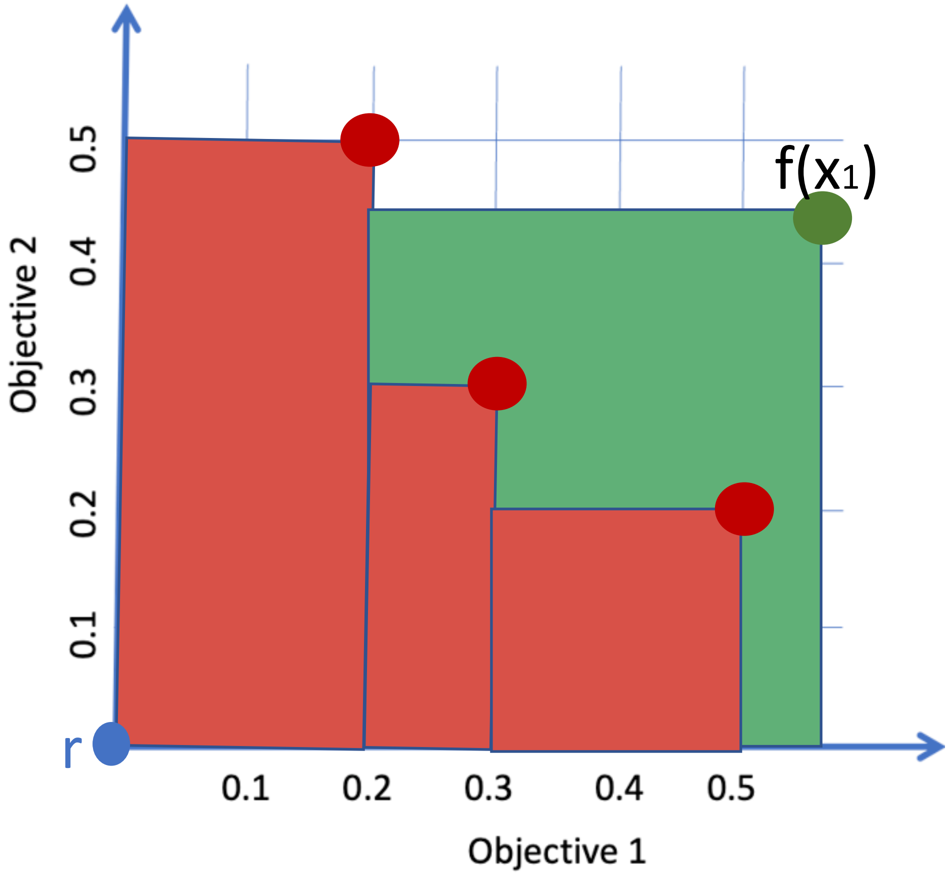
Hypervolume Indicator

Multi-Objective Bayesian Optimization with Expected Hypervolume Improvement



Hypervolume Improvement (HVI)

Multi-Objective Bayesian Optimization with Expected Hypervolume Improvement



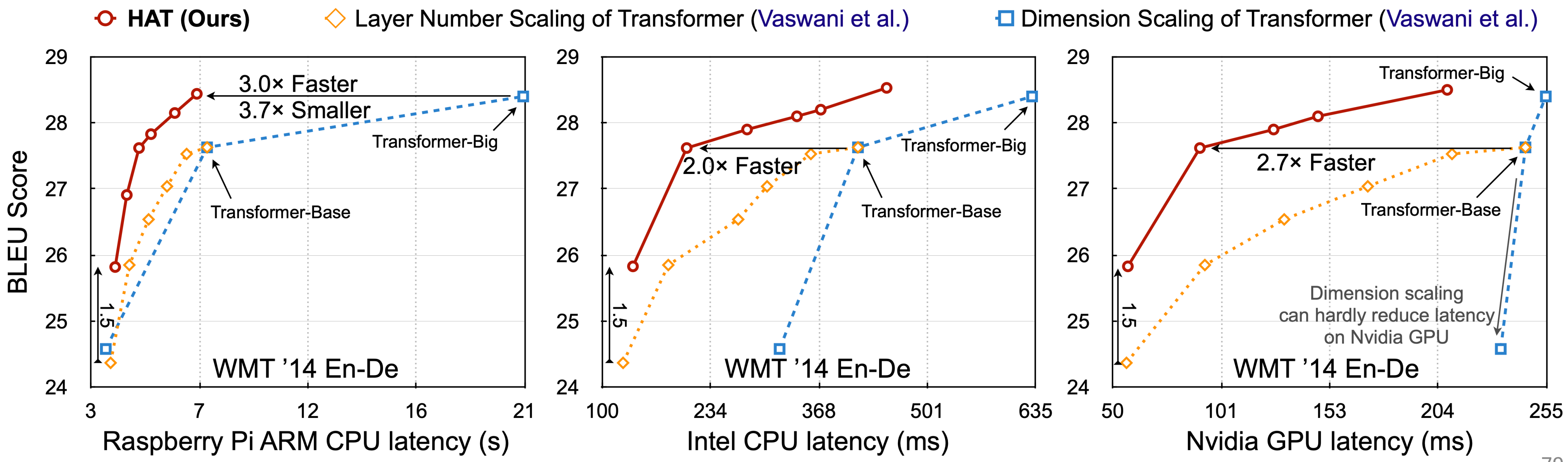
Objective function can be modeled as a multivariate Gaussian Process.

Expected Hypervolume Improvement:

$$\alpha_{\mathrm{EHVI}}(\mathcal{X}_{\text{cand}}) = \mathbb{E}\left[\mathrm{HVI}(\boldsymbol{f}(\mathcal{X}_{\text{cand}}))\right]$$

Section Summary

- Pareto Optimality and multi-objective HPO/NAS
- Multi-objective is one of the strongest selling points of AutoML
  - Suppose Transformer-Big/Base doesn't fit your deployment scenario:



From: Wang, et. al. HAT: Hardware-aware Transformers for Efficient NLP. ACL2020

Roadmap

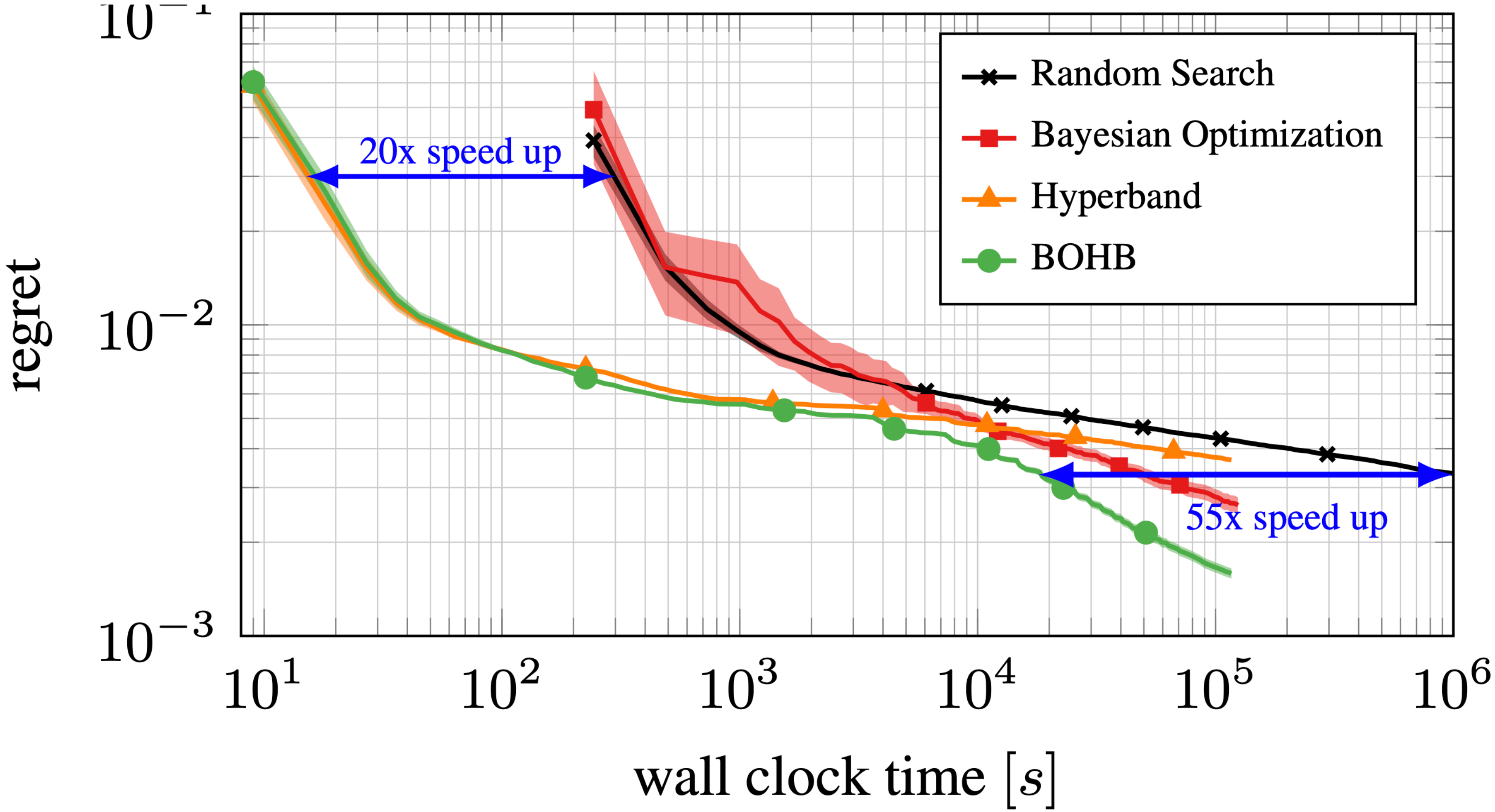
1. Motivation for AutoML
2. Hyperparameter Optimization (HPO)
3. Neural Architecture Search (NAS)
4. Extension to Multiple Objectives
5. Evaluation
    - Brief literature survey
    - Challenge of rigorous evaluation
    - Carbon footprint and broader issues
6. Toolkits
7. Survey of HPO/NAS in NLP

Which HPO/NAS method is best?

- This question is difficult to answer, perhaps even ill-defined.
  - Depends on budget, evaluation metric, task
- We'll survey 4 papers that compare HPO & NAS (on computer vision and simulation tasks), just to get a sense of the landscape
- We'll then describe competition result of the AutoML'22 MT benchmark.
- The message:
  - Evaluation of HPO/NAS methods is difficult due to computational constraints
  - The "best" solution for your problem will depend not just on the HPO/NAS method, but also on "best practices" for implementation (discussed later).

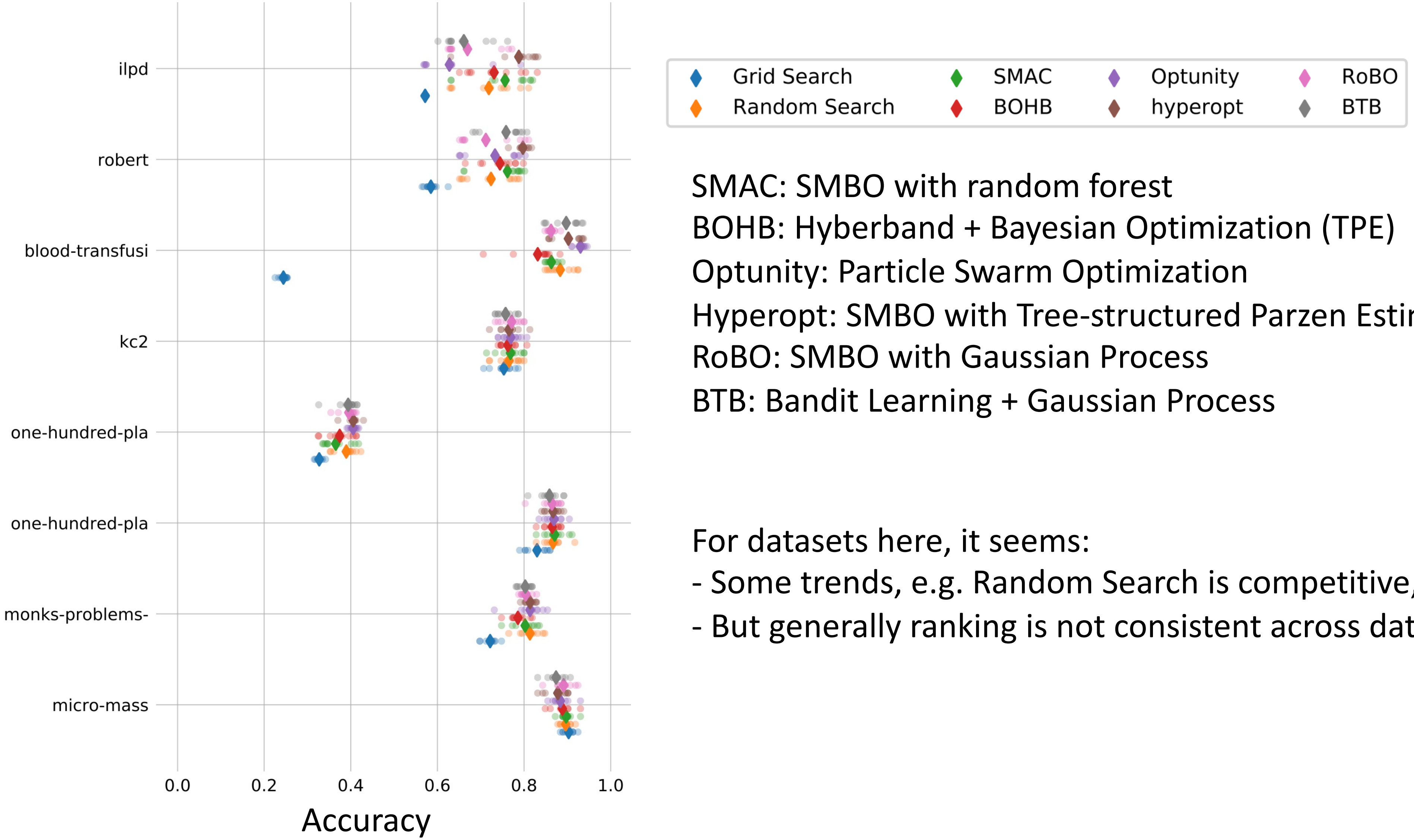
HPO comparison 1: Falkner, et. Al. BOHB: Robust and Efficient Hyperparameter Optimization at Scale. ICML2018

- "Best" method depends on your budget
- Compare methods by fixing budget, or "anytime" performance



Kohavi96 Adult dataset: predict whether a person makes over 50k per year (features from Census)

HPO comparison 2: Zoller & Huber, Benchmark and Survey of Automated Machine Learning Frameworks, JAIR 2021



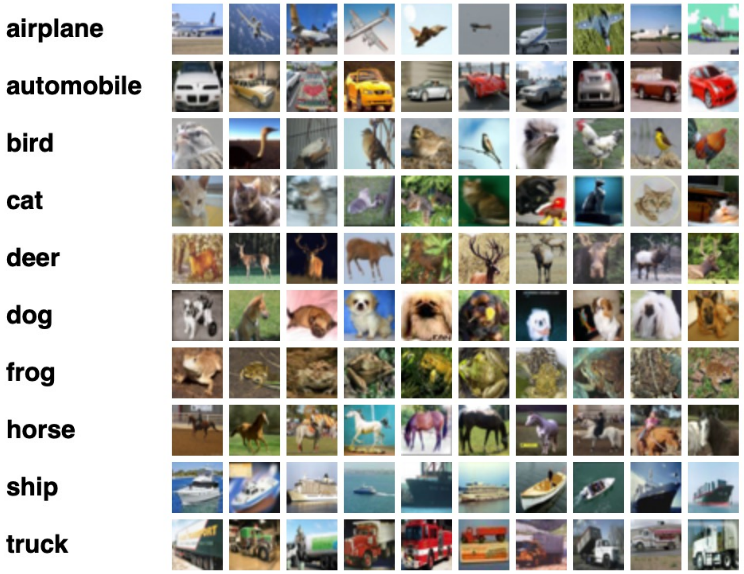
SMAC: SMBO with random forest
BOHB: Hyberband + Bayesian Optimization (TPE)
Optunity: Particle Swarm Optimization
Hyperopt: SMBO with Tree-structured Parzen Estimator (TPE)
RoBO: SMBO with Gaussian Process
BTB: Bandit Learning + Gaussian Process

For datasets here, it seems:
- Some trends, e.g. Random Search is competitive, Grid search isn't
- But generally ranking is not consistent across datasets, variance is high

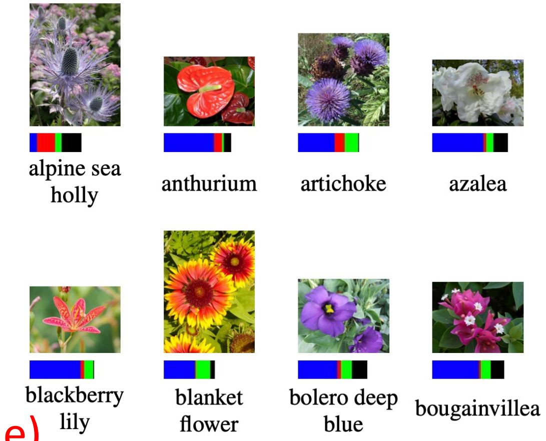
NAS Comparison 1: Yang et. al. NAS Evaluation is Frustratingly Hard, ICLR 2020

Object/scene classification data:

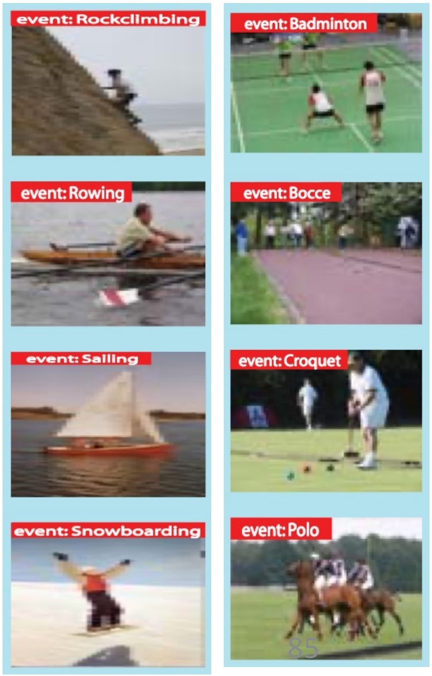
CIFAR10 & CIFAR100, 60k 32x32 images



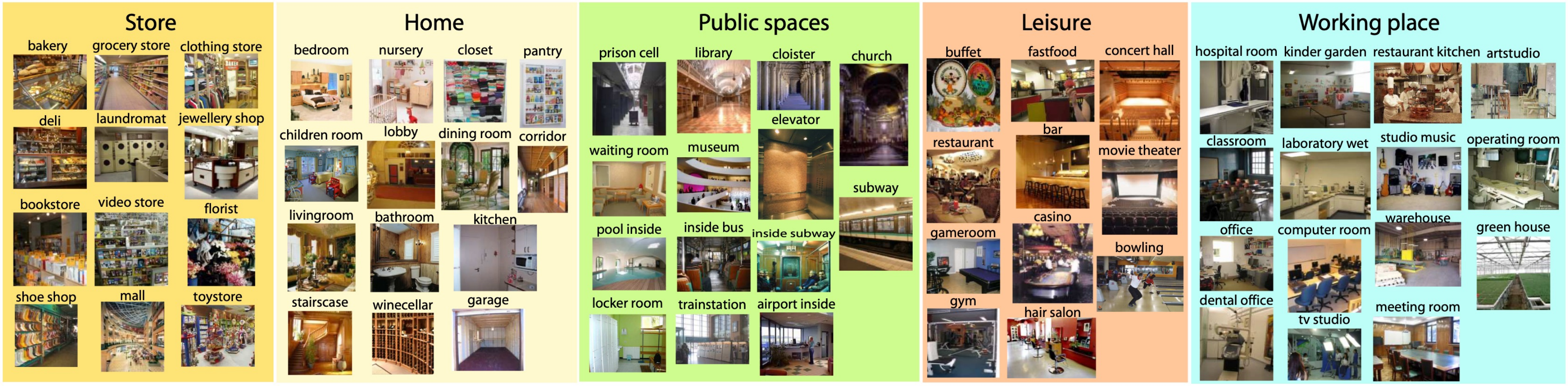
FLOWERS102

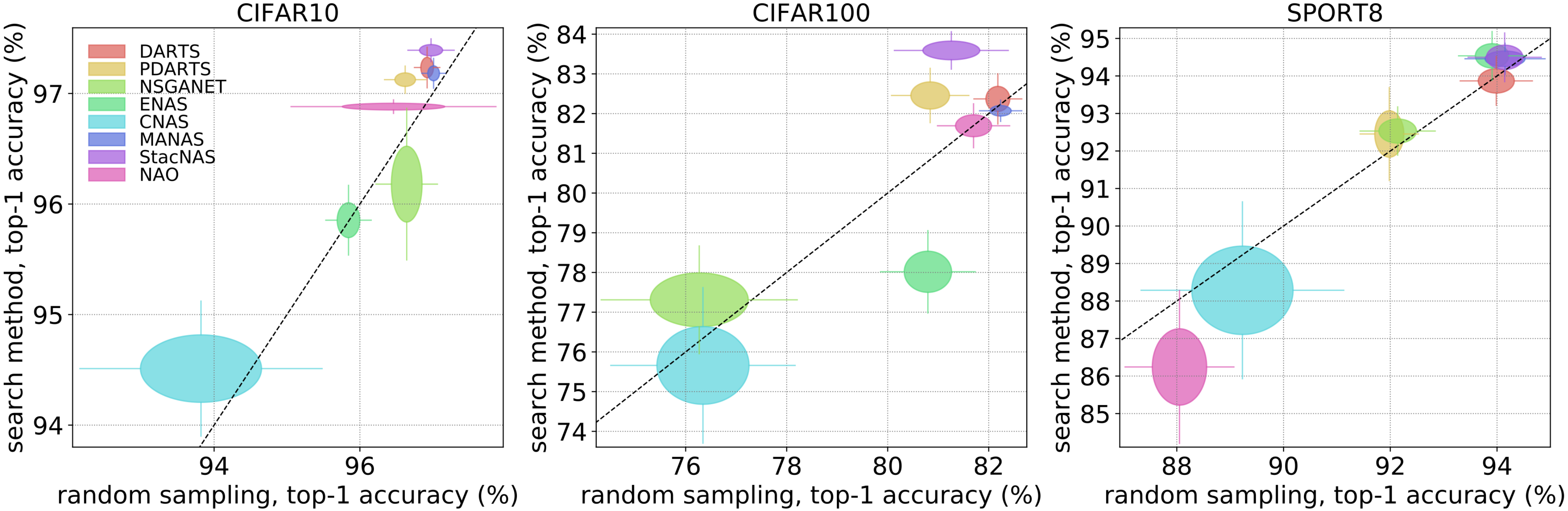


SPORT8

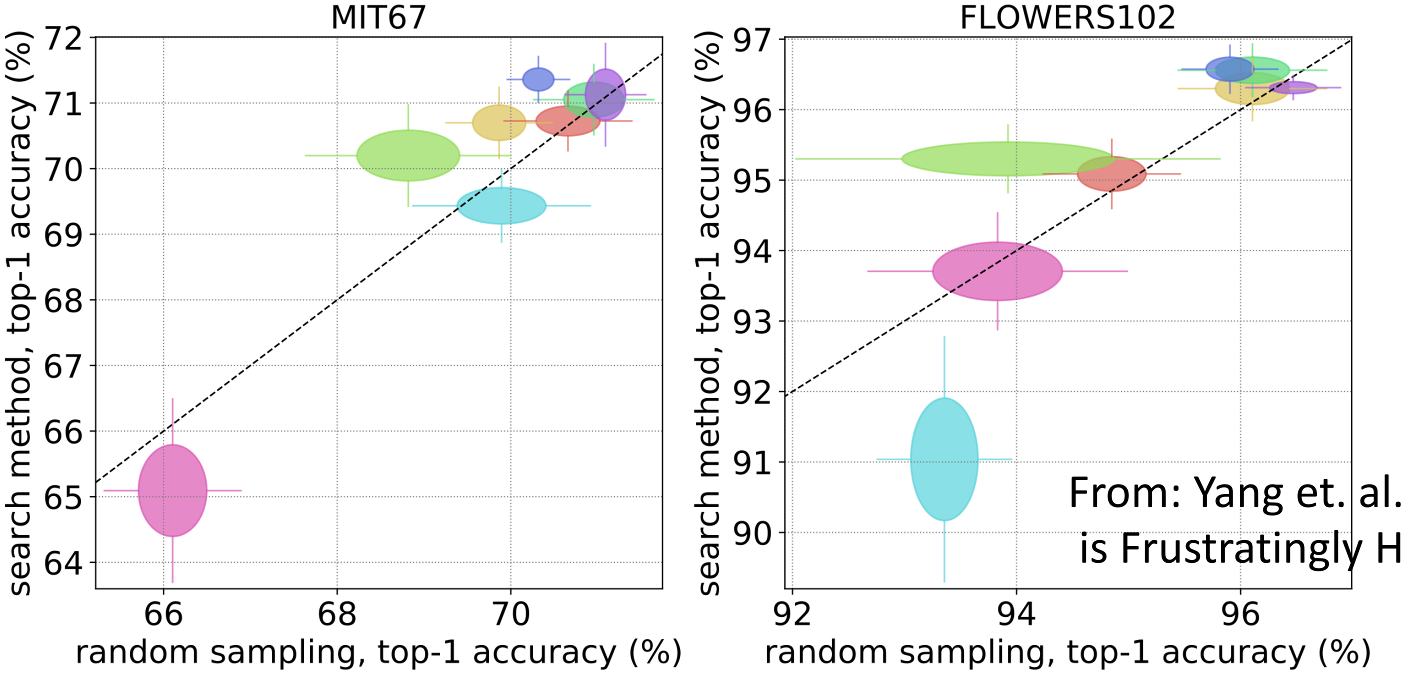


MIT67 (indoor scene)





- Compare NAS with random sampling in same space (not random search)
- Improvements not large/consistent...
- Paper argues training protocol more important



From: Yang et. al. NAS Evaluation is Frustratingly Hard, ICLR 2020

CIFAR-10: 32x32 pixel image, 10 classes, 60k samples

Reference	Published in	#Params (Millions)	Top-1 Acc(%)	GPU Days	#GPUs	AO
ResNet-110 [2]	ECCV16	1.7	93.57	-	-	Manually designed
PyramidNet [207]	CVPR17	26	96.69	-	-	
DenseNet [127]	CVPR17	25.6	96.54	-	-	
GeNet#2 (G-50) [30]	ICCV17	-	92.9	17	-	EA
Large-scale ensemble [25]	ICML17	40.4	95.6	2,500	250	
Hierarchical-EAS [19]	ICLR18	15.7	96.25	300	200	
CGP-ResSet [28]	IJCAI18	6.4	94.02	27.4	2	
AmoebaNet-B (N=6, F=128)+c/o [26]	AAAI19	34.9	97.87	3,150	450 K40	
AmoebaNet-B (N=6, F=36)+c/o [26]	AAAI19	2.8	97.45	3,150	450 K40	
Lemonade [27]	ICLR19	3.4	97.6	56	8 Titan	
EENA [149]	ICCV19	8.47	97.44	0.65	1 Titan Xp	
EENA (more channels)[149]	ICCV19	54.14	97.79	0.65	1 Titan Xp	
NASv3[12]	ICLR17	7.1	95.53	22,400	800 K40	RL
NASv3+more filters [12]	ICLR17	37.4	96.35	22,400	800 K40	
MetaQNN [23]	ICLR17	-	93.08	100	10	
NASNet-A (7 @ 2304)+c/o [15]	CVPR18	87.6	97.60	2,000	500 P100	
NASNet-A (6 @ 768)+c/o [15]	CVPR18	3.3	97.35	2,000	500 P100	
Block-QNN-Connection more filter [16]	CVPR18	33.3	97.65	96	32 1080Ti	
Block-QNN-Depthwise, N=3 [16]	CVPR18	3.3	97.42	96	32 1080Ti	
ENAS+macro [13]	ICML18	38.0	96.13	0.32	1	
ENAS+micro+c/o [13]	ICML18	4.6	97.11	0.45	1	
Path-level EAS [139]	ICML18	5.7	97.01	200	-	
Path-level EAS+c/o [139]	ICML18	5.7	97.51	200	-	
ProxylessNAS-RL+c/o[132]	ICLR19	5.8	97.70	-	-	
FPNAS[208]	ICCV19	5.76	96.99	-	-	
DARTS(first order)+c/o[17]	ICLR19	3.3	97.00	1.5	4 1080Ti	GD
DARTS(second order)+c/o[17]	ICLR19	3.3	97.23	4	4 1080Ti	
sharpDARTS [178]	ArXiv19	3.6	98.07	0.8	1 2080Ti	
P-DARTS+c/o[128]	ICCV19	3.4	97.50	0.3	-	
P-DARTS(large)+c/o[128]	ICCV19	10.5	97.75	0.3	-	
SETN[209]	ICCV19	4.6	97.31	1.8	-	
GDAS+c/o [154]	CVPR19	2.5	97.18	0.17	1	
SNAS+moderate constraint+c/o [155]	ICLR19	2.8	97.15	1.5	1	
BayesNAS[210]	ICML19	3.4	97.59	0.1	1	
ProxylessNAS-GD+c/o[132]	ICLR19	5.7	97.92	-	-	
PC-DARTS+c/o [211]	CVPR20	3.6	97.43	0.1	1 1080Ti	
MiLeNAS[153]	CVPR20	3.87	97.66	0.3	-	
SGAS[212]	CVPR20	3.8	97.61	0.25	1 1080Ti	
GDAS-NSAS[213]	CVPR20	3.54	97.27	0.4	-	
NASBOT[160]	NeurIPS18	-	91.31	1.7	-	SMBO
PNAS [18]	ECCV18	3.2	96.59	225	-	
EPNAS[166]	BMVC18	6.6	96.29	1.8	1	
GHN[214]	ICLR19	5.7	97.16	0.84	-	
NAO+random+c/o[169]	NeurIPS18	10.6	97.52	200	200 V100	RS
SMASH [14]	ICLR18	16	95.97	1.5	-	
Hierarchical-random [19]	ICLR18	15.7	96.09	8	200	
RandomNAS [180]	UAI19	4.3	97.15	2.7	-	
DARTS - random+c/o [17]	ICLR19	3.2	96.71	4	1	
RandomNAS-NSAS[213]	CVPR20	3.08	97.36	0.7	-	
NAO+weight sharing+c/o [169]	NeurIPS18	2.5	97.07	0.3	1 V100	GD+SMBO
RENASNet+c/o[42]	CVPR19	3.5	91.12	1.5	4	EA+RL
CARS[40]	CVPR20	3.6	97.38	0.4	-	EA+GD

Evolutionary

Reinforcement Learning

Gradient

SMBO, e.g. Bayesian

Random Search

ImageNet (subset): 224x224 pixel image, 1000 classes, 1million samples

Reference	Published in	#Params (Millions)	Top-1/5 Acc(%)	GPU Days	#GPUs	AO
ResNet-152 [2]	CVPR16	230	70.62/95.51	-	-	Manually designed
PyramidNet [207]	CVPR17	116.4	70.8/95.3	-	-	
SENet-154 [126]	CVPR17	-	71.32/95.53	-	-	
DenseNet-201 [127]	CVPR17	76.35	78.54/94.46	-	-	
MobileNetV2 [215]	CVPR18	6.9	74.7/-	-	-	
GeNet#2[30]	ICCV17	-	72.13/90.26	17	-	EA
AmoebaNet-C(N=4,F=50)[26]	AAAI19	6.4	75.7/92.4	3,150	450 K40	
Hierarchical-EAS[19]	ICLR18	-	79.7/94.8	300	200	
AmoebaNet-C(N=6,F=228)[26]	AAAI19	155.3	83.1/96.3	3,150	450 K40	
GreedyNAS [216]	CVPR20	6.5	77.1/93.3	1	-	
NASNet-A(4@1056)	ICLR17	5.3	74.0/91.6	2,000	500 P100	RL
NASNet-A(6@4032)	ICLR17	88.9	82.7/96.2	2,000	500 P100	
Block-QNN[16]	CVPR18	91	81.0/95.42	96	32 1080Ti	
Path-level EAS[139]	ICML18	-	74.6/91.9	8.3	-	
ProxylessNAS(GPU) [132]	ICLR19	-	75.1/92.5	8.3	-	
ProxylessNAS-RL(mobile) [132]	ICLR19	-	74.6/92.2	8.3	-	
MnasNet[130]	CVPR19	5.2	76.7/93.3	1,666	-	
EfficientNet-B0[142]	ICML19	5.3	77.3/93.5	-	-	
EfficientNet-B7[142]	ICML19	66	84.4/97.1	-	-	
FPNAS[208]	ICCV19	3.41	73.3/-	0.8	-	
DARTS (searched on CIFAR-10)[17]	ICLR19	4.7	73.3/81.3	4	-	GD
sharpDARTS[178]	Arxiv19	4.9	74.9/92.2	0.8	-	
P-DARTS[128]	ICCV19	4.9	75.6/92.6	0.3	-	
SETN[209]	ICCV19	5.4	74.3/92.0	1.8	-	
GDAS [154]	CVPR19	4.4	72.5/90.9	0.17	1	
SNAS[155]	ICLR19	4.3	72.7/90.8	1.5	-	
ProxylessNAS-G[132]	ICLR19	-	74.2/91.7	-	-	
BayesNAS[210]	ICML19	3.9	73.5/91.1	0.2	1	
FBNet[131]	CVPR19	5.5	74.9/-	216	-	
OFA[217]	ICLR20	7.7	77.3/-	-	-	
AtomNAS[218]	ICLR20	5.9	77.6/93.6	-	-	
MiLeNAS[153]	CVPR20	4.9	75.3/92.4	0.3	-	
DSNAS[219]	CVPR20	-	74.4/91.54	17.5	4 Titan X	
SGAS[212]	CVPR20	5.4	75.9/92.7	0.25	1 1080Ti	
PC-DARTS [211]	CVPR20	5.3	75.8/92.7	3.8	8 V100	
DenseNAS[220]	CVPR20	-	75.3/-	2.7	-	
FBNetV2-L1[221]	CVPR20	-	77.2/-	25	8 V100	
PNAS-5(N=3,F=54)[18]	ECCV18	5.1	74.2/91.9	225	-	SMBO
PNAS-5(N=4,F=216)[18]	ECCV18	86.1	82.9/96.2	225	-	
GHN[214]	ICLR19	6.1	73.0/91.3	0.84	-	
SemiNAS[202]	CVPR20	6.32	76.5/93.2	4	-	
Hierarchical-random[19]	ICLR18	-	79.6/94.7	8.3	200	RS
OFA-random[217]	CVPR20	7.7	73.8/-	-	-	
RENASNet[42]	CVPR19	5.36	75.7/92.6	-	-	EA+RL
Evo-NAS[41]	Arxiv20	-	75.43/-	740	-	EA+RL
CARS[40]	CVPR20	5.1	75.2/92.5	0.4	-	EA+GD

NAS comparison 2: He, et. Al. AutoML: A Survey of the State-of-the-Art, 2021

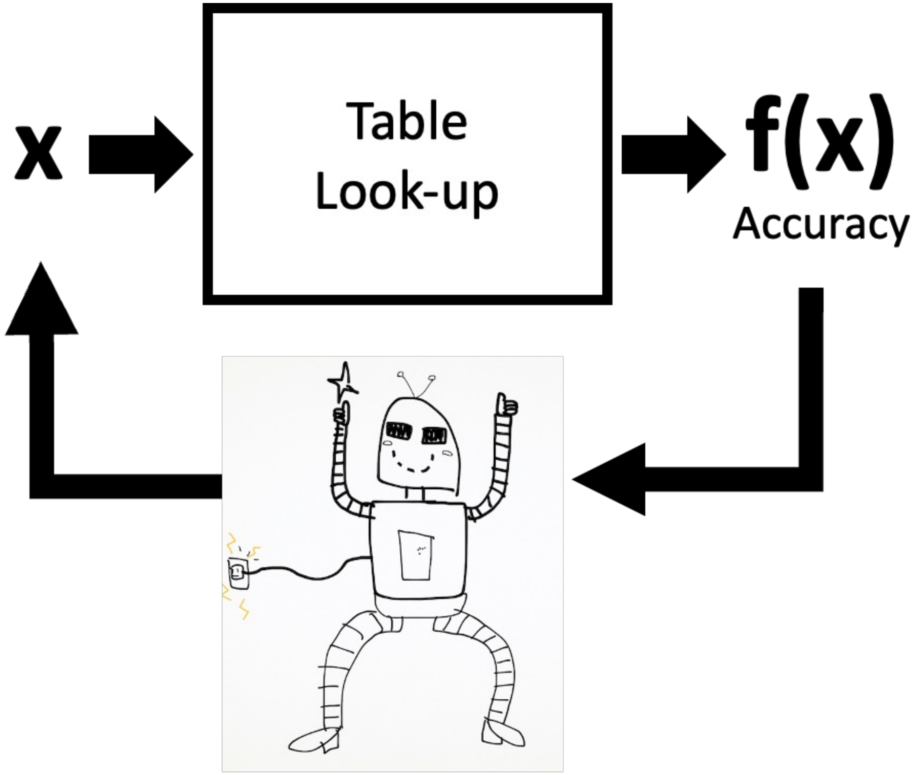
Many results! Different budgets...

Evaluation in HPO/NAS is extremely hard!

- Note previous papers focused on mostly on smaller datasets
- Evaluation is hard due to computational constraint:
  - Suppose it takes 1 week to train one model
  - Each HPO algorithm samples and trains 100 models at best
- Cannot do head-to-head comparison, repeated trials don't know if an algorithm really works!
  - Li & Talwalkar (2019) Random search & Reproducibility for Neural Architecture Search: *"Of the 12 papers published since 2018 at NeurIPS, ICML, and ICLR that introduce novel NAS methods, none are exactly reproducible."*
  - *Also: Lindauer & Hutter. Best Practices for Scientific Research on Neural Architecture Search, JMLR 2021. https://www.jmlr.org/papers/volume21/20-056/20-056.pdf*

(Crazy) Solution: Tabular Benchmarks

- One-time fixed cost:
  - Run grid/random search, training MANY models on some dataset
  - Publish all {x,f(x)} pairs in a table
- HPO algorithm developers:
  - Experiment with HPO on finite universe
  - Can run repeated trials quickly



Tabular Benchmark for NMT (Zhang & Duh, TACL2020)

Hyperparameter Type	Possible Values
# BPE Subword Units	1k, 2k, 4k, 8k, 16k, 32k, 50k
# Transformer Layers	1, 2, 4, 6
Word embedding	256, 512, 1024
# Hidden Units	1024, 2048
# Attention Heads	8, 16
Initial Learning Rate for ADAM	3×10^{-4}, 6×10^{-4}, 10×10^{-4}

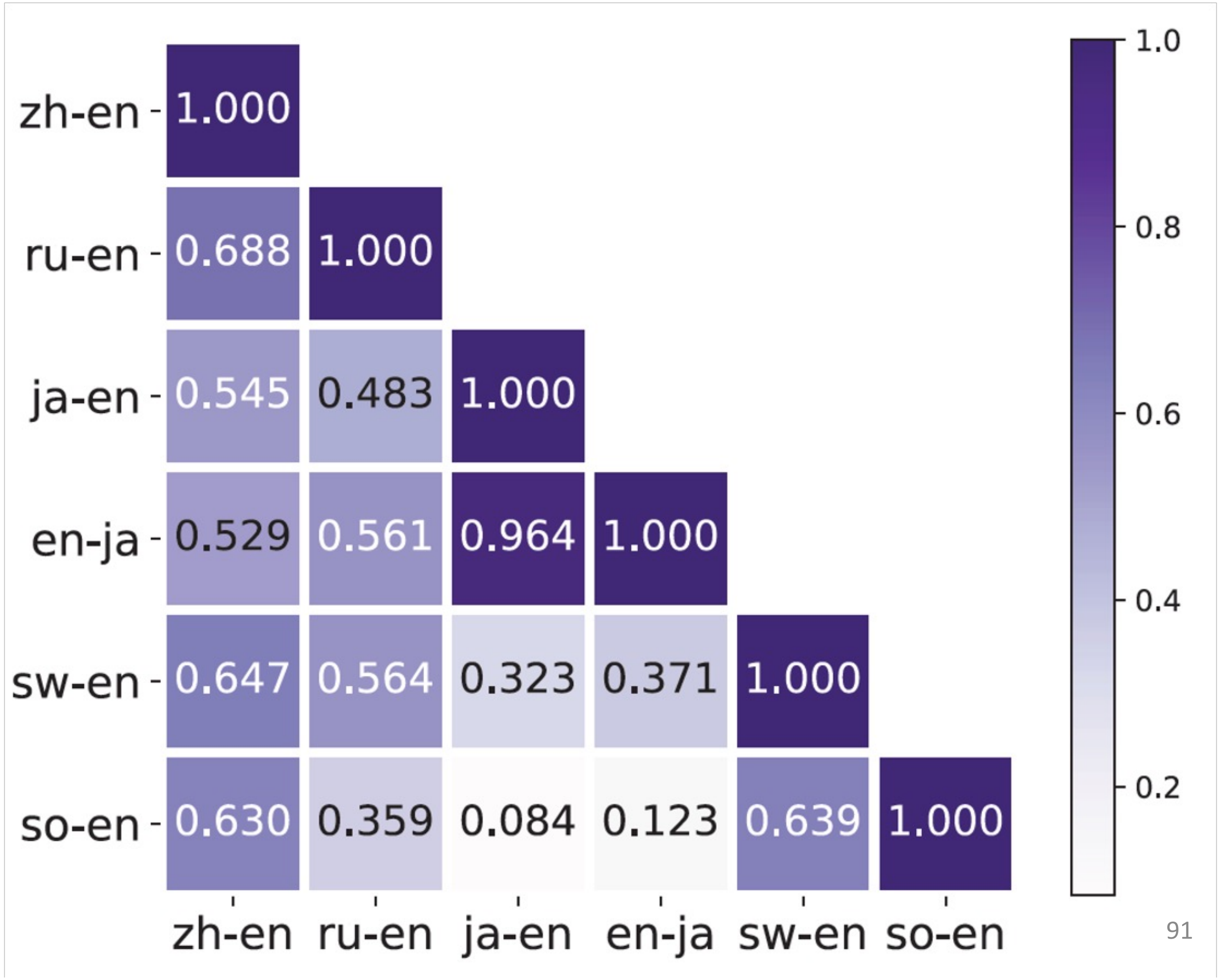
Total: 2245 Transformer models, trained on ~1550 GPU days; record BLEU, train/test time, etc.
https://github.com/Este1le/hpo_nmt

Dataset	Domain	#models
zh-en	TED	118
ru-en	TED	176
ja-en	WMT	150
en-ja	WMT	168
sw-en	MATERIAL	767
so-en	MATERIAL	605

Diversity in dataset

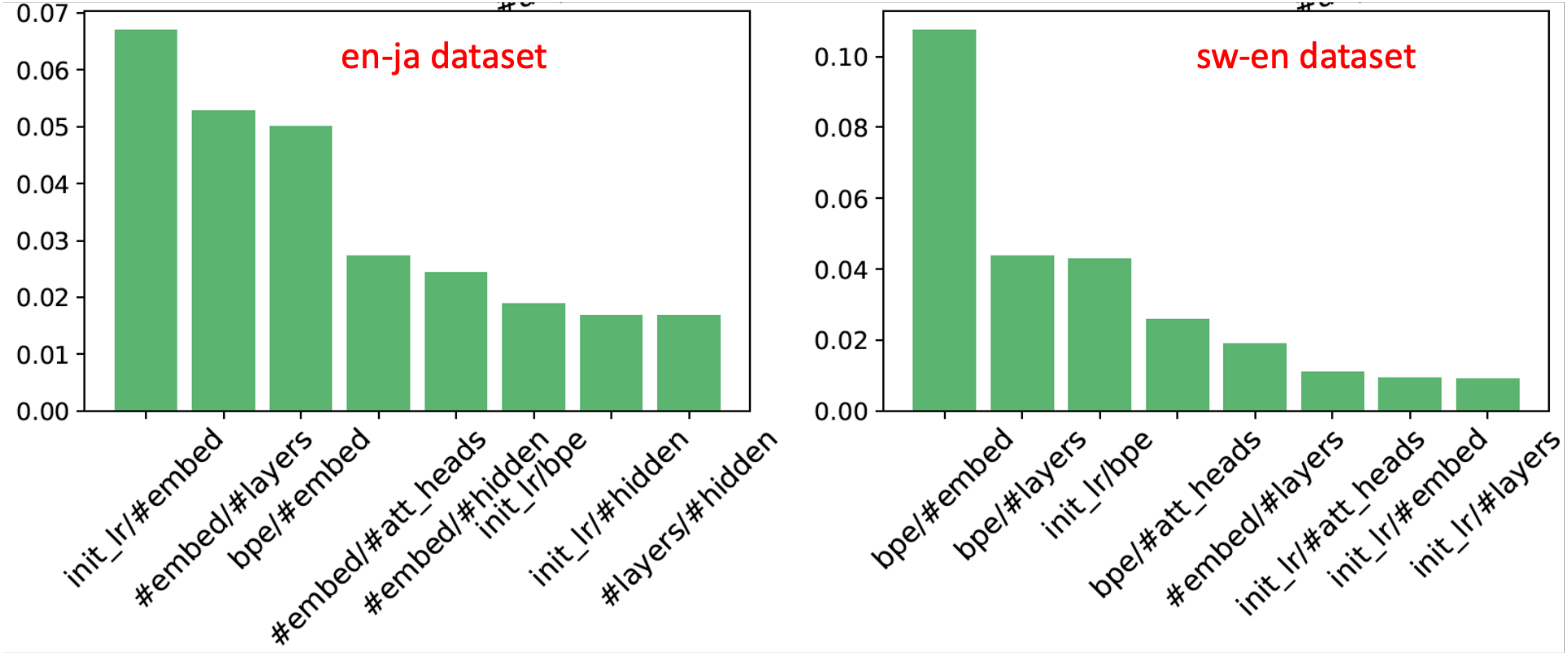
For each dataset, we order hyperparameter configurations by BLEU, then compare these rankings across datasets

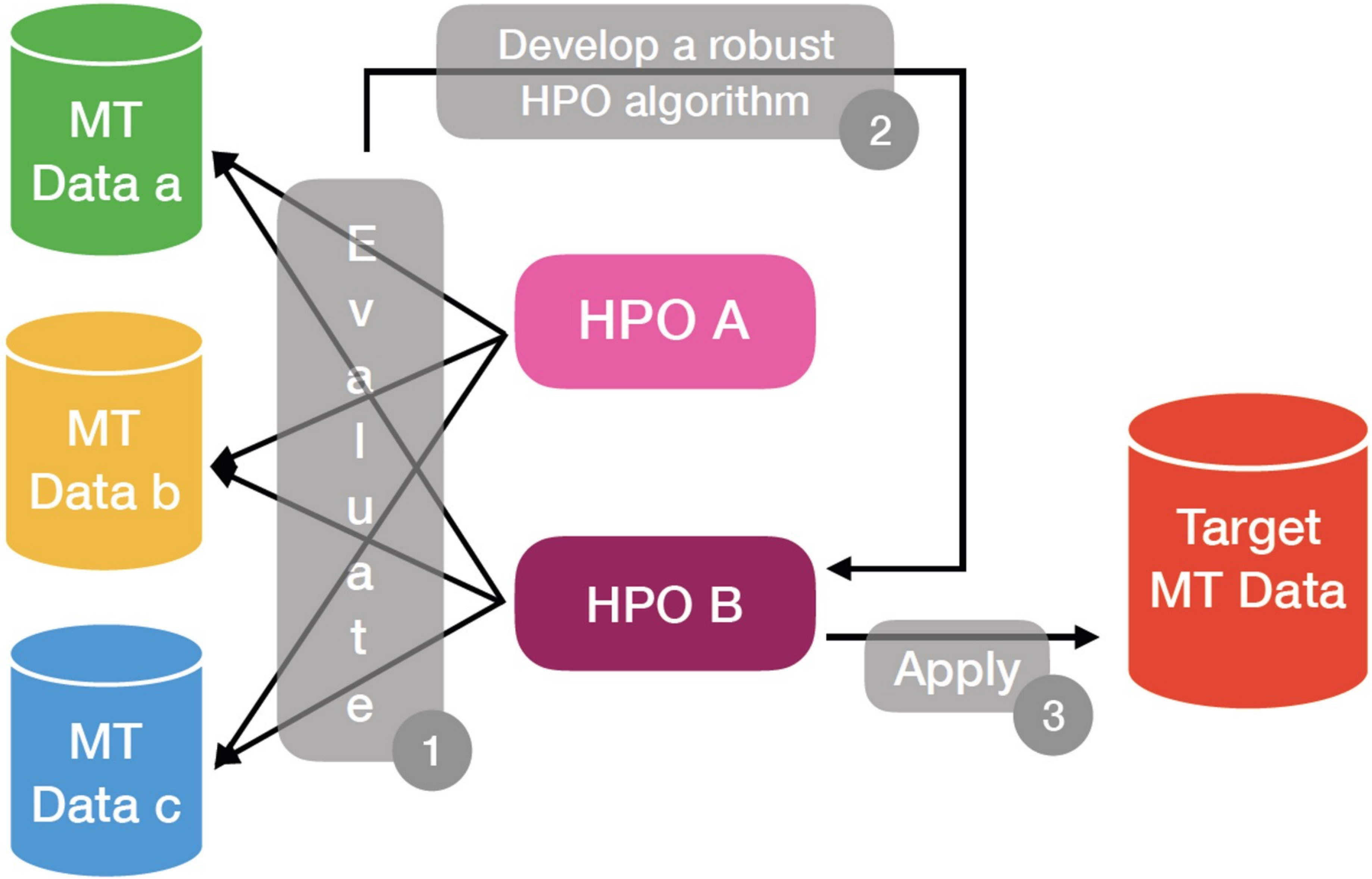
Low Spearman's correlation imply no single best set of Transformer model across datasets



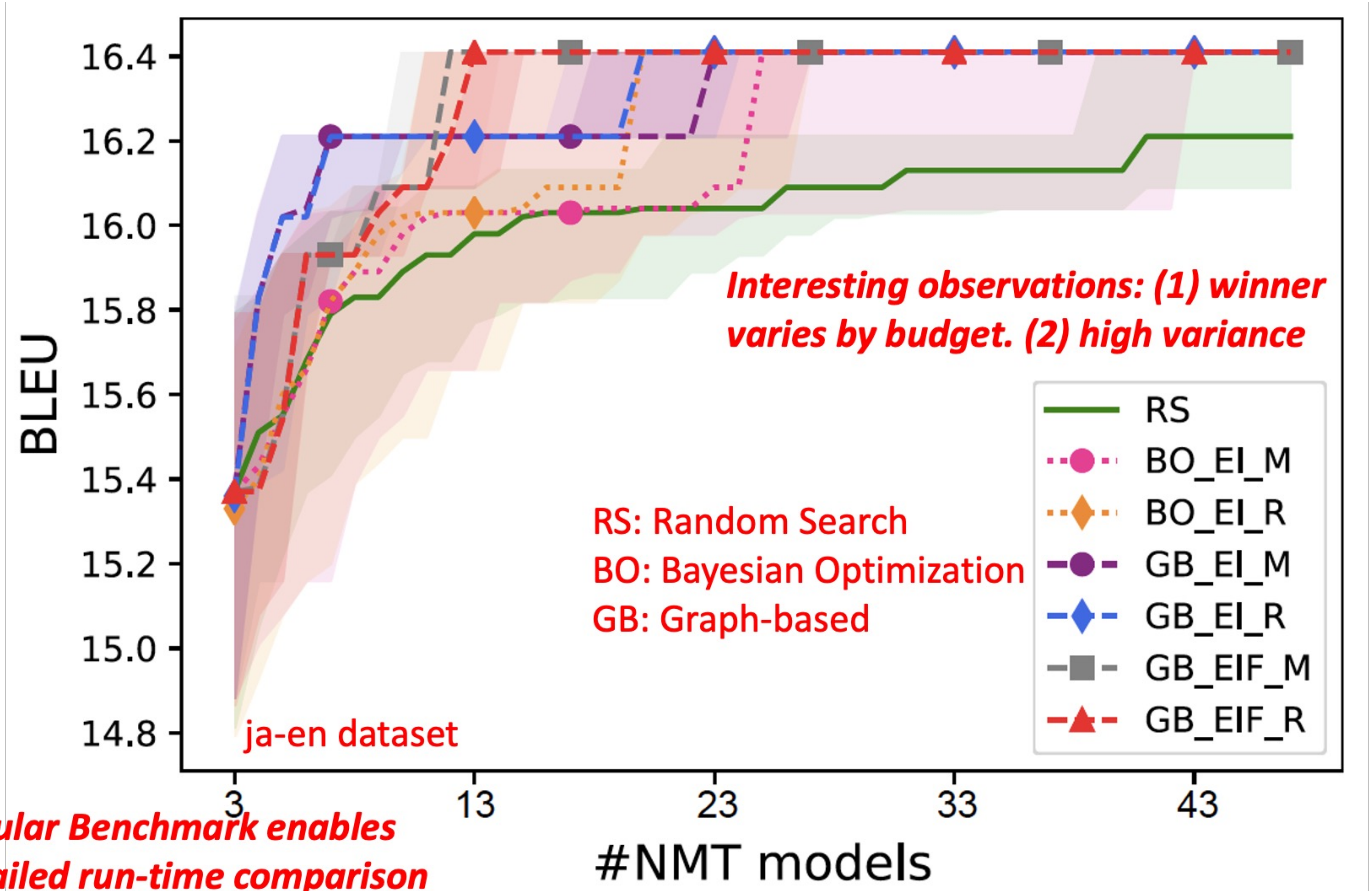
Diversity in dataset:

Hyperparameter importance by fANOVA, measuring BLEU variance when changing a specific hyperparameter value pairs





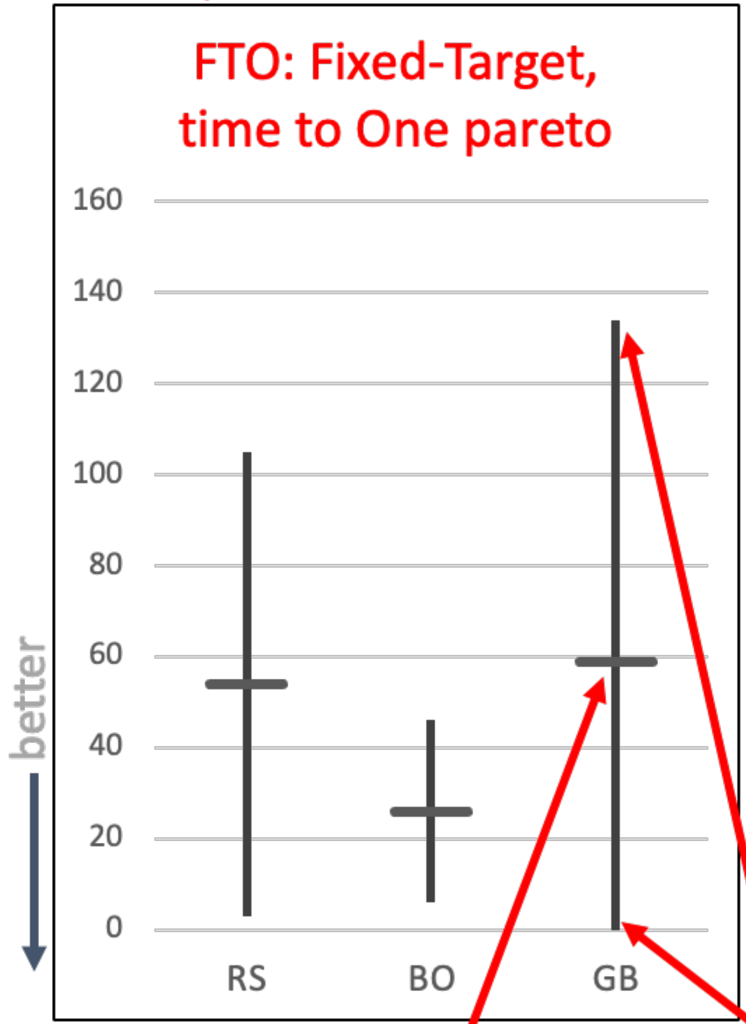
Evaluation philosophy: Find HPO methods that are robust over multiple datasets before applying to target real-world data

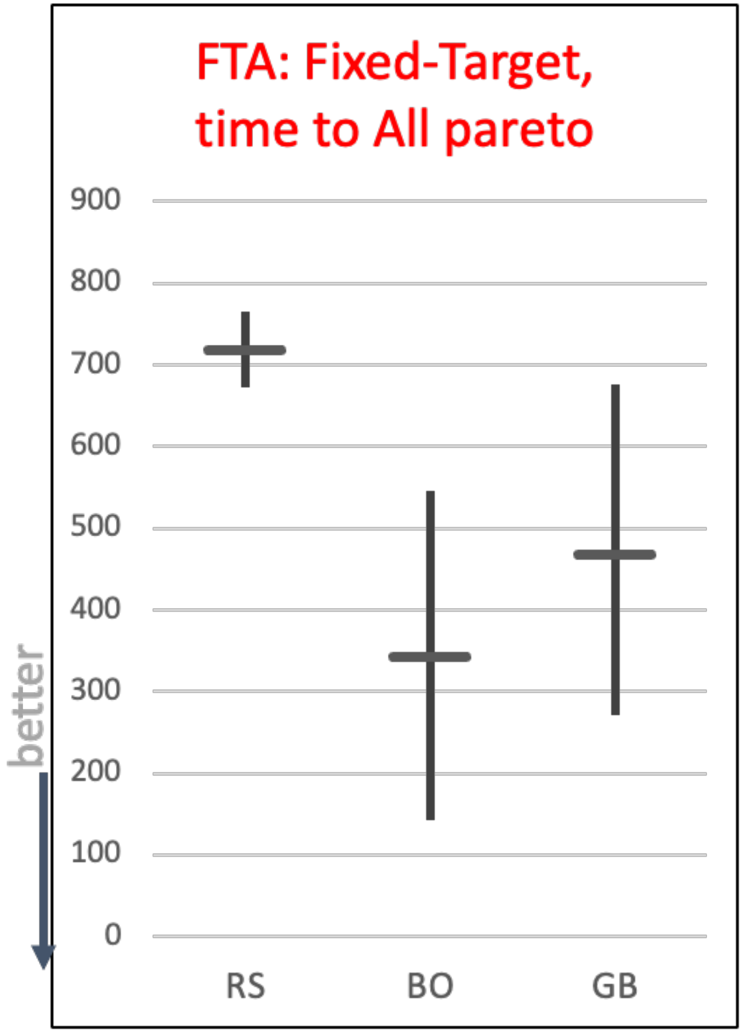


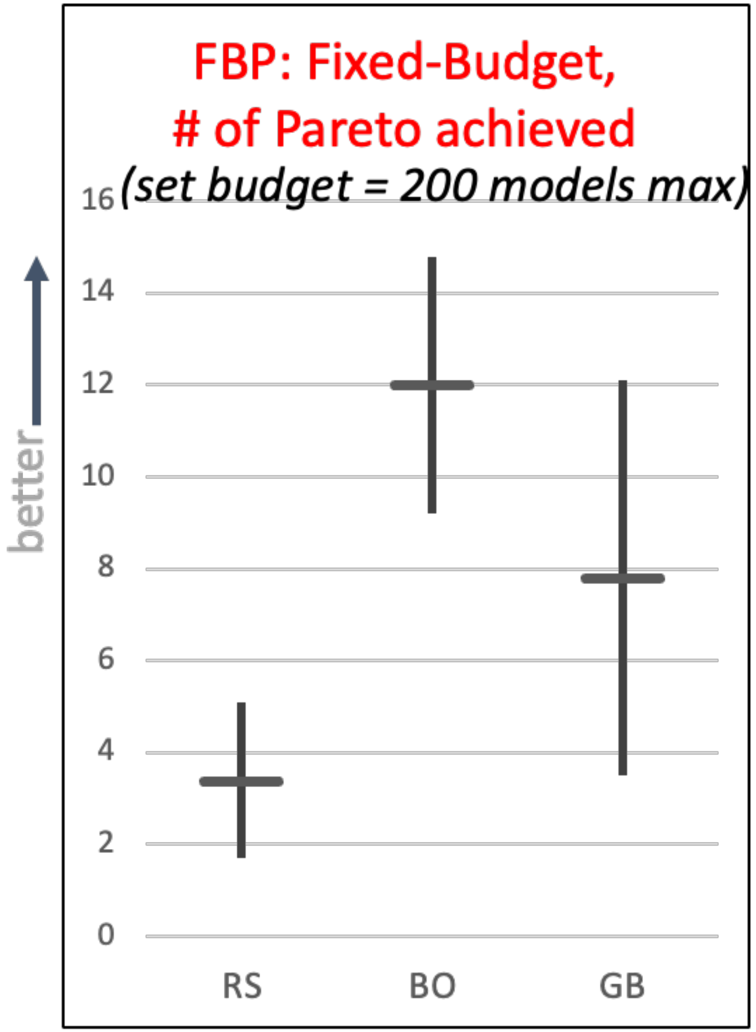
Tabular Benchmark enables detailed run-time comparison

Multi-objective evaluation metrics

Example results on sw-en data, 700+ models in tabular benchmark, 14 pareto points

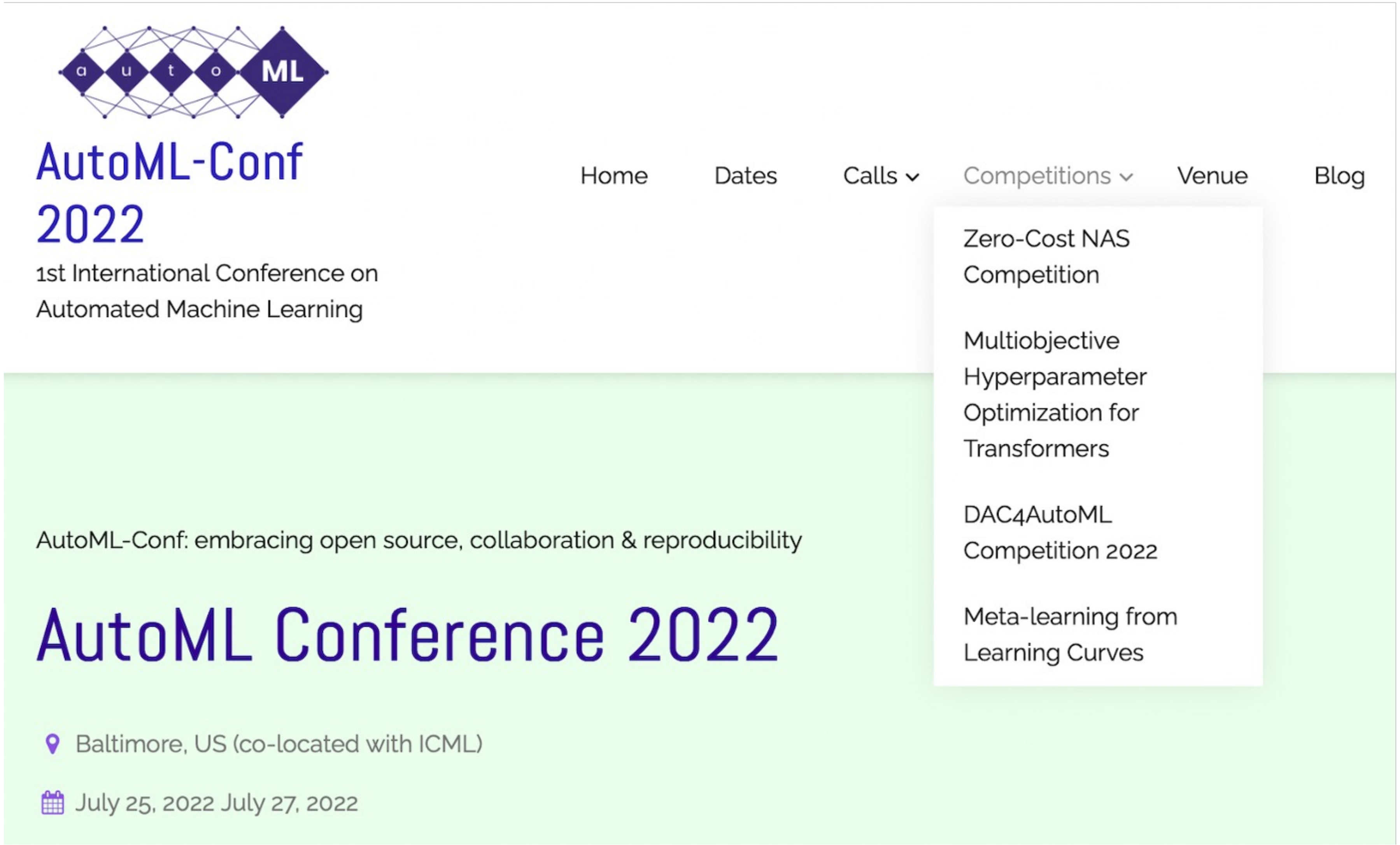






For each method, plot mean and one standard deviation bounds over 100 random runs

AutoML 2022 Competition https://automl.cc



Top performers in AutoML'22 Competition

- ESI Algiers and LAMIH/CNRS France – Evolutionary approach
  - Latin Hypercube Sampling for initial population
  - XGBRank for fitting x --> f(x), then creating “surrogate function”
  - Find next generation by optimizing NSGA-II on surrogate function
- AutoML@Freiburg – Bayes Opt. approach, with transfer learning
  - Tree-structured Parzen Estimator (TPE) for Bayes Optimization
  - Transfer learning from multiple MT datasets
  - Define task similarity by how often similar hyperparameters perform well

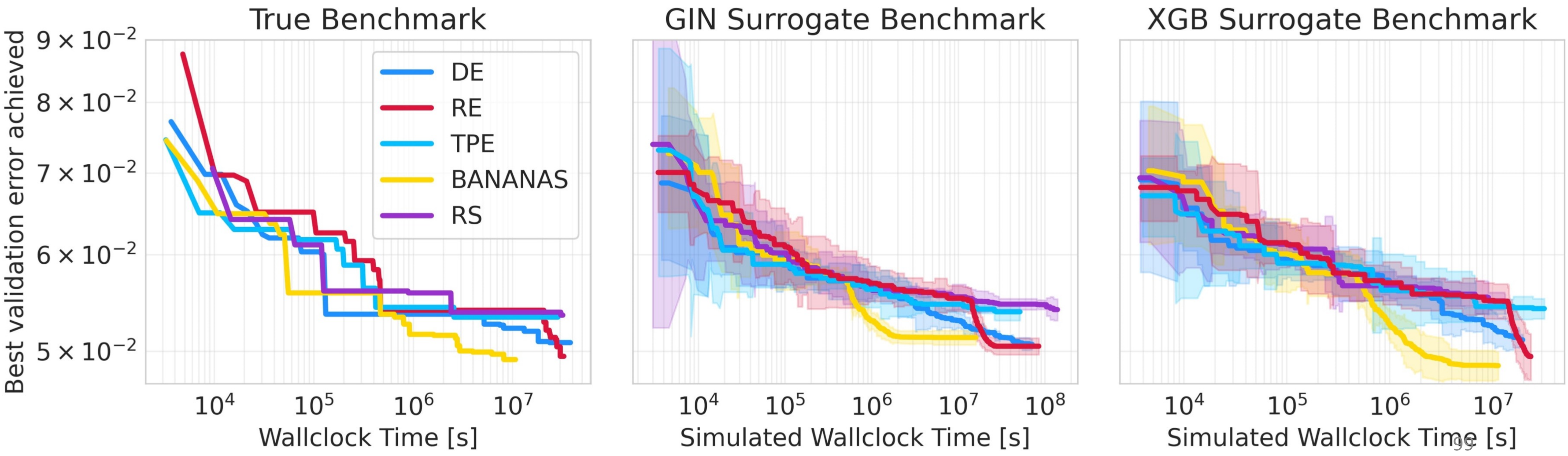
Beyond tabular benchmarks?

- Surrogate benchmark:
  - Use external ML model to estimate f(x)
  - These can create infinitely many new "rows" in table
- Open questions:
  - How many {x, f(x)} pairs are needed to train an accurate surrogate?
  - Will the surrogate model introduce bias?
  - IMHO, I'm not convinced we can do this for complex and large tasks like Transformer hyperparameters for NMT.

Surrogate benchmark

	R^2
LGBoost	**0.892**
XGBoost	0.832
GIN	0.832
NGBoost	0.810
μ-SVR	0.709
MLP (Path enc.)	0.704
RF	0.679
ϵ-SVR	0.675

- Zela, et. Al. Surrogate NAS Benchmarks, ICLR2022
- Argues that ranking of NAS methods are similar when comparing true benchmark to surrogate benchmarks (on different external models)



Discussion: CO2e footprint and energy cost

- AutoML is basically trading human effort with computer time
- What is the cost of compute?
  - We may enjoy the convenience of AutoML, but we should be aware of the cost and potentially inefficiencies
  - To put things in perspective, let's discuss how different HPO/NAS compare in terms of CO2 footprint and energy cost
  - AutoML has the potential to have both positive and negative impact!

Strubell et. al., Energy and Policy Considerations for Deep Learning in NLP, ACL2019

Estimating CO2e footprint

Consumption	**CO_2e (lbs)**
Air travel, 1 person, NY↔SF	1984
Human life, avg, 1 year	11,023
American life, avg, 1 year	36,156
Car, avg incl. fuel, 1 lifetime	126,000
Training one model (GPU)	
NLP pipeline (parsing, SRL)	39
w/ tuning & experiments	78,468
Transformer (big)	192
w/ neural arch. search	626,155

Table 1: Estimated CO_2 emissions from training common NLP models, compared to familiar consumption.[1]

Strubell et. al., Energy and Policy Considerations for Deep Learning in NLP, ACL2019

Estimating CO2e footprint

Power Usage Effectiveness (PUE) - energy for infrastructure (cooling)

Avg power draw (watts) from CPU, RAM, #g GPUs

Power consumption (kWh) from training a model

$$p_t = \frac{1.58t(p_c + p_r + gp_g)}{1000}$$

CO2e: CO2 equivalent emission (includes other greenhouse gases)

$$\mathrm{CO_2e} = 0.954 p_t$$

EPA's estimate of avg CO2 (in lb per kWh) based on U.S. non-renewable vs renewable sources

Strubell et. al., Energy and Policy Considerations for Deep Learning in NLP, ACL2019

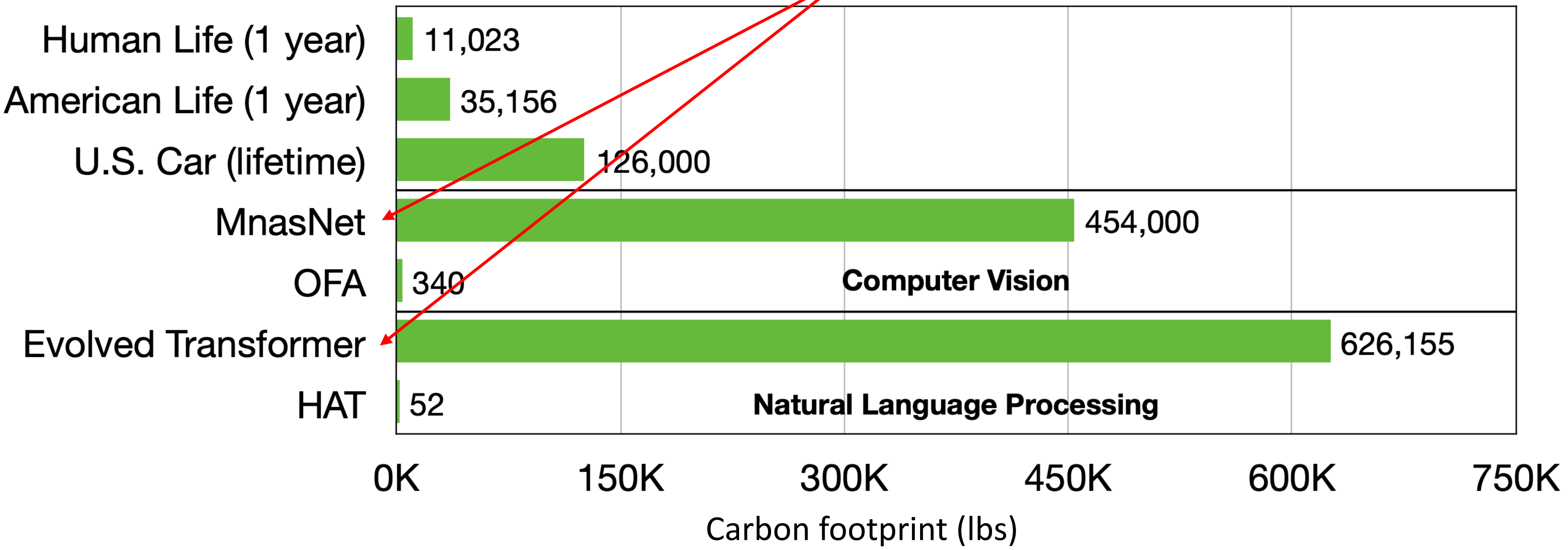
Estimating CO2e footprint

Model	Hardware	Power (W)	Hours	kWh·PUE	CO_2e	Cloud compute cost
$T2T_{base}$	P100x8	1415.78	12	27	26	\$41–\$140
$T2T_{big}$	P100x8	1515.43	84	201	192	\$289–\$981
ELMo	P100x3	517.66	336	275	262	\$433–\$1472
$BERT_{base}$	V100x64	12,041.51	79	1507	1438	\$3751–\$12,571
$BERT_{base}$	TPUv2x16	—	96	—	—	\$2074–\$6912
NAS	P100x8	1515.43	274,120	656,347	626,155	\$942,973–\$3,201,722
NAS	TPUv2x1	—	32,623	—	—	\$44,055–\$146,848
GPT-2	TPUv3x32	—	168	—	—	\$12,902–\$43,008

Table 3: Estimated cost of training a model in terms of CO_2 emissions (lbs) and cloud compute cost (USD).[7] Power and carbon footprint are omitted for TPUs due to lack of public information on power draw for this hardware.

AutoML can have both positive and negative impact on carbon footprint

To be fair, these NAS methods aren't optimizing for training cost, but the difference with those that do can be large. Also, see next slide for revised estimate



Cai et. Al. Enable Deep Learning on Mobile Devices: Methods, Sytems, and Applications, ACM Trans. Design Automation of Electronic Systems, 2022

Estimating carbon footprint, revisited

- Recommended reading if interested: Patterson, et. al. Carbon Emissions and Large Neural Network Training
  - It's challenging to estimate CO2e retrospectively; ideal for each paper author to measure it
  - Specific data center & time matters
  - Inference may take more energy in the aggregate than training/AutoML
  - Note CO2e for Evolved Transformer is very different from previous papers!

Number of Parameters (B)	0.064 per model
Percent of model activated on every token	100%
Developer	
Datacenter of original experiment	Google Georgia
When model ran	Dec 2018
Datacenter Gross CO_2e/KWh (kg/KWh when it was run)	0.431
Datacenter Net CO2e/KWh (kg/KWh when it was run)	0.431
Datacenter PUE (when it was run)	1.10
Processor	TPU v2
Chip Thermal Design Power (TDP in Watts)	280
Measured System Average Power per Accelerator, including memory, network interface, fans, host CPU (W)	208
Measured Performance (TFLOPS/s)[12]	24.8
Number of Chips	200
Training time (days)	6.8
Total Computation (floating point operations)	2.91E+21
Energy Consumption (MWh)	7.5
% of Google 2019 total energy consumption (12.2 TWh = 12,200,000 MWh) [Goo20]	0.00006%
Gross tCO_2e for Model Training	3.2
Net tCO_2e for Model Training	3.2
Fraction of NAS Estimate in [Str19] (284 tCO2e)	0.011
Fraction of equivalent jet plane CO_2e round trip San Francisco ↔ New York (~180 t; see Ap. A)	0.018

3.2x2200 = 7040 lbs

Section Summary

- Evaluation of HPO/NAS is non-trivial in two aspects
- First, what do you want to look at?
  - Fixed budget, or anytime performance
  - What metric? What datasets?
- Second, can you even run the evaluation in a rigorous fashion?
  - Tabular & Surrogate benchmark
  - NMT example
- Awareness of CO2e footprint discussions, potential of AutoML for positive and negative impact

Roadmap

1. Motivation for AutoML
2. Hyperparameter Optimization (HPO)
3. Neural Architecture Search (NAS)
4. Extension to Multiple Objectives
5. Evaluation
6. Toolkits
7. Survey of HPO/NAS in NLP

Software Implementation of AutoML

- HPO/NAS algorithms are in general simple to implement.
- Challenge is the interface with the ML toolkit and the underlying computing infrastructure.
- Design considerations:
  - Automatically submit jobs
  - Automatically check job states
  - Automatically evaluate and collect results
  - Parallelization
  - Maximize the GPU utilization
  - Allow users to customize the AutoML runs by specifying arguments, e.g. #GPU, #configuration, #epochs

Existing AutoML Toolkits

Google Vizier

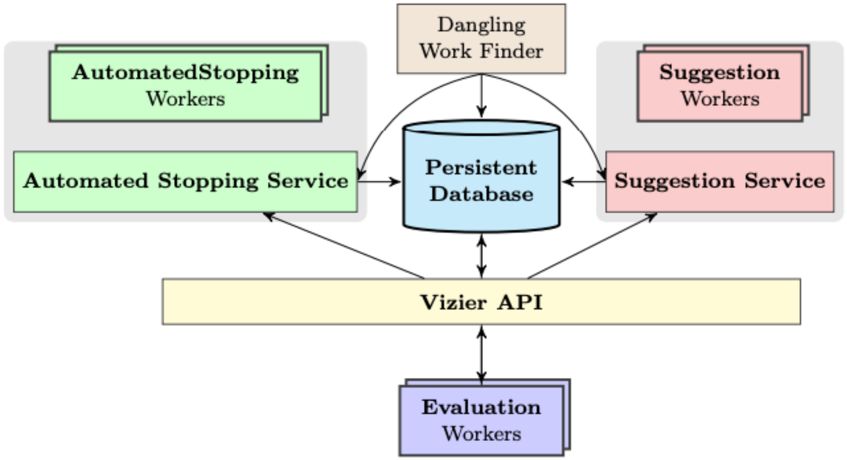
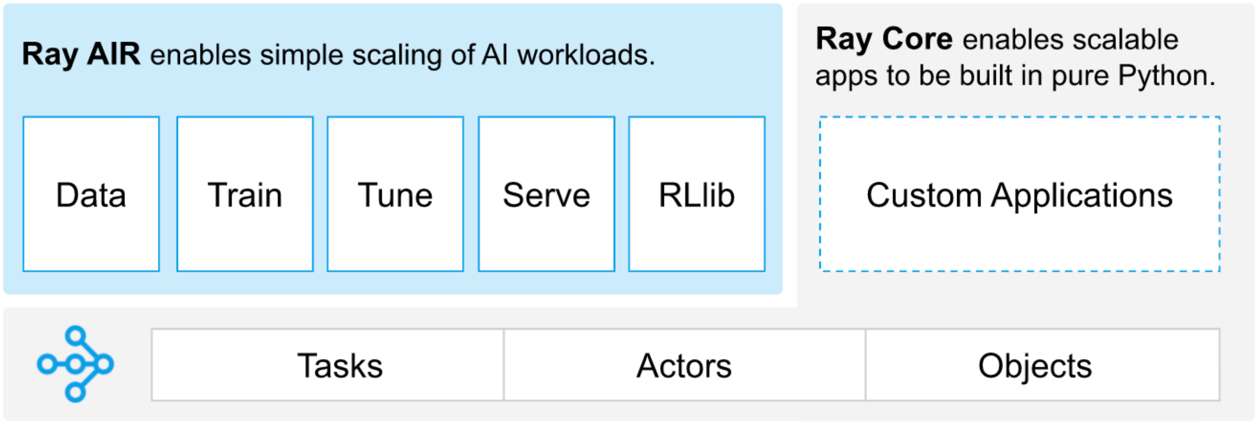


Figure 1: Architecture of Vizier service: Main components are (1) Dangling work finder (restarts work lost to preemptions) (2) Persistent Database holding the current state of all Studies (3) Suggestion Service (creates new Trials), (4) Early Stopping Service (helps terminate a Trial early) (5) Vizier API (JSON, validation, multiplexing) (6) Evaluation workers (provided and owned by the user).

Ray Tune



From: Google Vizier: A Service for Black-Box Optimization, Golovin et al. 2017
https://docs.ray.io/en/latest/

Use existing AutoML toolkits or Implement your own?

- Choice 1:

  Take an existing AutoML toolkit, and reimplement your training pipeline.

- Choice 2:

  Already have a training pipeline, e.g. Amazon Sockeye for MT, add an AutoML wrapper on top of it.

  It's worth implementing AutoML from scratch in this case.

Case Study: Amazon Sockeye with AutoML

- Amazon Sockeye:

  An open-source sequence-to-sequence framework for NMT built on PyTorch.

  https://github.com/awslabs/sockeye

- Sockeye-recipes (Duh et al.):

  Training scripts and recipes for the Sockeye toolkit.

  https://github.com/kevinduh/sockeye-recipes3

- Sockeye-recipes with AutoML:

  Automatic hyperparameter search with asynchronous successive halving on top of sockeye-recipes.

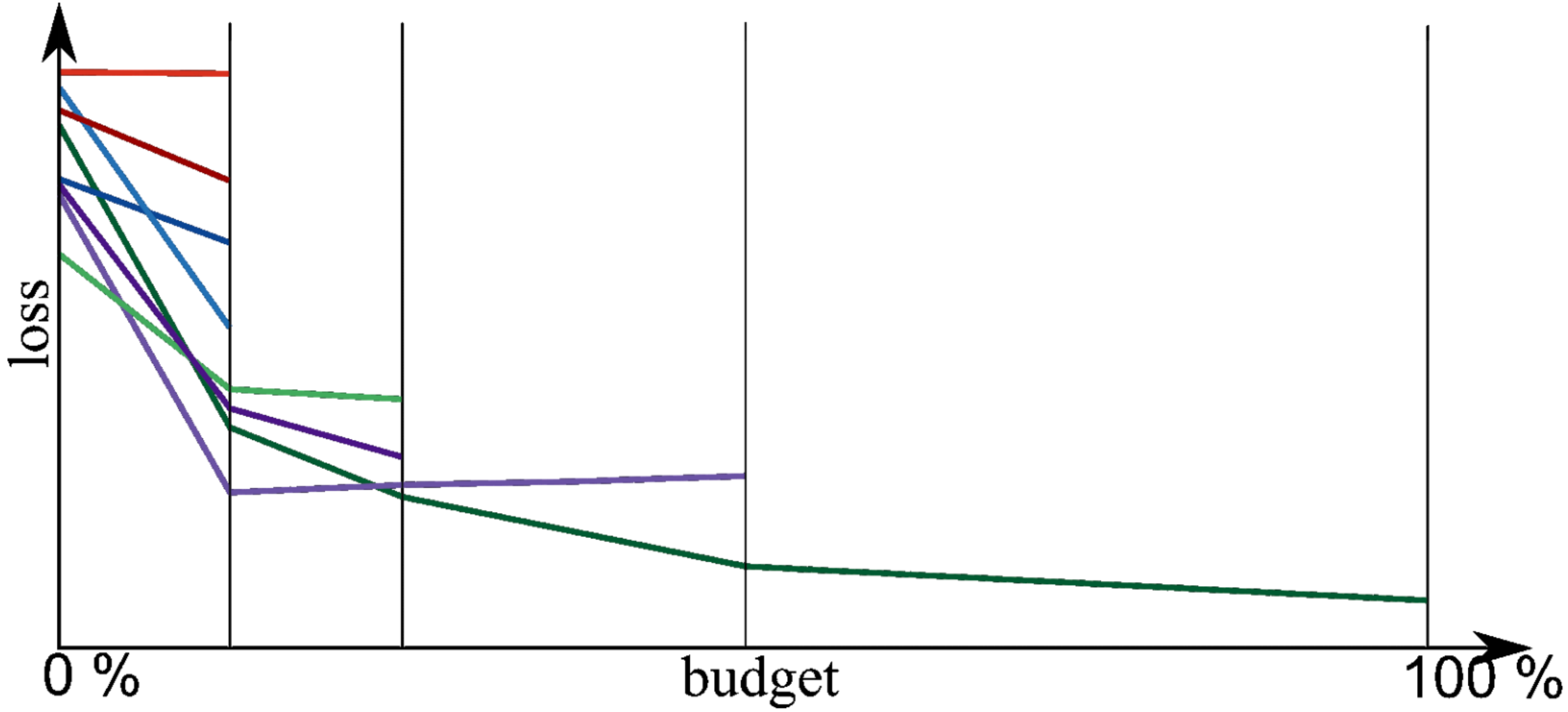
  https://github.com/kevinduh/sockeye-recipes3/tree/automl

Outline for Case Study

- Asynchronous Successive Halving (ASHA)
- Software design
- Use case

Recall: Successive Halving (SHA)

-- multi-armed bandit algorithm to perform early stopping



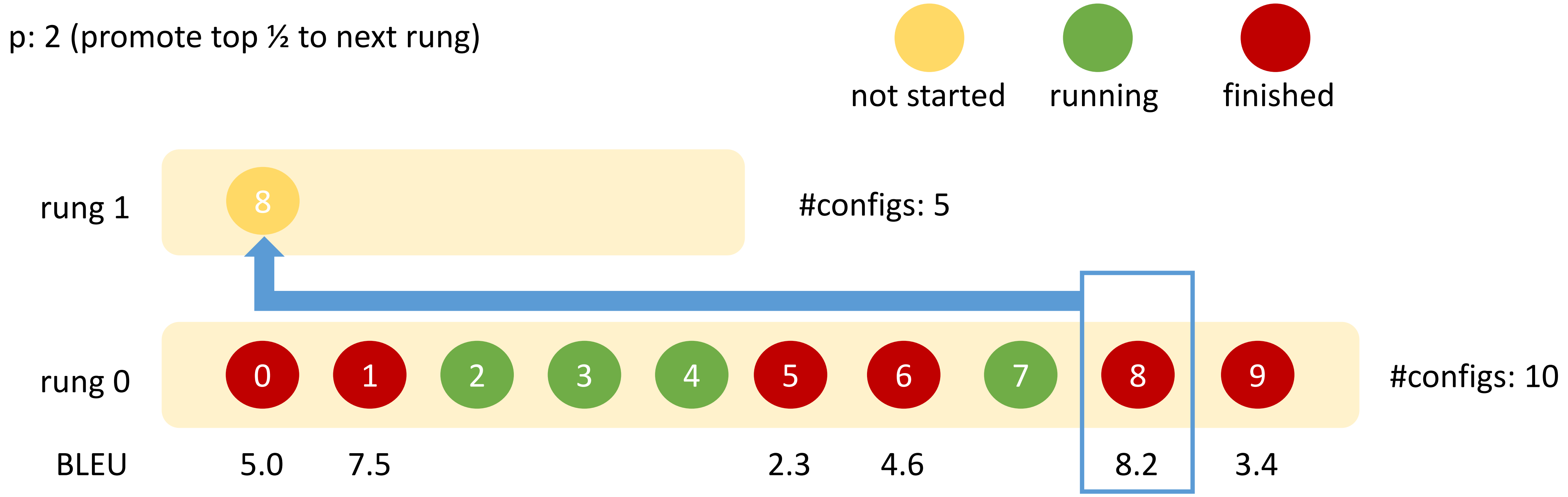
From : automl.org

Asynchronous Successive Halving (ASHA)

- In the sequential SHA, the algorithm waits for all configurations in a rung to complete before promoting configurations to next rung.
- ASHA removes the bottleneck created by synchronous promotions.
- It would promote a configuration to next rung when
  - There's an idle worker.
  - There's a configuration that is secured a position in the top 1/p of this rung.
- Parallelization with maximal GPU utilization
- References:
  - Li, et. al., A system for massively parallel hyperparameter tuning https://arxiv.org/pdf/1810.05934.pdf
  - https://blog.ml.cmu.edu/2018/12/12/massively-parallel-hyperparameter-optimization/

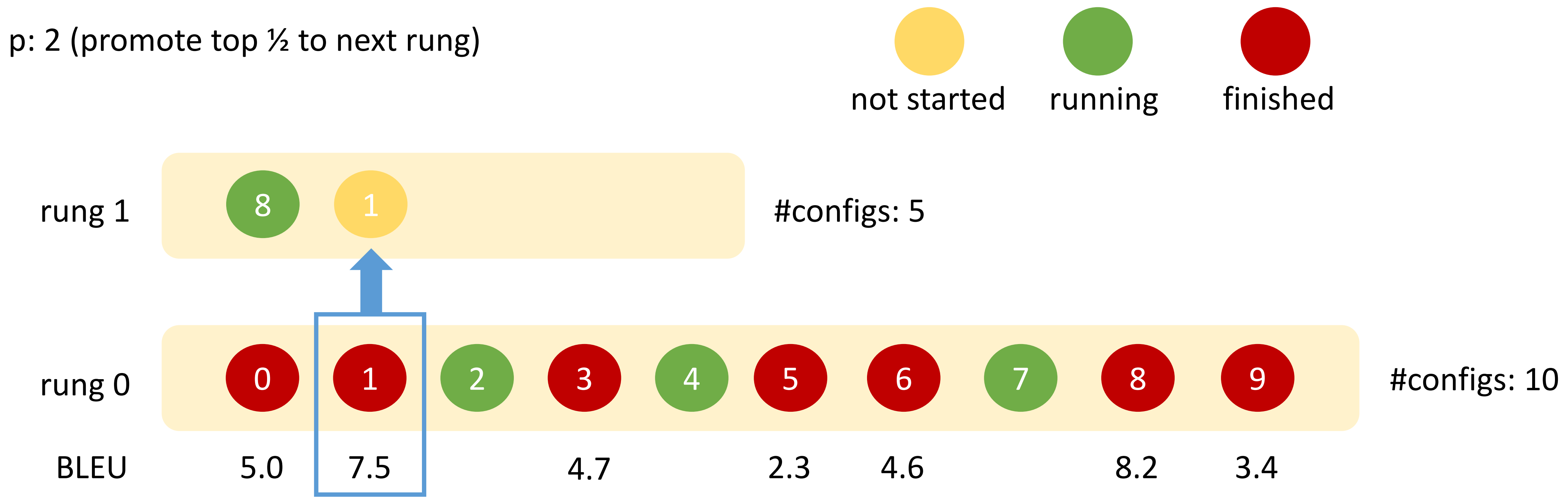
Asynchronous Successive Halving (ASHA)

- ASHA promotes a configuration to next rung when there's a configuration that is secured a position in the top 1/p of this rung.



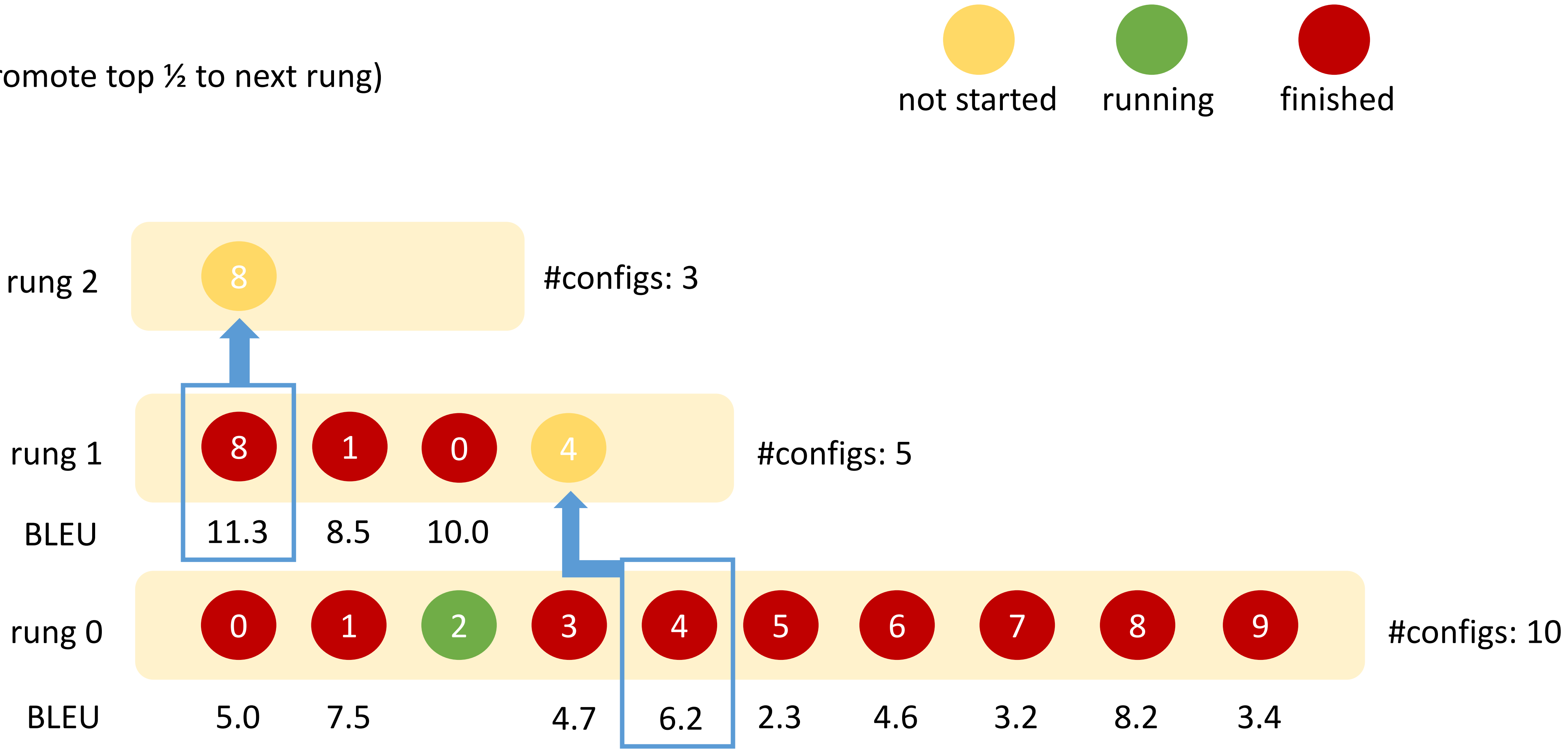
Asynchronous Successive Halving (ASHA)

- ASHA promotes a configuration to next rung when there's a configuration that is secured a position in the top 1/p of this rung.



Asynchronous Successive Halving (ASHA)

p: 2 (promote top ½ to next rung)



Pseudo-Code

```
Input: configurations configs, state checking time interval t,
          minimum training checkpoints r, checkpoints within each rung u,
          maximum training checkpoints R, reduction rate p, number of GPUs G

If runtime % t == 0 do
    For each config do
          state = check_state(config)
          react_to_state(config, state, r, R)
    end
    If ASHA is finished do
       Return
    end
    For each idle GPU do
          candidate = get_candidate(configs, p)
          promote(candidate)
          submit_train(candidate, GPU, u)
     end
end
```

Pseudo-Code

Input: configurations ***configs***, state checking time interval ***t***,
minimum training checkpoints ***r***, checkpoints within each rung ***u***,
maximum training checkpoints ***R***, reduction rate ***p***, number of GPUs ***G***

```
If runtime % t == 0 do
    For each config do
        state = check_state(config)
        react_to_state(config, state, r, R)
    end
    If ASHA is finished do
        Return
    end
    For each idle GPU do
        candidate = get_candidate(configs, p)
        promote(candidate)
        submit_train(candidate, GPU, u)
    end
end
```

At each time step, we check the state of each config, and submit jobs to idle GPUs

Pseudo-Code

Input: configurations ***configs***, state checking time interval ***t***,
minimum training checkpoints ***r***, checkpoints within each rung ***u***,
maximum training checkpoints ***R***, reduction rate ***p***, number of GPUs ***G***

```
If runtime % t == 0 do
    For each config do
        state = check_state(config)
        react_to_state(config, state, r, R)
    end
    If ASHA is finished do
        Return
    end
    For each idle GPU do
        candidate = get_candidate(configs, p)
        promote(candidate)
        submit_train(candidate, GPU, u)
    end
end
```

We check the state of each configurations, and react accordingly to different states

Pseudo-Code

Input: configurations ***configs**,* state checking time interval ***t**,*
minimum training checkpoints ***r**,* checkpoints within each rung ***u***,
maximum training checkpoints ***R***, reduction rate ***p***, number of GPUs ***G***

```
If runtime % t == 0 do
    For each config do
        state = check_state(config)
        react_to_state(config, state, r, R)
    end
    If ASHA is finished do
        Return
    end
    For each idle GPU do
        candidate = get_candidate(configs, p)
        promote(candidate)
        submit_train(candidate, GPU, u)
    end
end
```

Find config candidates and submit training jobs.

Pseudo-Code

Input: configurations ***configs***, state checking time interval ***t***,
minimum training checkpoints ***r***, checkpoints within each rung ***u***,
maximum training checkpoints ***R***, reduction rate ***p***, number of GPUs ***G***

If runtime % *t* == 0 **do**
  For each *config* **do**
    state = check_state(*config*)
    react_to_state(*config*, *state*, *r*, *R*)
  end
  If ASHA is finished **do**
    Return
  end
  For each idle GPU **do**
    candidate = get_candidate(*configs*, *p*)
    promote(*candidate*)
    submit_train(*candidate*, *GPU*, *u*)
  end
end

It is done by reading the train log.

```
for l in lines:
    if "Maximum number of not improved checkpoints" in l:
        return CONVERGED
    elif "CUDA error: all CUDA-capable devices are busy or unavailable" in l:
        return GPU_ERROR
    elif "CUDA out of memory" in l:
        return MEM_ERROR
    elif "OverflowError" in l:
        return MATH_ERROR
    elif "Best validation perplexity: inf" in l or "Train-ppl=nan" in l:
        return DIVERGED
    elif "Stale file handle" in l:
        return STORAGE_ERROR
if "Training finished" in lines[0]:
    return SUCCESS
return RUNNING
```

Pseudo-Code

Input: configurations ***configs***, state checking time interval ***t***,
minimum training checkpoints ***r***, checkpoints within each rung ***u***,
maximum training checkpoints ***R***, reduction rate ***p***, number of GPUs ***G***

```
If runtime % t == 0 do
    For each config do
        state = check_state(config)
        react_to_state(config, state, r, R)
    end
    If ASHA is finished do
        Return
    end
    For each idle GPU do
        candidate = get_candidate(configs, p)
        promote(candidate)
        submit_train(candidate, GPU, u)
    end
end
```

State	Reaction
RUNNING	N/A
SUCCESS / CONVERGED	Submit valid job or Collect evaluation results
GPU ERROR	Submit again
MEM ERROR / DIVERGED	Delete job and add it to blacklist

Pseudo-Code

Input: configurations ***configs***, state checking time interval ***t***,
minimum training checkpoints ***r***, checkpoints within each rung ***u***,
maximum training checkpoints ***R***, reduction rate ***p***, number of GPUs ***G***

```
If runtime % t == 0 do
    For each config do
        state = check_state(config)
        react_to_state(config, state, r, R)
    end
    If ASHA is finished do
        Return
    end
    For each idle GPU do
        candidate = get_candidate(configs, p)
        promote(candidate)
        submit_train(candidate, GPU, u)
    end
end
```

Get configs that are ready to move to next rung.
(ASHA: no need to wait till all the configs in current run to finish.)

Pseudo-Code

Input: configurations ***configs***, state checking time interval ***t***,
minimum training checkpoints ***r***, checkpoints within each rung ***u***,
maximum training checkpoints ***R***, reduction rate ***p***, number of GPUs ***G***

If runtime % *t* == 0 **do**
  For each *config* **do**
    state = check_state(*config*)
    react_to_state(*config*, *state*, *r*, *R*)
  end
  If ASHA is finished **do**
    Return
  end
  For each idle GPU **do**
    candidate = get_candidate(*configs*, *p*)
    promote(*candidate*)
    submit_train(*candidate*, *GPU*, *r*, *u*, *R*)
  end
end

Pick one from all the candidates.
Random search or Bayesian Optimization.

Pseudo-Code

Input: configurations ***configs***, state checking time interval ***t***,
minimum training checkpoints ***r***, checkpoints within each rung ***u***,
maximum training checkpoints ***R***, reduction rate ***p***, number of GPUs ***G***

```
If runtime % t == 0 do
    For each config do
        state = check_state(config)
        react_to_state(config, state, r, R)
    end
    If ASHA is finished do
        Return
    end
    For each idle GPU do
        candidate = get_candidate(configs, p)
        promote(candidate)
        submit_train(candidate, GPU, r, u, R)
    end
end
```

Submit a train job and let it run for
min(r, u*rung, R)-min(r, u*(rung-1), R) checkpoints

Implementation Challenges

- How to get the job state?

  We check the job log.
- How to automatically check the job state?

  We set up a timer running in a background thread.
- How to interact with the grid / GPU cluster?

  Besides job states, we also check GPU states.

  We debug carefully with possible errors.
- How to deal with failed jobs?

  We either resubmit it or delete it.

Example Run

```
(sockeye3) xzhang@test1:/exp/xzhang/sockeye-recipes3/automl$ sh submit_run_automl.sh
2022-09-07 19:41:02,733 Run ASHA with Arguments:
minimum number of chckpoints (r): 1
number of checkpoints per rung (u): 1
maximum checkpoints (R): 6
reduction rate (p): 2
number of GPUs (G): 4
2022-09-07 19:41:02,733 work directory: /exp/xzhang/sockeye-recipes3/egs/asha/space1/run1
2022-09-07 19:41:02,733 job log directory: /exp/xzhang/sockeye-recipes3/egs/asha/space1/run1/job_logs
2022-09-07 19:41:02,733 Single-objective optimizagtion: BLEU will be optimized.
config id to real id:  {0: 0, 1: 1, 2: 2, 3: 3, 4: 4, 5: 5, 6: 6, 7: 7, 8: 8, 9: 9}

num_avail_gpu  4
Obtaining the candidates .....
Rung states:  {0: {'finished': [], 'running': []}, 1: {'finished': [], 'running': []}, 2: {'finished':
ing': []}}
Config states:  {0: {'rung': -1, 'bleu': -1, 'gpu_time': -1, 'bleus': [], 'gpu_times': [], 'converged'
e': -1, 'bleus': [], 'gpu_times': [], 'converged': False}, 2: {'rung': -1, 'bleu': -1, 'gpu_time': -1,
lse}, 3: {'rung': -1, 'bleu': -1, 'gpu_time': -1, 'bleus': [], 'gpu_times': [], 'converged': False}, 4
eus': [], 'gpu_times': [], 'converged': False}, 5: {'rung': -1, 'bleu': -1, 'gpu_time': -1, 'bleus': [
rung': -1, 'bleu': -1, 'gpu_time': -1, 'bleus': [], 'gpu_times': [], 'converged': False}, 7: {'rung':
gpu_times': [], 'converged': False}, 8: {'rung': -1, 'bleu': -1, 'gpu_time': -1, 'bleus': [], 'gpu_tim
'bleu': -1, 'gpu_time': -1, 'bleus': [], 'gpu_times': [], 'converged': False}}
rung  0 candidates  {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
rung  1 candidates  set()
rung  2 candidates  set()
rung  3 candidates  set()
Picked candidate:  4
2022-09-07 19:41:02,825 qsub -l mem_free=200G,h_rt=10:00:00,num_proc=10,gpu=1 -q gpu.q -o /exp/xzhang/
/4_train.log.o -e /exp/xzhang/sockeye-recipes3/egs/asha/space1/run1/job_logs/4_train.log.e -N ashat4
pes3/egs/asha/space1/run1/hpms/4.hpm -e sockeye3
Your job 10022180 ("ashat4") has been submitted
```

arguments

Pick up a candidate

Submit a train job

Example Run

```
2022-09-07 19:52:10,840 Saved ASHA states to /exp/xzhang/sockeye-recipes3/egs/asha/space1/run1/ckpt.json
config id to real id:  {0: 0, 1: 1, 2: 2, 3: 3, 4: 4, 5: 5, 6: 6, 7: 7, 8: 8, 9: 9}
config 1 train_job_state:  SUCCESS val_job_state:  RUNNING train_gpu_state:  NOTEXIST val_gpu_state:  RUNNING
config 7 train_job_state:  SUCCESS val_job_state:  RUNNING train_gpu_state:  NOTEXIST val_gpu_state:  RUNNING
config 6 train_job_state:  RUNNING val_job_state:  NOTSTARTED train_gpu_state:  RUNNING val_gpu_state:  NOTEXIST
config 3 train_job_state:  RUNNING val_job_state:  NOTSTARTED train_gpu_state:  RUNNING val_gpu_state:  NOTEXIST

num_avail_gpu  0
2022-09-07 19:53:41,440 ————————————————————
Rung 0:
Finished Jobs    2         4         8         9
Ids              2         4         8         9
BLEU             1.7       2.1       1.3       3.1


2022-09-07 19:53:41,443 Saved ASHA states to /exp/xzhang/sockeye-recipes3/egs/asha/space1/run1/ckpt.json
```

Check
job state &
GPU state

Finished jobs

Example Run

```
2022-09-07 20:31:07,887 ASHA finished successfully!
2022-09-07 20:31:07,888 ────────────────────
Rung 0:
Finished Jobs   0       1       2       3       4       5       6       7       8       9
Ids             0       1       2       3       4       5       6       7       8       9
BLEU            0.7     2.3     1.7     0.7     2.1     1.0     3.4     1.8     1.3     3.1

────────────────────
Rung 1:
Finished Jobs   1       4       6       7       9
Ids             1       4       6       7       9
BLEU            3.5     3.2     5.1     1.3     5.1

────────────────────
Rung 2:
Finished Jobs   6       9
Ids             6       9
BLEU            7.3     7.2

────────────────────
Rung 3:
Finished Jobs   6
Ids             6
BLEU            8.3

2022-09-07 20:31:07,889 Best config: 6  BLEU: 8.3
```

ASHA finished successfully.
The best config is 6 with 8.3 BLEU score.

Roadmap

1. Motivation for AutoML
2. Hyperparameter Optimization (HPO)
3. Neural Architecture Search (NAS)
4. Extension to Multiple Objectives
5. Evaluation
6. Toolkits
7. Survey of HPO/NAS in NLP

HPO on input text features

- Bayesian Optimization on hyperparameters related to text preprocessing.
- Logistic Regression classifier
- Positive results on sentiment and topic classification
  - e.g. Stanford sentiment treebank, movie reviews, 20 Newsgoups

Hyperparameter	**Values**
n_{min}	$\{1, 2, 3\}$
n_{max}	$\{n_{min}, \ldots, 3\}$
weighting scheme	{tf, tf-idf, binary}
remove stop words?	{True, False}
regularization	$\{\ell_1, \ell_2\}$
regularization strength	$[10^{-5}, 10^{5}]$
convergence tolerance	$[10^{-5}, 10^{-3}]$

Table 1: The set of hyperparameters considered in our experiments. The top half are hyperparameters related to text representation, while the bottom half are logistic regression hyperparameters, which also interact with the chosen representation.

HPO on subsets of training data

- Each HPO blackbox evaluation can be expensive. Reduce the time cost by running Bayesian Optimization on subsets of training data.

	Hyper-parameters
SVM	bias, cost parameter, and regularization parameter
Boosted regression trees	feature sampling rate, data sampling rate, learning rate, # trees, # leaves, and minimum # instance per leaf

Table 1: Hyper-parameters used in SVM and boosted regression trees.

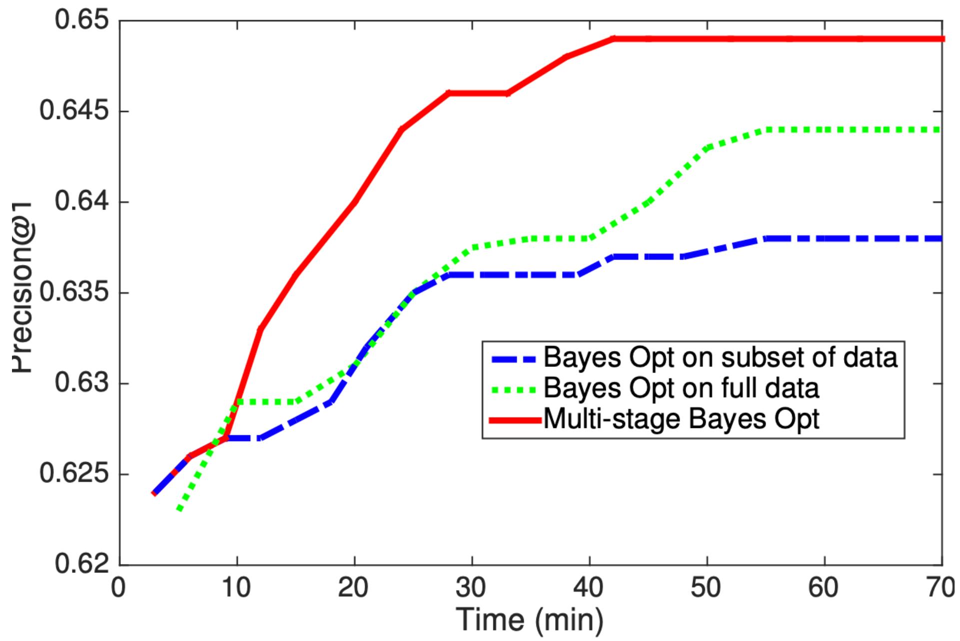


Figure 1: QA task: test accuracy vs tuning time.

From: Beck, et. al. Speed-Constrained Tuning for Statistical Machine Translation Using Bayesian Optimization. NAACL 2016

HPO for Statistical MT speed-accuracy

- Statistical MT decoder hyperparameters:
  - distortion limit
  - stack size (in histogram pruning during decoding)
  - number of translation options per source phrase
- These affect decoding speed & accuracy, but are hard to optimize
- Apply Bayesian Optimization

	Tuning		Test		θ		
	BLEU	speed	BLEU	speed	d	s	n
MERT	44.3	–	42.5	51	10	1K	500
BO-S	43.8	2.0K	41.9	1.9K	10	25	100
MERT-flat	43.8	2.0K	41.4	1.9K	10	25	100
MERT-opt	**44.3**	**2.0K**	**42.4**	**1.9K**	10	25	100

Table 4: Chinese-to-English results of re-running MERT using parameters that satisfy the 2K wpm speed constraint.

From: Zhang, et. al. KGTuner: Efficient Hyper-parameter Search for Knowledge Graph Learning. ACL 2022

HPO for Knowledge Graph problems

- Knowledge graph embedding techniques encodes nodes or edges as vectors. Useful for link prediction, etc.
- Training choices optimized by HPO. (First on small graph, then large)

Table 1: The HP space. Conditioned HPs are in parenthesize. "adv." and "reg." are short for "adversarial" and "regularization", respectively. Please refer to the Appendix A for more details.

component	name	type	range
negative sampling	# negative samples	cat	{32, 128, 512, 2048, `1VsAll`, `kVsAll`}
loss function	loss function	cat	{MR, BCE_(mean, sum, adv), CE}
	gamma (MR)	float	[1, 24]
	adv. weight (BCE_adv)	float	[0.5, 2.0]
regularization	regularizer	cat	{FRO, NUC, DURA, None}
	reg. weight (not None)	float	$[10^{-12}, 10^{2}]$
	dropout rate	float	$[0, 0.5]$
optimization	optimizer	cat	{Adam, Adagrad, SGD}
	learning rate	float	$[10^{-5}, 10^{0}]$
	initializer	cat	{uniform, normal, xavier_uniform, xavier_norm}
	batch size	int	{128, 256, 512, 1024}
	dimension size	int	{100, 200, 500, 1000, 2000}
	inverse relation	bool	{True, False}

Table 3: MRR of models with HPs tuned in different methods. The bold numbers mean the best performance of the same model.

source	models	WN18RR	FB15k-237
original	TransE	0.226	0.296
	ComplEx	0.440	0.247
	ConvE	0.430	0.325
LibKGE	TransE	0.228	0.313
	ComplEx	0.475	0.348
	ConvE	**0.442**	**0.339**
KGTuner	TransE	**0.233**	**0.327**
	ComplEx	**0.484**	**0.352**
	ConvE	0.437	0.335

From: Liu & Wang. An Empirical Study on Hyperparameter Optimization for Fine-Tuning Pre-trained Language Models. ACL 2021.

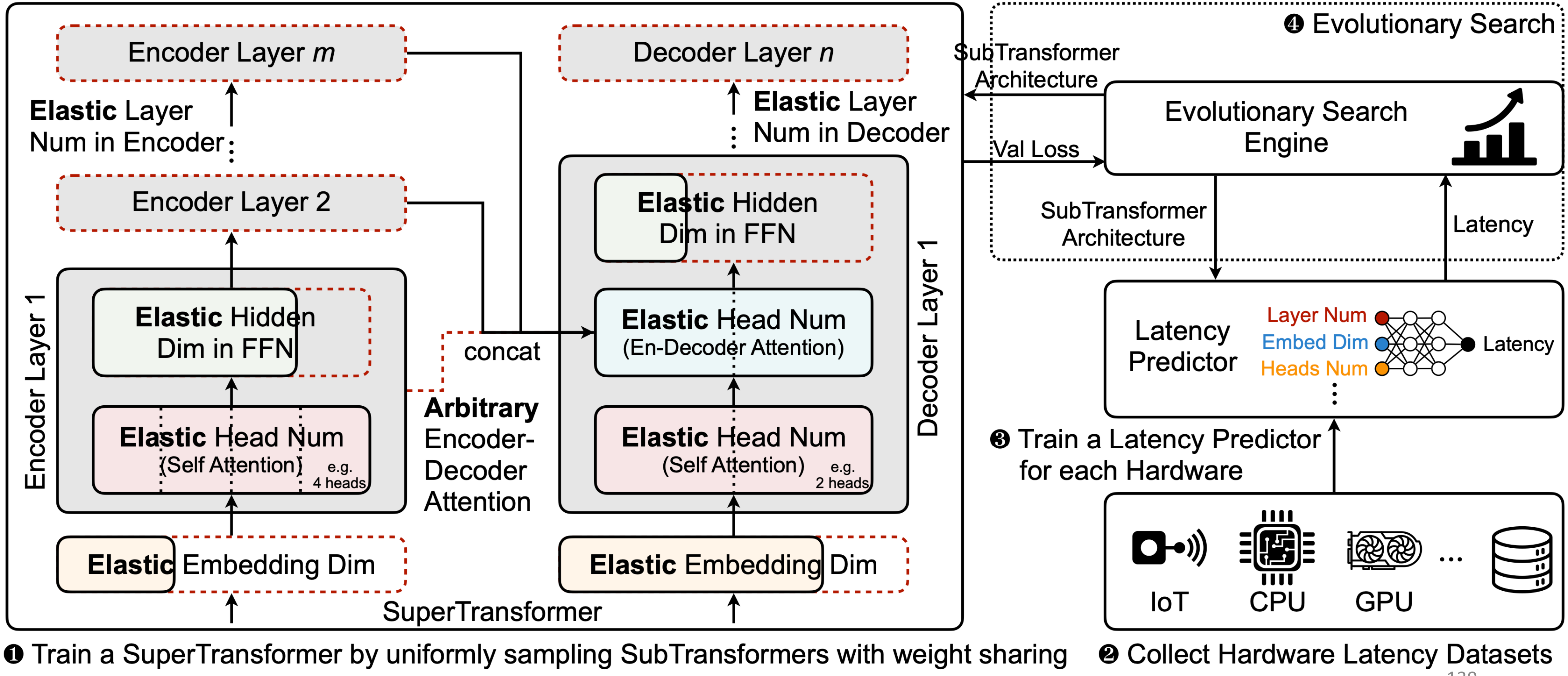
HPO for fine-tuning hyperparameters

- Compares grid search, random search, ASHA, ASHA with Bayes Opt
- Given limited budget, results are mixed. Setting search space per task is important.
- Refer to paper for details.

Hyperparameter	Electra-grid	Electra-HPO
learning rate	{3e-5,1e-4,1.5e-4}	log((2.99e-5,1.51e-4))
warmup ratio	0.1	(0, 0.2)
attention dropout	0.1	(0, 0.2)
hidden dropout	0.1	(0, 0.2)
weight decay	0	(0, 0.3)
batch size	32	{16, 32, 64}
epochs	10 for RTE/STS-B, 3 for other	10 for RTE/STS-B, 3 for other

	WNLI	**RTE**	**MRPC**	**CoLA**	**STS-B**
Electra-base, validation					
grid	56.3	**84.1**	92.3/89.2	67.2	**91.5/91.4**
RS	56.8	82.2	**93.0/90.4**	68.8	90.1/90.2
RS+ASHA	57.2	80.3	93.0/90.3	67.9	91.4/91.3
BO+ASHA	**58.2**	82.6	**93.1/90.4**	**69.4**	91.5/91.3
Electra-base, test					
grid	**65.1**	**76.8**	**91.1/87.9**	58.5	**89.7/89.2**
RS	64.4	75.6	90.7/87.5	63.0	88.0/87.6
RS+ASHA	62.6	74.1	90.6/87.3	61.2	89.5/89.1
BO+ASHA	61.6	75.1	90.7/87.4	**64.1**	89.7/89.1

Once-for-All NAS applied to Transformers



From: Hu, et. al. RankNAS: Efficient Neural Architecture Search by Pairwise Ranking. EMNLP 2021

Improving training time for Once-for-All Transformers (HAT, previous slide)

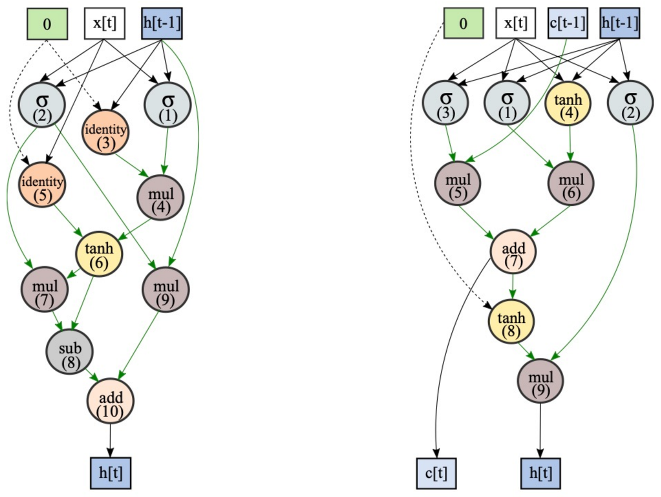
Hardware	Task	Method	Latency (ms)	#Params	FLOPs (G)	BLEU	Search Cost (hours)
Intel Xeon Silver 4114 CPU	WMT	Transformer	1031.4	213.0M	12.7	28.4	-
		HAT	396.8	**67.9M**	4.2	28.5	335.1
		RankNAS	**384.2**	68.1M	**4.0**	**28.6**	**31.8**
	IWSLT	Transformer	353.5	34.9M	1.6	34.4	-
		HAT	**190.5**	**27.9M**	**1.4**	34.5	31.7
		RankNAS	197.4	29.6M	1.5	**34.6**	**7.2**
NVIDIA GTX 1080Ti GPU	WMT	Transformer	249.6	213.0M	12.7	28.4	-
		HAT	214.8	66.2M	4.1	**28.5**	302.1
		RankNAS	**201.7**	**62.1M**	**3.9**	28.4	**30.2**
	IWSLT	Transformer	200.9	34.9M	1.6	34.4	-
		HAT	159.4	**33.9M**	1.6	34.7	24.5
		RankNAS	**148.2**	35.4M	**1.4**	**34.7**	**5.8**

Table 1: Comparisons of latency, model size, FLOPs, BLEU, and the overall search cost on machine translation tasks for the standard Transformer, HAT, and discovered architectures by our method. We mark the best results in bold for all metrics. Search costs are measured on a single RTX 2080Ti GPU.

From: Pasunuru & Bansal. FENAS: Flexible and Expressive Neural Architecture Search. EMNLP Findings 2020.

NAS on Text Classification (sentence pair tasks)

- 2-layer bidirectional LSTM-RNN + MLP classifier.
  - LSTM hidden size=1500. Word embedding dim=300. MLP hidden=300



(a) GRU cell (b) LSTM cell

Figure 2: GRU/LSTM represented in FENAS space.

Architecture	CoLA	SST-2	MRPC	QQP	STS-B	MNLI	QNLI	RTE	WNLI	AVG
LSTM	17.1	86.9	71.0/78.9	83.2/62.7	67.8/65.6	64.9/65.8	77.4	52.1	65.1	64.3
ENAS-RL	14.7	84.1	74.5/82.6	83.8/63.0	72.6/70.7	66.0/66.6	78.5	51.0	65.1	64.8
ENAS-RS	16.7	85.6	73.7/81.6	81.9/61.5	72.5/70.4	66.9/67.5	78.8	53.1	65.1	65.3
FENAS	16.4	86.6	71.0/78.9	84.9/63.7	73.2/71.0	66.6/66.0	79.1	52.7	65.1	65.6

Table 3: Results on GLUE task test sets, obtained from `https://gluebenchmark.com/`.

More nuanced results

From: MacLaughlin, et. al. Evaluating the Effectiveness of Efficient Neural Architecture Search for Sentence-Pair Tasks. Workshop on Insights from Negative Results in NLP, 2020.

- NAS outperforms LSTMs and random search only some (dataset, embedding, model) configurations

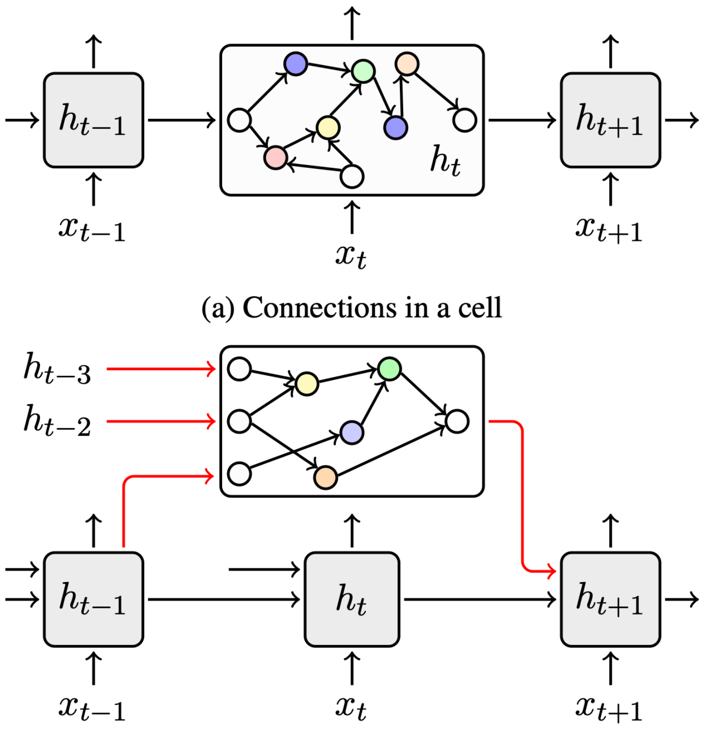
	Author	Embedding	Model	RNN	Dev Performance SICK-R	Dev Performance MRPC	Dev Performance STS-B	Test Performance SICK-R	Test Performance MRPC	Test Performance STS-B
1.	Devlin et al. (2019)	BERT	fine-tuned	–	–	–	–	**88.7**	**84.8**	**87.1**
2.	Peters et al. (2019)	BERT	ESIM	L / L	–	–	–	86.4	78.1	82.9
3.	Ours	BERT	ESIM	L / L	88.9	**88.0**	88.0	87.0	80.2	82.0
4.	Ours	BERT	ESIM	E / E	88.6	87.0	**88.5**	86.8	80.8	82.2
5.	Ours	BERT	ESIM	E / L	**89.3**	87.5	**88.5**	**87.4**	**81.0**	**83.0**
6.	Ours	BERT	ESIM	L / E	88.0	87.0	88.2	86.5	79.8	81.8
7.	Ours	BERT	BLM	L	87.4	88.0	88.1	84.8	80.4	80.1
8.	Ours	BERT	BLM	E	**87.8**	**88.5**	**88.7**	**85.5**	**82.8**	**83.3**
9.	Ours	Glove	ESIM	L / L	88.6	**79.9**	**83.3**	**86.1**	**73.7**	75.5
10.	Ours	Glove	ESIM	E / E	88.2	76.0	**83.3**	85.1	69.0	75.3
11.	Ours	Glove	ESIM	E / L	**88.7**	77.2	83.0	85.7	71.0	72.7
12.	Ours	Glove	ESIM	L / E	88.2	78.2	83.0	85.2	72.5	**76.0**
13.	Ours	Glove	BLM	L	86.3	**78.4**	79.7	82.5	71.8	73.0
14.	Ours	Glove	BLM	E	**87.0**	**78.4**	**81.6**	**84.1**	**73.4**	**74.8**

Table 1: Dev & Test set performances for LSTM and ENAS-RNN based models. Following Peters et al. (2019), we report pearson correlation for SICK-R and STS-B and accuracy for MRPC. In the RNN collumn, “E” stands for ENAS-RNN and “L” stands for LSTM. For ESIM there can be different of cells in different layers, e.g. E / L stands for ENAS-RNN in the 1st layer and LSTM in the 2nd layer.

From: Li, et. al. Learning Architectures from an Extended Search Space for Language Modeling. ACL 2020.

NAS on LM task, transfer to NER task

- Search both intra- and inter-cell architectures



(a) Connections in a cell

(b) Connections among cells

Figure 1: Examples of intra and inter-cell architectures.

- Search on WikiText-103, then fine-tune for NER

Models	F1
LSTM-CRF (Lample et al., 2016)	90.94
LSTM-CRF + ELMo (Peters et al., 2018)	92.22
LSTM-CRF + Flair (Akbik et al., 2019)	93.18
GCDT + $\text{BERT}_{\text{LARGE}}$ (Liu et al., 2019b)	93.47
CNN Large + ELMo (Baevski et al., 2019)	**93.50**
DARTS + Flair (Jiang et al., 2019)	93.13
I-DARTS + Flair (Jiang et al., 2019)	93.47
ESS	91.78
ESS + Flair	**93.62**

Table 4: F1 scores on CoNLL-2003 NER task. Bi-LSTM

Address memory limitations in NAS methods

- NAS methods like DARTS stores intermediate outputs from operations; out-of-memory with e.g. >400 dimension hidden size
- Reversible networks for reduced memory → larger search space

Model	**Pooling**	**Search s Layers**	**Model Size**	**BLEU**
Transformer	-	-	61.1M	27.7
ET	-	-	64.1M	28.2
Sampling	max	2	60.1M	18.7
Sampling	avg	2	61.6M	16.8
DARTSformer	max	1	64.5M	27.9
DARTSformer	max	2	65.2M	**28.4**
DARTSformer	avg	1	66.0M	28.3
DARTSformer	avg	2	63.4M	28.3

Table 1: BLEU scores of various search setups on WMT'14 En-De test set. ET is the Evolved Transformer (So et al., 2019). We use a 2-split encoder and a 3-split decoder.

Additional related work: HPO/NAS for NLP

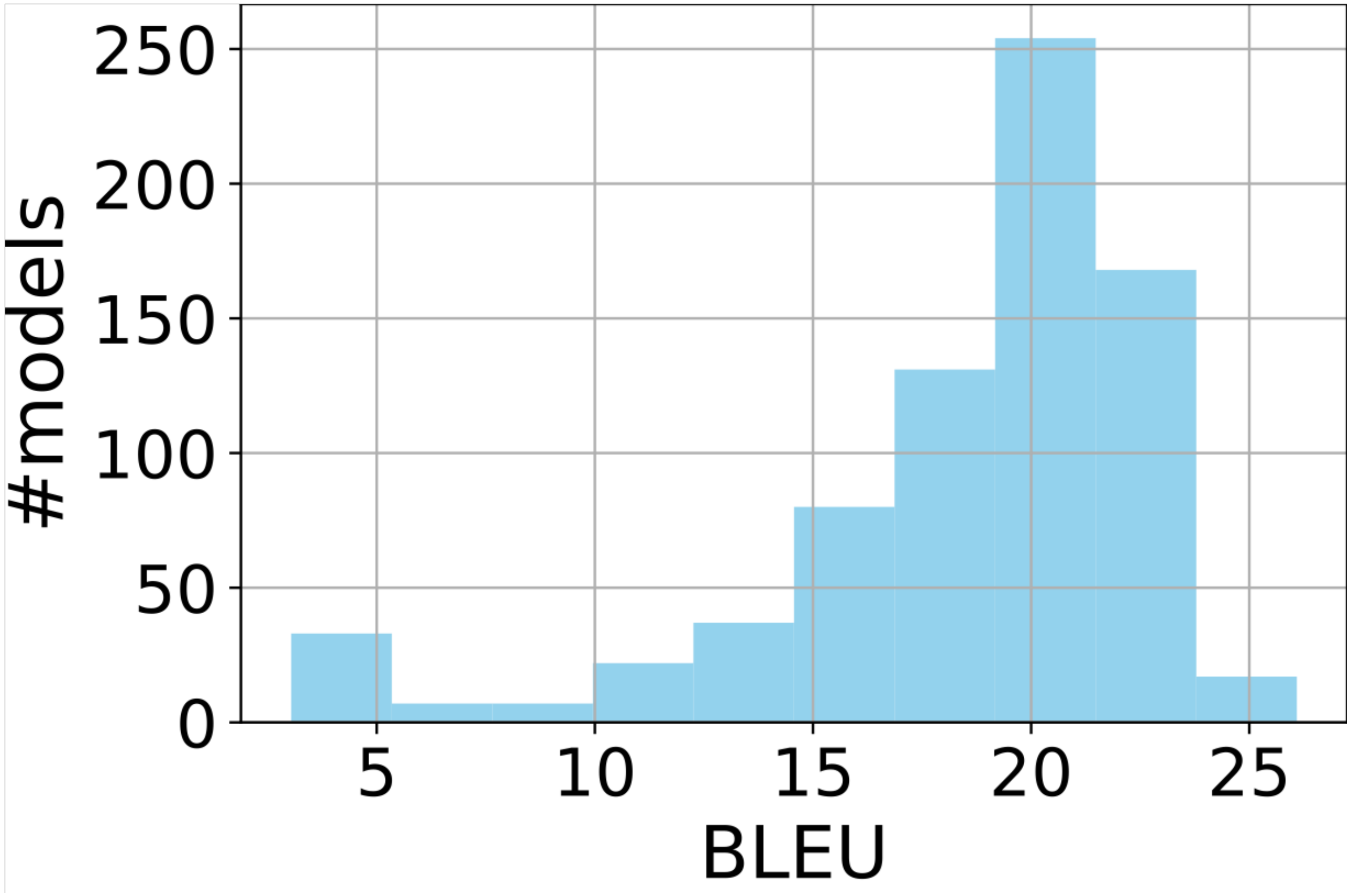
- Dodge, et. al. Show Your Work: Improved Reporting of Experimental Results. EMNLP 2019.
- Chen, et. al. AdaBERT: Task-Adaptive BERT Compression with Differentiable Neural Architecture Search. IJCAI 2020.
- Dong, et. al. EfficientBERT: Progressively Searching Multilayer Perceptron via Warm-up Knowledge Distillation. EMNLP Findings 2021
- Xu, et. al. NAS-BERT: Task-Agnostic and Adaptive-Size BERT Compression with Neural Architecture Search. KDD 2021.
- Puvis de Chavannes, et. al. Hyperparameter Power Impact in Transformer Language Model Training. Workshop on Simple & Efficient NLP 2021.

- Let us know if you know of other interesting work!

Review

1. Motivation for AutoML
2. Hyperparameter Optimization (HPO)
3. Neural Architecture Search (NAS)
4. Extension to Multiple Objectives
5. Evaluation
6. Toolkits
7. Survey of HPO/NAS in NLP

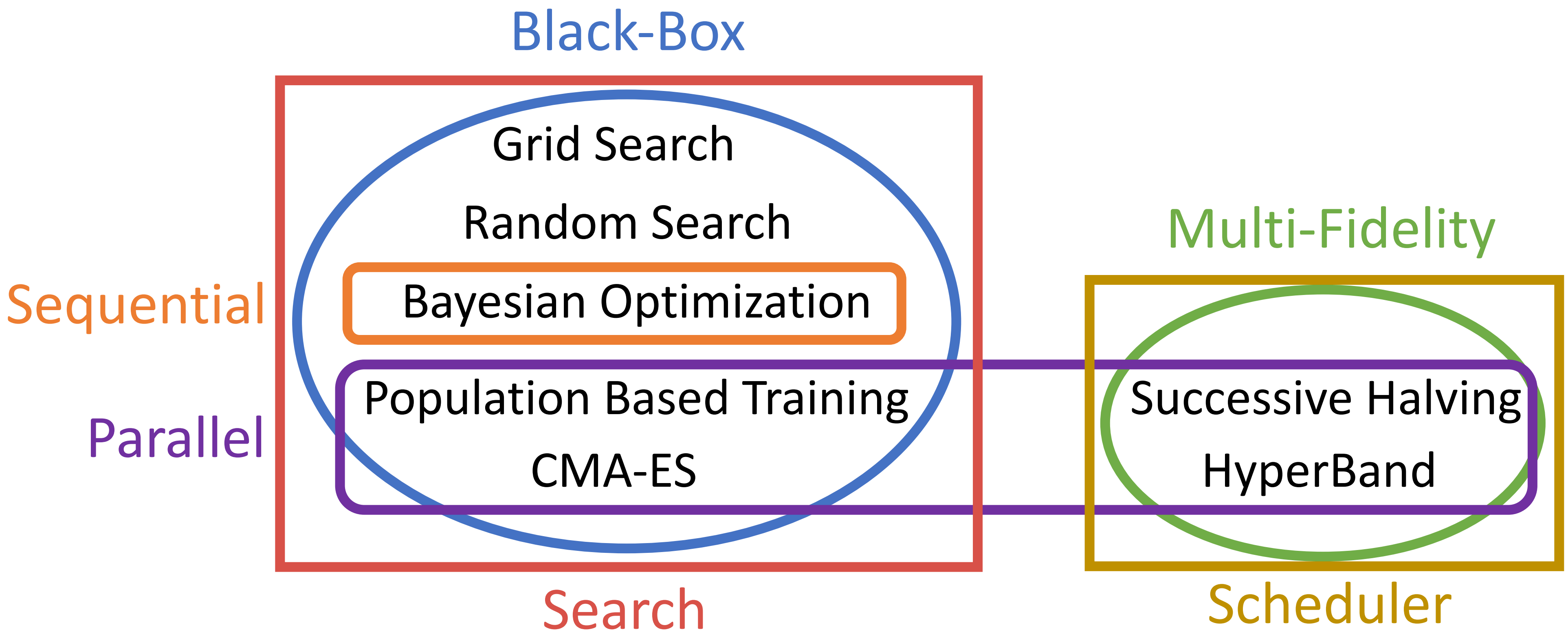
It's important to tune hyperparameters!



Histogram of BLEU scores for 700+ Swahili-English Neural Machine Translation (NMT) models

Note the large variance!

Hyperparameter Optimization (HPO)



Neural Architecture Search (NAS)

We discussed:

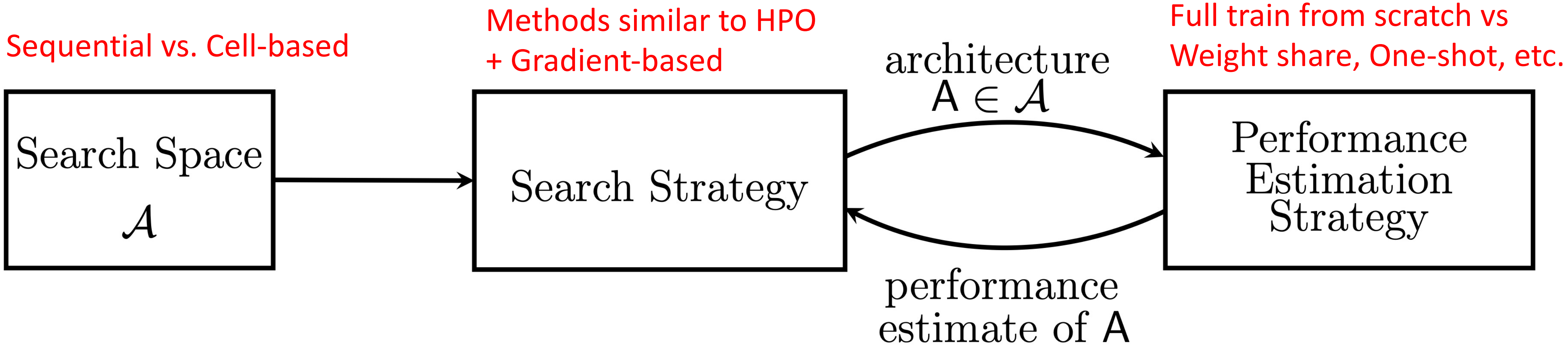
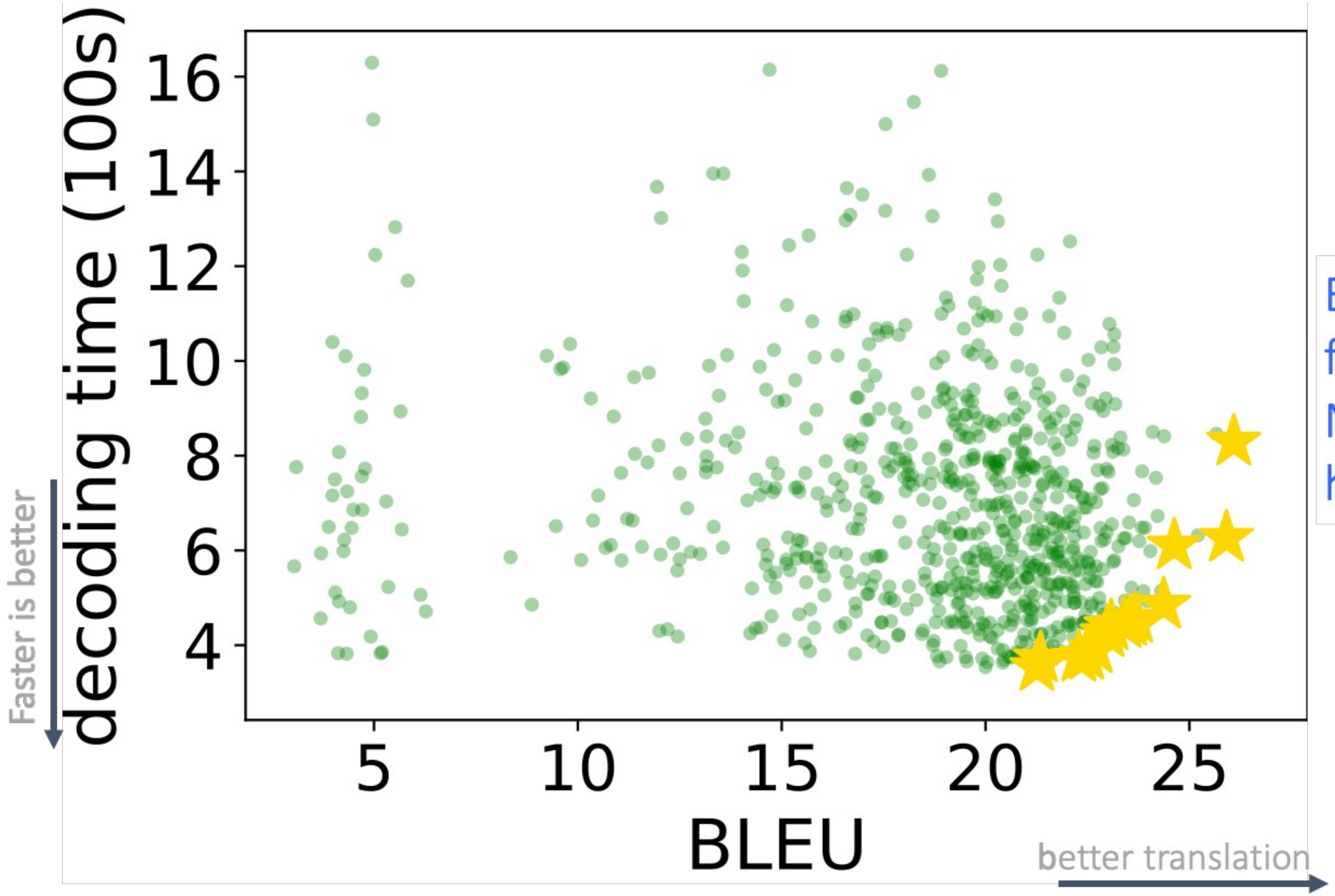


Figure 1: Abstract illustration of Neural Architecture Search methods. A search strategy selects an architecture A from a predefined search space $\mathcal{A}$. The architecture is passed to a performance estimation strategy, which returns the estimated performance of A to the search strategy.

From: Elsken, Metzen, Hutter. Neural Architecture Search, A Survey, JMLR 2019

When deploying models, we care about multiple objectives. But it's complex.



BLEU vs Time Scatterplot for **700+** Swahili-English NMT models: unclear how to get best tradeoff

Evaluation is hard, so Tabular Benchmark for NMT (Zhang & Duh, TACL2020)

Hyperparameter Type	Possible Values
# BPE Subword Units	1k, 2k, 4k, 8k, 16k, 32k, 50k
# Transformer Layers	1, 2, 4, 6
Word embedding	256, 512, 1024
# Hidden Units	1024, 2048
# Attention Heads	8, 16
Initial Learning Rate for ADAM	3×10^{-4}, 6×10^{-4}, 10×10^{-4}

Total: 2245 Transformer models, trained on ~1550 GPU days; record BLEU, train/test time, etc.
https://github.com/Este1le/hpo_nmt

Dataset	Domain	#models
zh-en	TED	118
ru-en	TED	176
ja-en	WMT	150
en-ja	WMT	168
sw-en	MATERIAL	767
so-en	MATERIAL	605

Use existing AutoML toolkits or Implement your own?

- Choice 1:

  Take an existing AutoML toolkit, and reimplement your training pipeline.

- Choice 2:

  Already have a training pipeline, e.g. Amazon Sockeye for MT, add an AutoML wrapper on top of it.

  It's worth implementing AutoML from scratch in this case.

Survey of HPO/NAS in NLP

- We surveyed a few papers with promising results in LM, MT, NER, sentence pair classification
- Compared to computer vision & machine learning, little work in NLP
- We think there are many research opportunities in NLP, especially multi-objective HPO/NAS of large models.

Questions or Comments?