Lecture 7: Automated Machine Learning

CS 256: Systems and Machine Learning

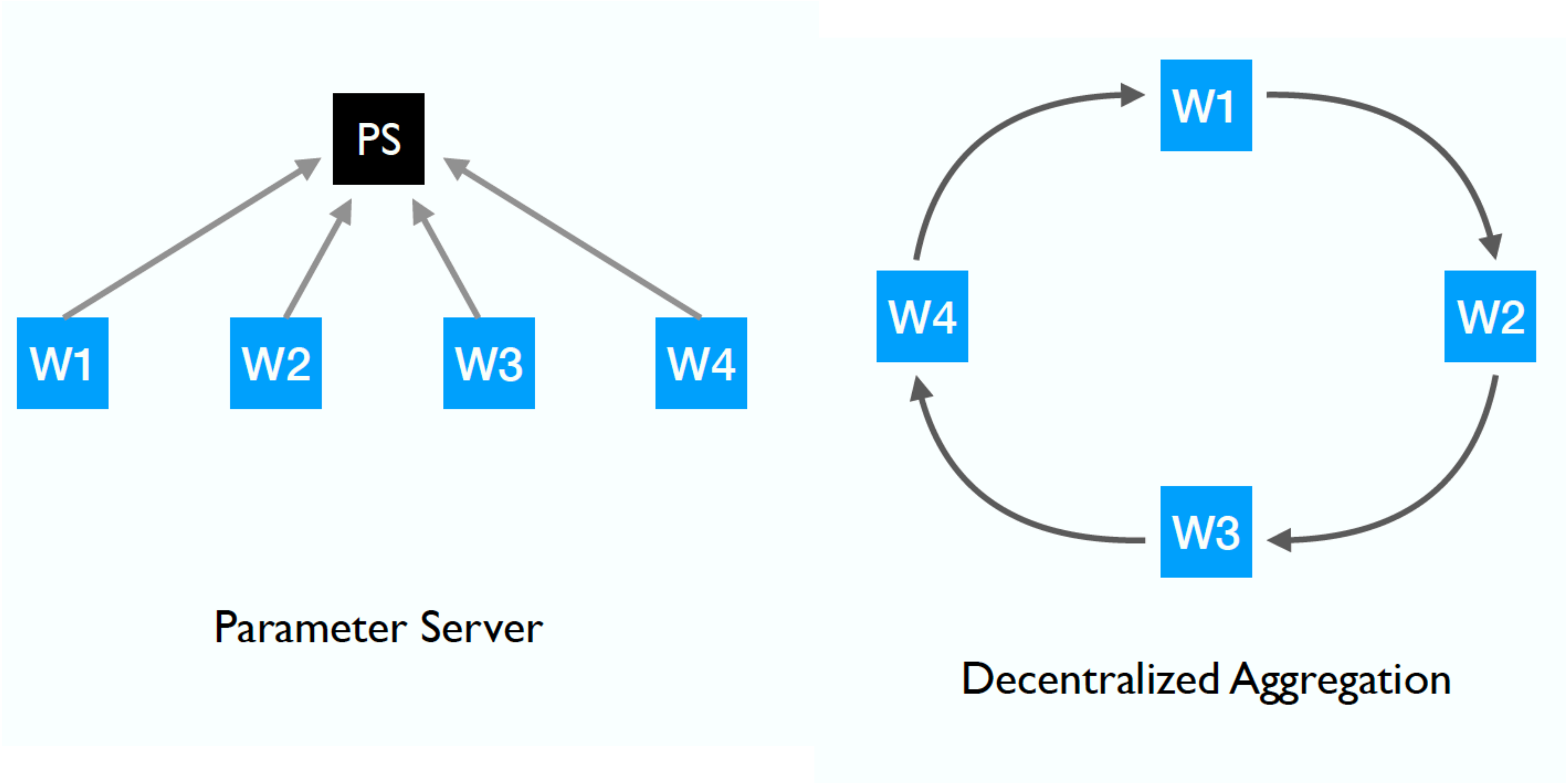
Sangeetha Abdu Jyothi

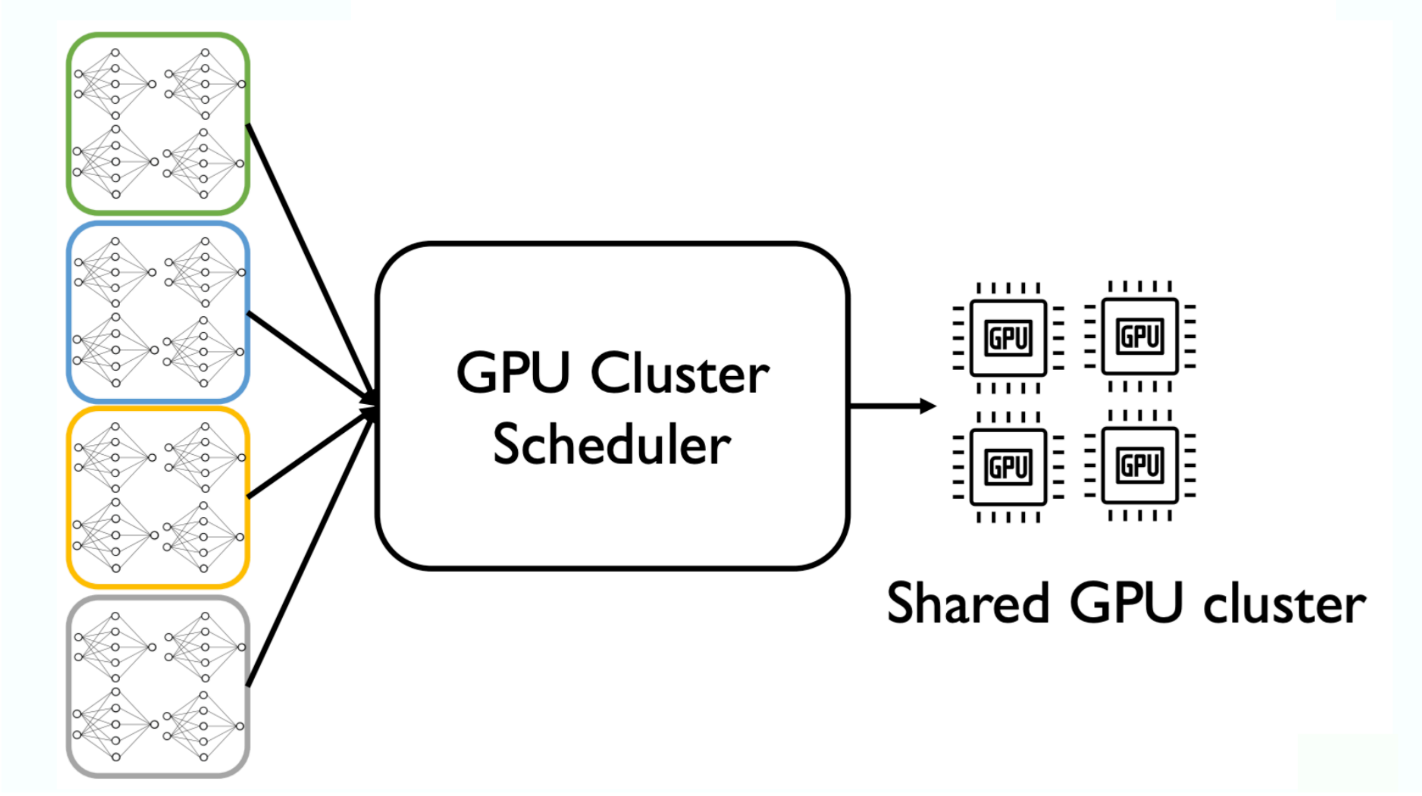
UCIrvine

Parts of this lecture were adapted from talks from CSE 291D/234 at UCSD

Quick Recap


Deep Learning Frameworks
TensorFlow
PyTorch
mxnet
Microsoft Cognitive Toolkit
Caffe2
Deep Learning Compilers
tvm
nGraph
GLOW
MLIR
Hardware


PS
W1
W2
W3
W4
Parameter Server
W1
W2
W3
W4
Decentralized Aggregation


GPU Cluster Scheduler
GPU
GPU
GPU
GPU
Shared GPU cluster

Automated Machine Learning or AutoML

AutoML: Automating end-to-end process of applying ML

DataRobot

Amazon SageMaker

H_2O AutoML

Google's AutoML

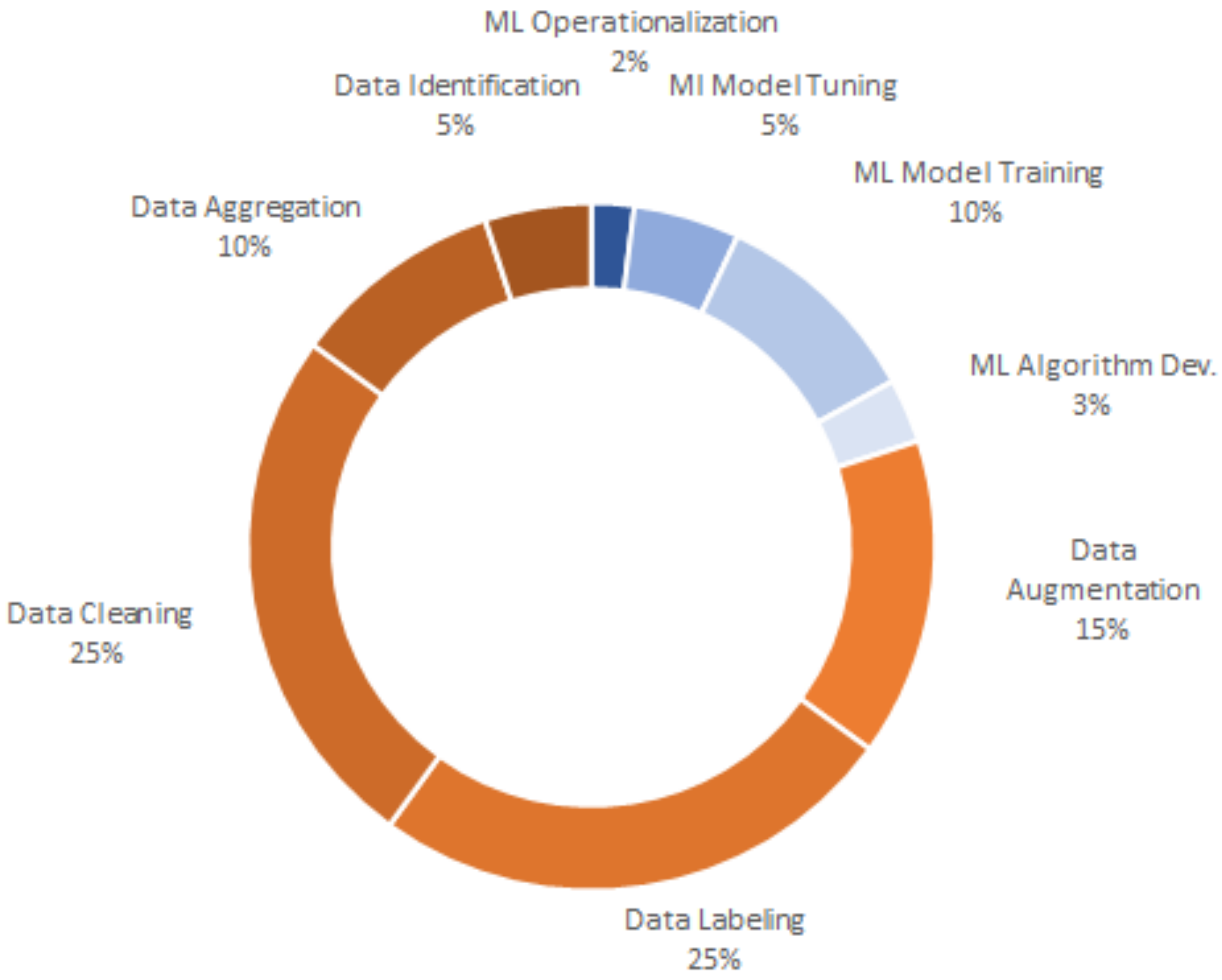
H_2O.ai DRIVERLESS AI

TPOT

Auto-Sklearn

AutoKeras

Why is Automated ML Necessary?



Source: Cognylitica; Factordaily

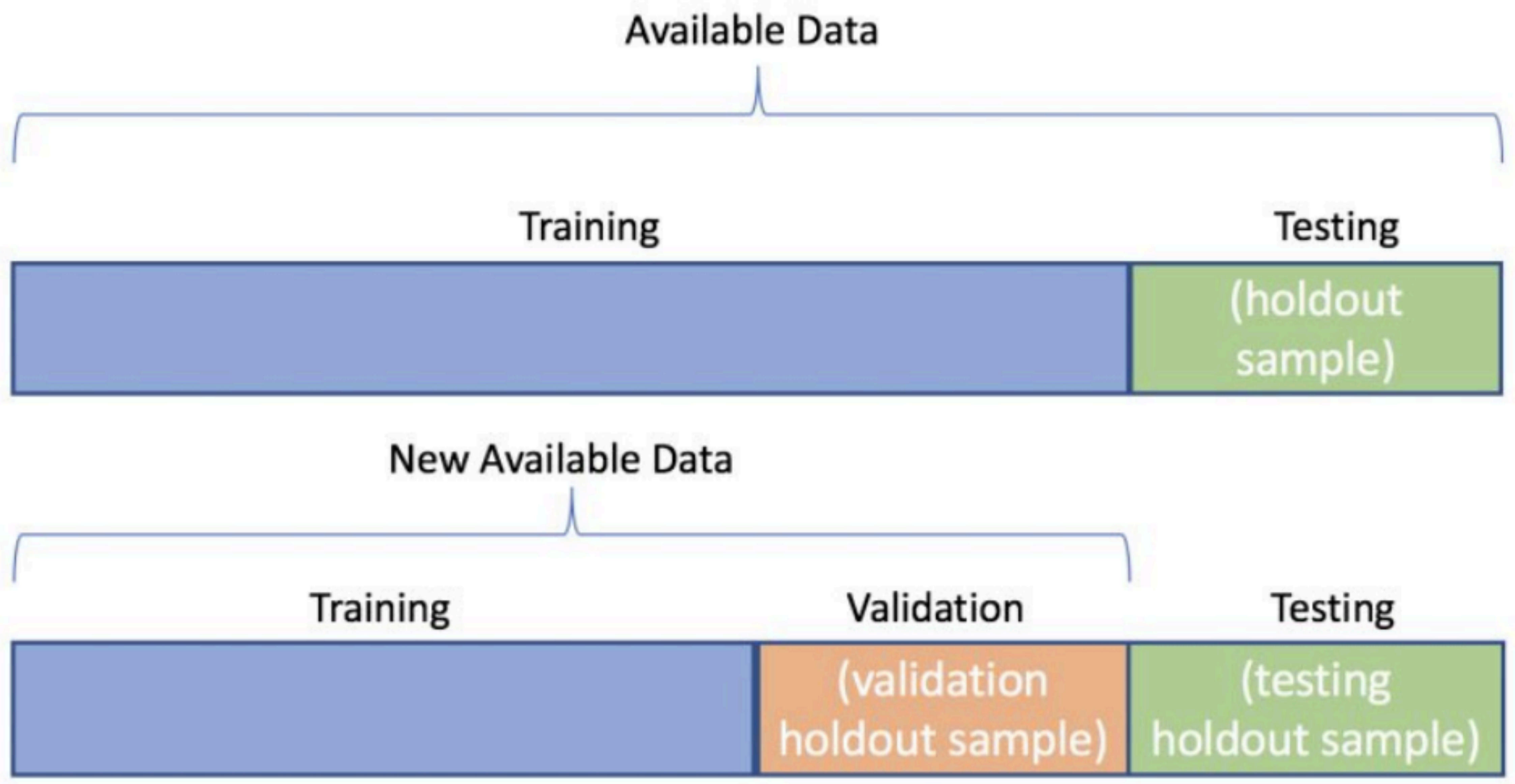
Algorithm development is only 3% of the total time!

AutoML Key Components

- Feature Engineering
- Algorithm/Architecture Selection
- Hyperparameter Tuning

Unpredictability of Model Selection

- The exact raises/drops in errors on given training task and sample are not predictable
- Need empirical comparisons of configurations on data
- Train-validation-test splits



Feature Engineering Systems: Brief Overview

- Feature Engineering: Converting raw data into a feature vector representation for ML training/inference
- Automated Feature Engineering Systems are less popular than hyperparameter tuning and AS
- Key issues addressed
    - Usability: Higher level specification of feature engineering operations
    - Efficiency: Automated systems-level optimization
- Challenges
    - Heterogeneity: Difficult to build a one-size-fits-all tool
    - Turing-complete code: Difficult to automatically optimize
- Some Tools: FeatureTools, AutoFeat, TsFresh, Cognito, OneBM, ExploreKit, PyFeat

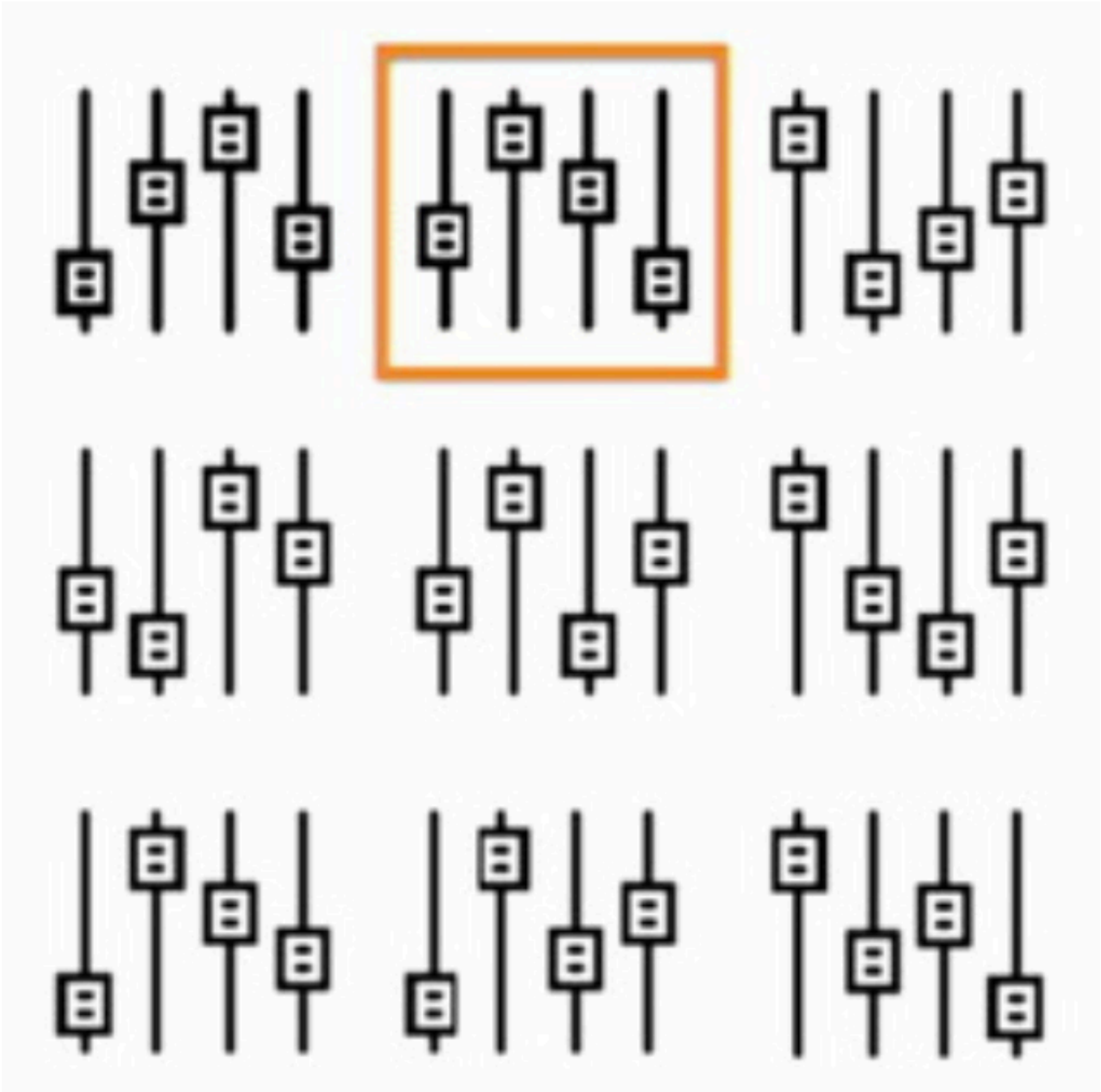
Feature Engineering Systems

Tools/Measures	Support for type of databases	Feature engineering	Feature selection	Open source implementation	Support for time series
Featuretools	Relational Tables	Yes	Yes	Yes	Yes
AutoFeat	Single Table	Yes	Yes	Yes	No
TSFresh	Single Table	Yes	Yes	Yes	Yes
FeatureSelector	Single Table	No	Yes	Yes	No
OneBM	Relational Tables	Yes	Yes	No	Yes
Cognito	Single Table	Yes	Yes	No	No

Hyperparameter Optimization

- Hyperparameters are parameters which define the model architecture

- Hyperparameter Optimization: Speed up the evaluation of different hyperparamter combinations and choose the most optimal one



Hyperparameter Optimization: High-Level Overview

Search Space → **Search Method** → **Evaluation Method**

Search Space	Search Method	Evaluation Method
Continuous and Discrete	Random and Grid Search	Partial Training
	Evolutionary Search	Full Training
	Bayesian Optimization	
	Gradient Based Optimization	

Cheap

Costly

Grid Search

- Search across a grid of configurations
    - Specify the bounds and steps between values of the hyperparameters
    - Start with a limited grid with relatively large steps between parameter values
    - Extend or make the grid finer at the best configuration
    - Continue searching on the new grid
- Costly approach
- Can be parallelized

Random Search

- Navigating the grid of hyperparameters randomly, one can obtain similar performance to a full grid search [1]

- If the close-to-optimal region of hyperparameters occupies at least 5% of the search space, then a random search with a certain number of trials will be able to find that region with high probability.

- Simple and effective

- Comparable performance to grid search with less number of trials

[1] Random Search for Hyper-Parameter Optimization, James Bergstra, Yoshua Bengio, JMLR 2012

Automated Hyperparameter Tuning

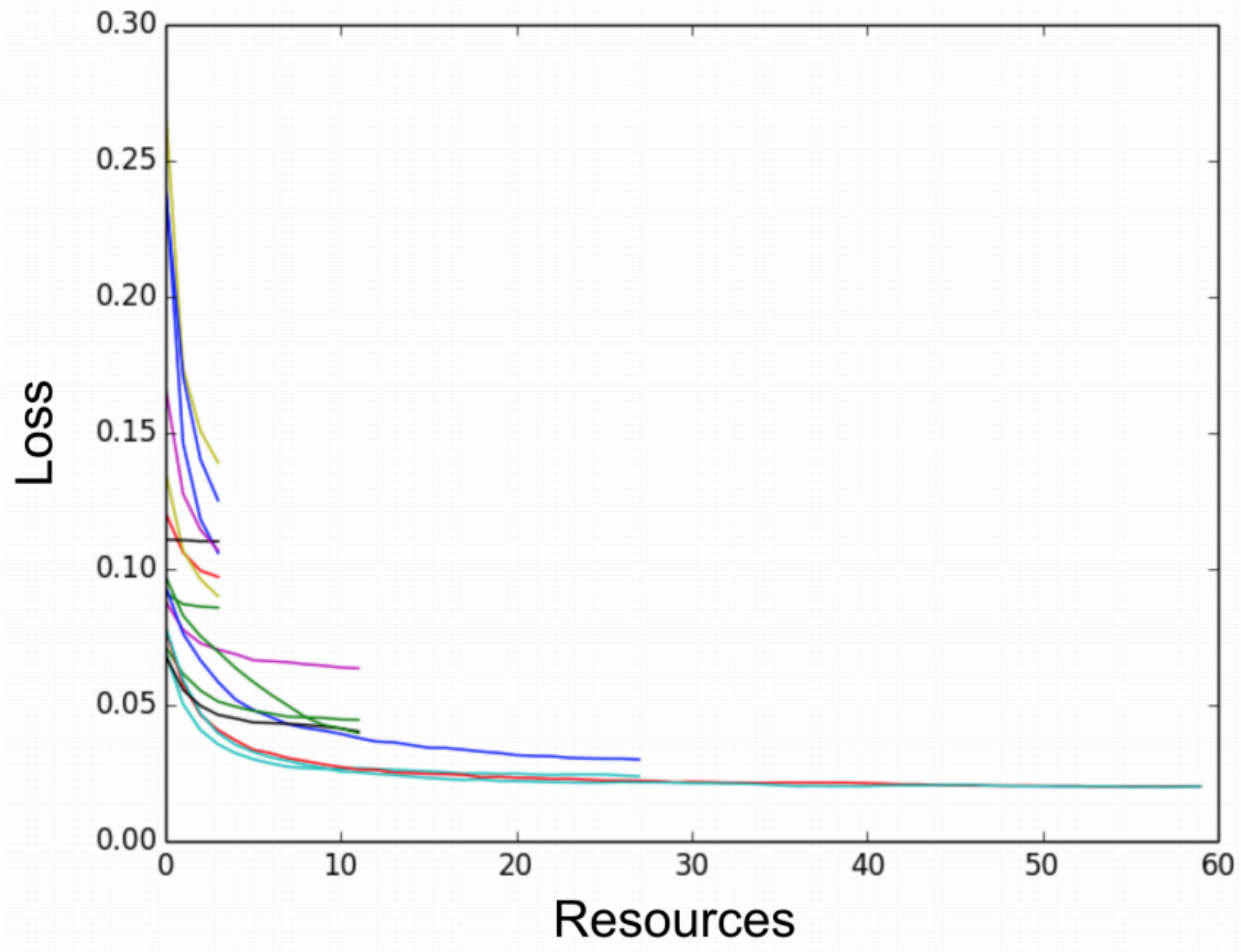
- In Grid Search and Random Search, the next trial is independent to all the trials done before.
- Goal of automated hyperparameter tuning: minimize the number of trials while finding a good optimum
- Use knowledge about the relation between the hyperparameter settings and model performance in order to make a smarter choice for the next parameter settings
- An optimization problem
- Sequential and not easily parallelizable.

Sequential Model-based Global Optimization(SMBO)

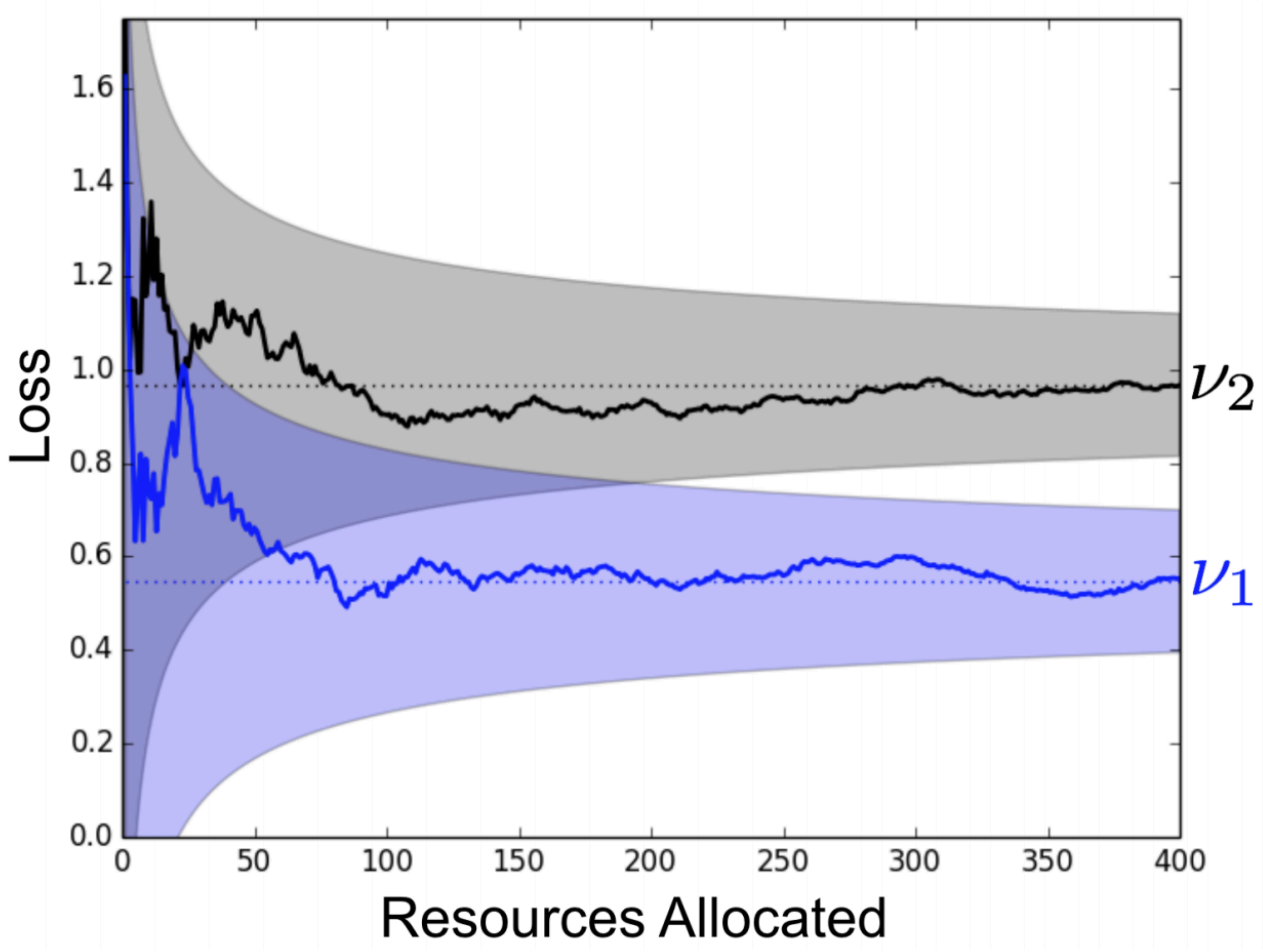
- Use a surrogate function to approximate the true blackbox function
- Use the surrogate model and an acquisition function to choose the next configuration to evaluate
- Several SMBO algorithms
    - Bayesian Optimization
        - Gaussian Process to model the surrogate
    - Sequential Model-based Algorithm Configuration (SMAC)
        - Random forest of regression trees to model the objective function
    - Tree-structured Parzen Estimator (TPE)
        - An improved version of SMAC where two separated models are used to model the posterior
- Acquisition functions: Expected Improvement (EI), probability of improvement, minimizing conditional entropy

Successive Halving

- Assumes that algorithm can be stopped early and an approx. validation score computed
- Randomly sample a number of configurations in parallel and run for short amount of time
- At the end of the interval, keep only a fraction of configurations with best performance
- Run remaining configurations longer
- Repeat until max budget is reached



Early Stopping



Hyperband: Bandit Approach

- Hyperband solves the robustness issue with Successive Halving
- Evenly split resources between running Successive Halving with multiple values of sensitive *hyper hyperparameters*
- Many values between extremes of no culling (random search) and aggressive culling (large #configs with multiple steps of culling) are tried
- Improves performance considerably, but still not most effective

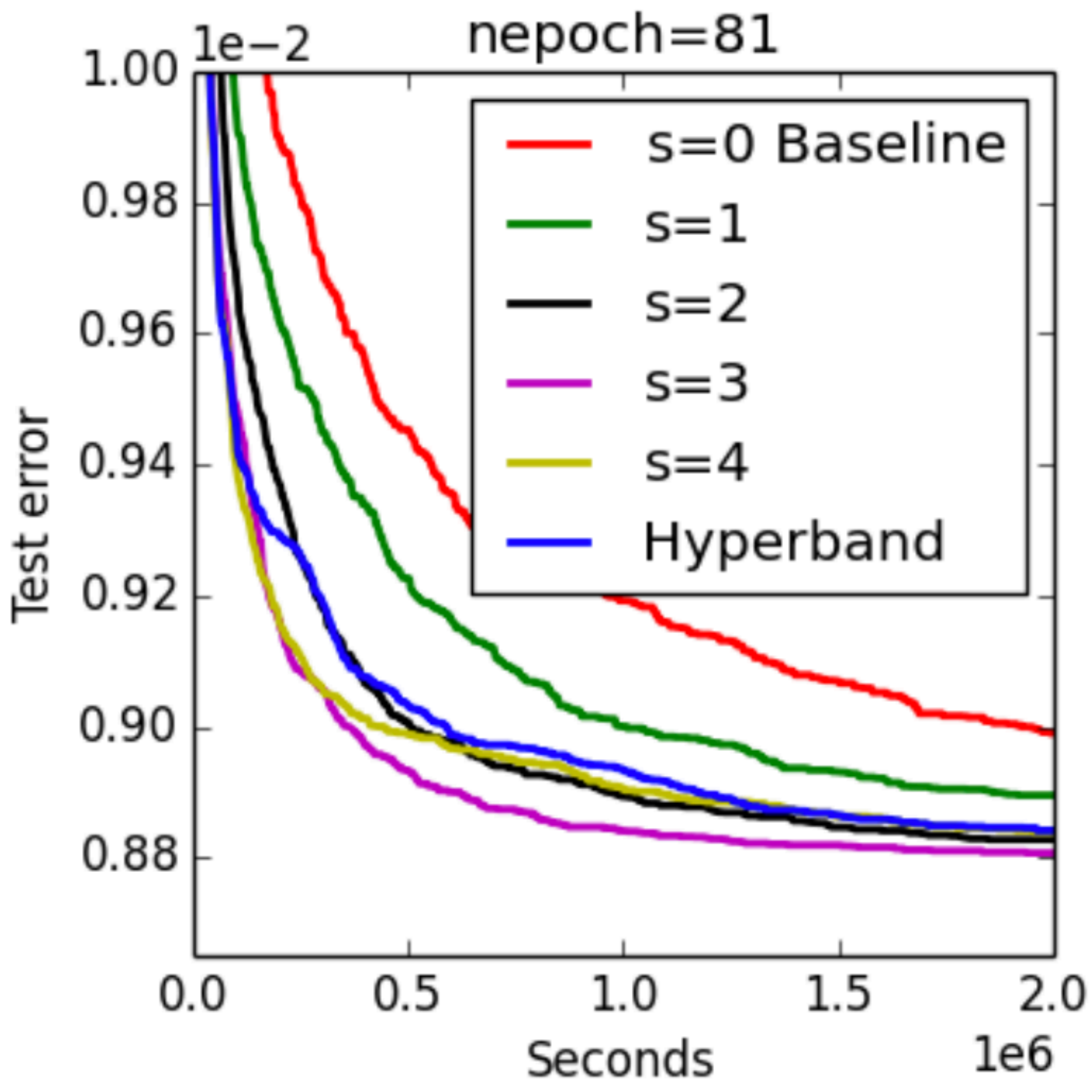
Hyperband Algorithm

Algorithm 1: HYPERBAND algorithm for hyperparameter optimization.

input : R, η (default $\eta = 3$)
initialization : $s_{\max} = \lfloor \log_\eta(R) \rfloor$, $B = (s_{\max} + 1)R$

for $s \in \{s_{\max}, s_{\max} - 1, \ldots, 0\}$ **do**
  $n = \lceil \frac{B}{R} \frac{\eta^s}{(s+1)} \rceil, \qquad r = R\eta^{-s}$
  `// begin SuccessiveHalving with` (n, r) `inner loop`
  $T =$`get_hyperparameter_configuration`(n)
  for $i \in \{0, \ldots, s\}$ **do**
    $n_i = \lfloor n\eta^{-i} \rfloor$
    $r_i = r\eta^i$
    $L = \{$`run_then_return_val_loss`$(t, r_i) : t \in T\}$
    $T =$`top_k`$(T, L, \lfloor n_i/\eta \rfloor)$
  end
end
return *Configuration with the smallest intermediate loss seen so far.*

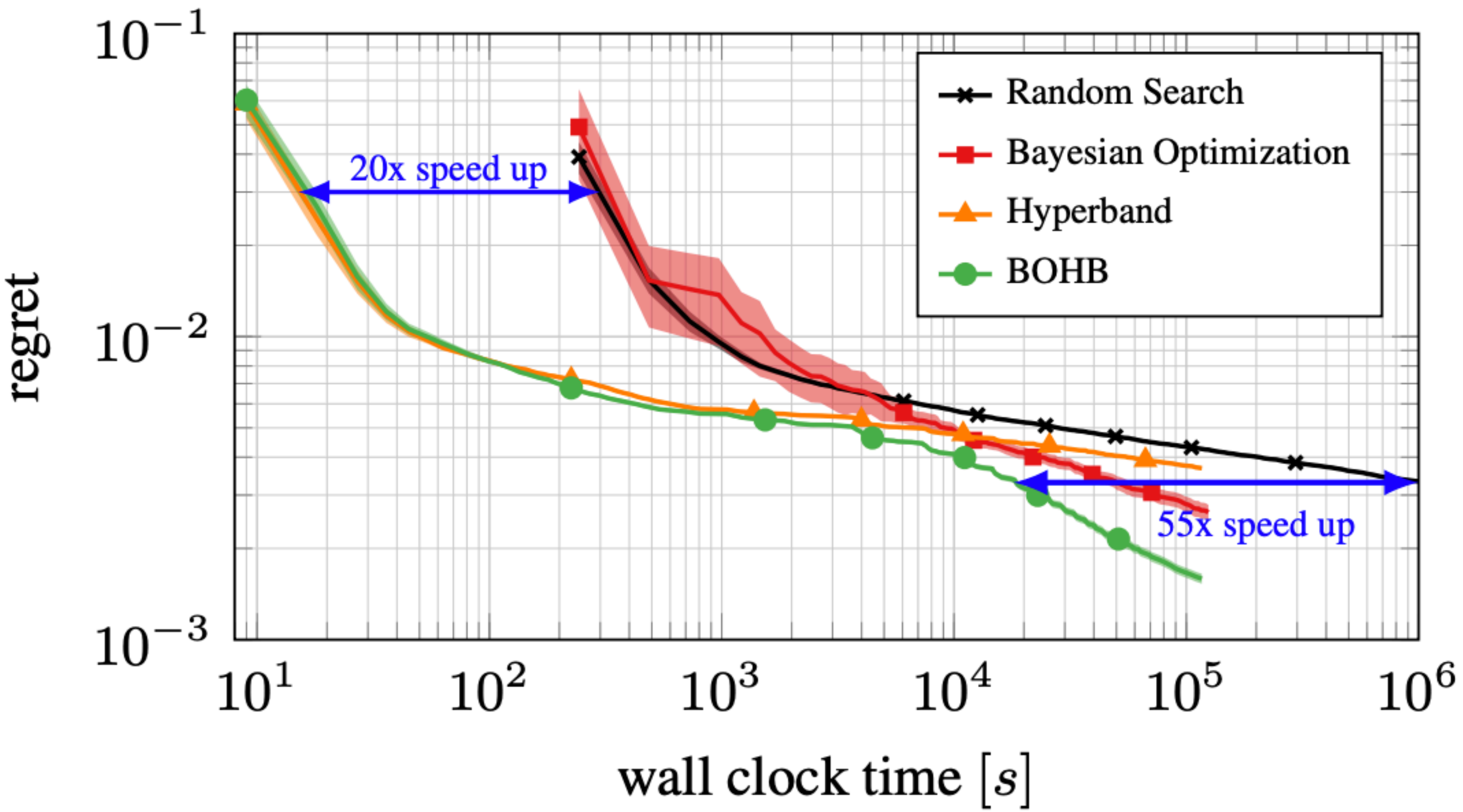
Hyperband Performance



Bayesian Optimization and Hyperband (BOHB, ICML'18)

- Start with vanilla Hyperband and store validation scores for all (config, budget) pair
- When sufficient amount of data is collected, fit a TPE surrogate model and use this for future configuration selection using EI
- Continue to sample random configurations with some small frequency for robustness

BOHB performance



Neural Architectural Search (NAS)

Search Space → **Search Method** → **Evaluation Method**

Search Space

- Continuous and Discrete
- Cell block and meta-architecture

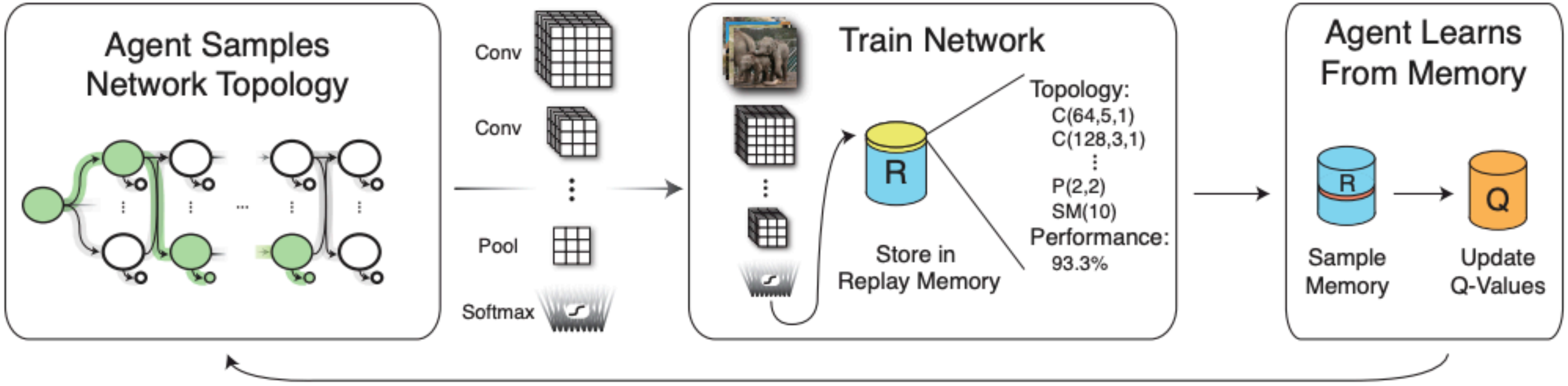
Search Method

- Random and Grid Search
- Evolutionary Search
- Bayesian Optimization
- Gradient Based Optimization
- Reinforcement Learning

Evaluation Method

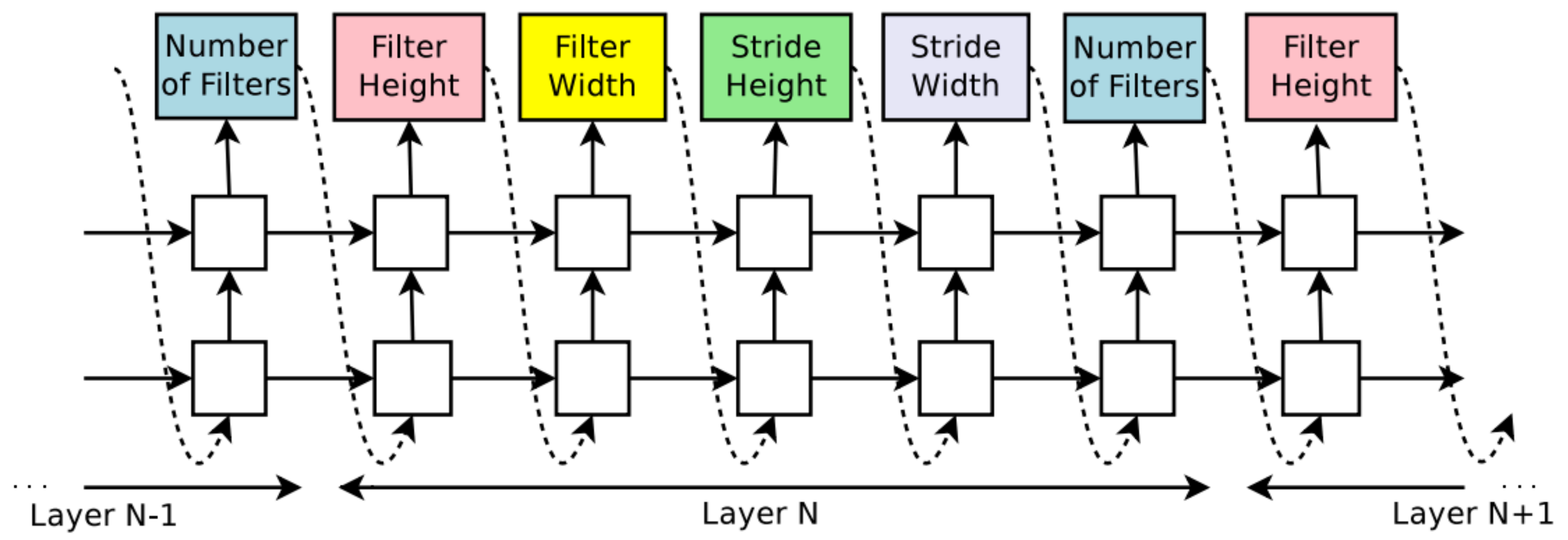
- Partial Training
- Full Training
- Weight Sharing
- Hypernetworks
- Network morphisms

NAS with RL: MetaQNN



NAS with Reinforcement Learning [Zoph, Le, ICLR'17]

- Uses a RNN to generate the model descriptions of neural networks
- Train the RNN with RL to maximize the expected accuracy of the generated architectures on a validation set
- Computational Overhead is high
  - 800 GPUs for 28 days on CIFAR dataset



Efficient NAS via Parameter Sharing [ICML'18]

- All of the graphs which NAS ends up iterating over can be viewed as sub-graphs of a larger graph
- Share parameters among all generated networks
- Each training stage is much shorter
- Much more efficient
    - 1 GPU for 0.45 days (CIFAR)
    - No Imagenet experiments

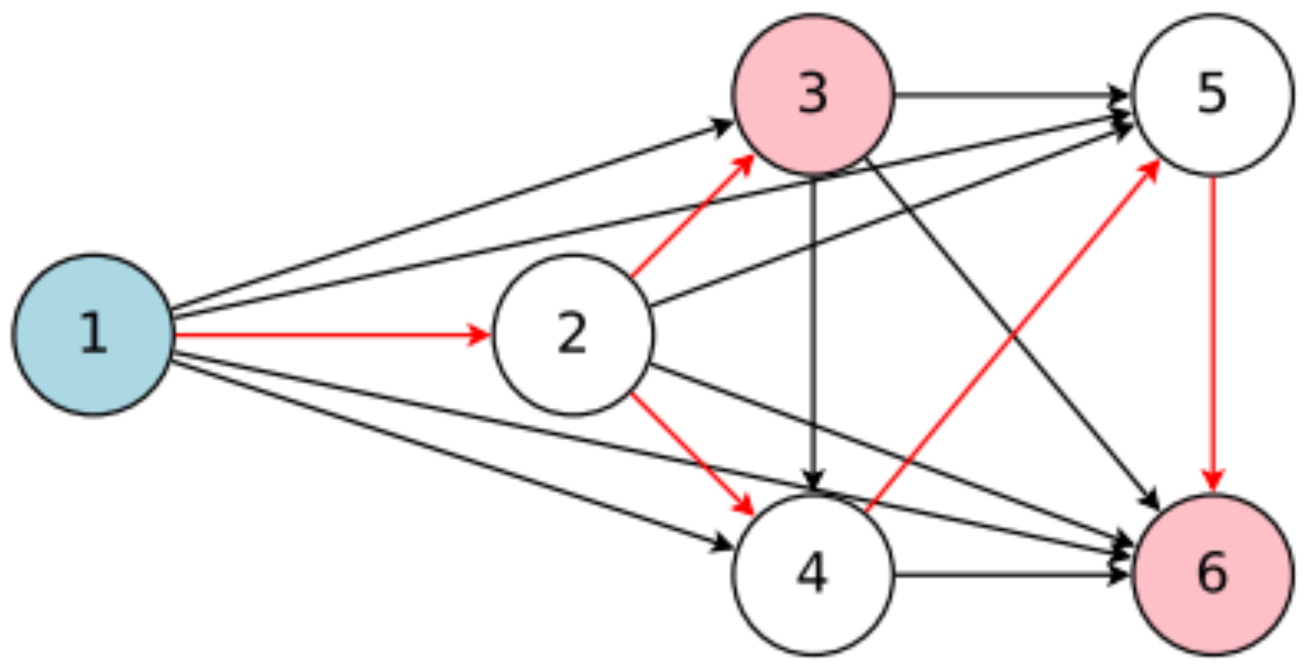


Figure 2. The graph represents the entire search space while the red arrows define a model in the search space, which is decided by a controller. Here, node 1 is the input to the model whereas nodes 3 and 6 are the model's outputs.

Regularized Evolution [AAAI'19]

- Evolution has comparable or better performance than RL
- Assign "aged individuals" with a higher probability for elimination
- Works best when computational budget is limited

ProxyLess NAS [ICLR'19]

- Learning weight parameters and binarized architectures simultaneously
- Specialized architectures for each platform
- Efficient
    - 1 GPU for 8 days
    - Reasonable performance

ProxyLess NAS [ICLR'19]

Model	Top-1 (%)	GPU latency	CPU latency	Mobile latency
Proxyless (GPU)	75.1	5.1ms	204.9ms	124ms
Proxyless (CPU)	75.3	7.4ms	138.7ms	116ms
Proxyless (mobile)	74.6	7.2ms	164.1ms	78ms

Other Related Directions

- Discovery of architectures that are robust against adversarial attacks
- Considering sample efficiency
- Interpretability of hyperparameter tuning process

Thanks!