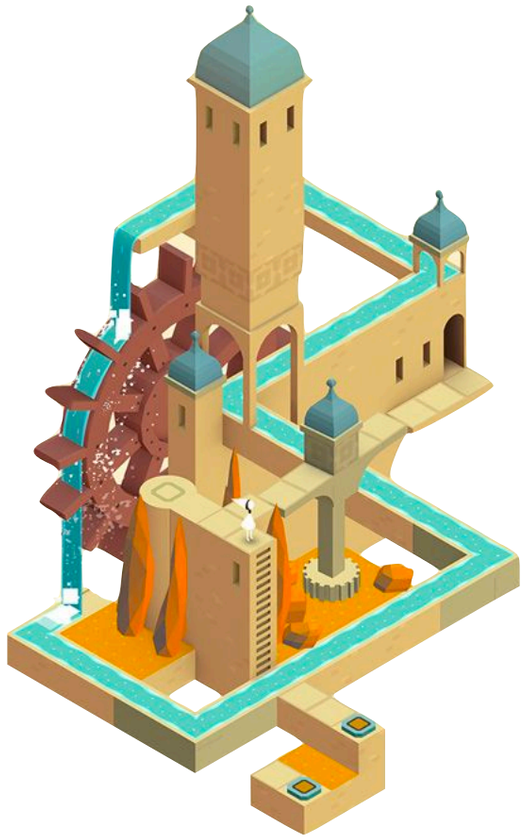
Open Data Science Conference
London, 2019



Automatic Machine Learning

A Tutorial

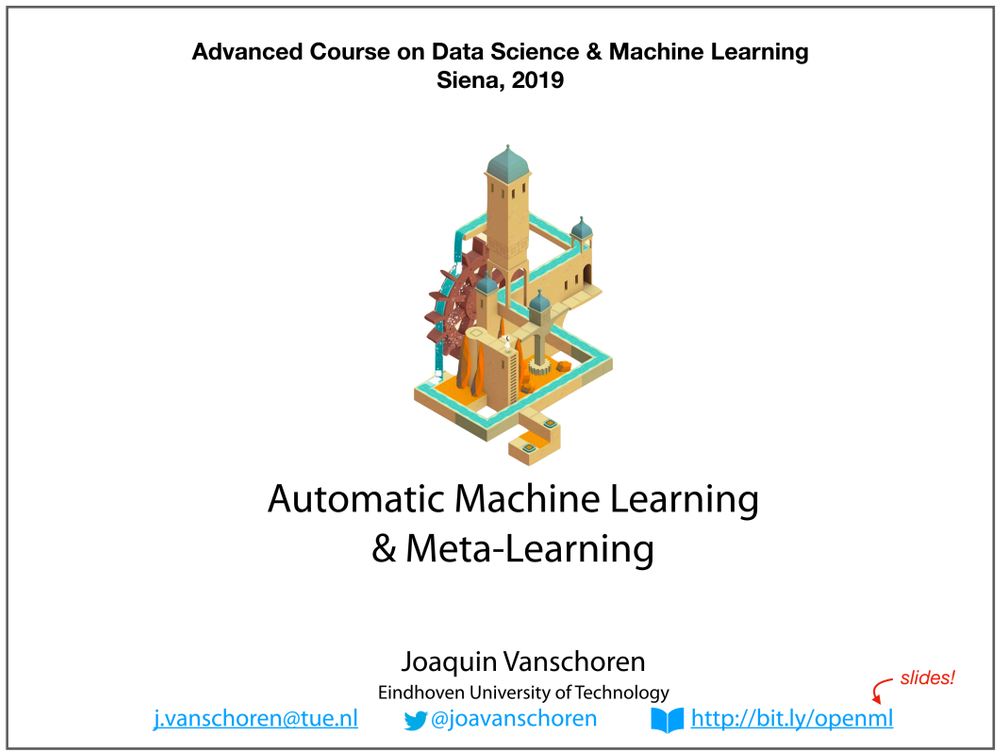
Joaquin Vanschoren & Pieter Gijsbers

Eindhoven University of Technology

j.vanschoren@tue.nl @joavanschoren http://bit.ly/openml

slides!

Slides / Book



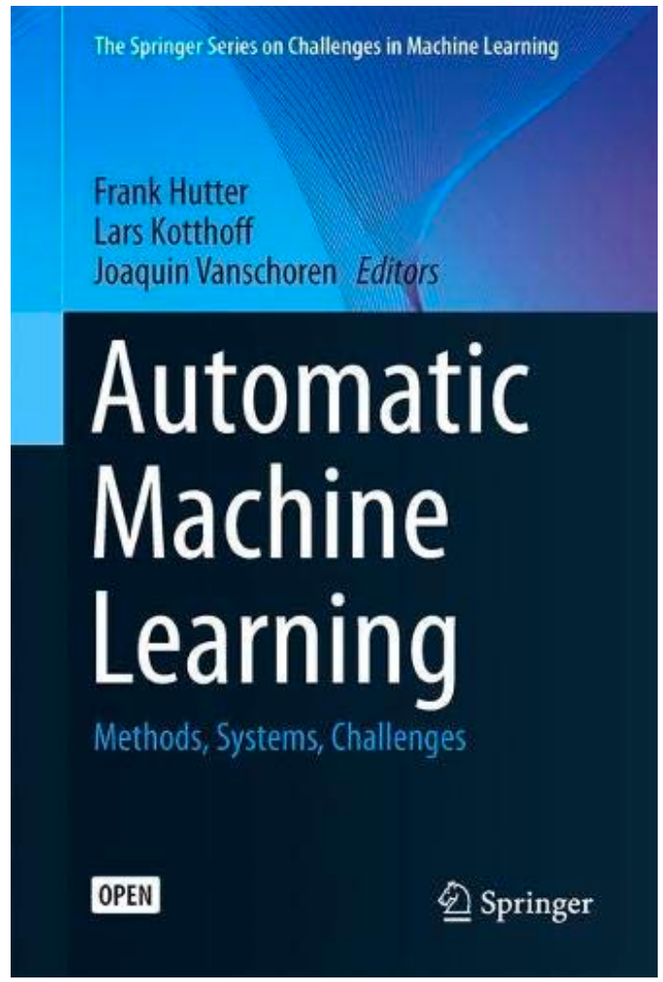
http://bit.ly/openml

More slides:

www.automl.org/events ->
AutoML Tutorial NeurIPS 2018

Video:

www.youtube.com/watch?v=0eBR8a4MQ30

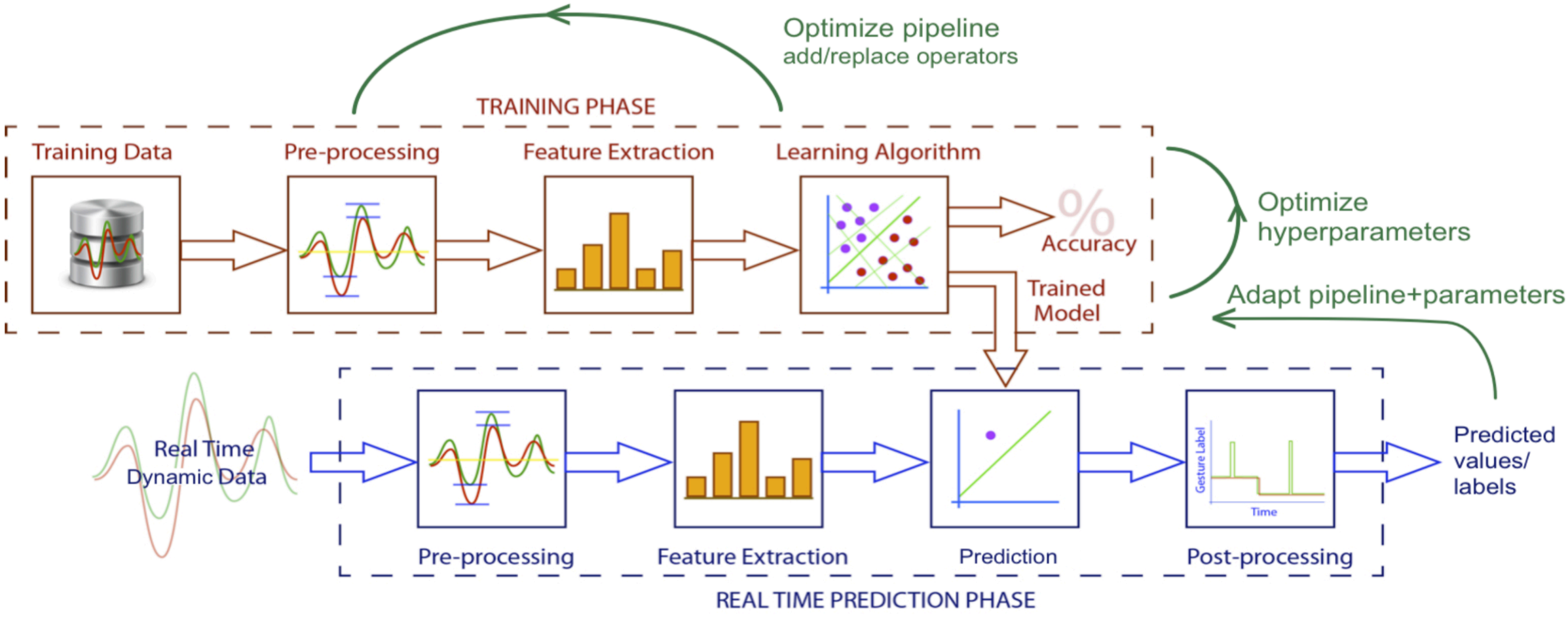


Open access book

PDF (free): www.automl.org/book
www.amazon.de/dp/3030053172

Doing machine learning requires a lot of expertise

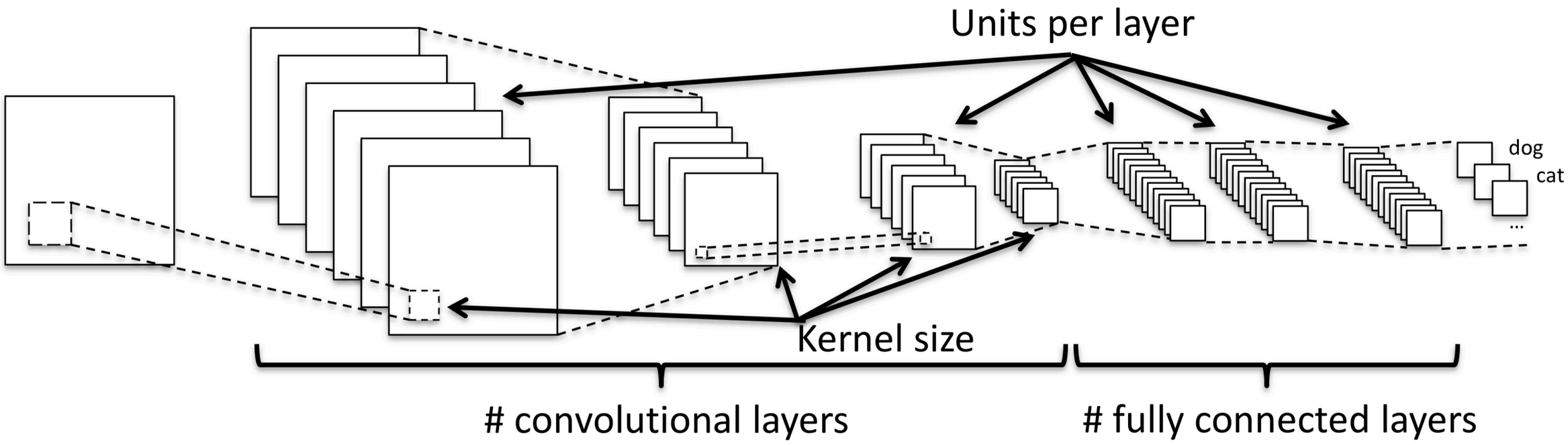
Designing pipelines



- Excellent tools (e.g. scikit-learn), but little guidance
- Cleaning, preprocessing, feature selection/engineering features, model selection, hyperparameter tuning, adapting to concept drift,…

Doing machine learning requires a lot of expertise

Designing neural architectures

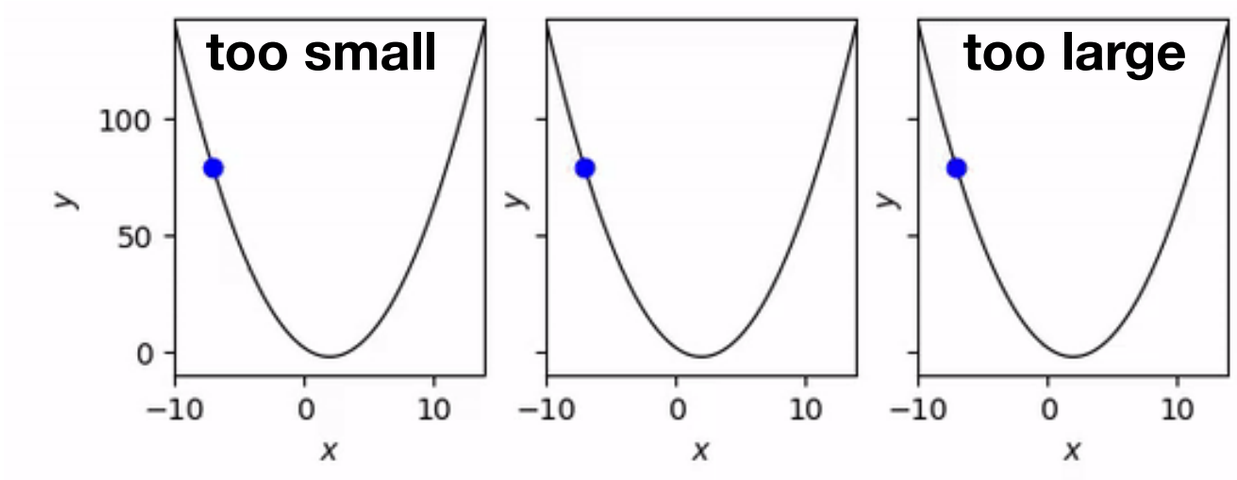


- Choose operators (layers), data preparation, data augmentation, hyperparameter tuning, regularization, early stopping,…

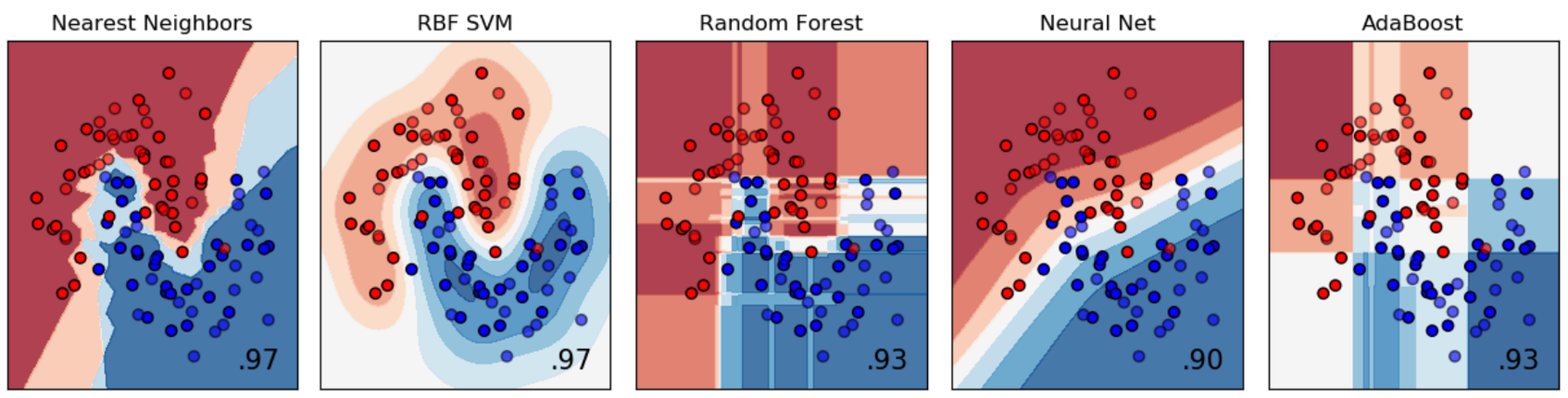
Hyperparameters

Every design decision made by the *user (architecture, operators, tuning,…)*

- Numeric
  - e.g. learning rate

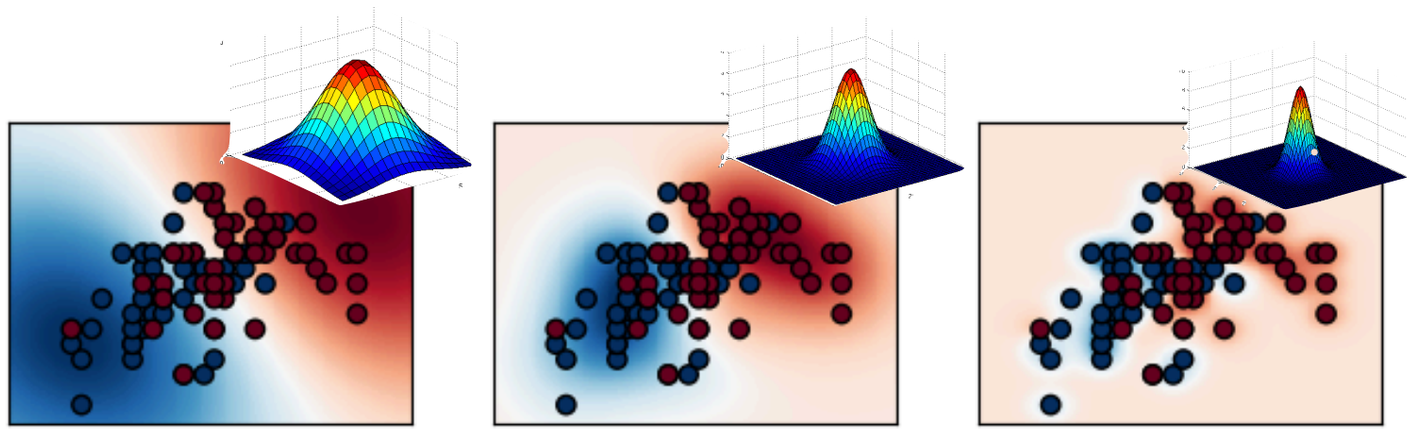


- Categorical
  - e.g. classifier



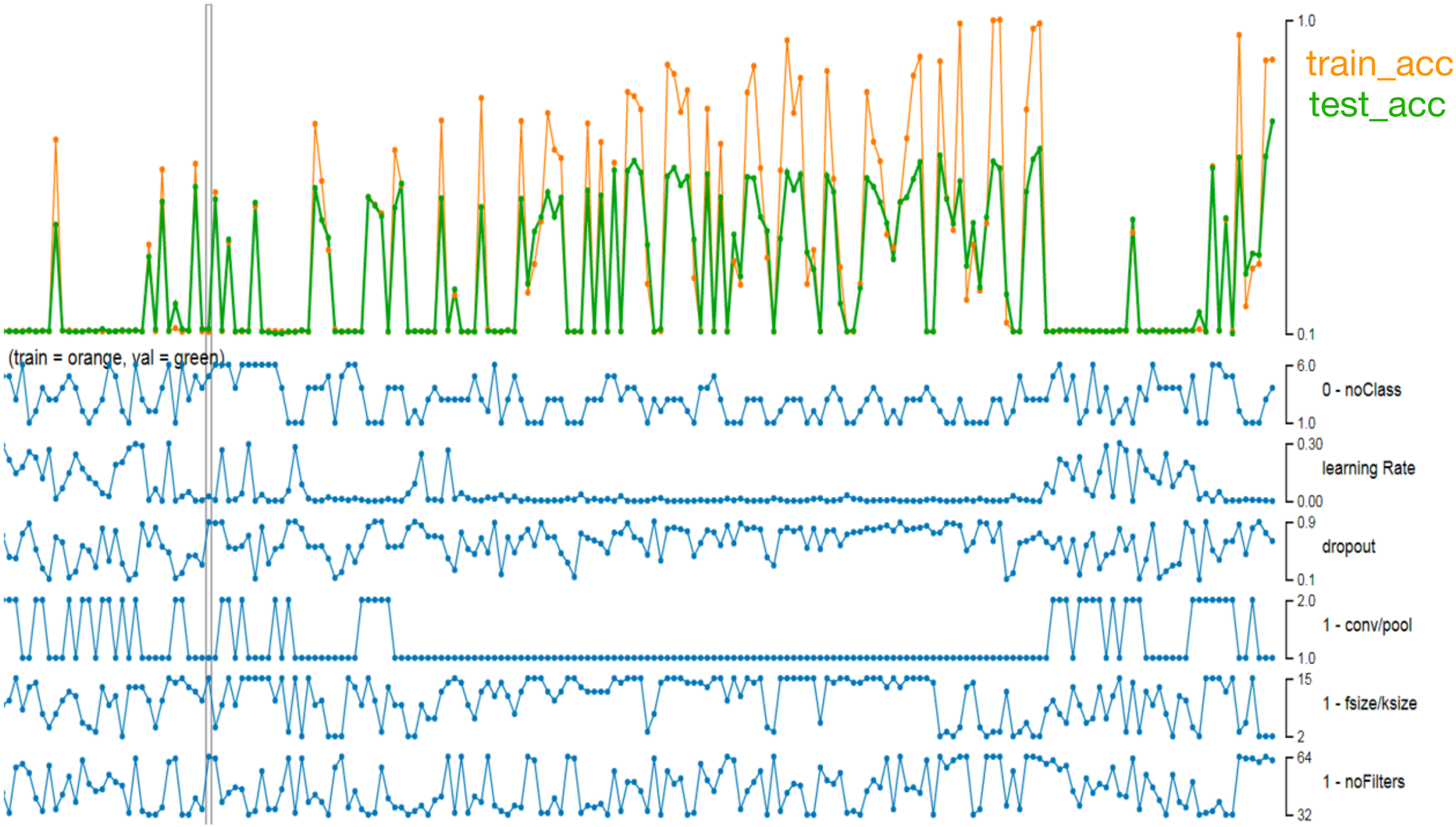
- Conditional
  - SVM -> kernel?

    RBF kernel -> gamma?



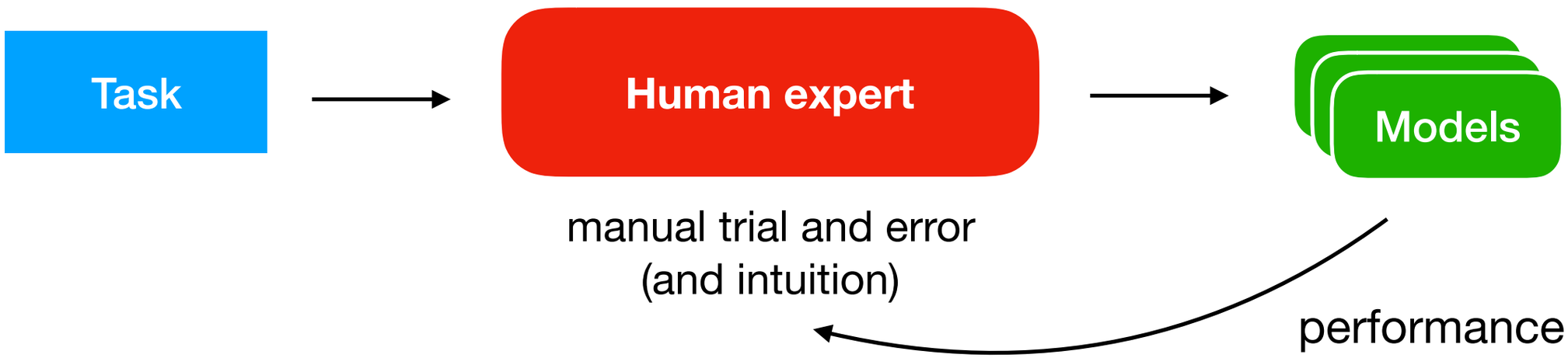
Hyperparameters

Some are very sensitive, others have no impact at all

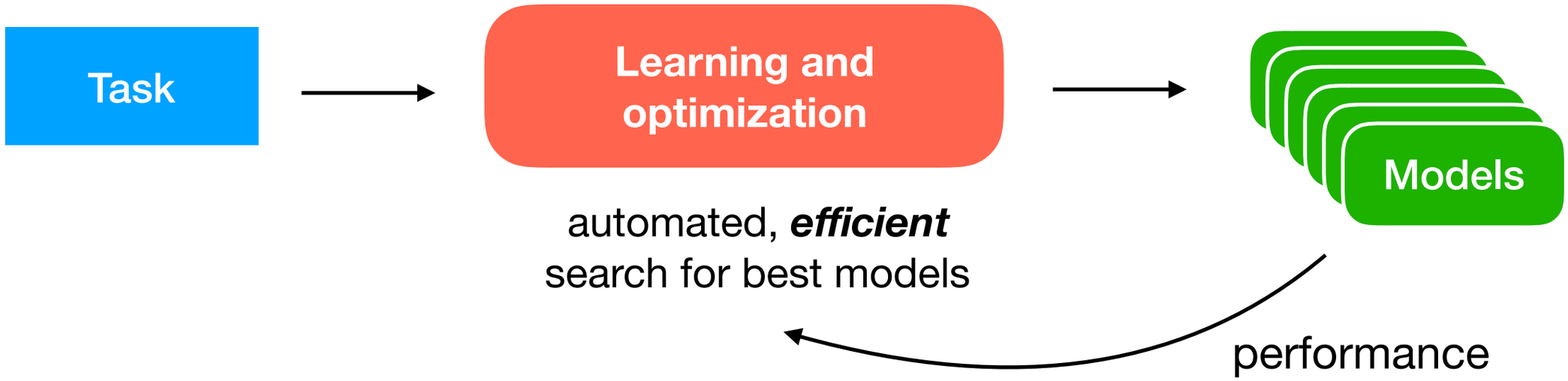


Automated machine learning

Manual machine learning

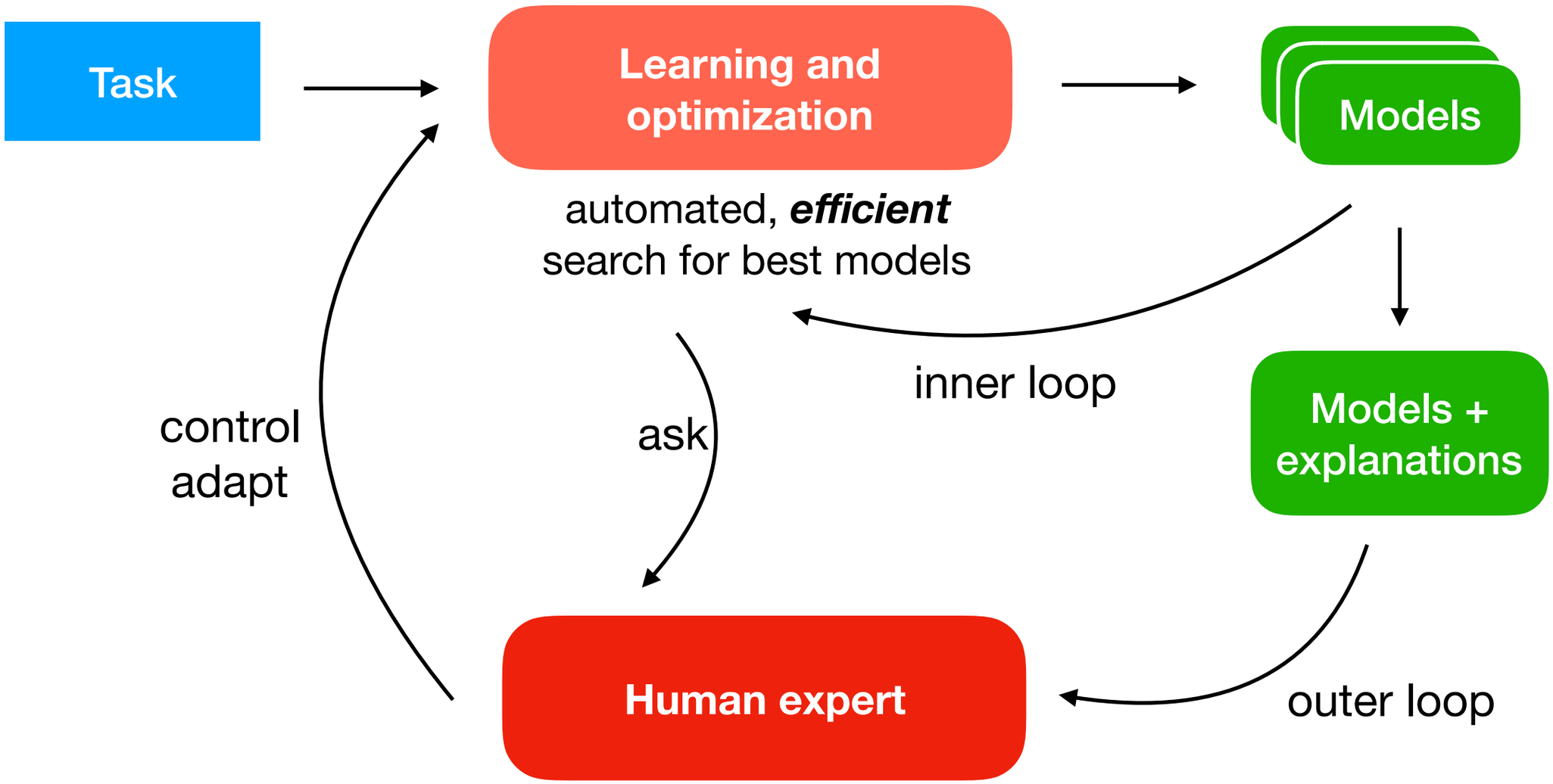


AutoML: build models in a data-driven, objective, and automated way



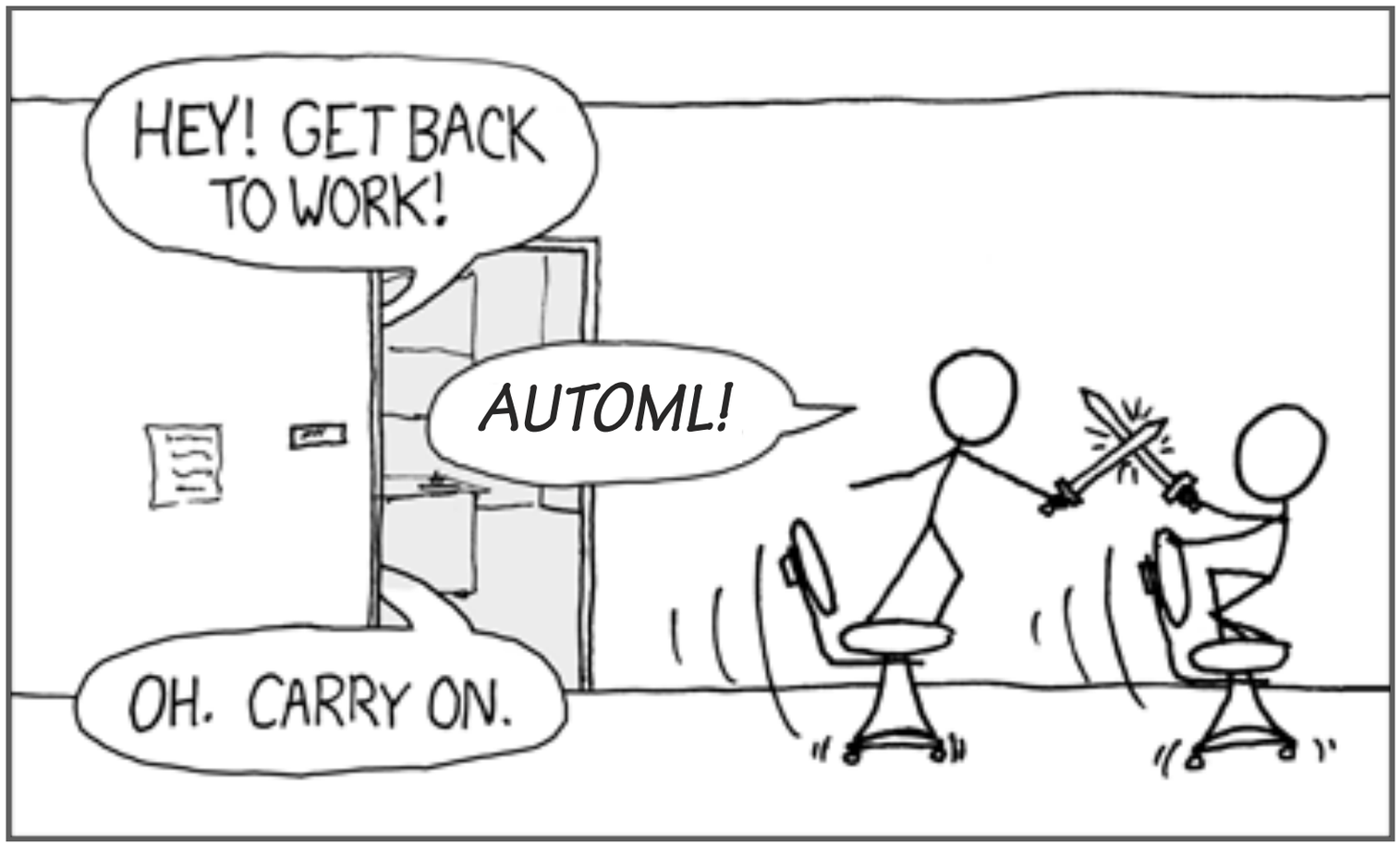
Semi-Automated machine learning

Human-in-the-loop



Domain knowledge and human expertise are very valuable
e.g. unknown unknowns, preference learning,…

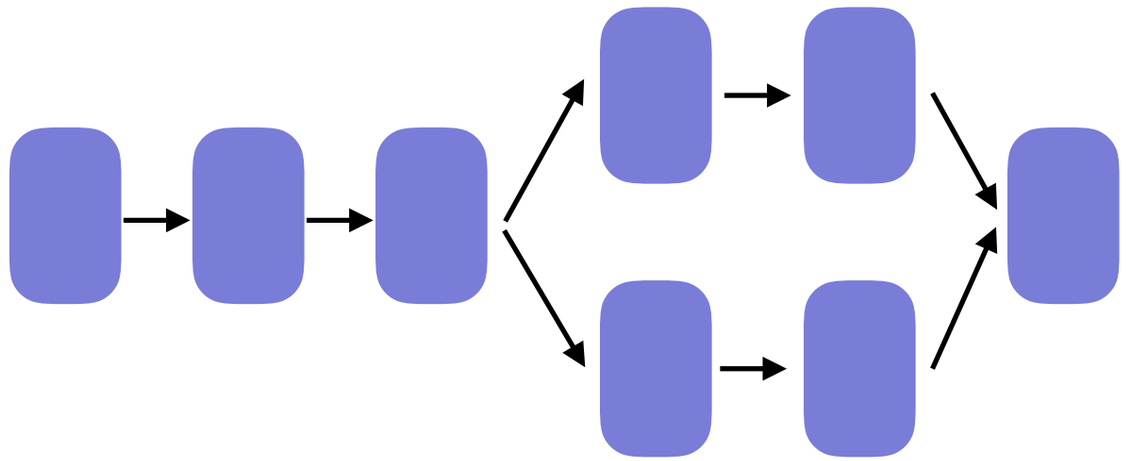
THE DATA SCIENTIST'S #1 EXCUSE FOR LEGITIMATELY SLACKING OFF: "THE AUTOML TOOL IS OPTIMIZING MY MODELS!"



Adapted from XKCD (Randall Munroe)

AutoML: subproblems

- **Architecture search**: *represent* and search possible architectures
  - Pipeline operators, neural layers, …
  - Defines the search space

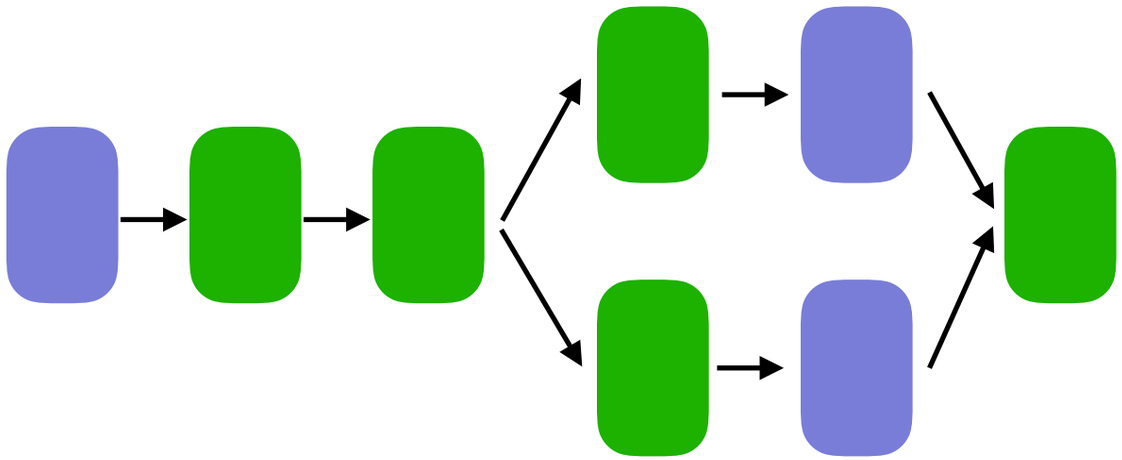


```
make_pipeline(
  OneHotEncoder(),
  Imputer(),
  StandardScaler(),
  SVC())
```

```
model.add(Conv2D(32, (3, 3))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3))
```

AutoML: subproblems

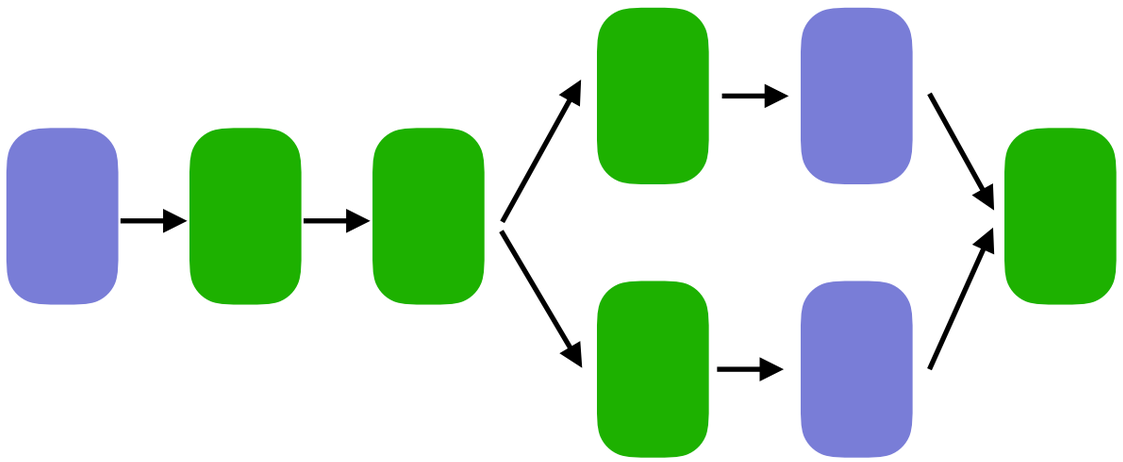
- **Architecture search**: *represent* and search all possible architectures
- **Hyperparameter optimization**:
  - Which hyperparameters are important? Which values to try?
  - How to *optimize* remaining hyperparameters



```
hyper_space = {'SVC__C': expon(scale=100),
               'SVC__gamma': expon(scale=.1)}
RandomizedSearchCV(pipe, param_distributions=hyper_space, n_iter=200)
```

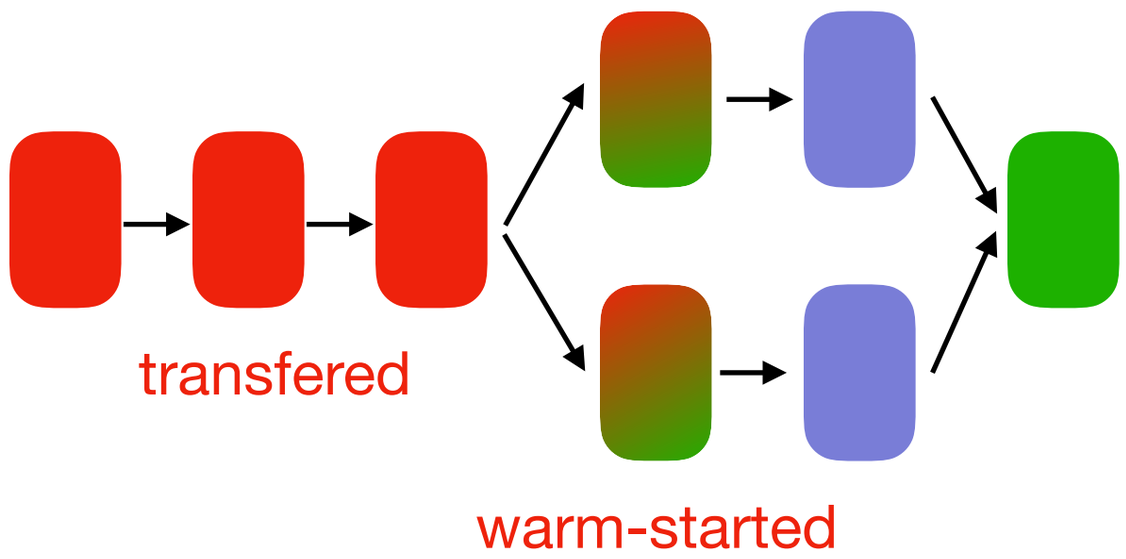
AutoML: subproblems

- **Architecture search**: *represent* and search all possible architectures
- **Hyperparameter optimization**: *optimize* remaining hyperparameters
- **Objective function**: *what* to optimize, can be multi-objective
  - E.g. high performance but also simplicity, interpretability,…



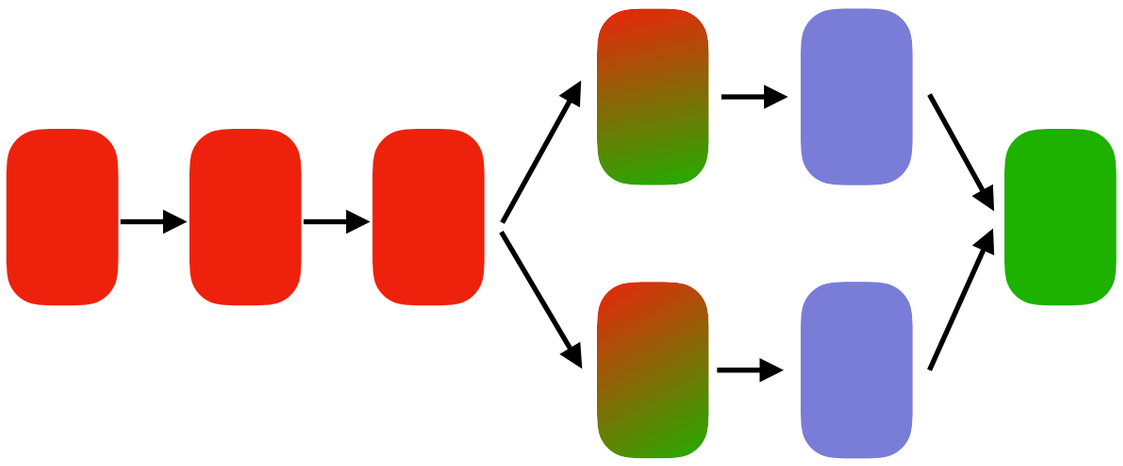
AutoML: subproblems

- **Architecture search**: *represent* and search all possible architectures
- **Hyperparameter optimization**: *optimize* remaining hyperparameters
- **Objective function**: *what* to optimize, can be multi-objective
- **Meta-learning**: how can we transfer experience from previous tasks?
  - Don’t start from scratch every time



AutoML: subproblems

- **Architecture search**: *represent* and search all possible architectures
- **Hyperparameter optimization**: *optimize* remaining hyperparameters
- **Objective function**: *what* to optimize, can be multi-objective
- **Meta-learning**: how can we transfer experience from previous tasks?
  - Don’t start from scratch every time



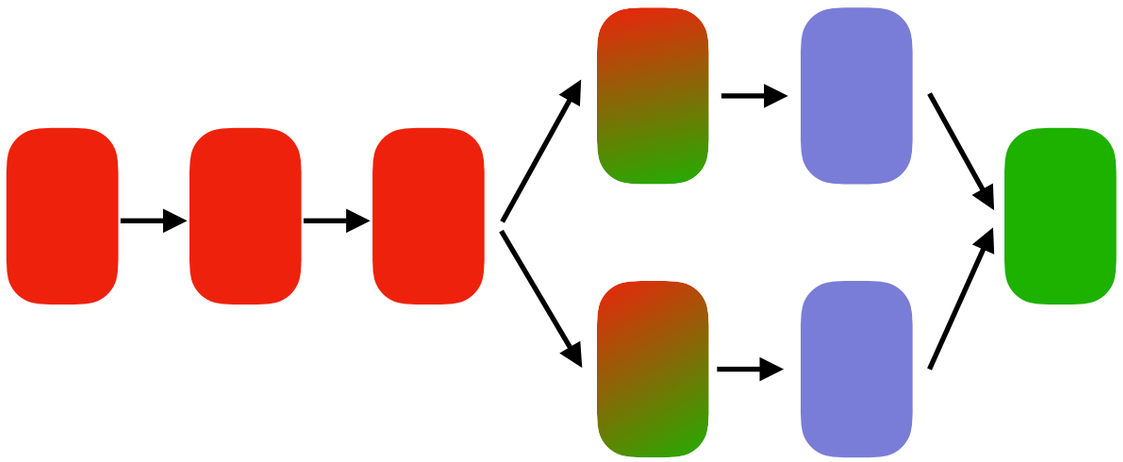
Goal:

```
model = AutoML_Learner().fit(X_train,y_train)
model2 = AutoML_Learner().fit(X_train2,y_train2)
```

meta-learn

AutoML: subproblems

- **Architecture search**: *represent* and search all possible architectures
- **Hyperparameter optimization**: *optimize* remaining hyperparameters
- **Meta-learning**: how can we transfer experience from previous tasks?
  - Don’t start from scratch every time

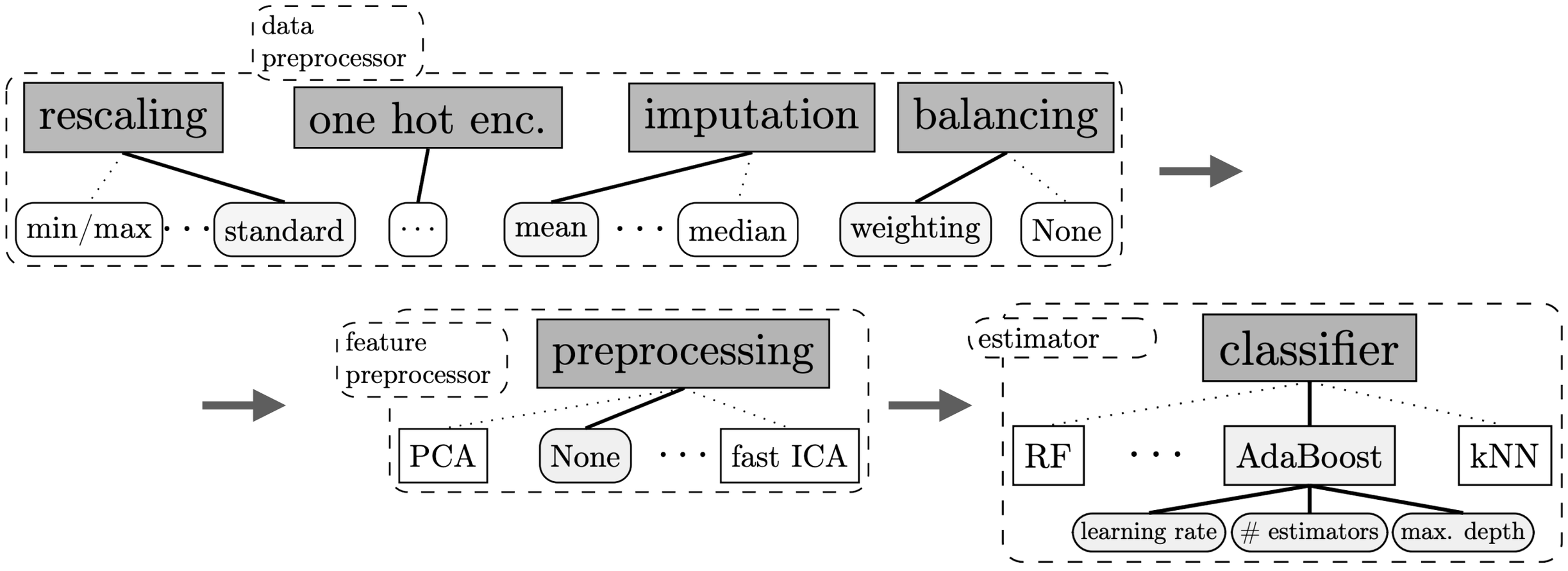


these 3 subproblems can be solved
consecutively, *simultaneously* or *interleaved*

autosklearn *Feurer et al. 2015*
autoWEKA *Thornton et al. 2013*
hyperopt-sklearn *Komer et al. 2014*

Parameterized architecture

- Observation: most successful pipelines have a similar structure
- Fixed (linear) architecture, all possible choices encoded as extra hyperparameters
- *Architecture search becomes hyperparameter optimization*



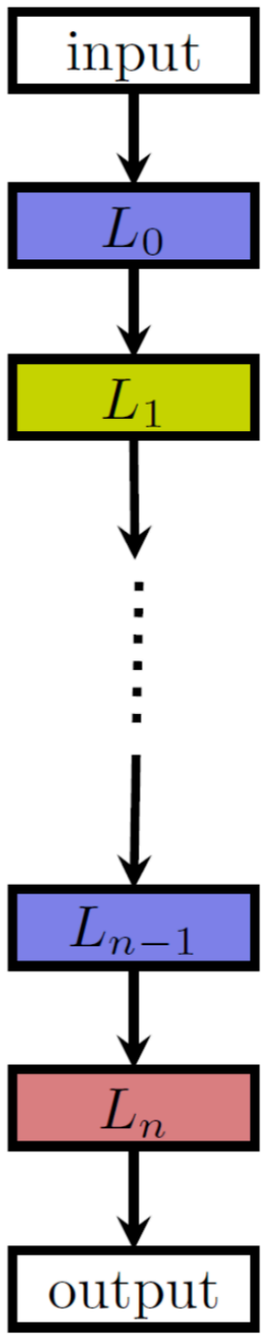
+ smaller search space
- you can't learn entirely new architectures

Neural Architecture Search

Parameterized Sequential

Choose:
- number of layers
- type of layers
  - dense
  - convolutional
  - max-pooling
  - …
- hyperparameters of layers

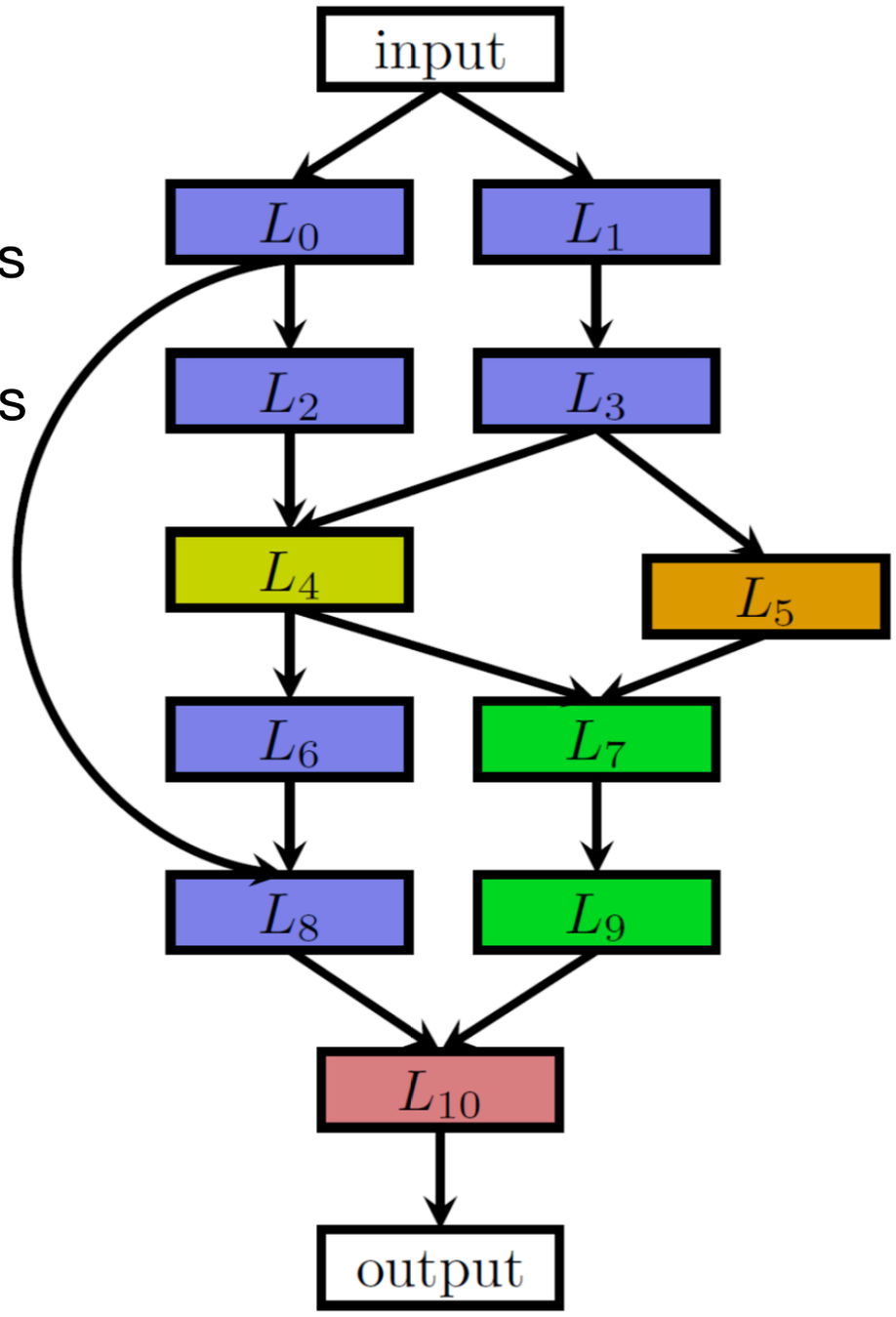
+ easier to search
- sometimes too simple



Parameterized Graph

Choose:
- branching
- joins
- skip connections
- types of layers
- hyperparameters of layers

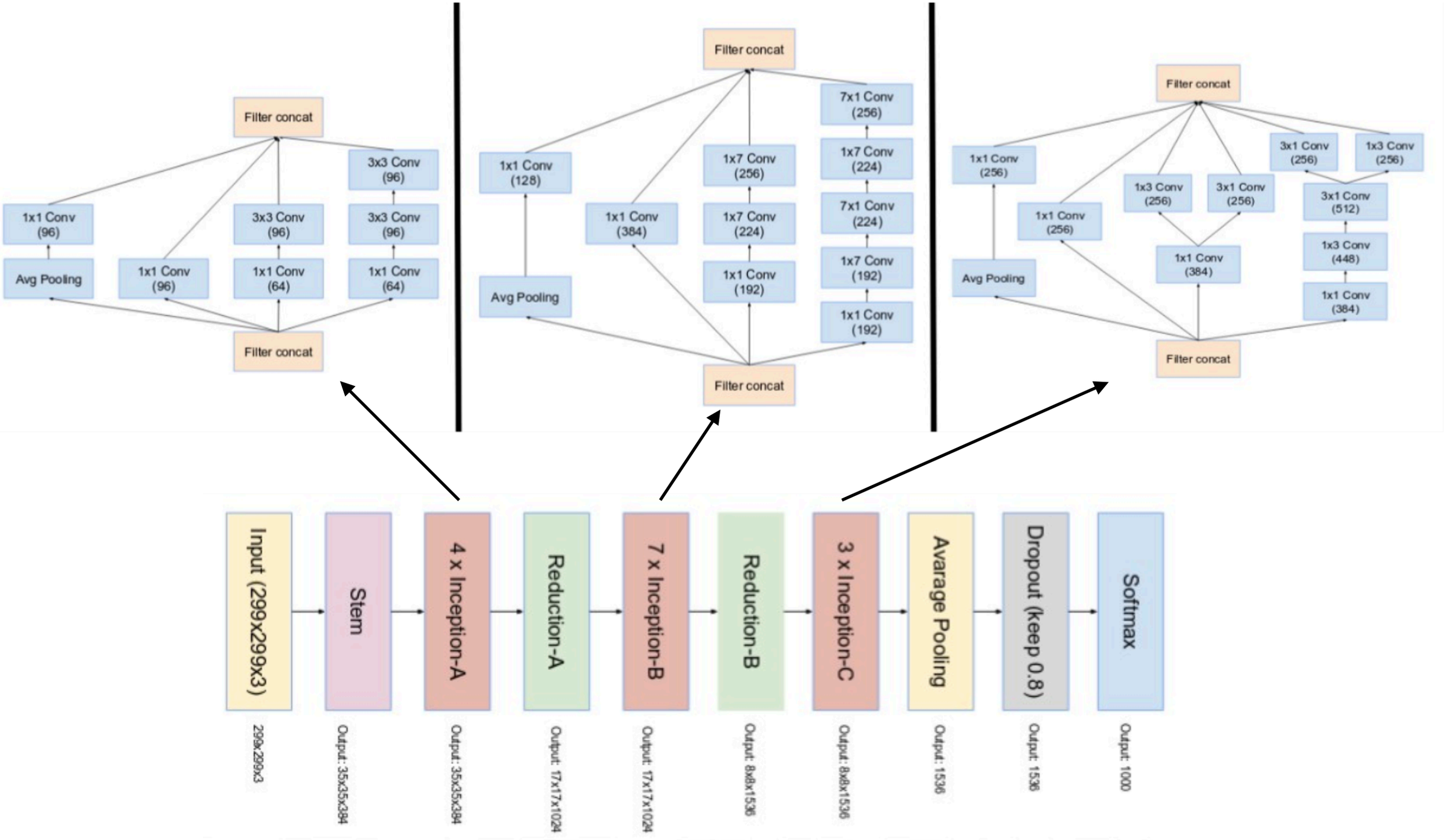
+ more flexible
- much harder to search



Neural Architecture Search

Observation: successful deep networks have repeated motifs (cells)

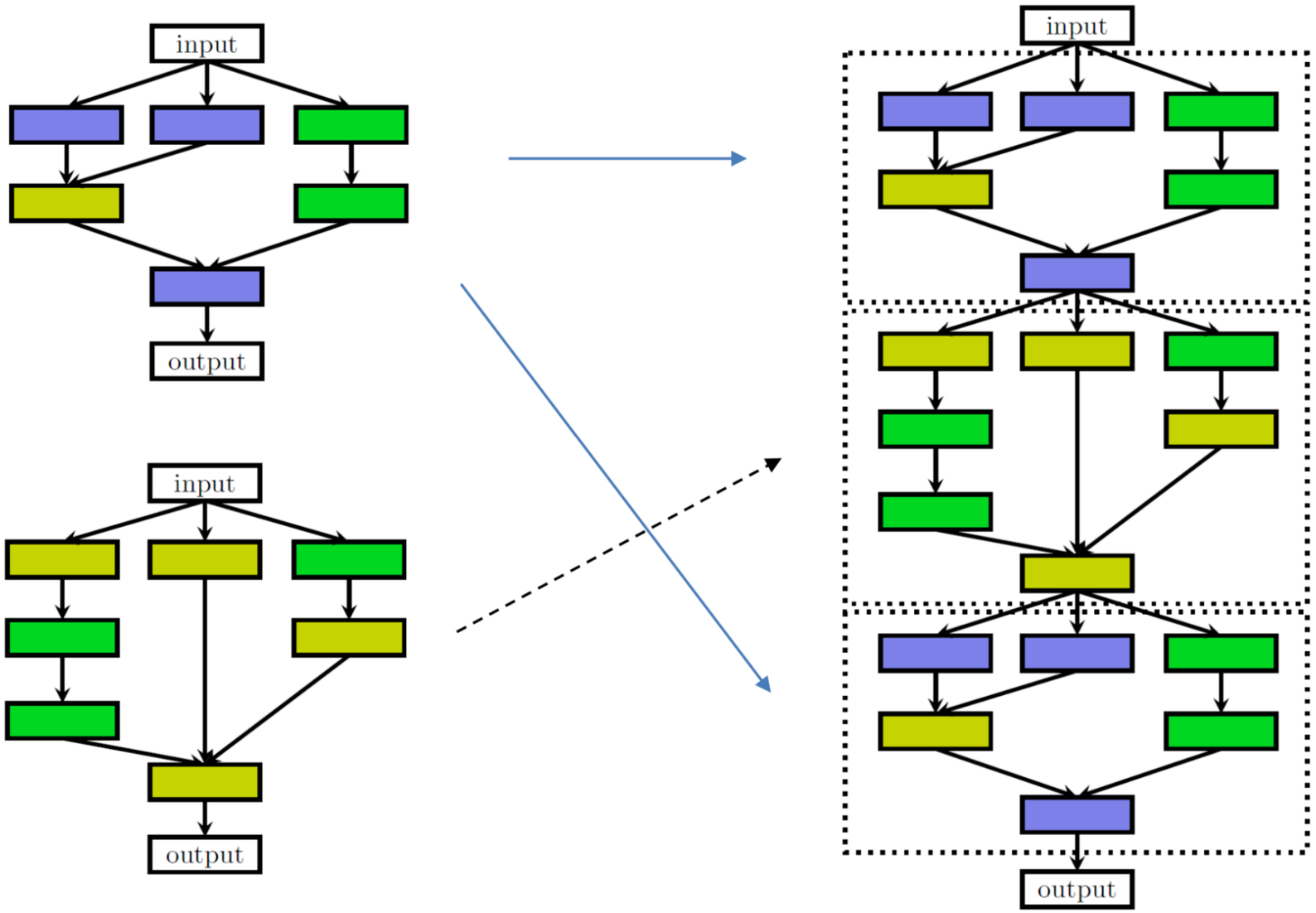
e.g. Inception v4:



Neural Architecture Search

Cell search space

- parameterized building blocks *(cells)*
- stack cells together in macro-architecture
  - usually a chain



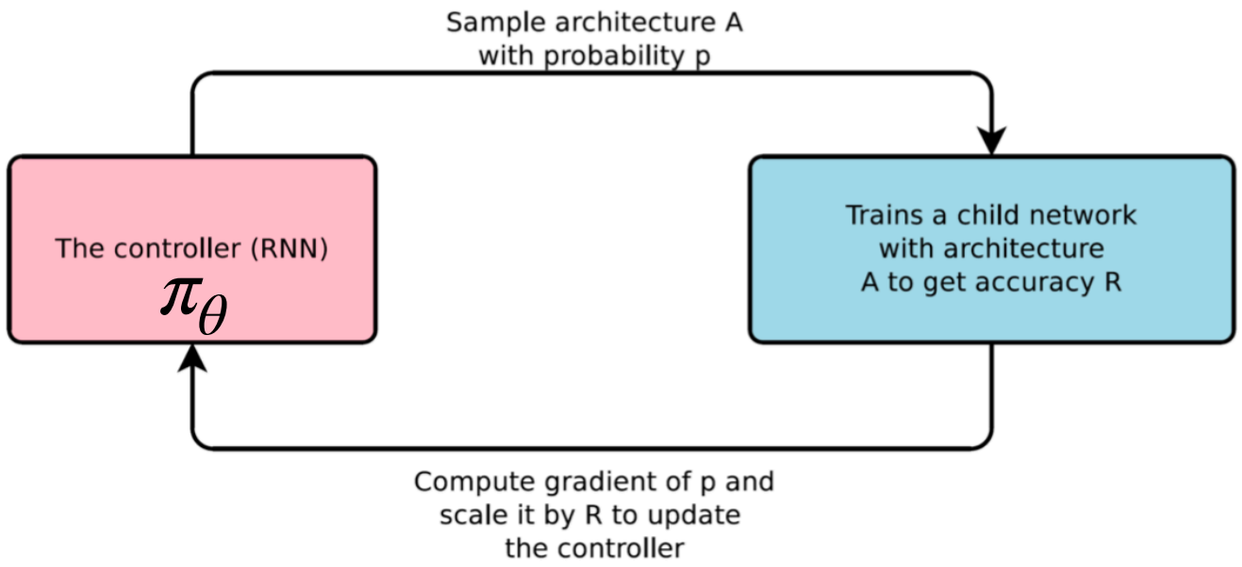
+ smaller search space
+ cells can be learned on a small dataset & transferred to a larger dataset
- you can't learn entirely new architectures

Zoph & Le 2017

NAS with Reinforcement learning

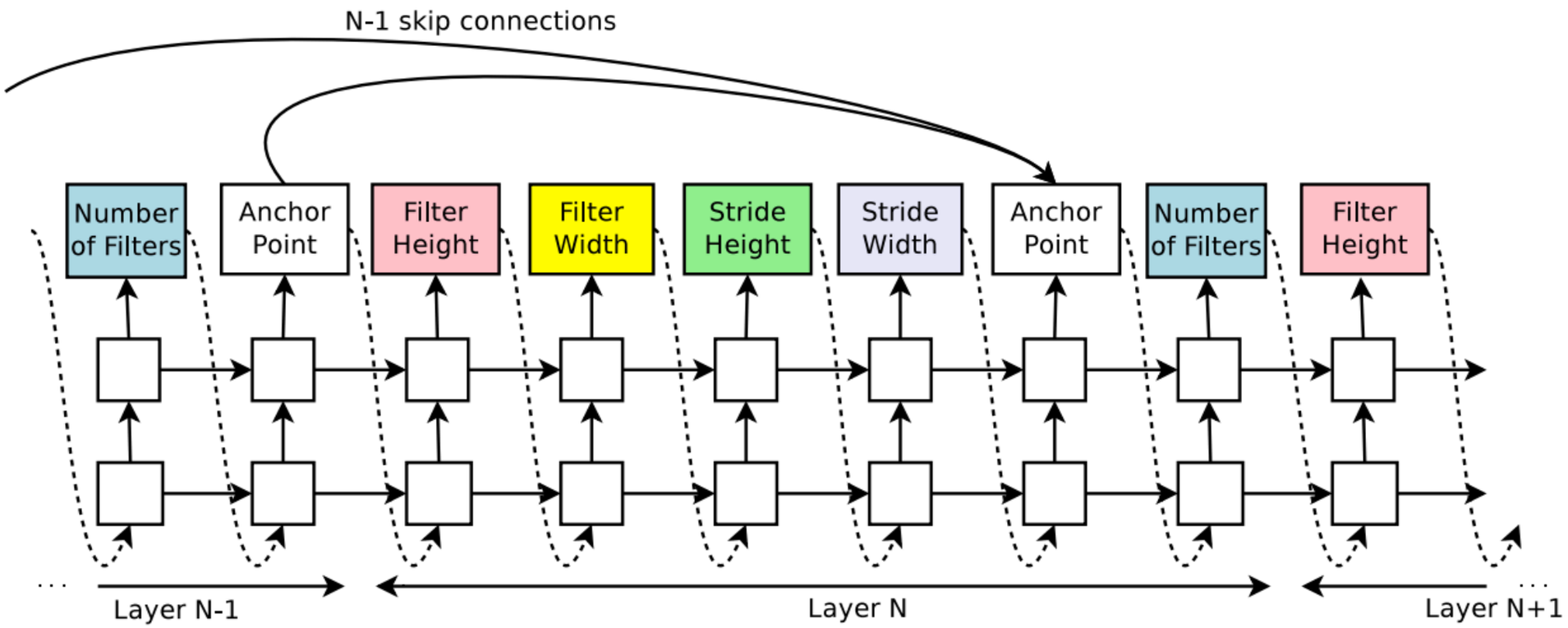
RNN policy network:

- generate architecture step by step
- evaluate to get reward
- update with policy gradient



2-layer LSTM (REINFORCE), chains of convolutional layers

- State of the art on CIFAR-10, Penn Treebank
- 800 GPUs for 3-4 weeks, 12800 architectures

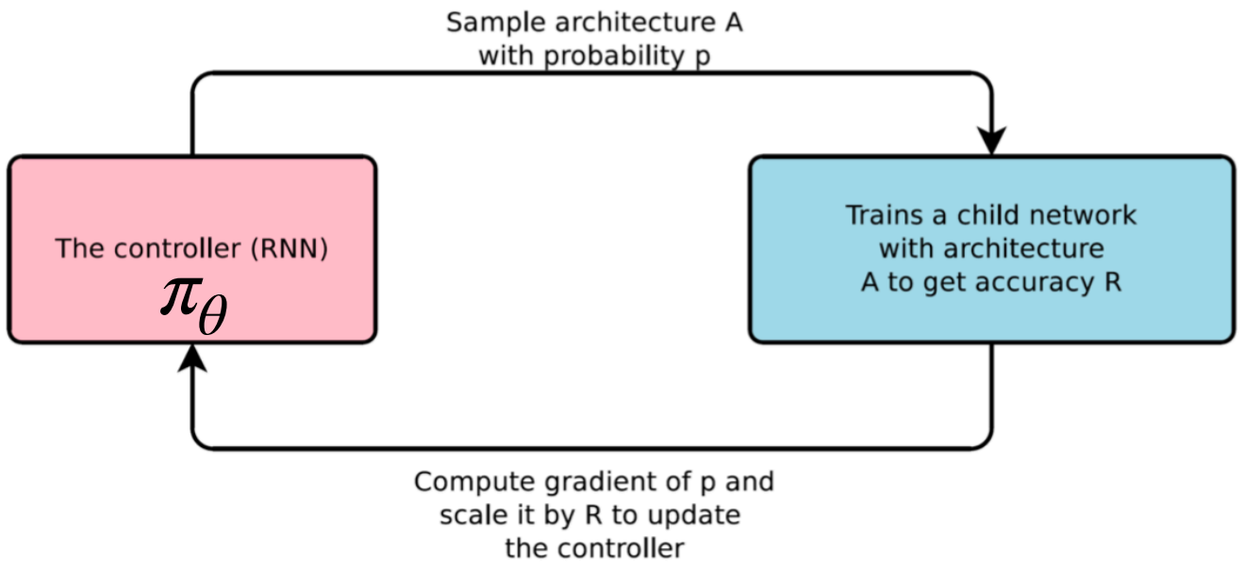


Zoph et al 2018

NAS with Reinforcement learning

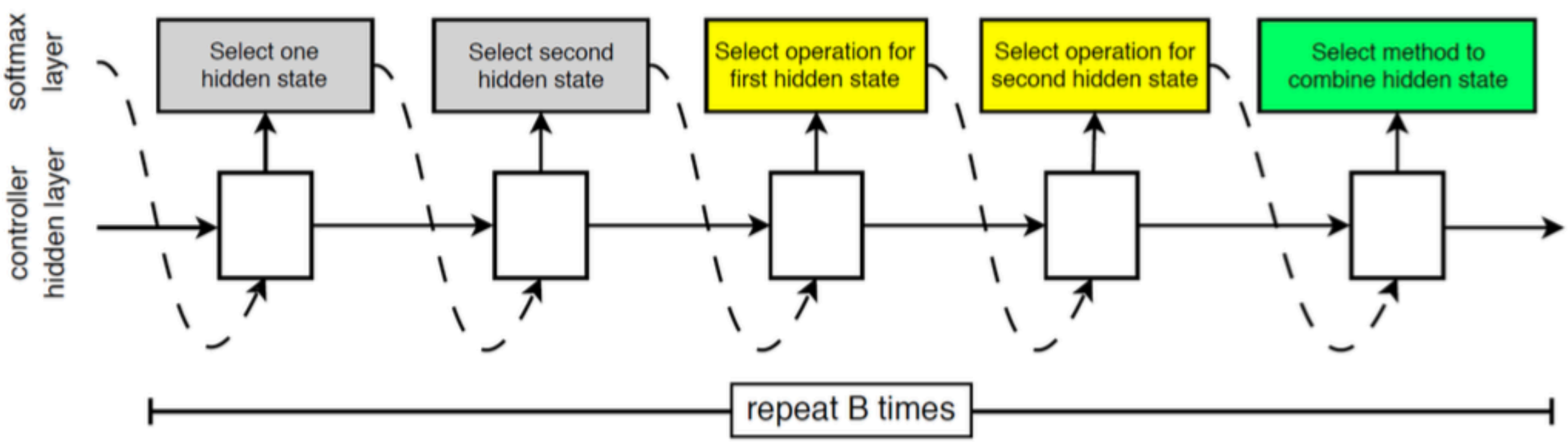
RNN policy network:

- generate architecture step by step
- evaluate to get reward
- update with policy gradient



1-layer LSTM (PPO), cell space search

- State of the art on ImageNet
- 450 GPUs, 20000 architectures

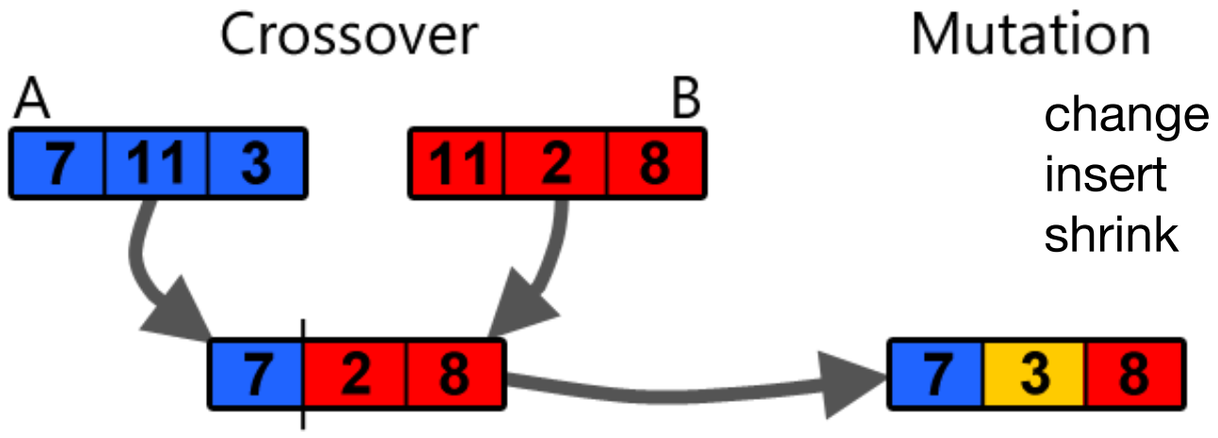


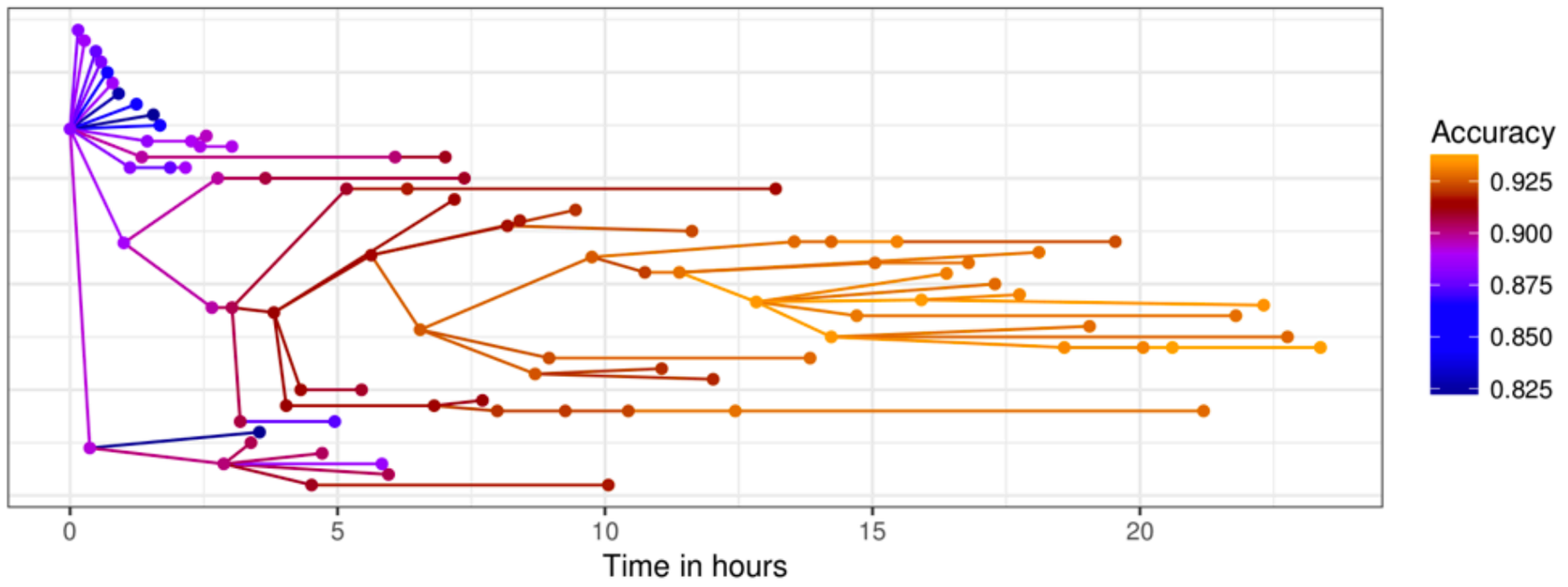
TPOT *Olson, Moore 2016, 2019*

GAMA *Gijsbers, Vanschoren 2019*

Evolving pipelines

- Start with initial architecture
- Best architectures *evolve*: cross-over or mutation
  - Per generation (TPOT)
  - Asynchronous (GAMA)
- Tree-shaped pipelines
- No fixed maximal length





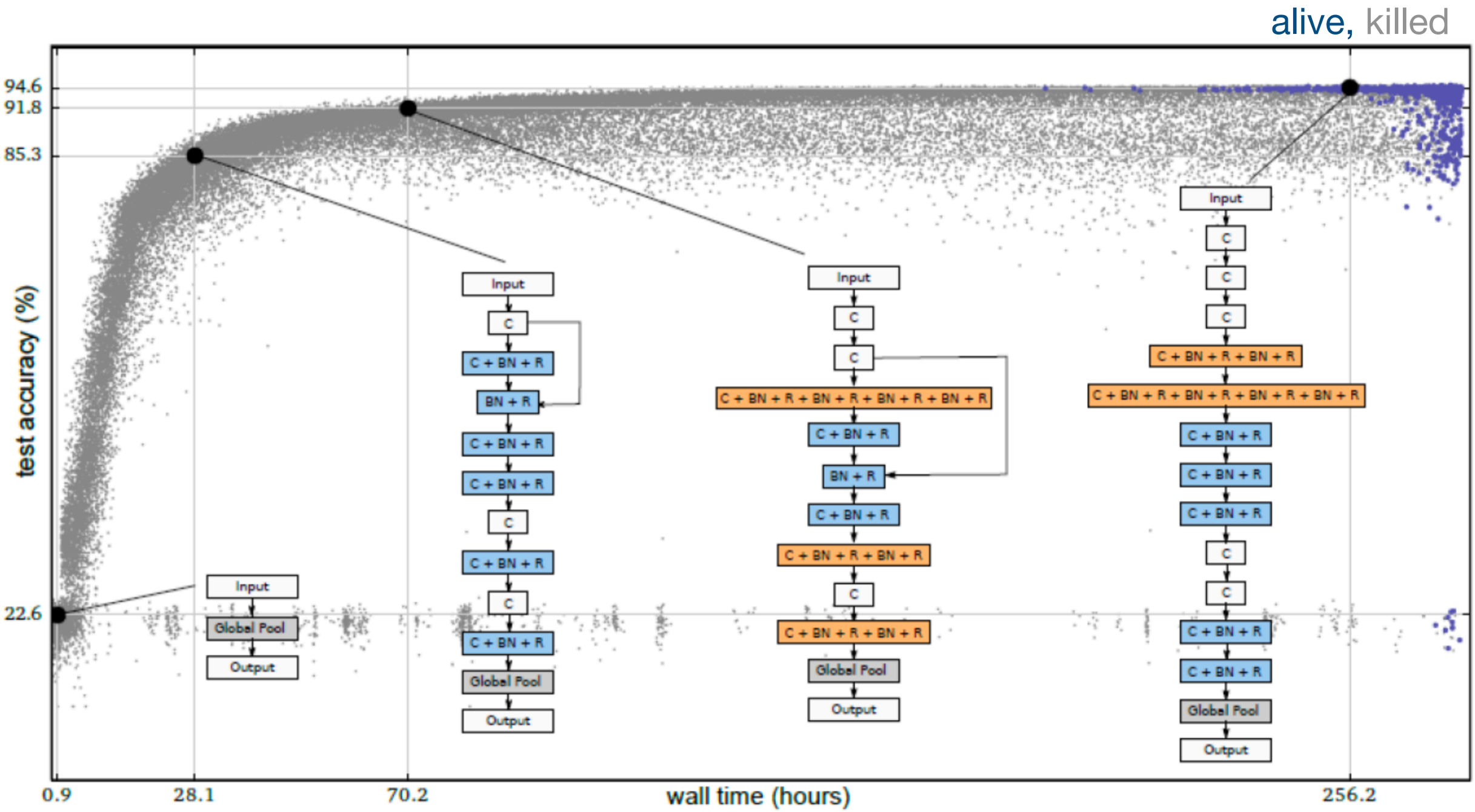
+ more flexible

- larger search space, can be slower

Neuro-evolution

Real et al 2017
Miikkulainen et al 2017
Wistuba et al 2018

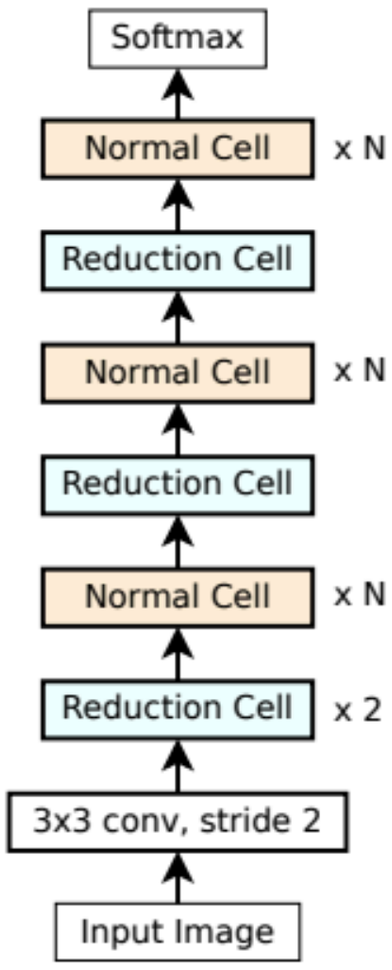
- same idea: learn the neural architecture through evolution
- mutations: add, change, remove layer

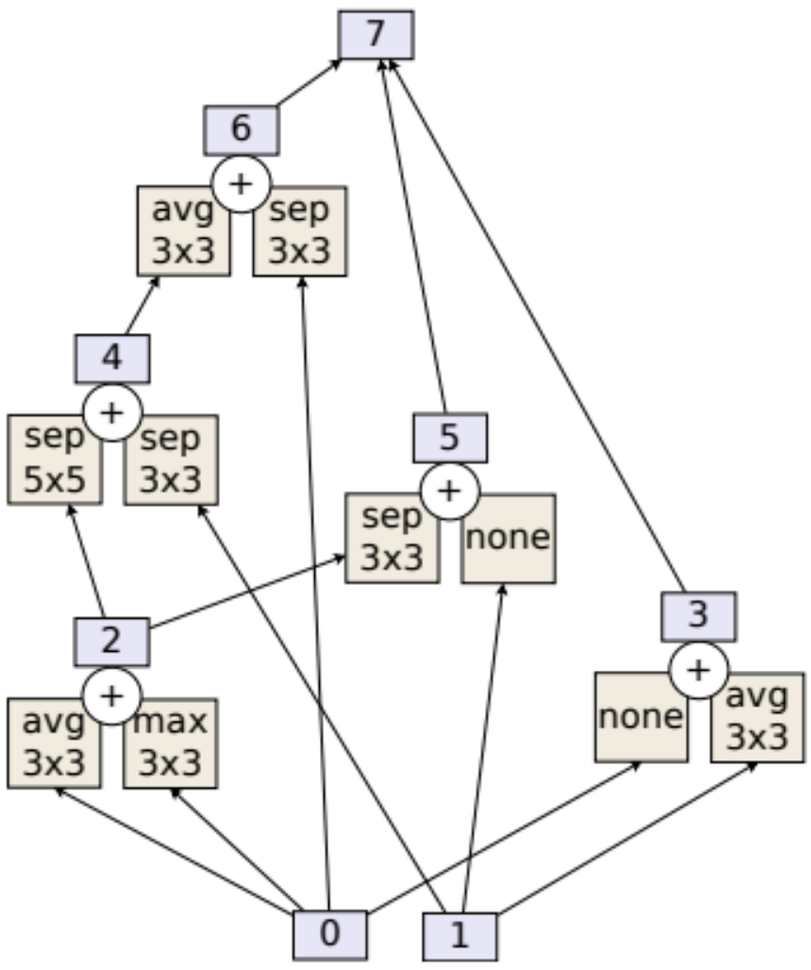


Neuro-evolution

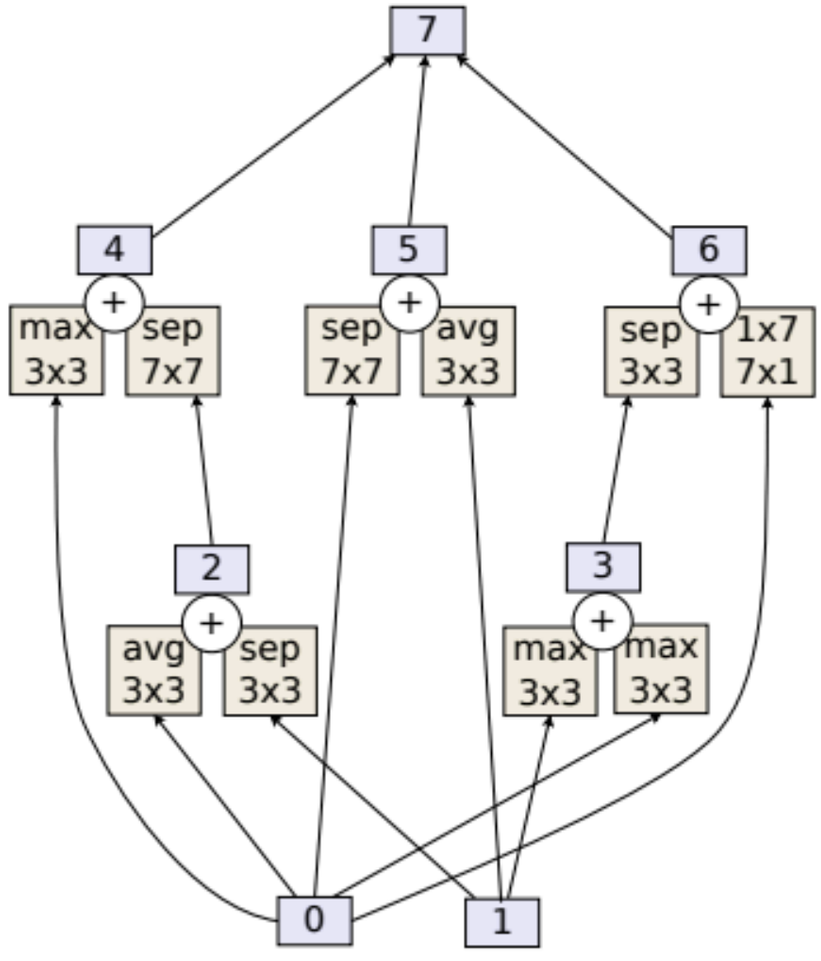
AmoebaNet: State of the art on ImageNet, CIFAR-10

- Cell search space, aging evolution (kill oldest networks)





normal cell

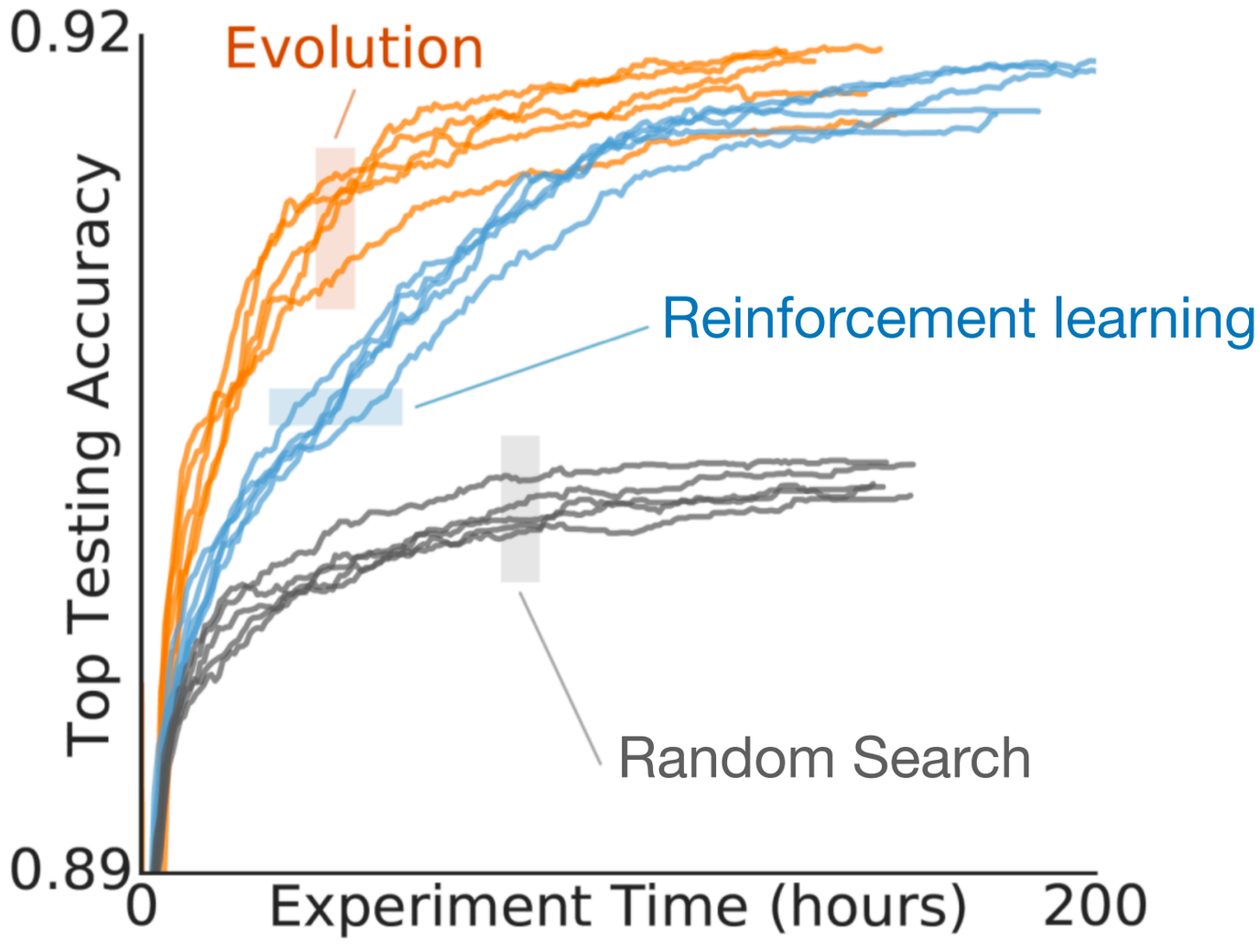


reduction cell

Neuro-evolution

AmoebaNet: State of the art on ImageNet, CIFAR-10

- Cell search space + aging evolution (kill oldest networks)
- More efficient than reinforcement learning



Grammars

- Rules define how pipeline could be built
  - Can include rules with background/domain knowledge
  - Can be combined with e.g. evolution (to avoid invalid pipelines)

production rule *optional* *non-terminal*

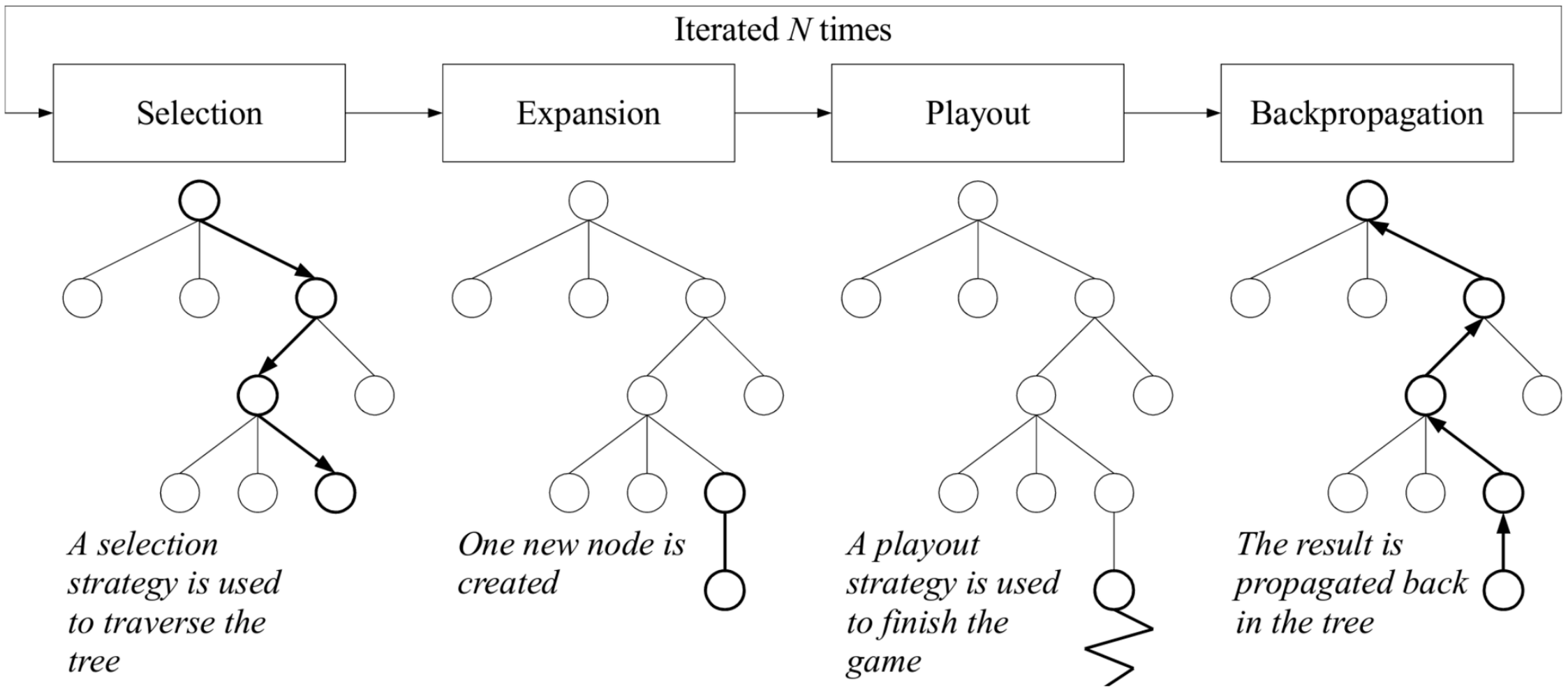
```
<Start> ::= [<Pre-processing>] <Algorithm>
<Pre-processing> ::= [<Imputation>] <DimensionalityDefinition>
<Imputation> ::= Mean | Median | Max
<DimensionalityDefinition> ::= <FeatureSelection>   [<FeatureConstruction>]
                               [<FeatureSelection>] <FeatureConstruction>
<FeatureSelection> ::= <Supervised> | <Unsupervised>
<Supervised> ::= SelectKBest <K> <score> | VarianceThreshold | [...]
<score> ::= f-classification | chi2
<K> ::= 1 | 2 | 3 | [...] | NumberOfFeatures - 1
<perc> ::= 1 | 2 | 3 | [...] | 99
<Unsupervised> ::= PCA | FeatureAgglomeration <affinity> | [...]
<affinity> ::= Euclidian | L1 | L2 | Manhattan | Cosine
<FeatureConstruction> ::= PolynomialFeatures
<Algorithm> ::=  <NaiveBayes> | <Trees> | [...]
<NaiveBayes> ::= GaussianNB | MultinomialNB | BernoulliNB
<Trees> ::= DecisionTree | RandomForest | [...]
```

terminal

MOSAIC *[Rakotoarison et al. 2019]*
AlphaD3M *[Drori et al. 2018]*

Monte Carlo Tree Search (MCTS)

- Select partial pipeline from tree of all pipelines
- Add one new operator, do random completion to estimate accuracy
- Update nodes according to performance

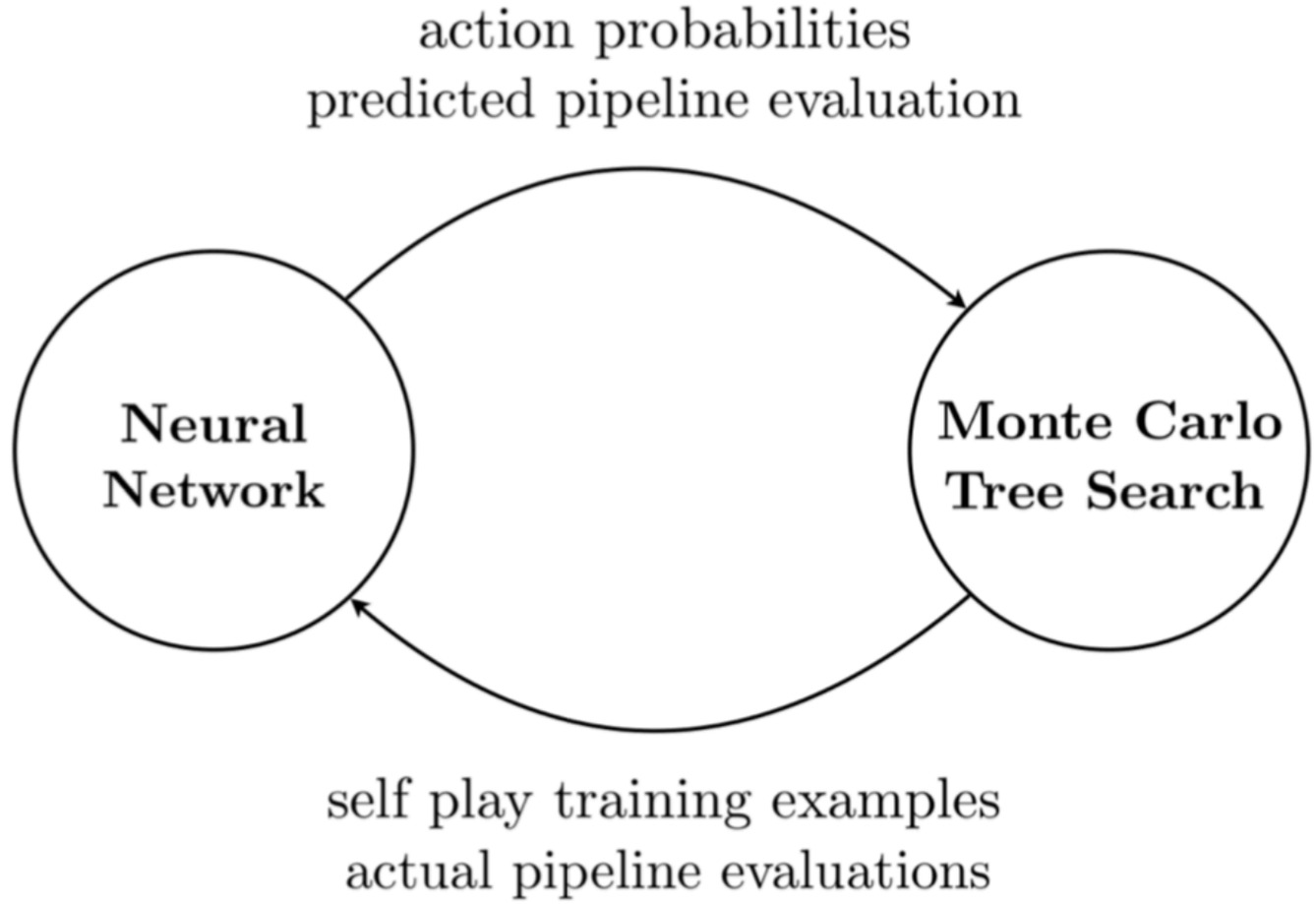


[Chaslot et al., 2008]

MOSAIC *[Rakotoarison et al. 2019]*
AlphaD3M *[Drori et al. 2018]*

MCTS + reinforcement learning

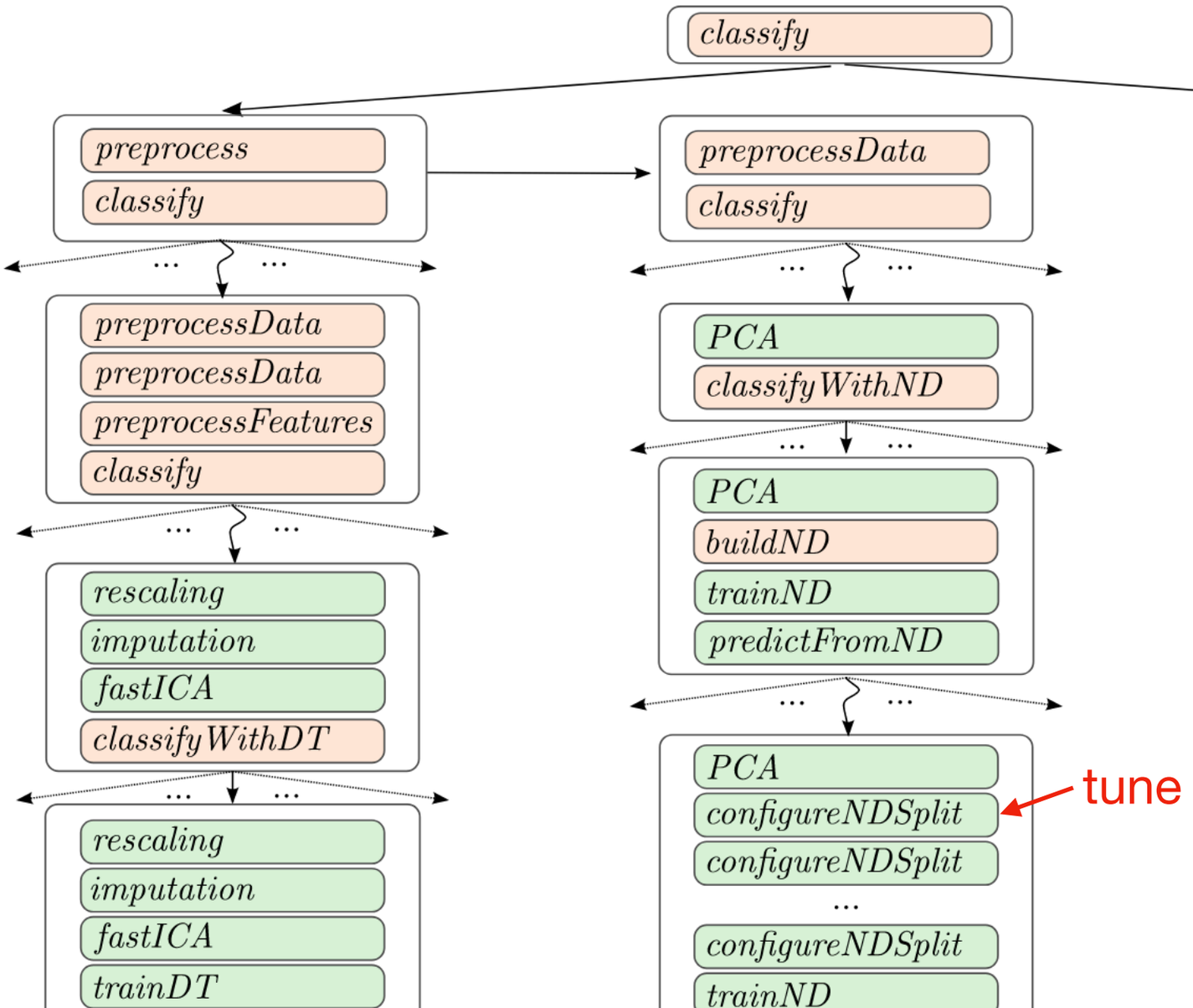
- Build pipelines by inserting, deleting, replacing components (actions)
- *Self-play:*
  - Monte Carlo Tree Search builds pipelines given action probabilities
  - Neural network (LSTM) Predicts pipeline performance



ML-Plan [Mohr et al. 2018]
P4ML [Gil et al. 2018]

Hierarchical planning

- Use planning to ‘plan' pipeline, e.g. with best first search
- Use random path completion (as in MCTS) to evaluate each node

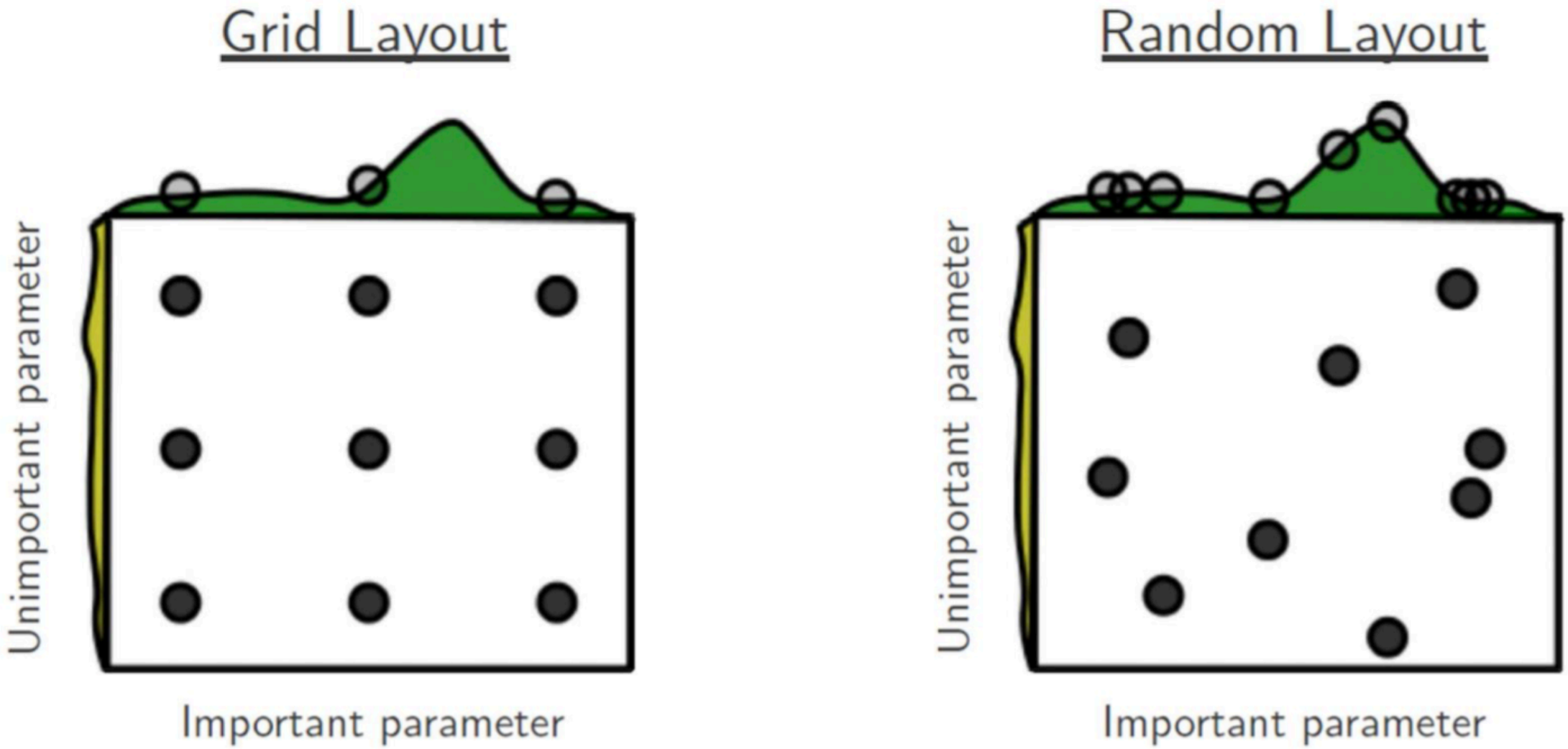


Hyperparameter Optimization

Searching the hyperparameter space ***efficiently***

Random search

- Handles unimportant dimensions better than grid search
- Easily parallelizable, but uninformed (no learning)

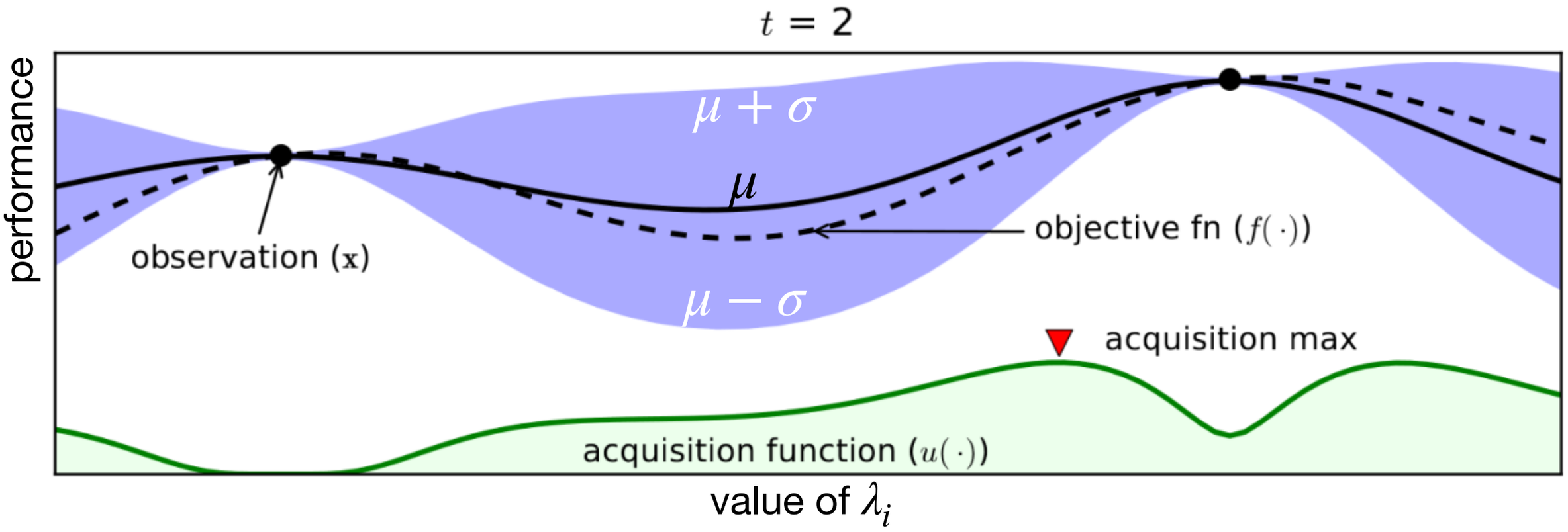


Bergstra & Bengio, JMLR 2012

Mockus [1974]

Bayesian Optimization

- Start with a few (random) hyperparameter configurations
- Fit a ***surrogate model*** to predict other configurations
- Probabilistic regression: mean μ and standard deviation σ (blue band)
- Use an ***acquisition function*** to trade off exploration and exploitation, e.g. Expected Improvement (EI)
- Sample for the best configuration under that function



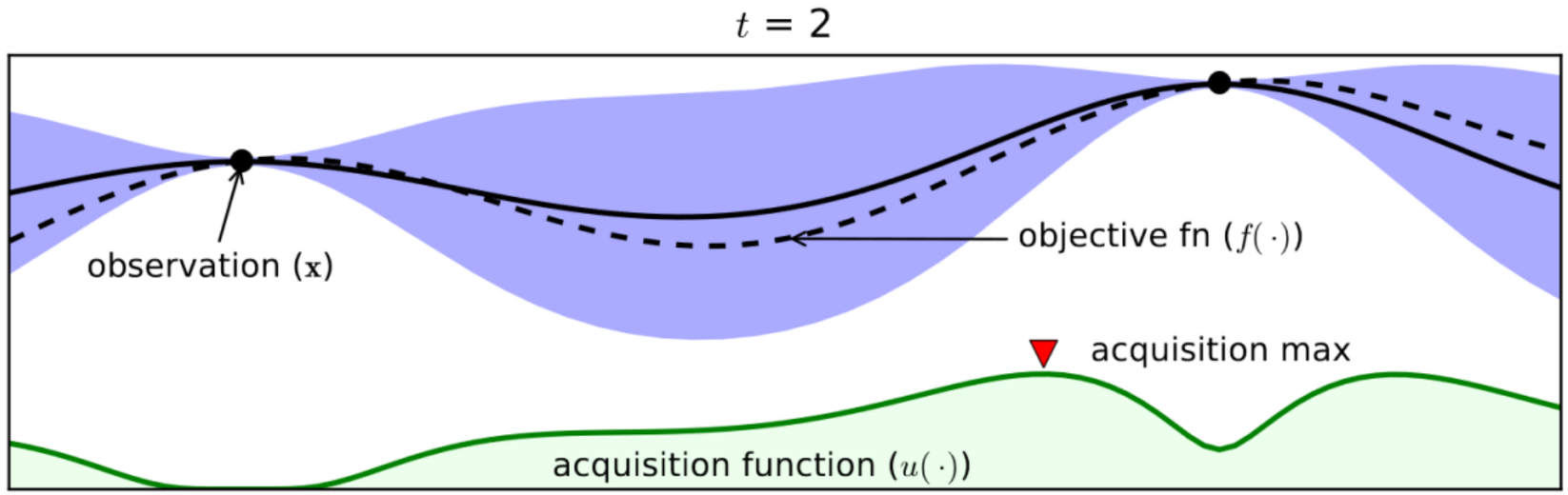
Bayesian Optimization

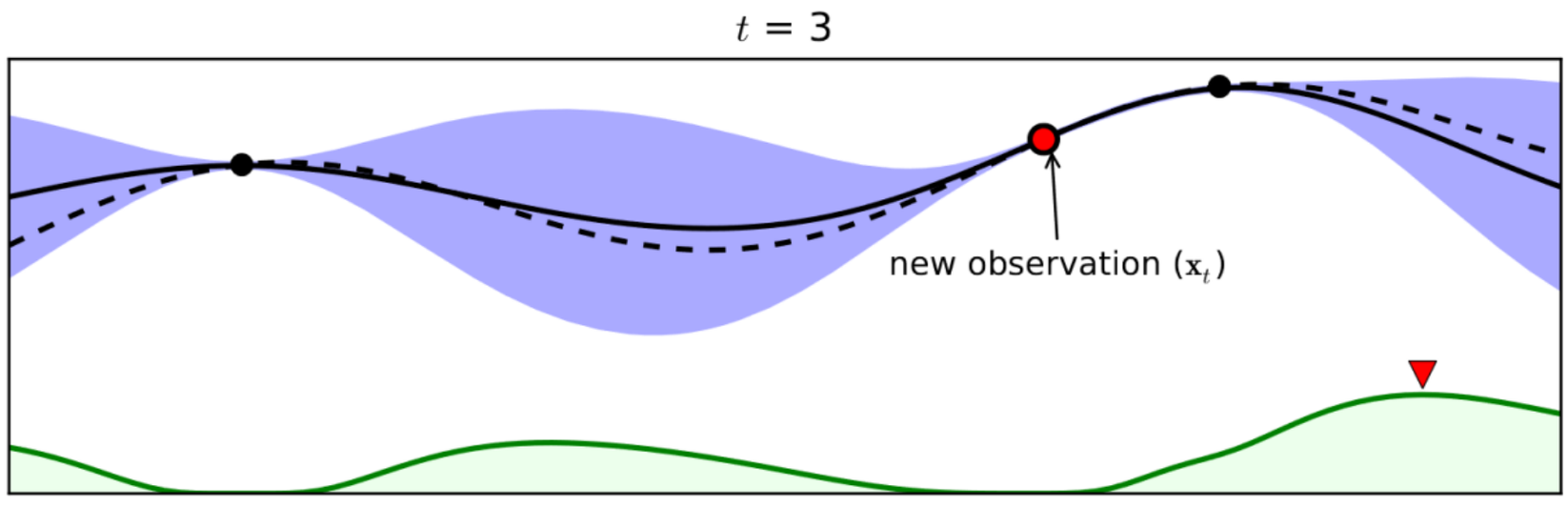
- Repeat until some stopping criterion:
  - Fixed budget
  - Convergence
  - EI threshold

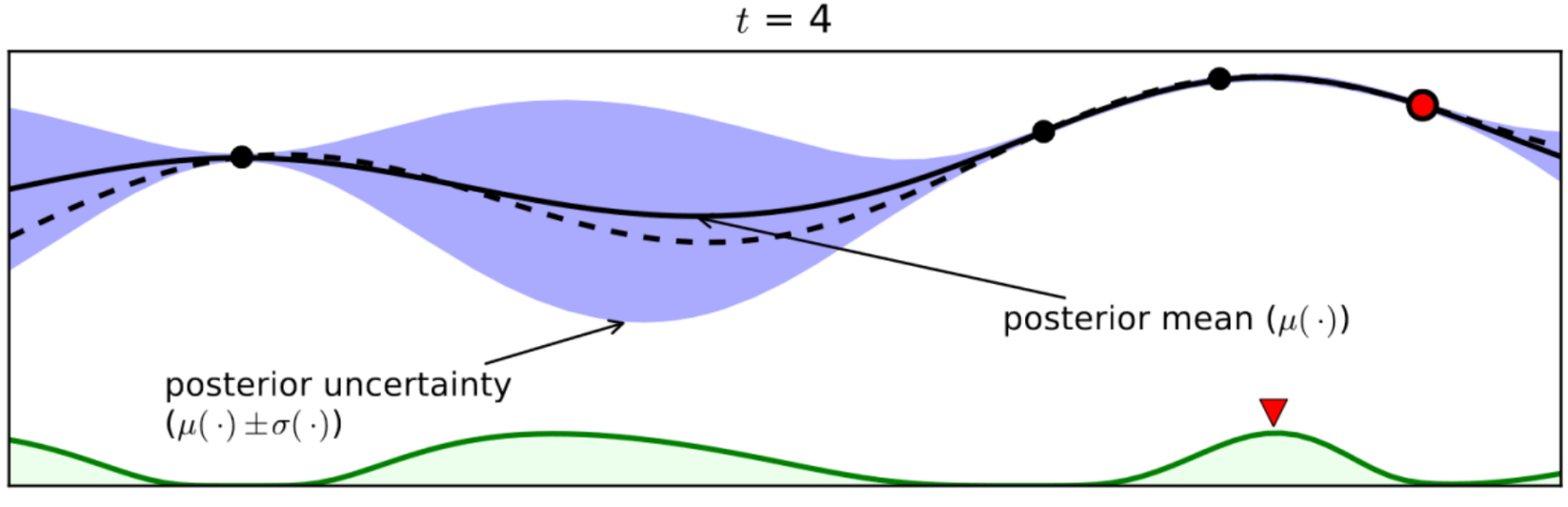
- Theoretical guarantees

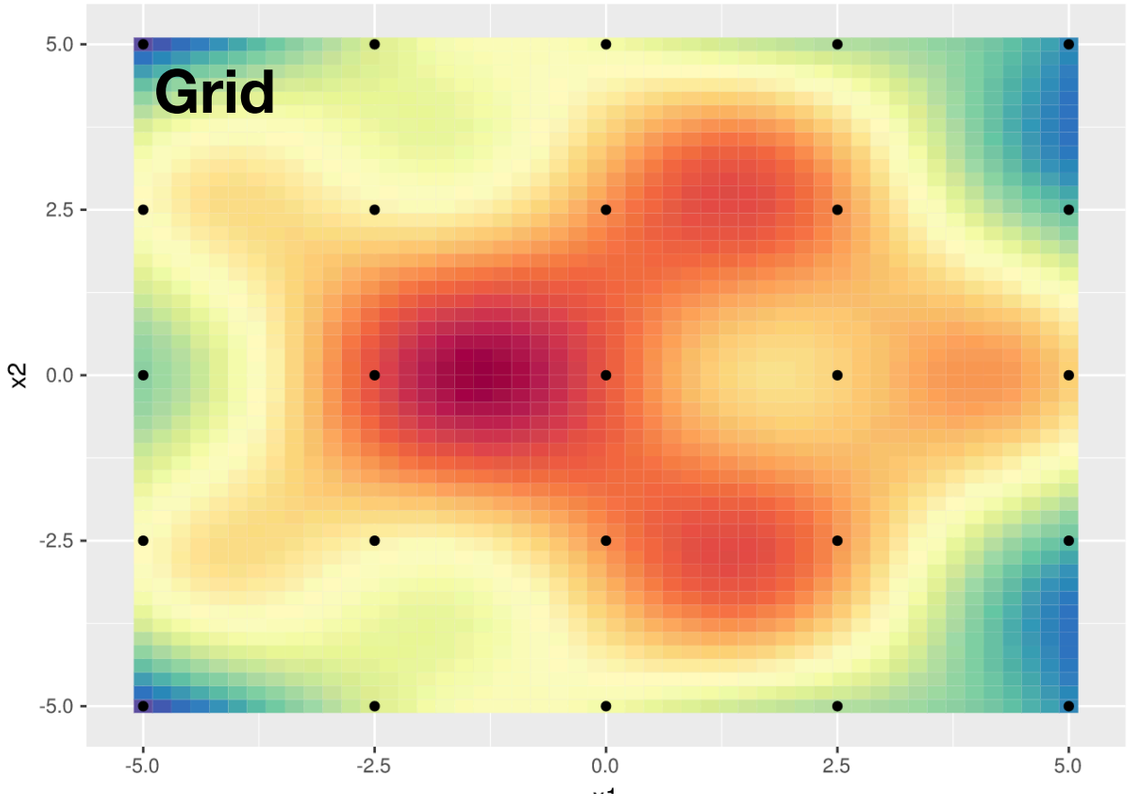
  Srinivas et al. 2010, *Freitas et al. 2012*, *Kawaguchi et al. 2016*

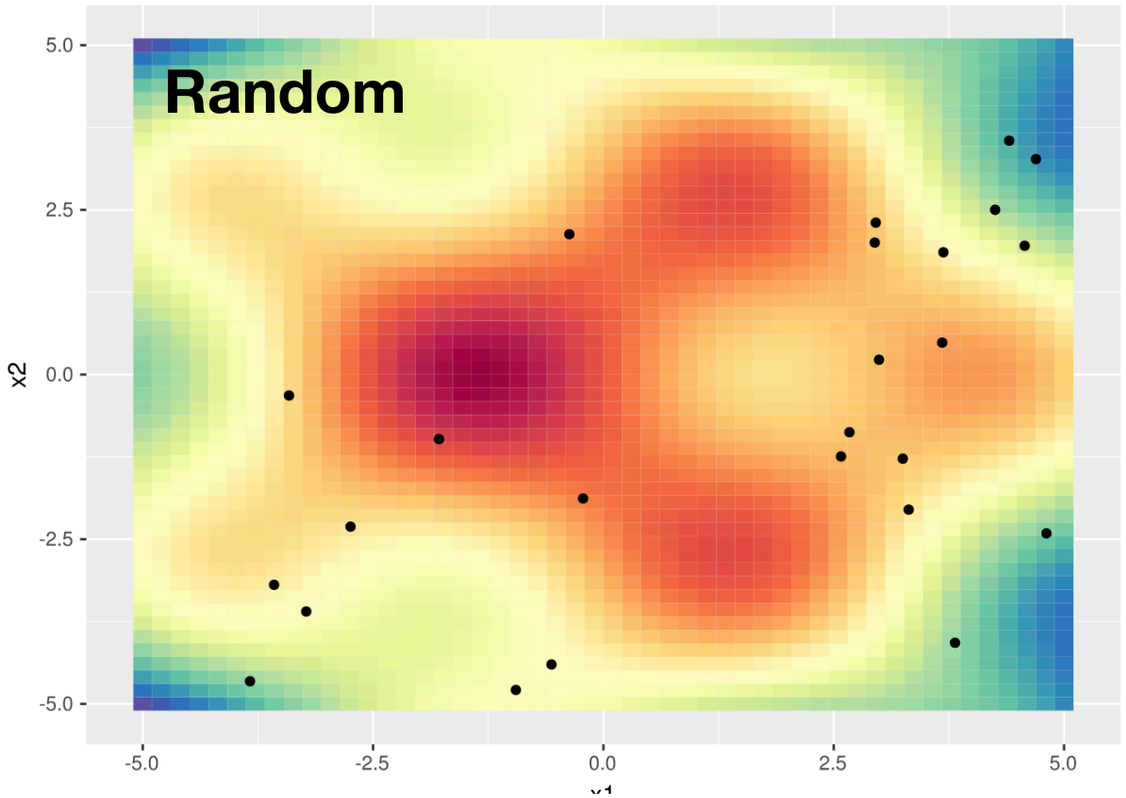
- Also works for non-convex, noisy data
- Used in AlphaGo

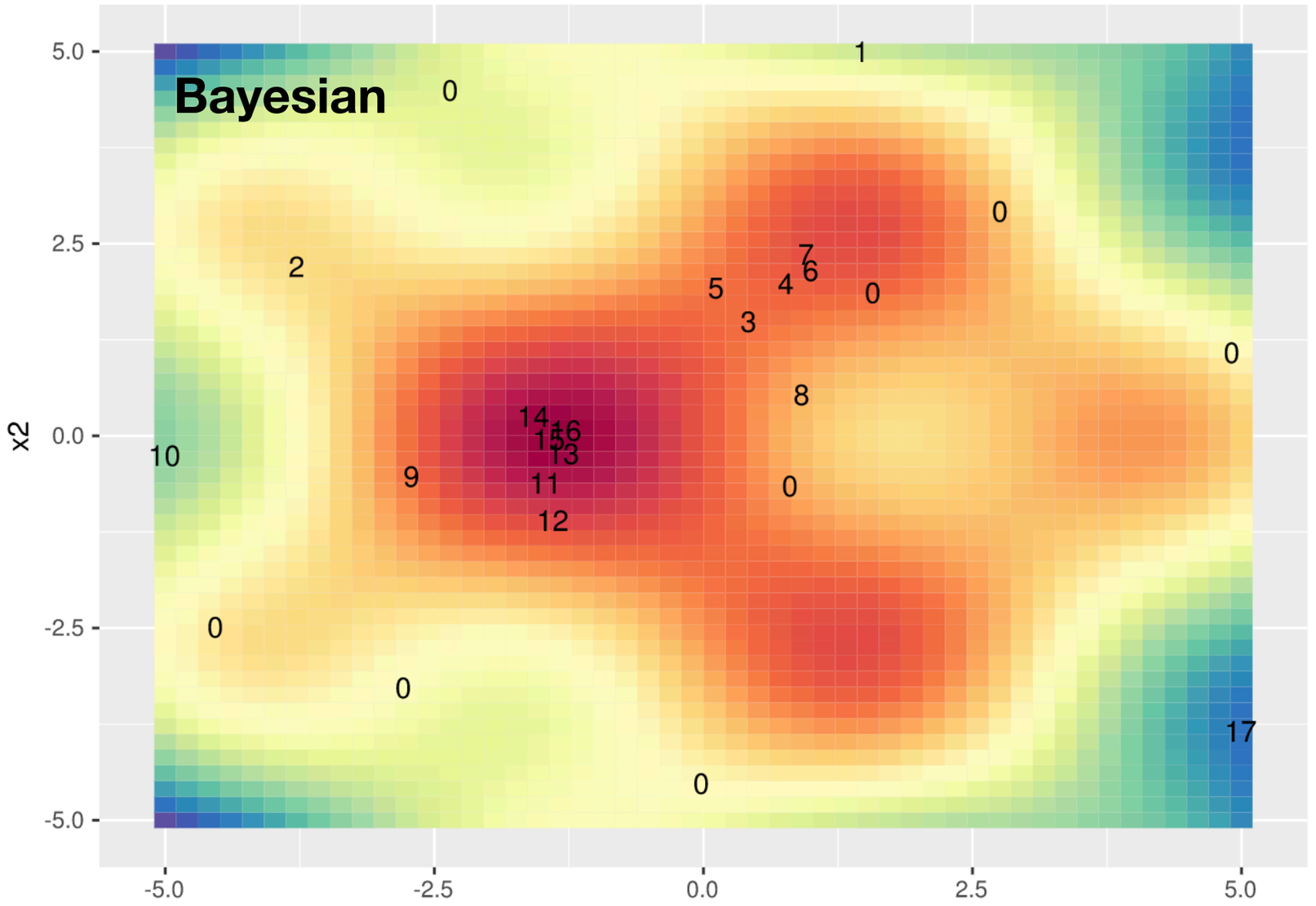






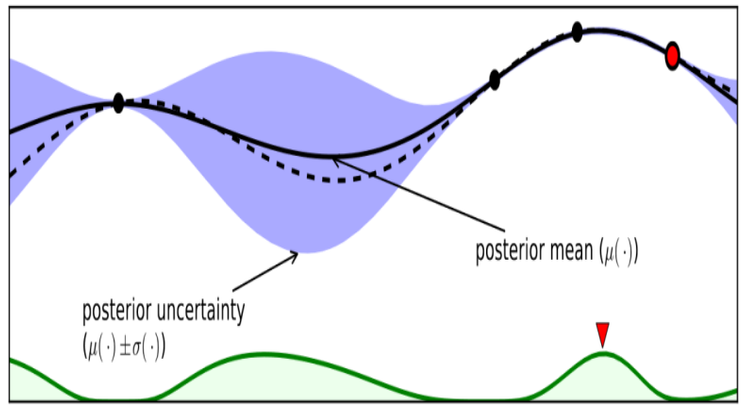

Grid
5.0
2.5
0.0
-2.5
-5.0
x2
-5.0
-2.5
0.0
2.5
5.0


Random
5.0
2.5
0.0
-2.5
-5.0
x2
-5.0
-2.5
0.0
2.5
5.0


Bayesian
5.0
2.5
0.0
-2.5
-5.0
x2
-5.0
-2.5
0.0
2.5
5.0
1
0
2
0
7
6
5
4
0
3
0
8
14
16
15
13
10
9
11
0
12
0
0
17

Bayesian Optimization: which surrogate?

Gaussian processes *[skopt]*

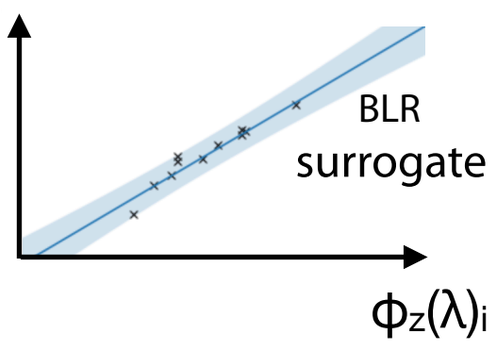


+ handles uncertainty, extrapolates well

+ ideal for few numeric hyperparameters

- scales badly (cubic), try random embeddings *[Wang et al. 2013]*

Bayesian Linear Regression *[Snoek et al. 2015]* *[Amazon AutoML]*



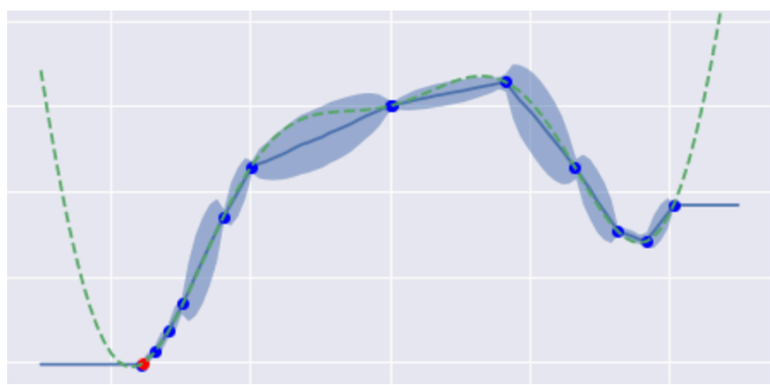
+ Scalable, Bayesian, - requires good basis expansion

Bayesian neural networks *[Springenberg et al. 2016]*

+ Bayesian - Scalability?

[autosklearn, autoWEKA]

Random Forests *[Hutter et al. 2011, Feurer et al. 2015]*



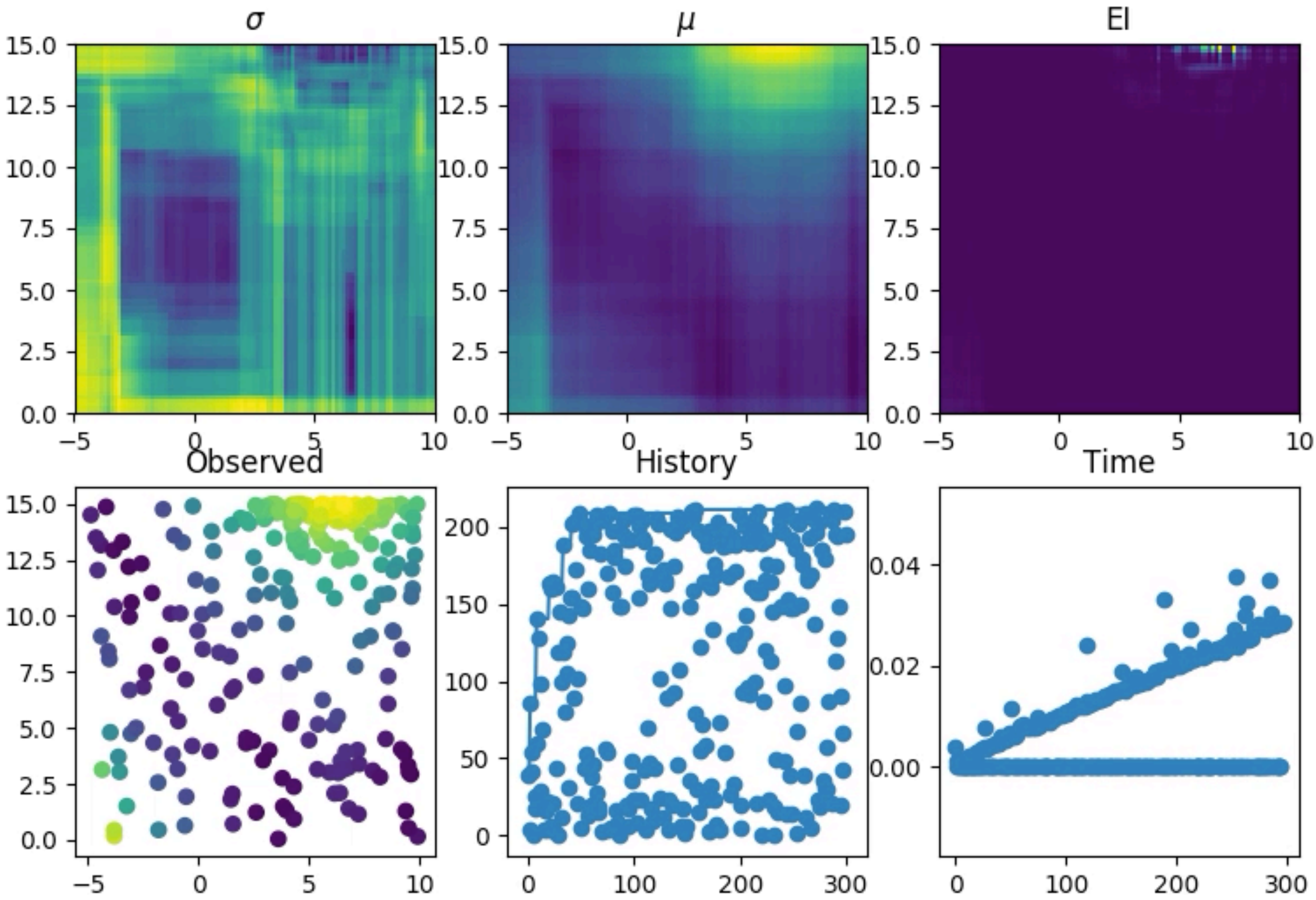
+ scalable, handles conditional hyperparams

- bad uncertainty estimates, extrapolation

Boosting + quantile regression *[van Hoof, Vanschoren 2019]* *[HyperBoost]*

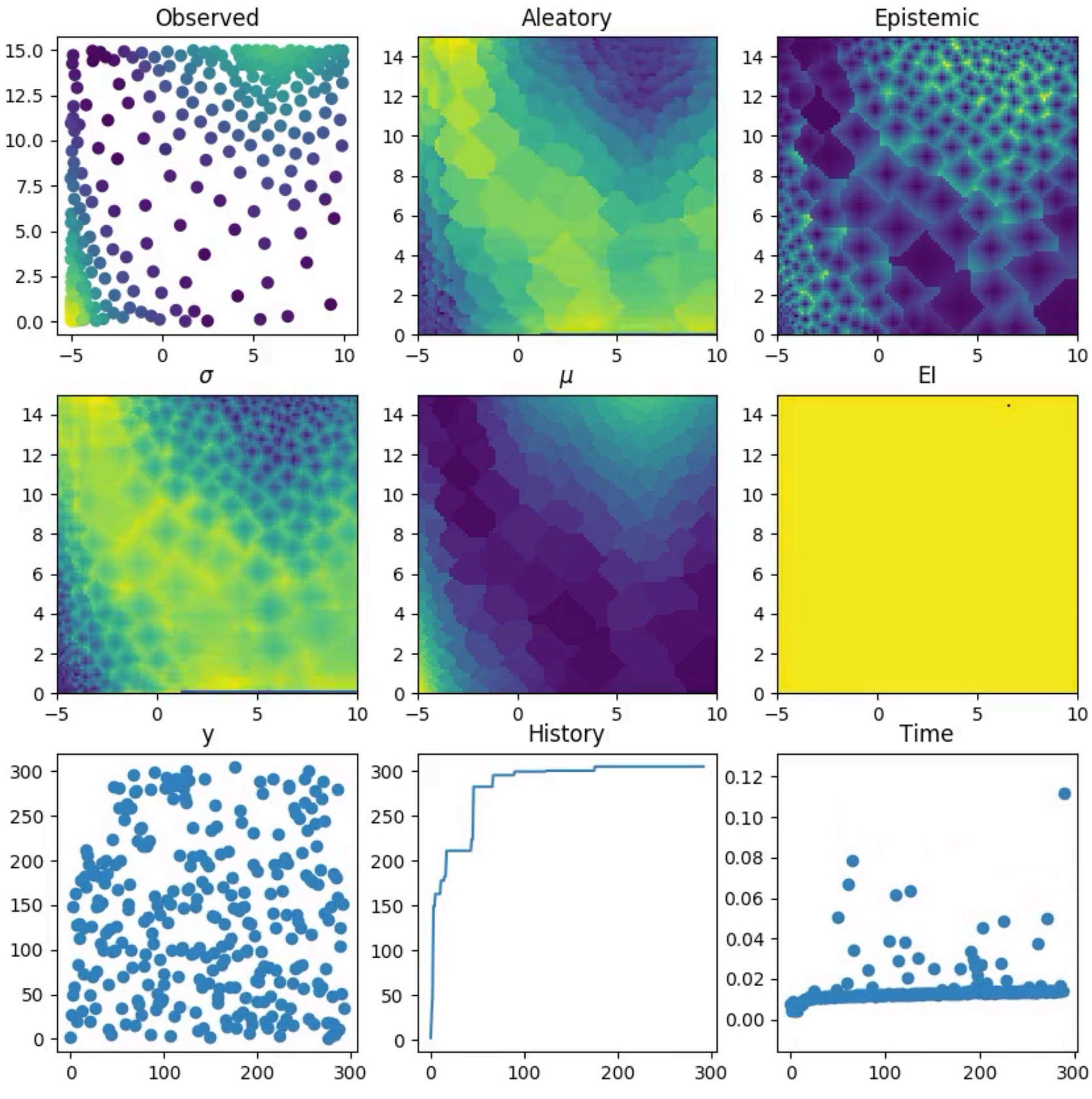
+ better fit than RF, handles conditionals, adapts to drift - new

Random Forest Surrogate



animation by Jeroen van Hoof

Gradient Boosting Surrogate



Tree of Parzen Estimators

1.Test some hyperparameters

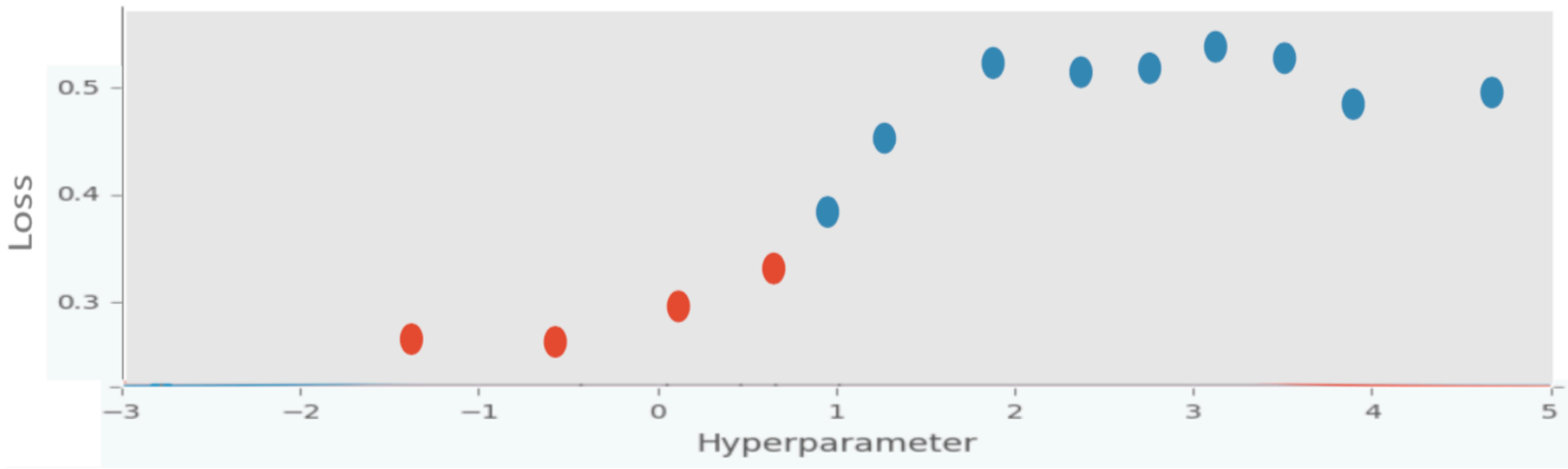
2. Separate into good and bad hyperparameters (with some quantile)

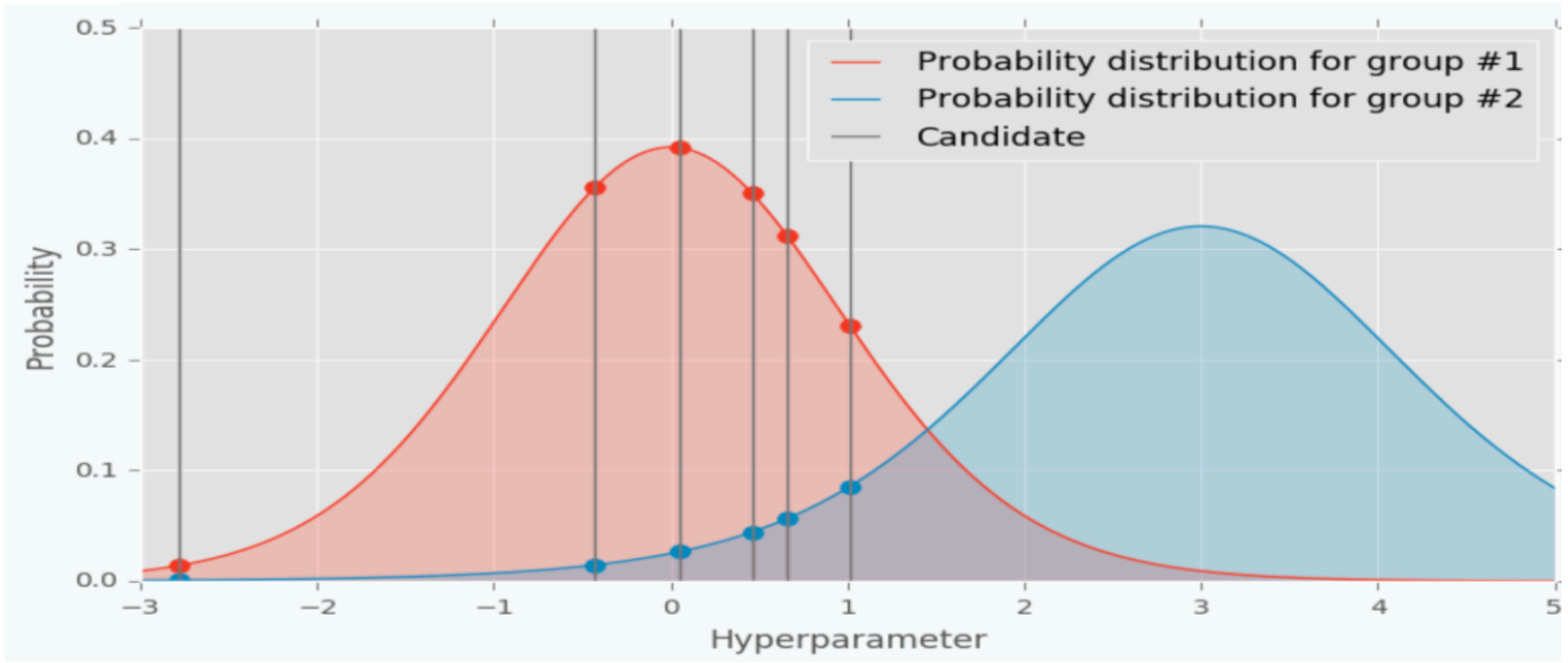
3. Fit non-parametric KDE for $p(\lambda = \text{good})$ and $p(\lambda = \text{bad})$

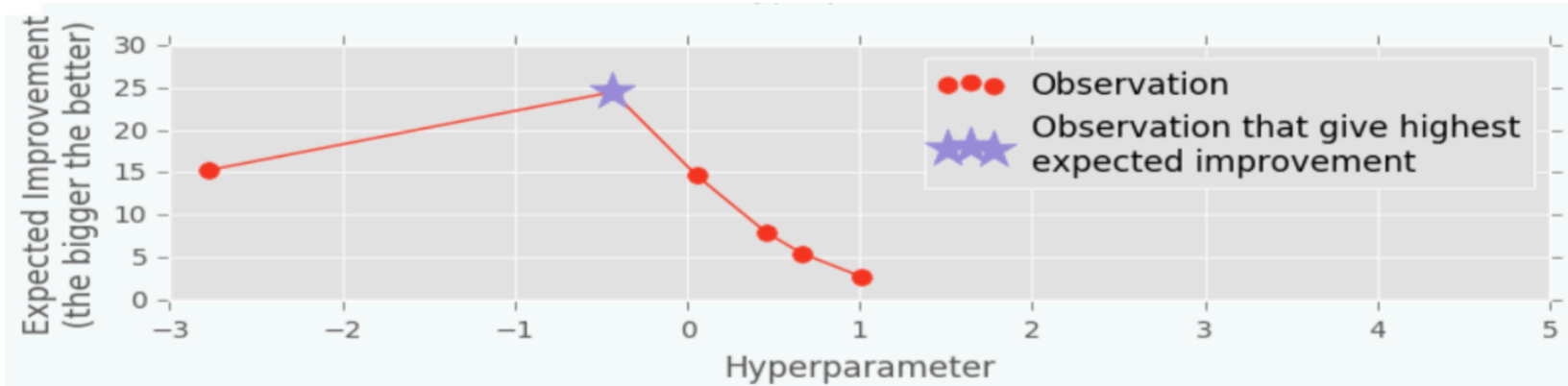
4. For a few samples, evaluate $\dfrac{p(\lambda = \text{good})}{p(\lambda = \text{bad})}$
Shown to be equivalent to EI!

Efficient, **parallelizable**, robust, but less sample efficient than GPs

Used in HyperOpt-sklearn
[Komer et al. 2019]



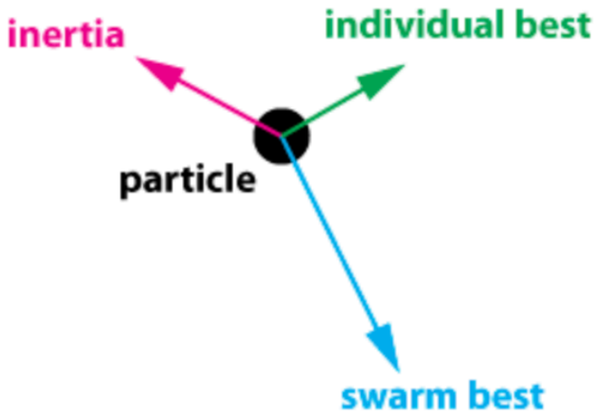


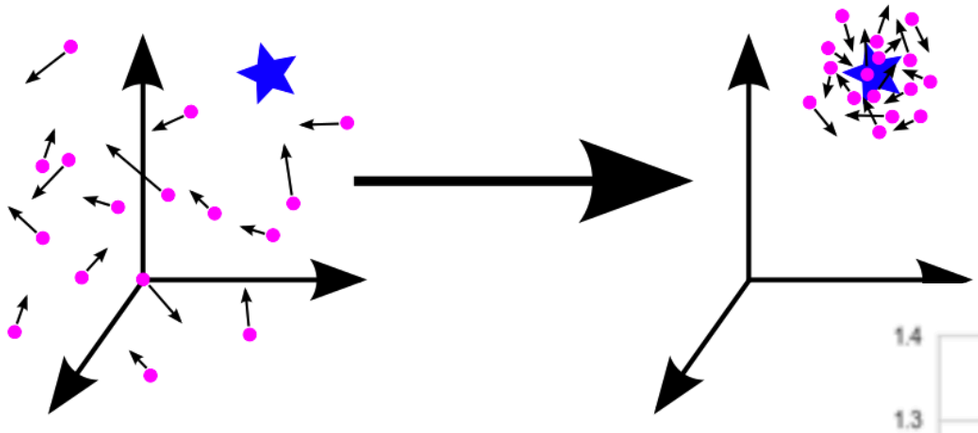


Population-based methods

- Less sample efficient, but easy to parallelize, and adapts to changes
- Genetic programming *Olson, Moore 2016, 2019*
  - Mutations: add, mutate/tune, remove HP

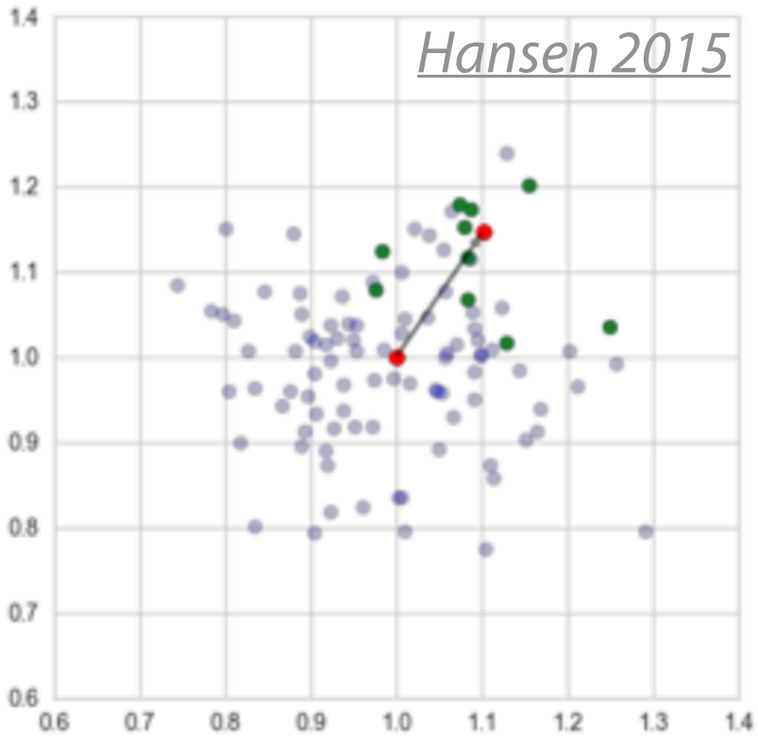
- Particle swarm optimization *Mantovani et al 2015*





- Covariance matrix adaptation evolution (CMA-ES)
  - Purely continuous, expensive
  - Very competitive to optimize deep neural nets

[Loshilov, Hutter 2016]

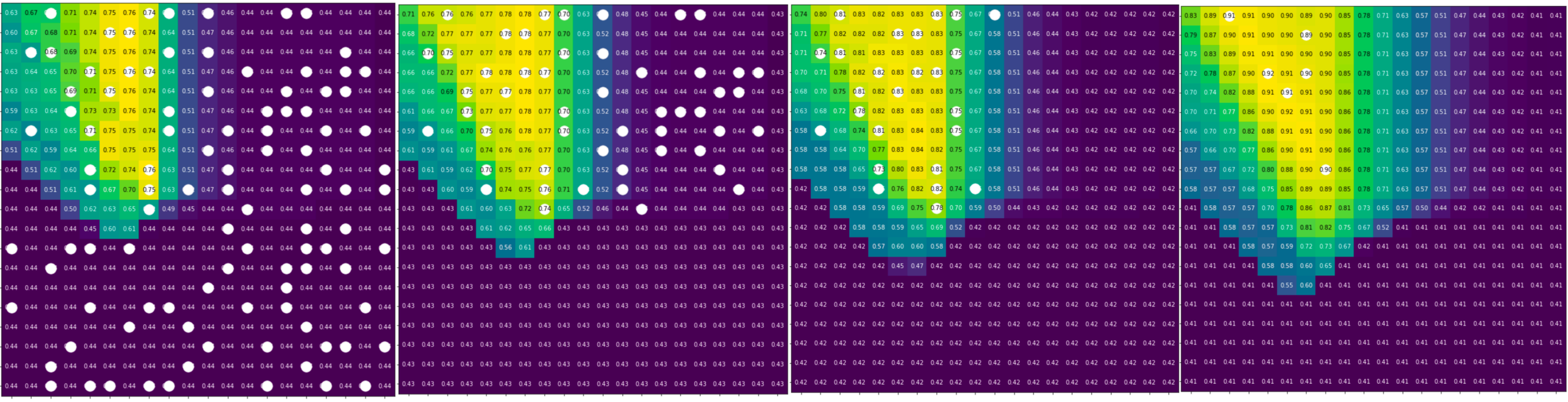


Jamieson & Talwalkar 2016

Multi-fidelity optimization

Successive halving:

- train on small subsets, infer which regions may be interesting to evaluate in more depth
- Randomly sample candidates and evaluate on a small data sample
- retrain the 50% best candidates on twice the data



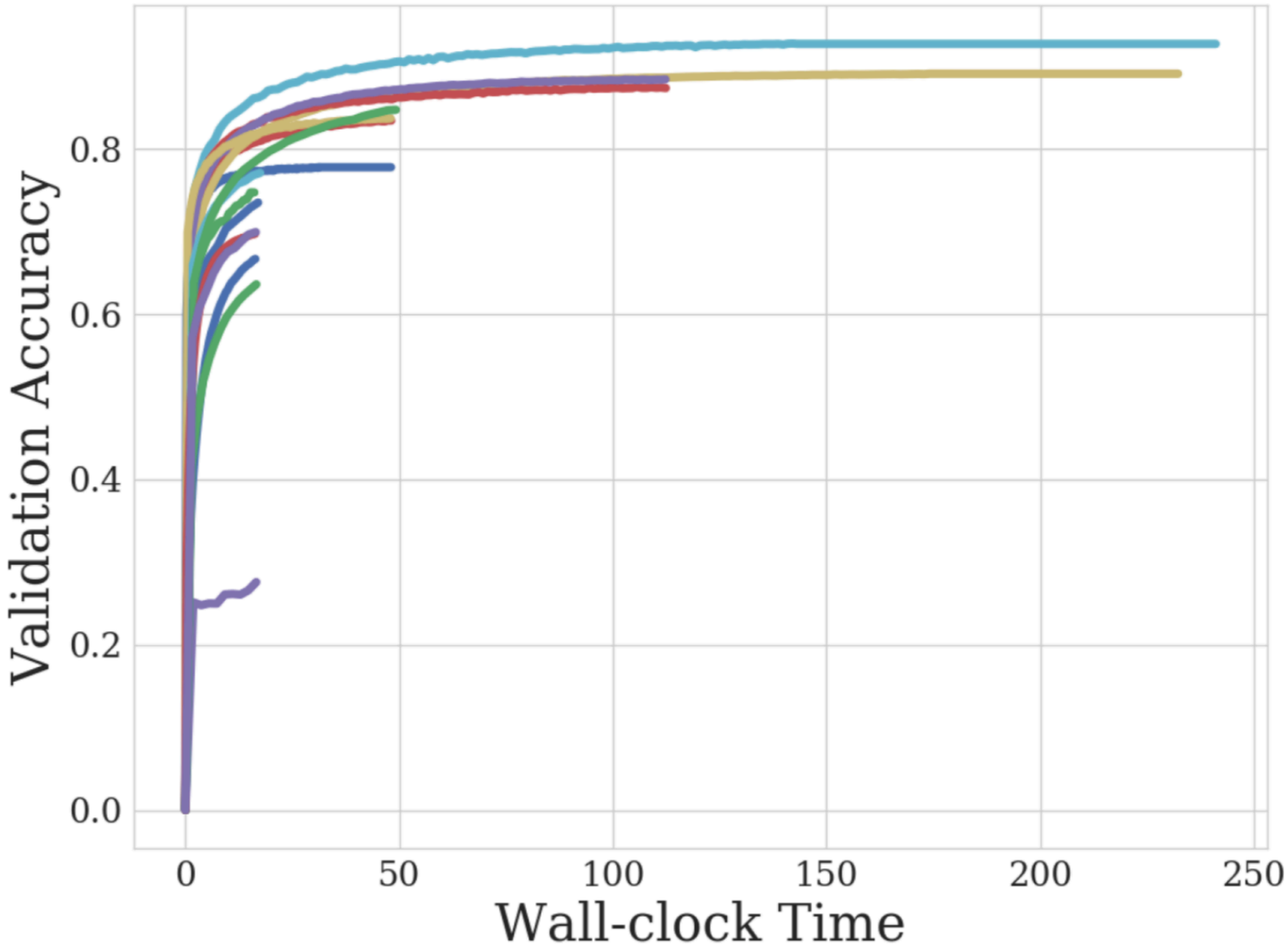
1/16 1/8 1/4 1/2

sample size

Multi-fidelity optimization

Successive halving:

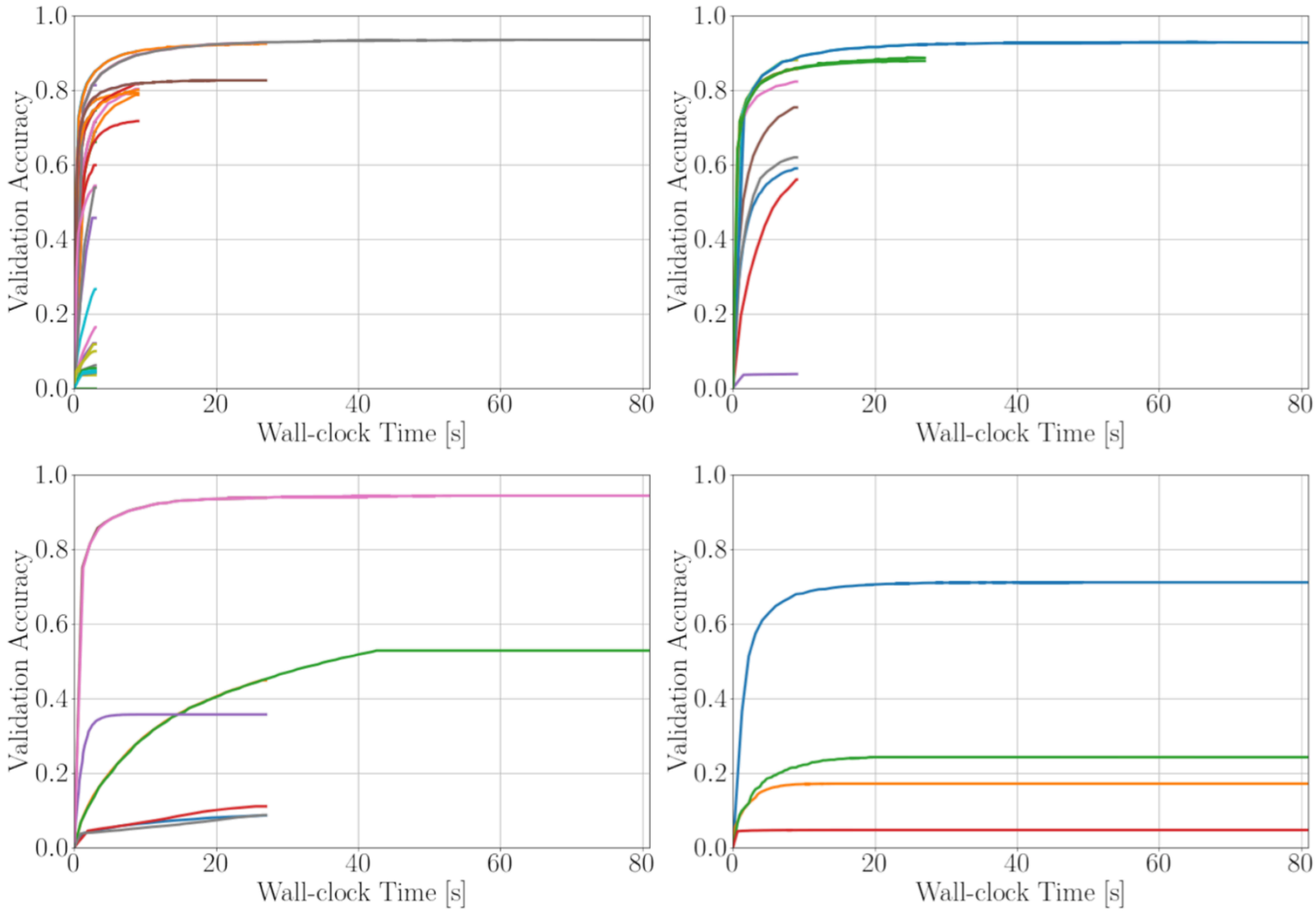
- Randomly sample candidates and evaluate on a small data sample
- retrain the best half candidates on twice the data



Multi-fidelity optimization

Hyperband (HB): Repeated, decreasingly aggressive successive halving

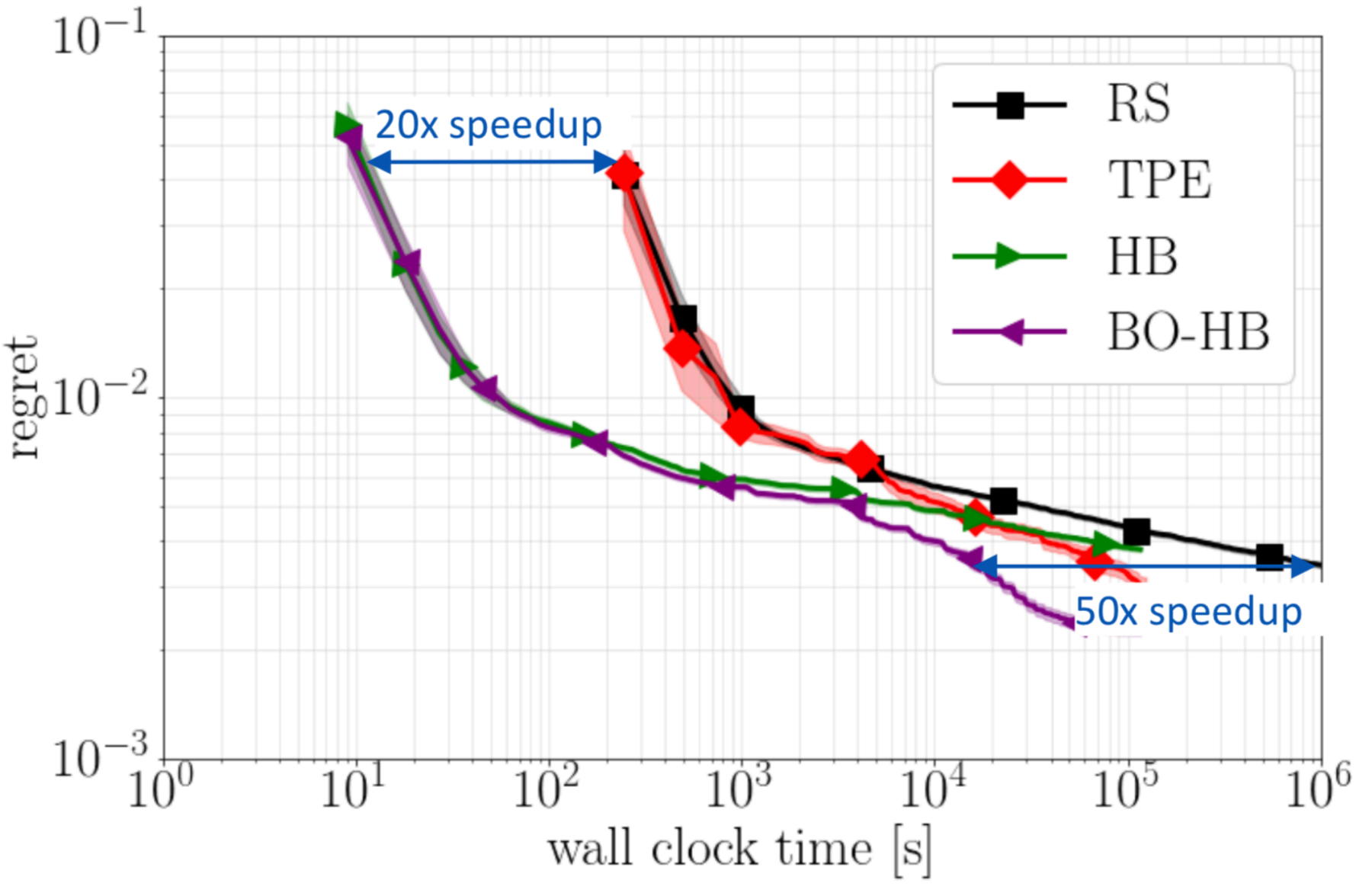
- Minimizes (doesn't eliminate) chance that candidate was pruned to early
- Strong anytime performance, easy to implement, scalable, parallelizable



Multi-fidelity optimization

Combined Bayesian Optimization and Hyperband (BO-HB)

- Choose which configurations to evaluate (with TPE)
- Hyperband: allocate budgets more efficiently
- Strong anytime and final performance



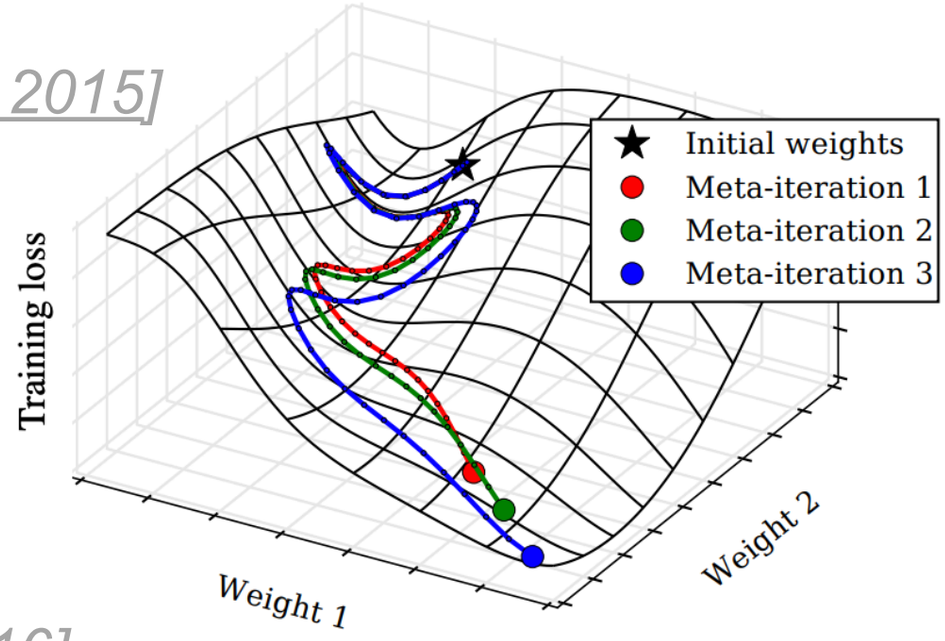
Hyperparameter gradient descent

Optimize neural network hyperparameters and weights simultaneously

- Bilevel program *[Franceschi et al 2018]*
  - Outer obj.: optimize λ
  - Inner obj.: optimize weights given λ

$$\min_\lambda \; \mathcal{L}_{val}\left(w^*(\lambda), \lambda\right)$$
$$s.t. \quad w^*(\lambda) = \operatorname{argmin}_w \mathcal{L}_{train}\left(w, \lambda\right)$$

- Derive through the entire SGD *[MacLaurin et al 2015]*
  - Get hypergradients wrt. validation loss
  - Expensive! But useful if you have many hyper parameters and high parallelism



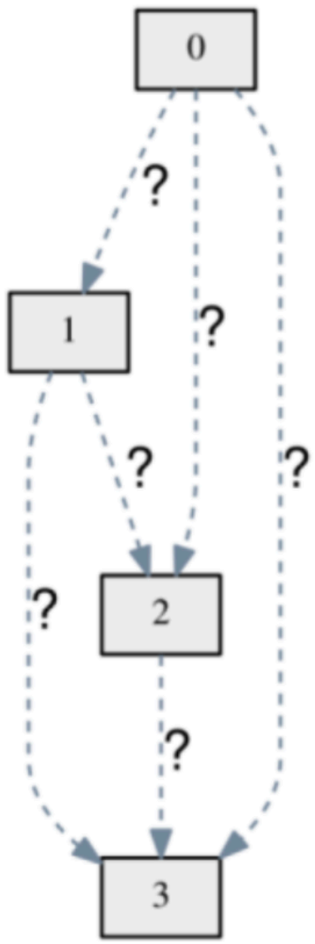
- Interleave optimization steps *[Luketina et al 2016]*
  - Alternate SGD steps for ω and λ

Hyperparameter gradient step w.r.t. $\nabla_\lambda \mathcal{L}_{val}$

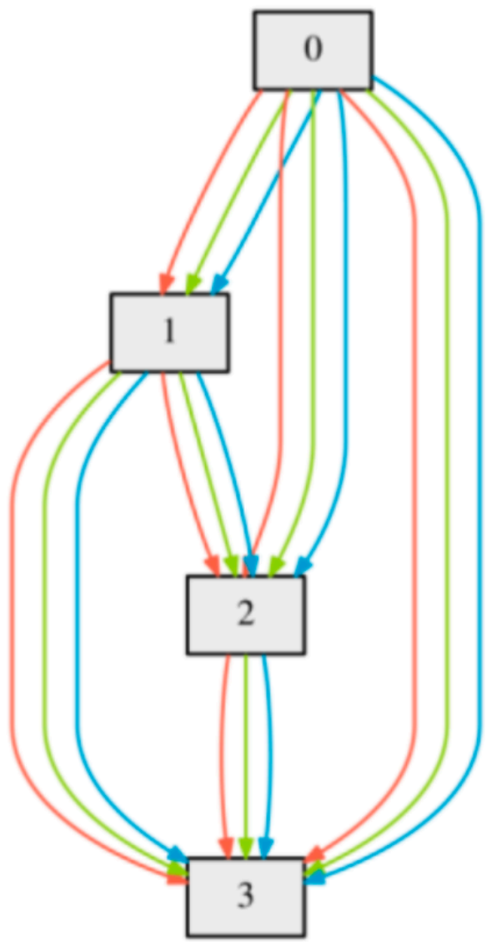
Parameter gradient step w.r.t. $\nabla_w \mathcal{L}_{train}$

DARTS: Differentiable NAS

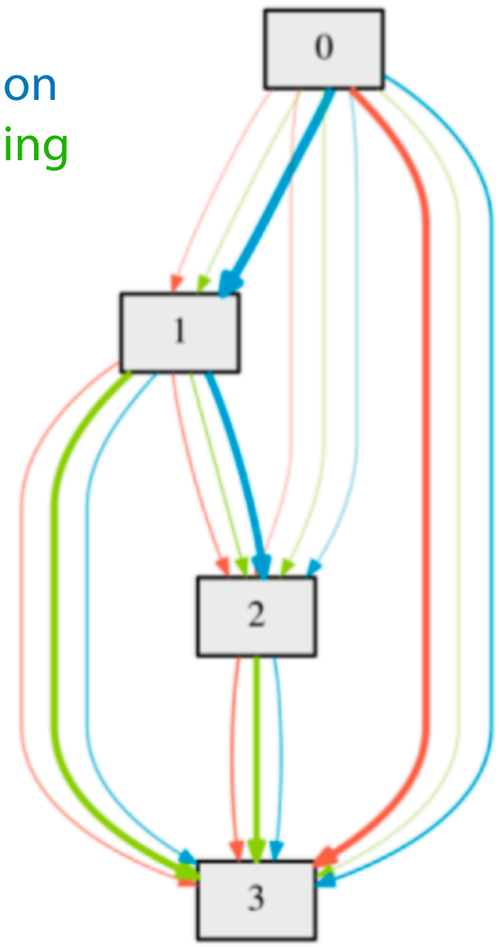
- Fixed (one-shot) structure, learn which operators to use
- Give all operators a weight α_i
- Optimize α_i and model weights ω_j using bilevel programming



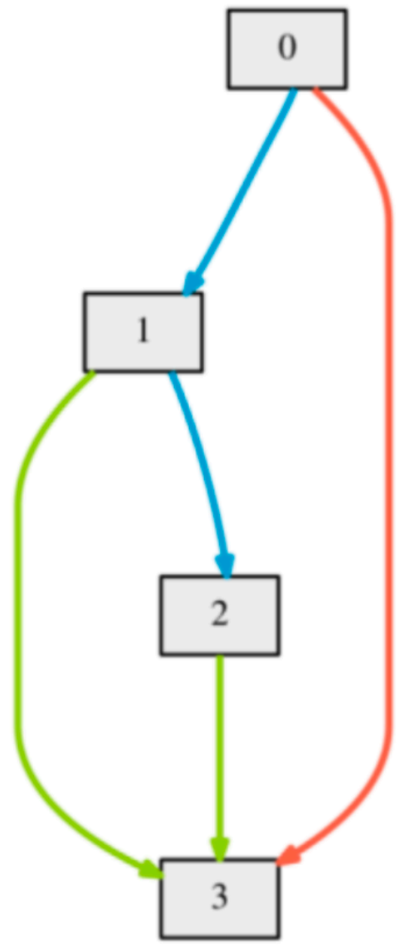
One-shot model



operator weights α_i



interleaved optimization of α_i and ω_j with SGD



argmax α_i

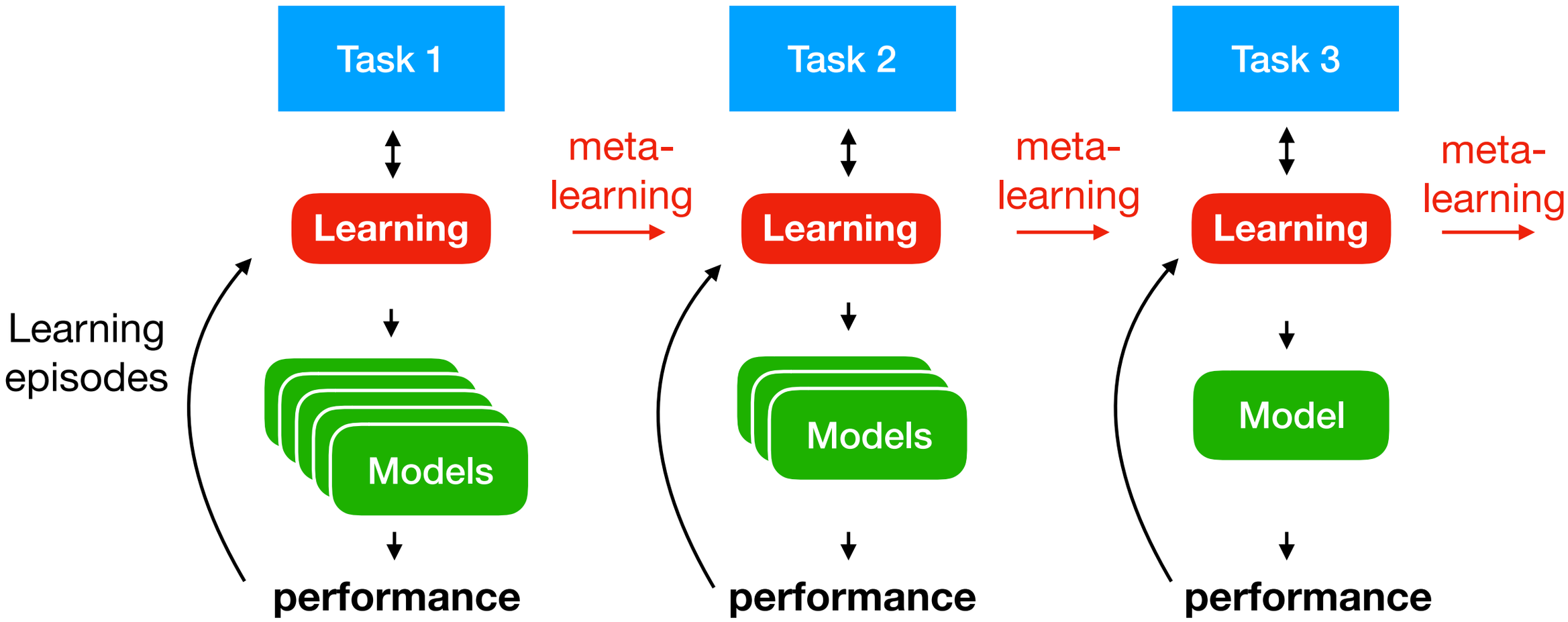
Learning is a never-ending process

Learning humans also seek/create related tasks

E.g. find many similar puzzles, solve them in different ways,…

Learning is a never-ending process

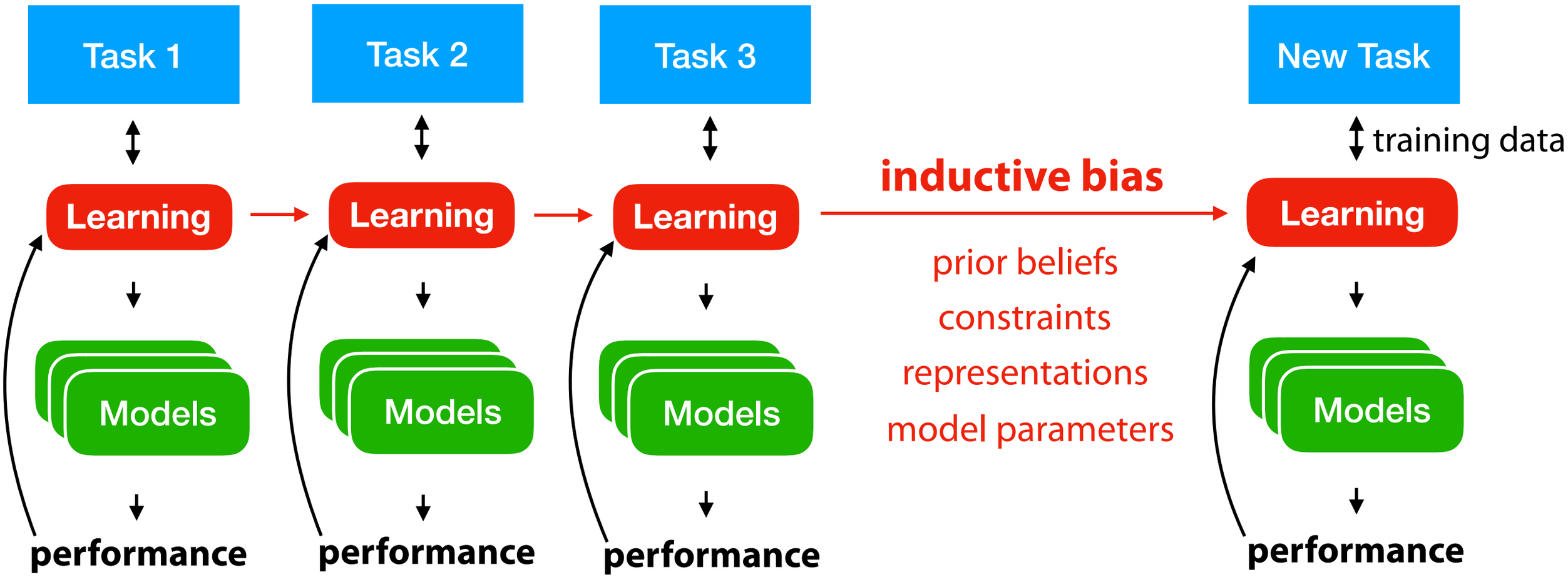
Humans learn *across* tasks
Why? Requires less trial-and-error, less data



Learning to learn

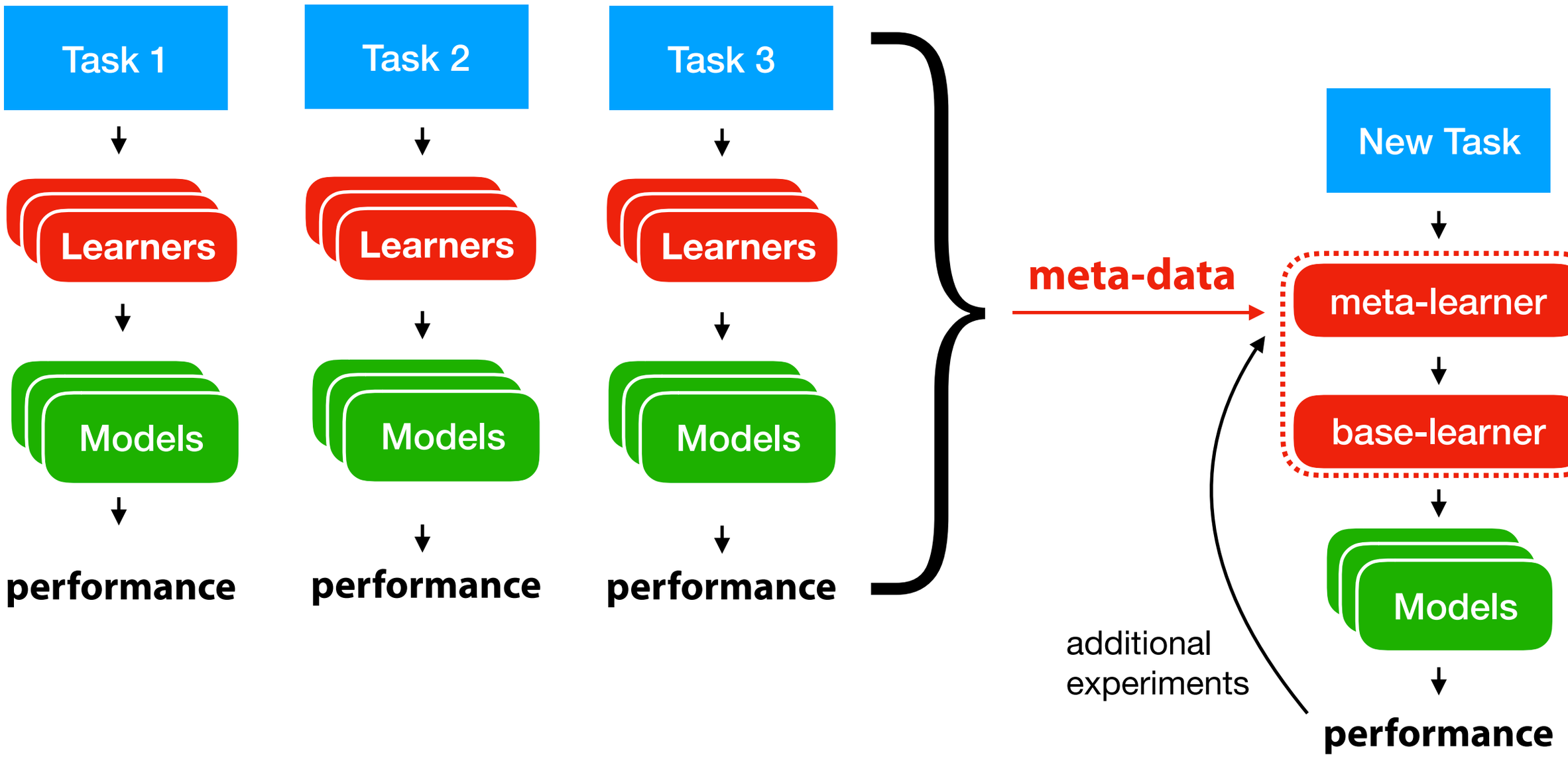
If prior tasks are *similar*, we can ***transfer*** prior knowledge to new tasks

Inductive bias: assumptions added to the training data to learn effectively



Meta-learning

Meta-learner *learns* a (base-)learning algorithm, based on *meta-data*



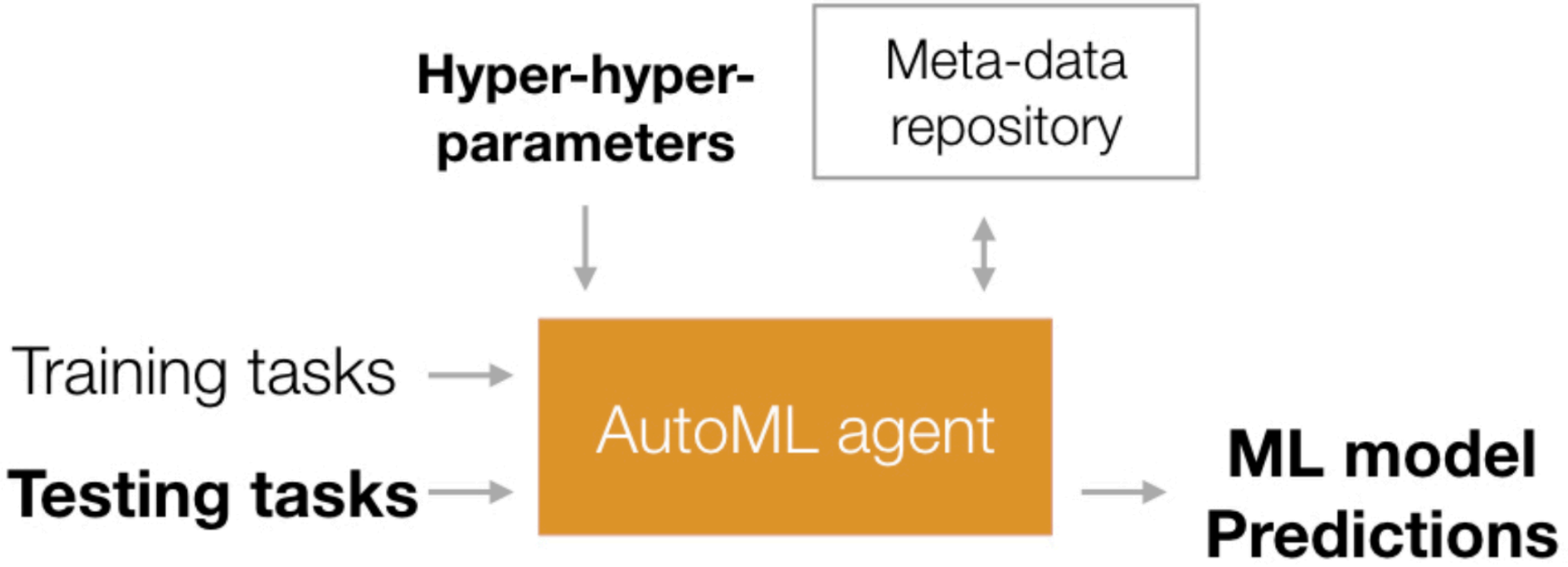
Meta-learning

Meta-learning can drastically speed up the architecture and hyperparameter search, in combination with the optimization techniques we just discussed

- Learn which hyperparameters are really important
- Learn which hyperparameter values should be tried first
- Learn which architectures will most likely work
- Learn how to clean and annotate data
- Learn which feature representations to use
- Learn which embeddings, pre-trained nets to use
- …

Meta-learning in practice

- We need a meta-data repository of relevant prior machine learning experiments
- Ideally, a *shared* memory that all AutoML tools can access



Meta-learning in practice

- OpenML.org
  - Thousands of uniform datasets, 100+ meta-features
  - Millions of evaluated runs
    - Same splits, 30+ metrics
    - Traces, models *(opt)*
  - **APIs in Python, R, Java,… Sklearn, Keras, PyTorch, MXNet,…**
  - Publish your own runs
  - **Never ending learning**
  - Benchmarks

```
import openml as oml
from sklearn import ensembles

task = oml.tasks.get_task(14951)
clf = ensembles.RandomForestClassifier()
run = oml.runs.run_model_on_task(task, flow)
myrun = run.publish()
```

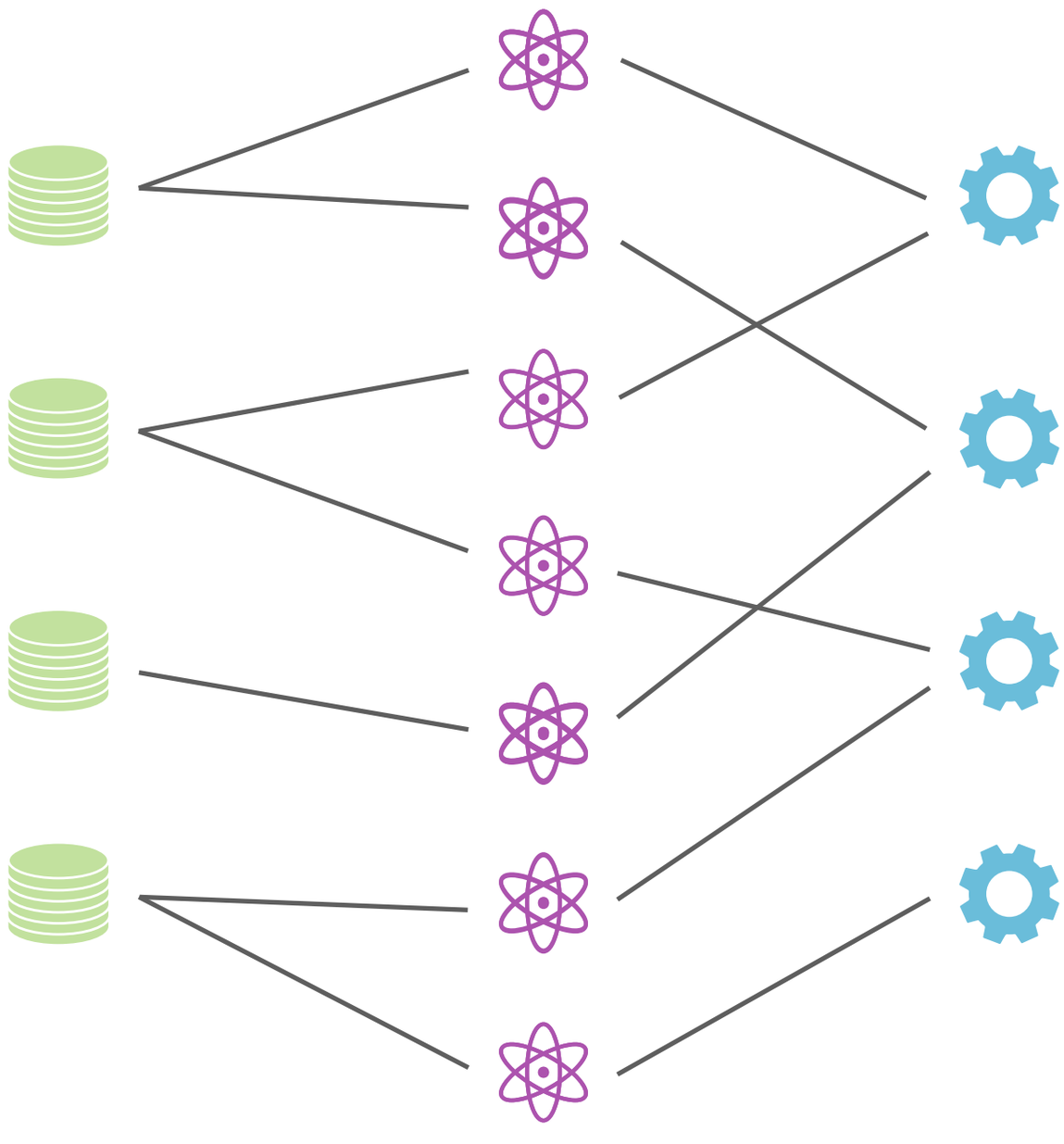
run *locally,* share *globally*

OpenML: Organize the world's ML information

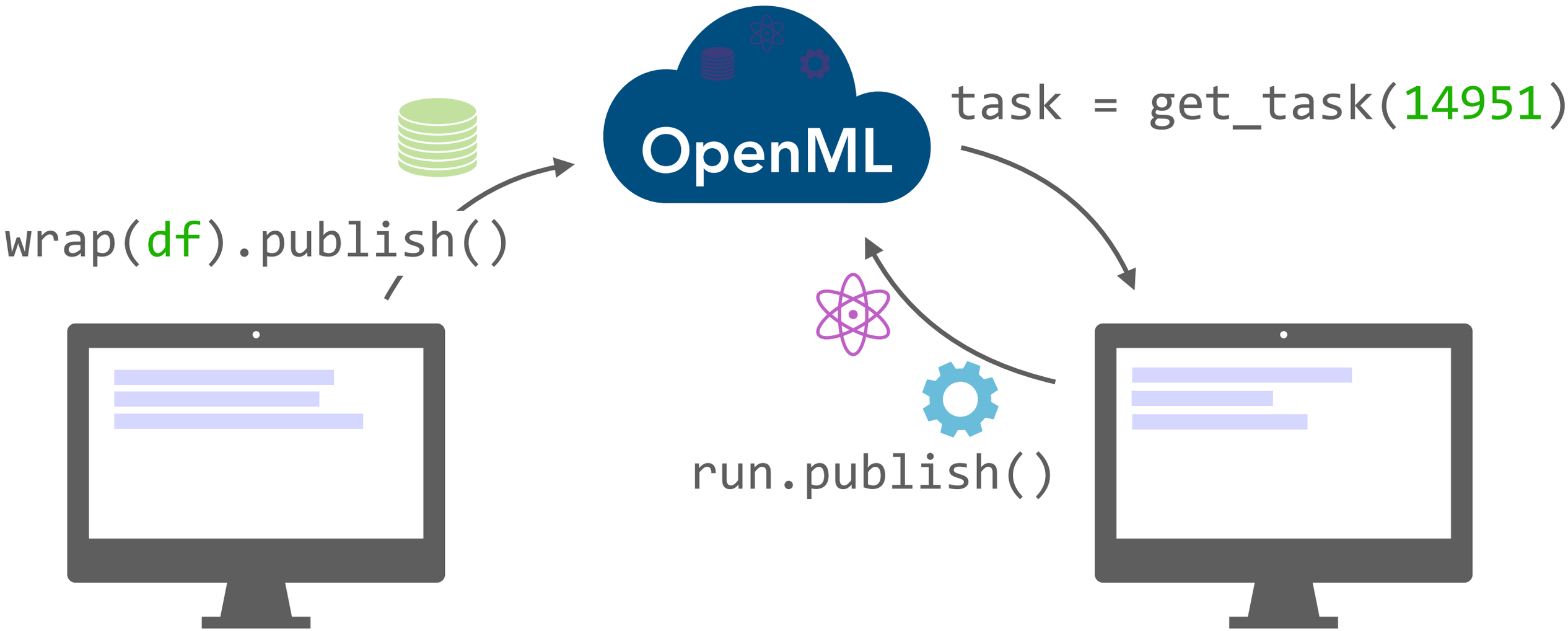
For every dataset, find all models built (and how good they are)

For every model, get the *exact* dataset and algorithm used

For every algorithm, find how useful it is for every dataset



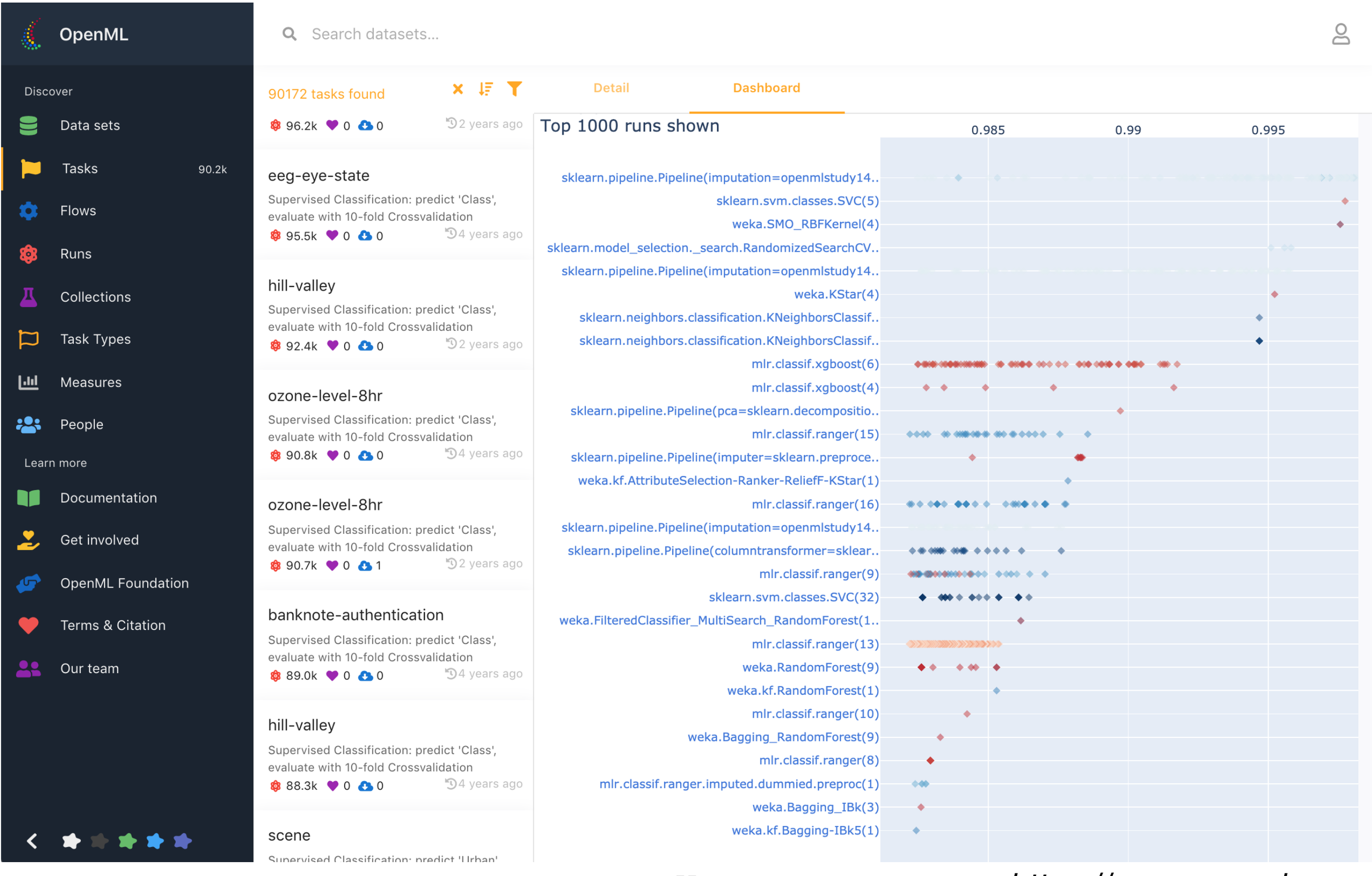
OpenML: Frictionless machine learning



APIs:

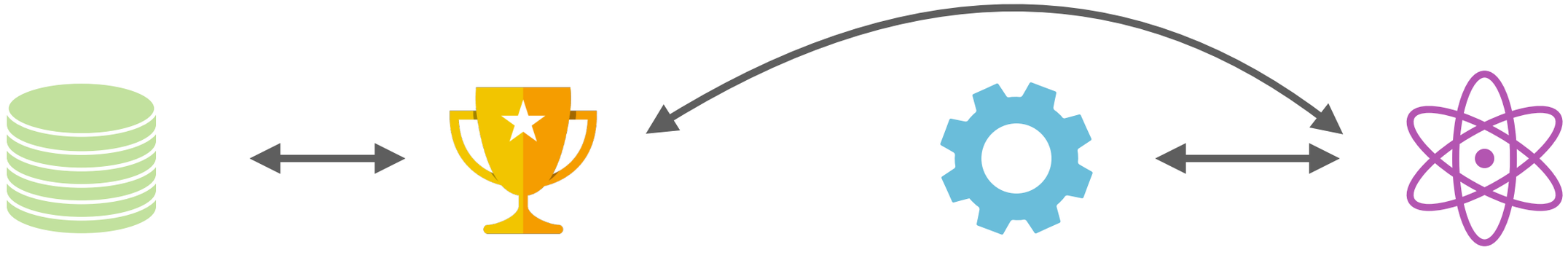
Integrations:

OpenML.org



https://new.openml.org

OpenML: Reproducibility



Datasets
(+ meta-data)

Tasks
(problem meta-data)

Flows
(algorithm meta-data)

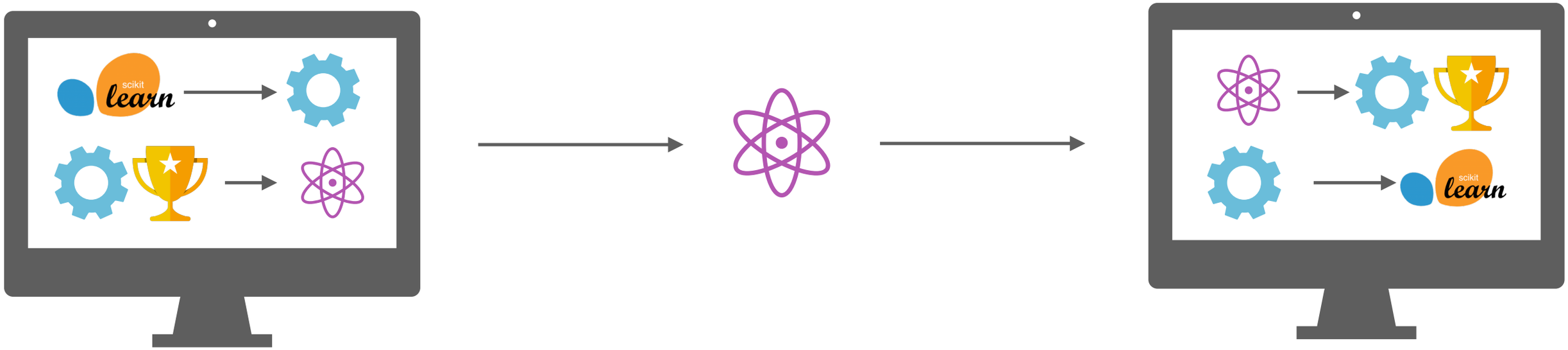
Runs
(model meta-data)

file, url, version

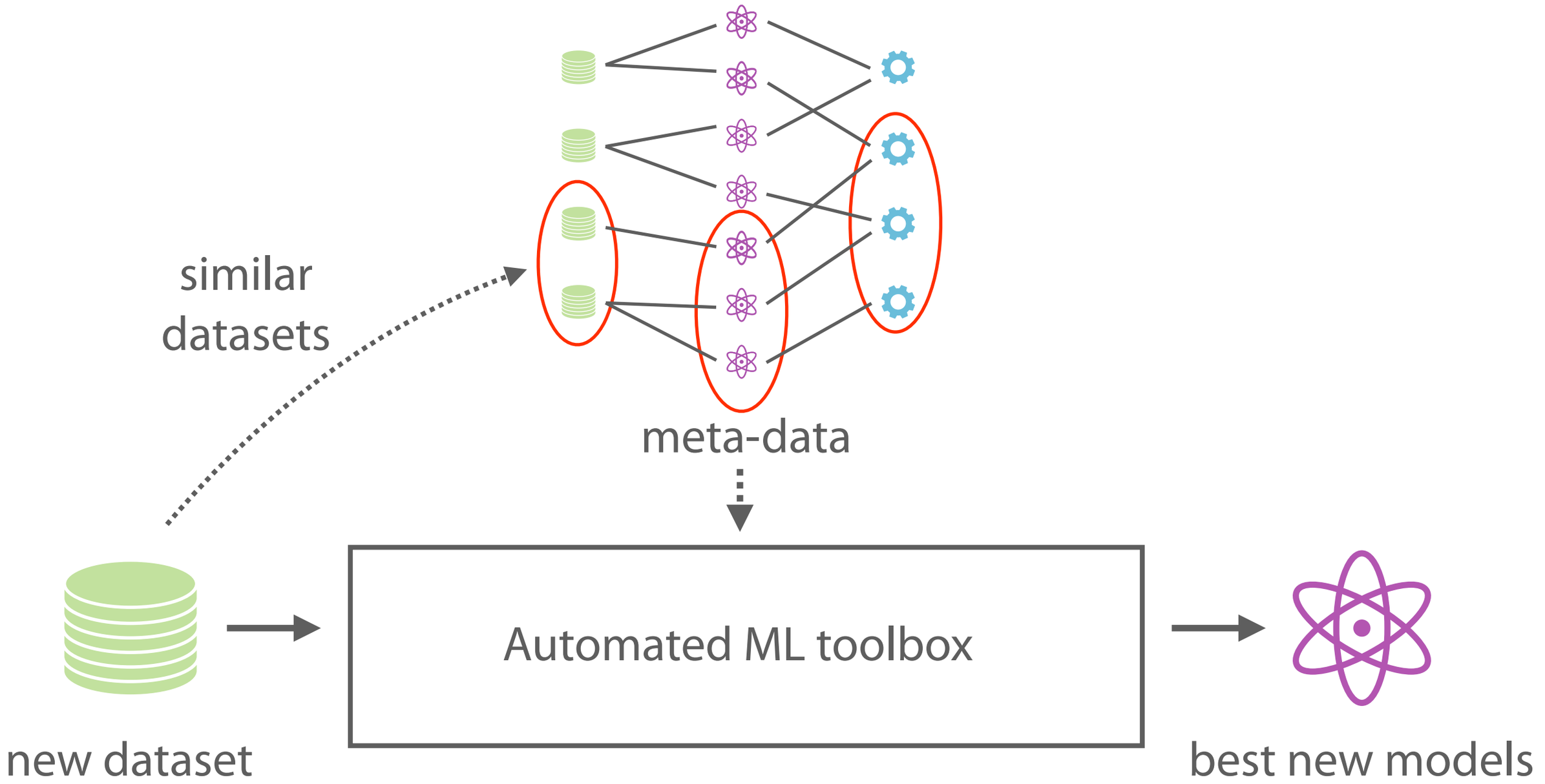
how to evaluate

dependencies,
detailed structure

configurations,
evaluations

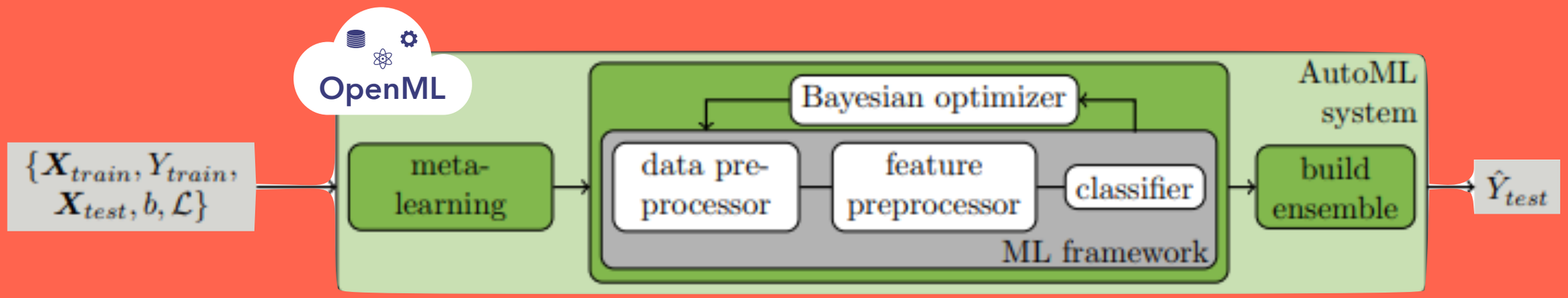


Meta-learning with OpenML



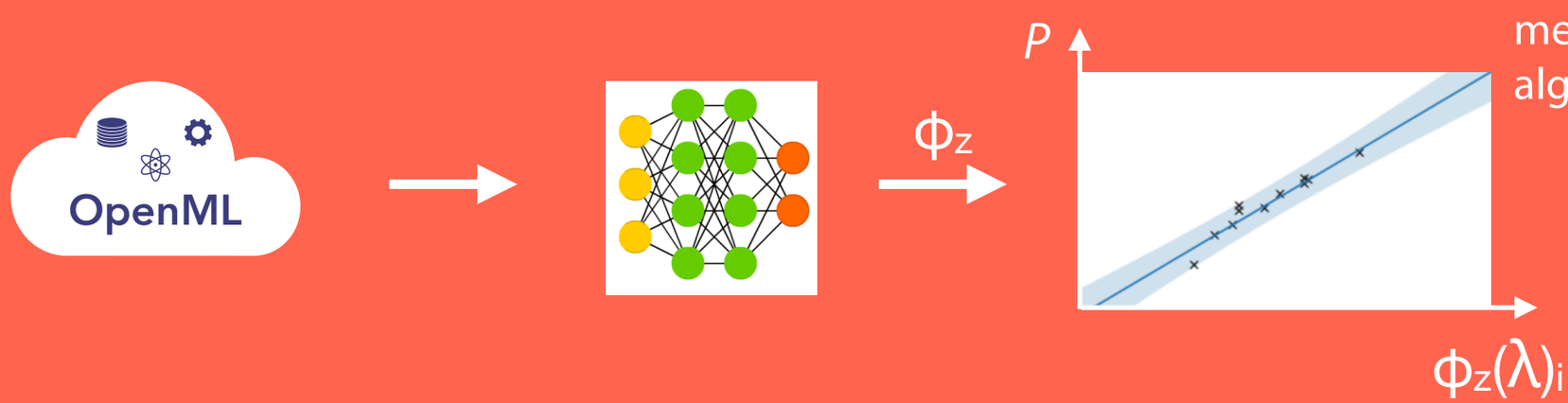
AutoML + metalearning

auto-sklearn: uses OpenML to *warm-start* the search for the best pipelines



Feurer et al. 2016

ABLR (Amazon): uses OpenML to learn how to search hyperparameters



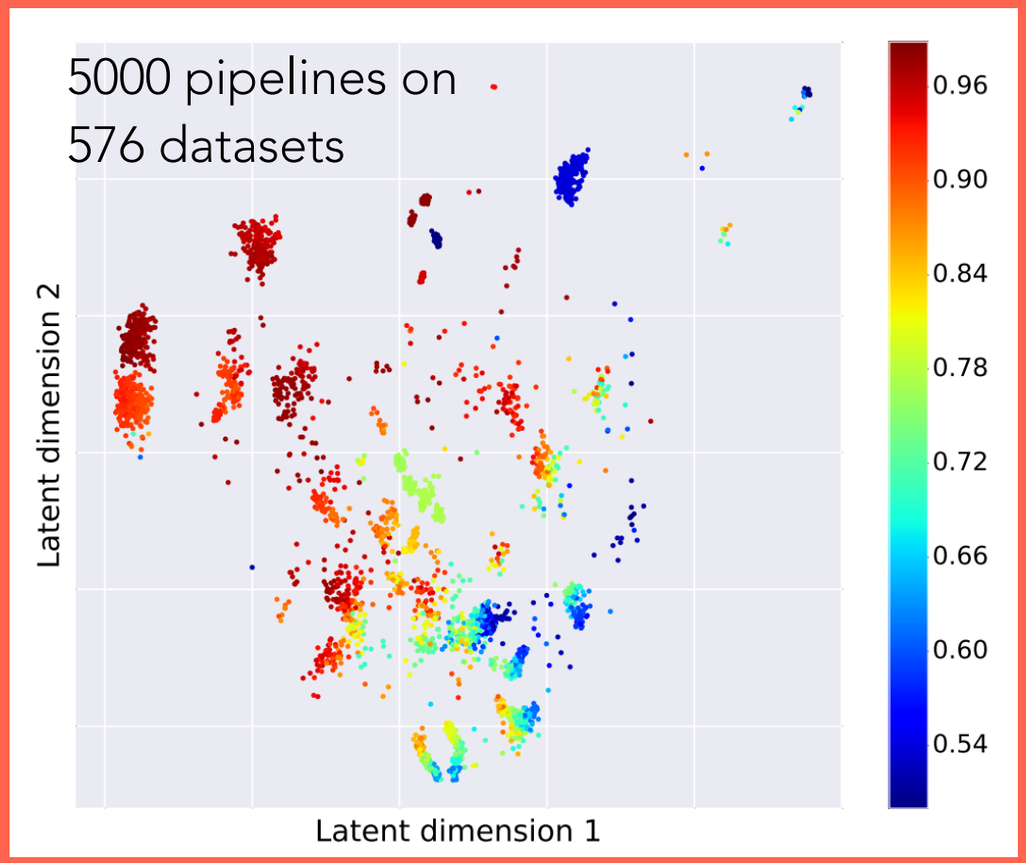
meta-model: predicts algorithm performance

Perrone et al. 2017

AutoML + metalearning

ProbMF (Microsoft): uses OpenML to recommend the best algorithms

datasets

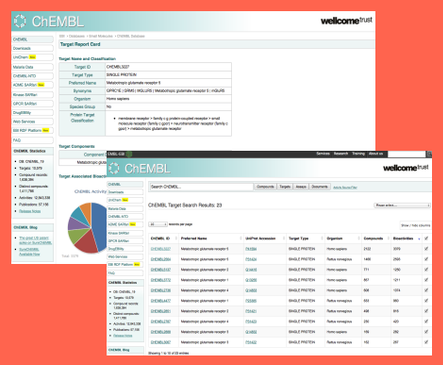


Recommender system

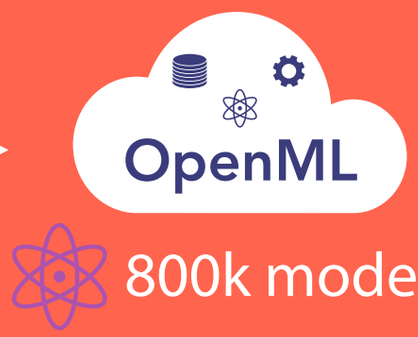
Fusi et al. 2018

AutoML + metalearning

Drug discovery for rare diseases

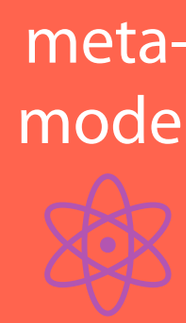


ChEMBL database

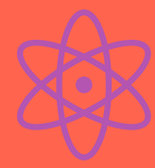


800k models

new target



meta-model

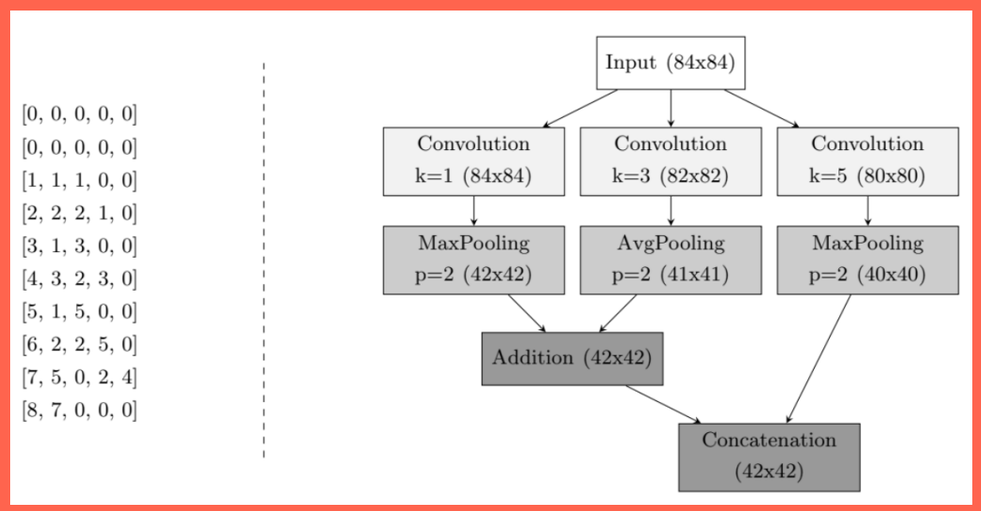


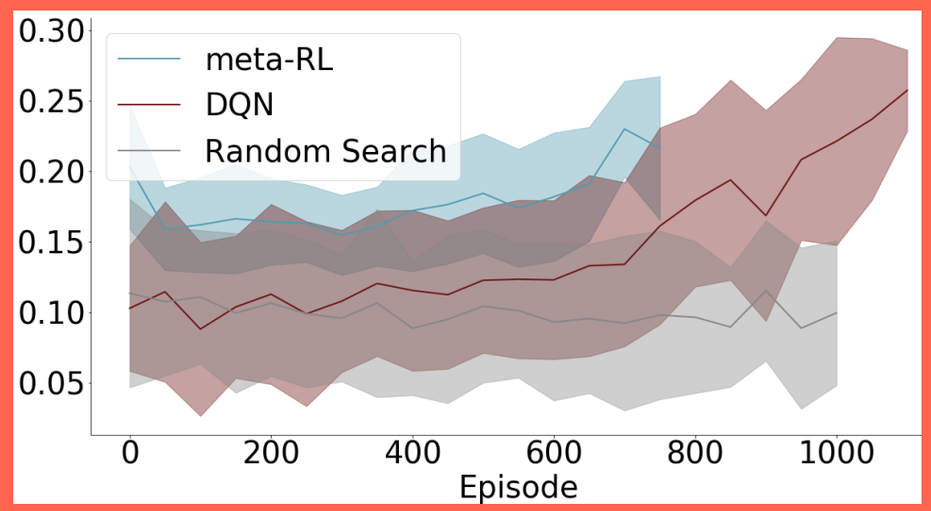
model

active ingredients

Olier et al. 2018

Meta-reinforcement learning for neural architecture search





Gomez & Vanschoren 2019

AutoML *open source* tools

	Architect. search	Operators	Hyperpar. search	Improvements	Metalearning
Auto-WEKA	Param. pipeline	WEKA	Bayesian Opt. (RF)		
auto-sklearn	Param. pipeline	sklearn	Bayesian Opt. (RF)	Ensemble	warm-start
mlr-mbo	Param. pipeline	mlr	Bayesian Opt.	multi-obj.	
BO-HB	Param. pipeline	sklearn	Tree of Parzen Estim.	Ensemble, HB	
hyperopt-sklearn	Param. pipeline	sklearn	Tree of Parzen Estim.		
skopt	Param. pipeline	sklearn	Bayesian Opt. (GP)		
TPOT	Evolving pipelines	sklearn	Population-based		
GAMA	Evolving pipelines	sklearn	Population-based	Ensemble, HB	
ML-Plan	Planning	sklearn	Interleaved		
MOSAIC	MCTS	config	Surrogate model		
H2O AutoML	Param. pipeline	H2O	Random search	Stacking	
OBOE	Single algorithms	sklearn	Low rank approx.	Ensembling	Runtime pred
Auto-Keras	Param. NAS	keras	Bayesian Opt.	Net Morphisms	
Auto-pyTorch	Param. pipeline	pyTorch	BO-HB		

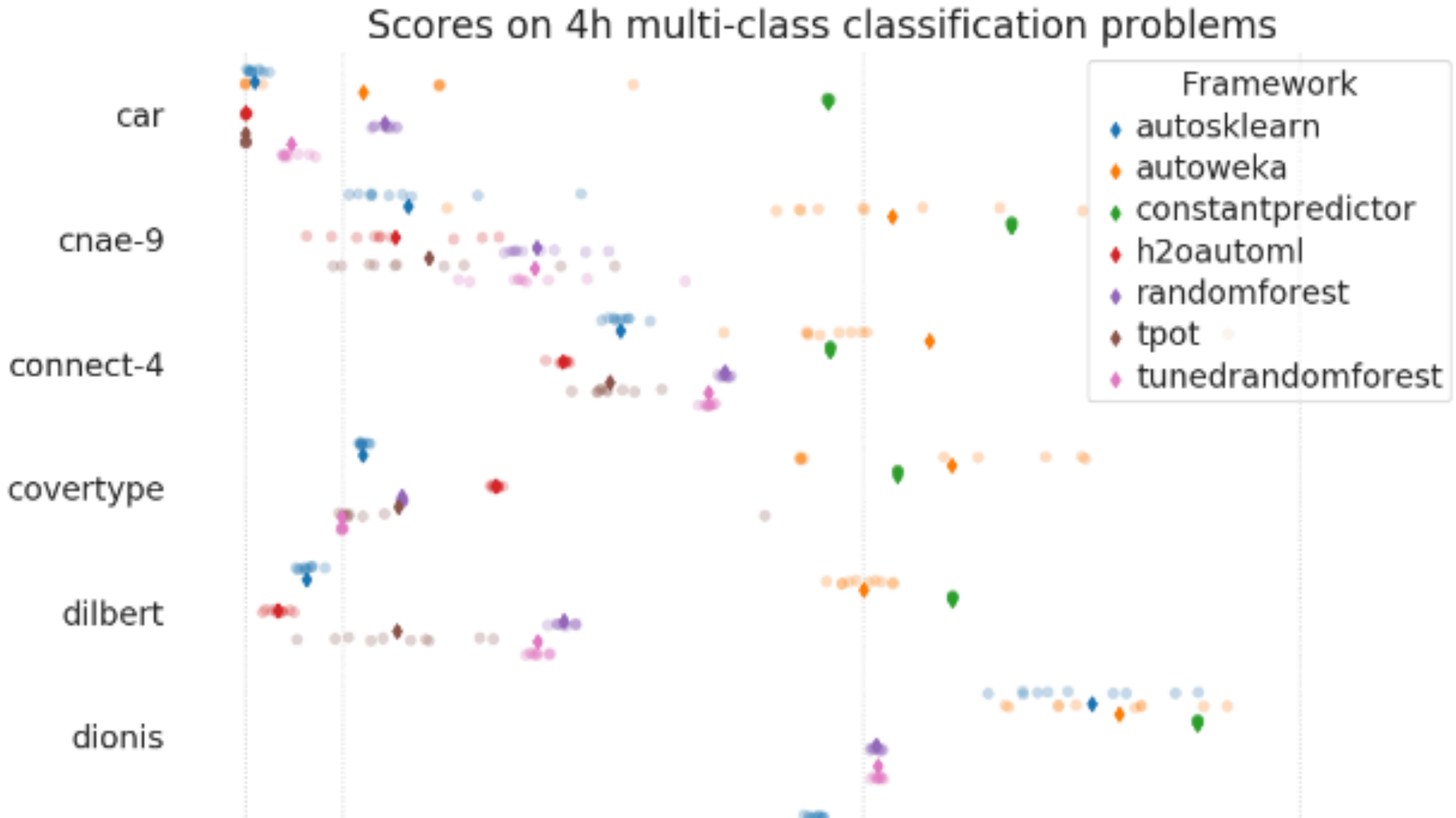
Many other tools for hyperparameter optimization alone

Gijsbers et al. 2019

AutoML benchmarks

Open source benchmark on https://openml.github.io/automlbenchmark/

- On OpenML datasets, will regularly be updated, anyone can add



Thank You

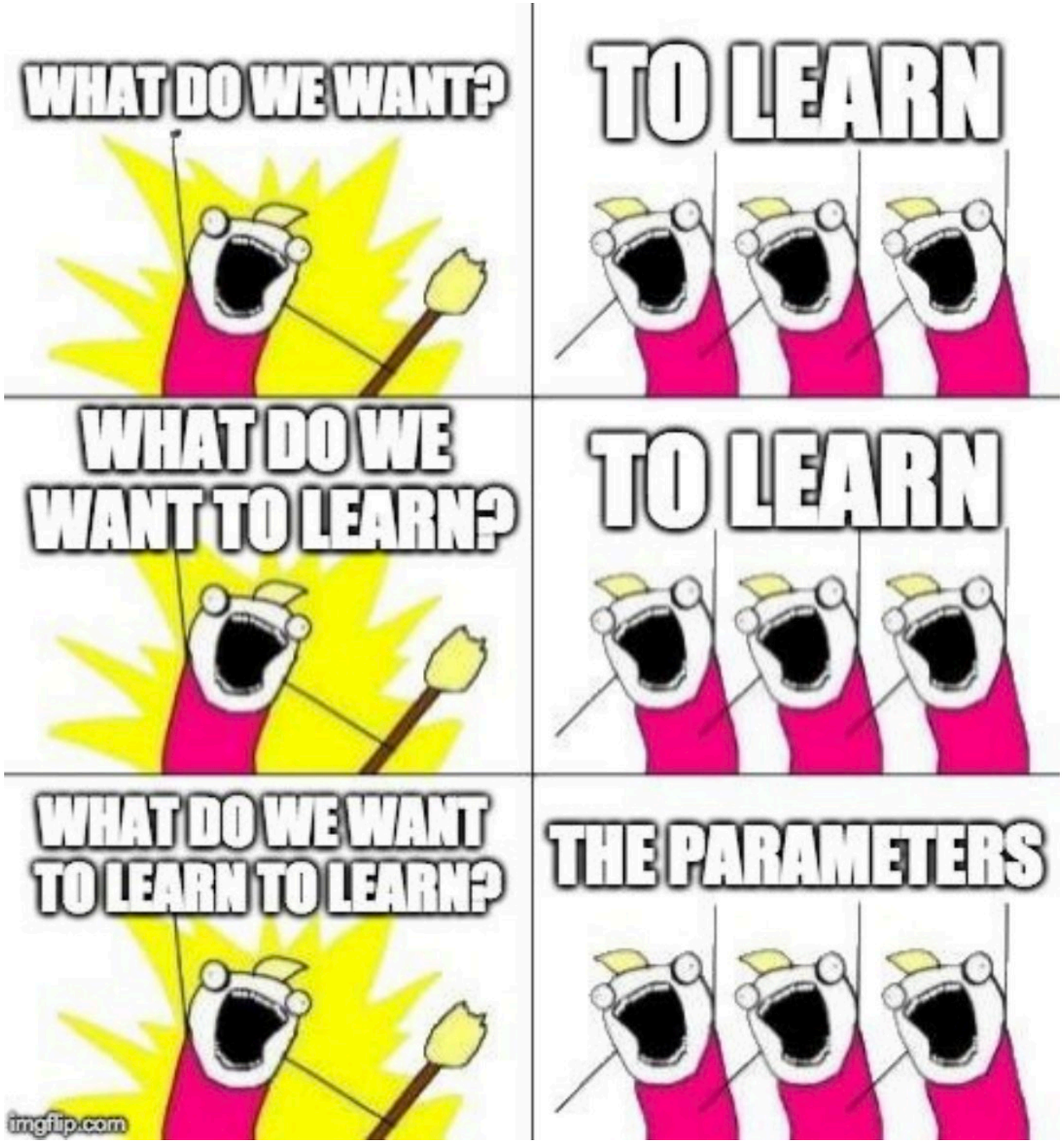


Image: Pesah et al. 2018