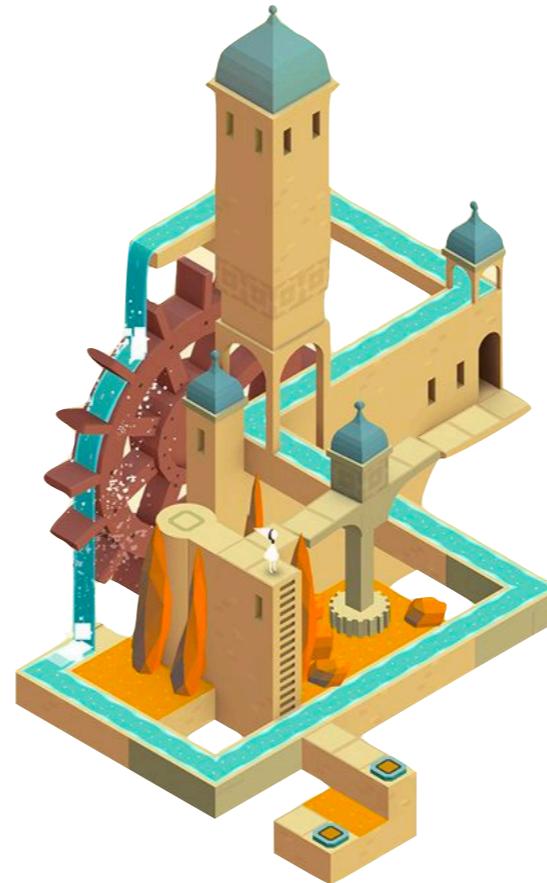


Advanced Course on Data Science & Machine Learning

Siena, 2019



Automatic Machine Learning & Meta-Learning

part 2

j.vanschoren@tue.nl

Joaquin Vanschoren
Eindhoven University of Technology
[@joavanschoren](https://twitter.com/joavanschoren)

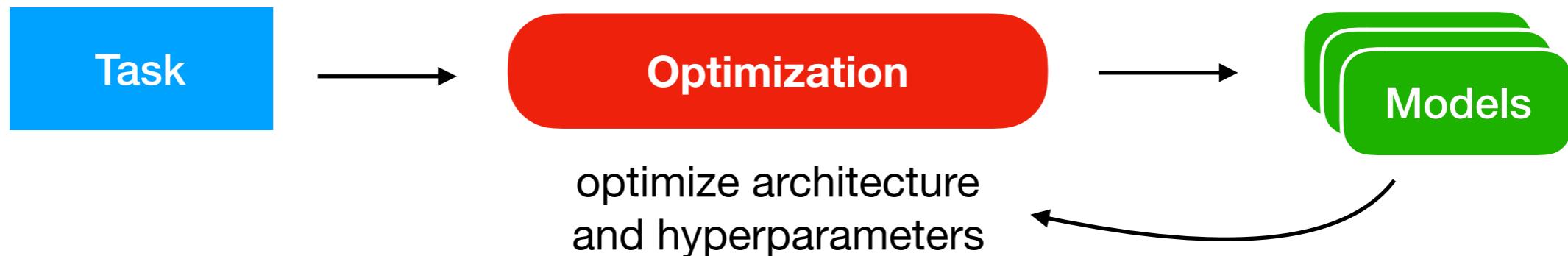


<http://bit.ly/openml>

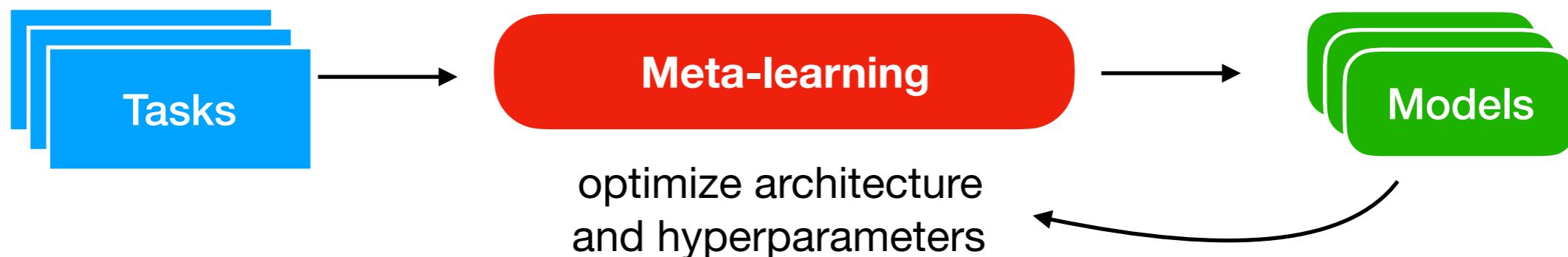
slides!

Overview

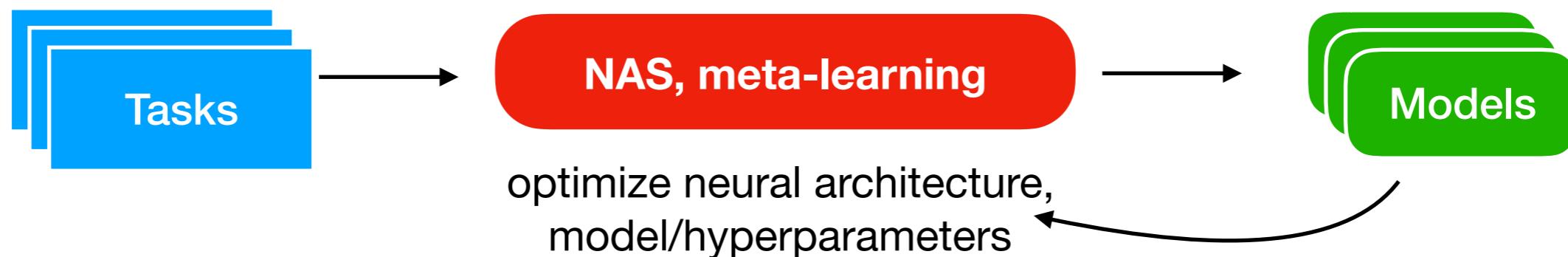
part I AutoML introduction, promising optimization techniques



part 2 Meta-learning

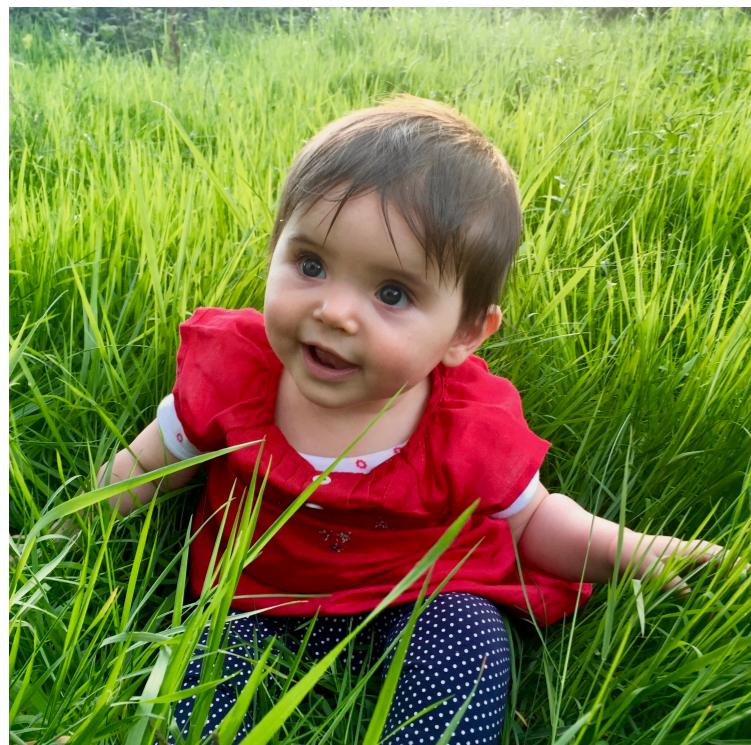


part 3 Neural architecture search, meta-learning on neural nets



Learning is a never-ending process

Humans don't learn from scratch



Learning is a never-ending process

Learning humans also seek/create related tasks

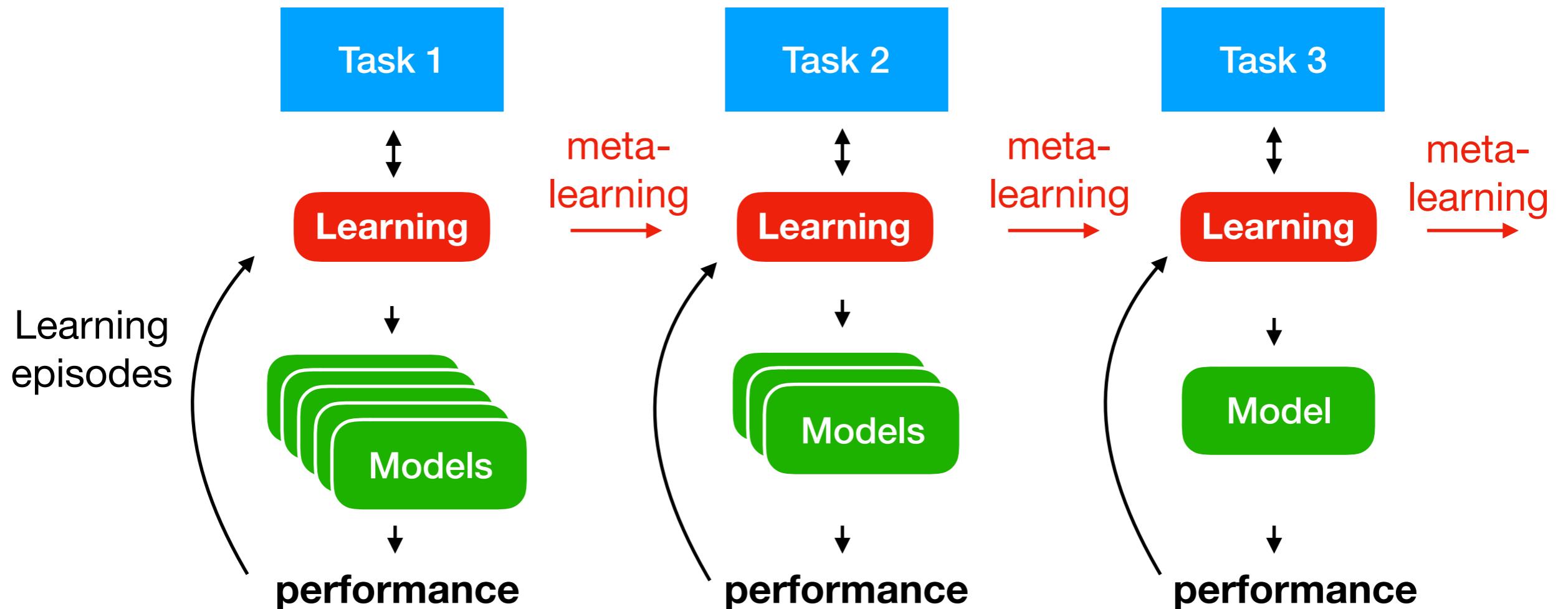


E.g. find many similar puzzles, solve them in different ways,...

Learning is a never-ending process

Humans learn *across* tasks

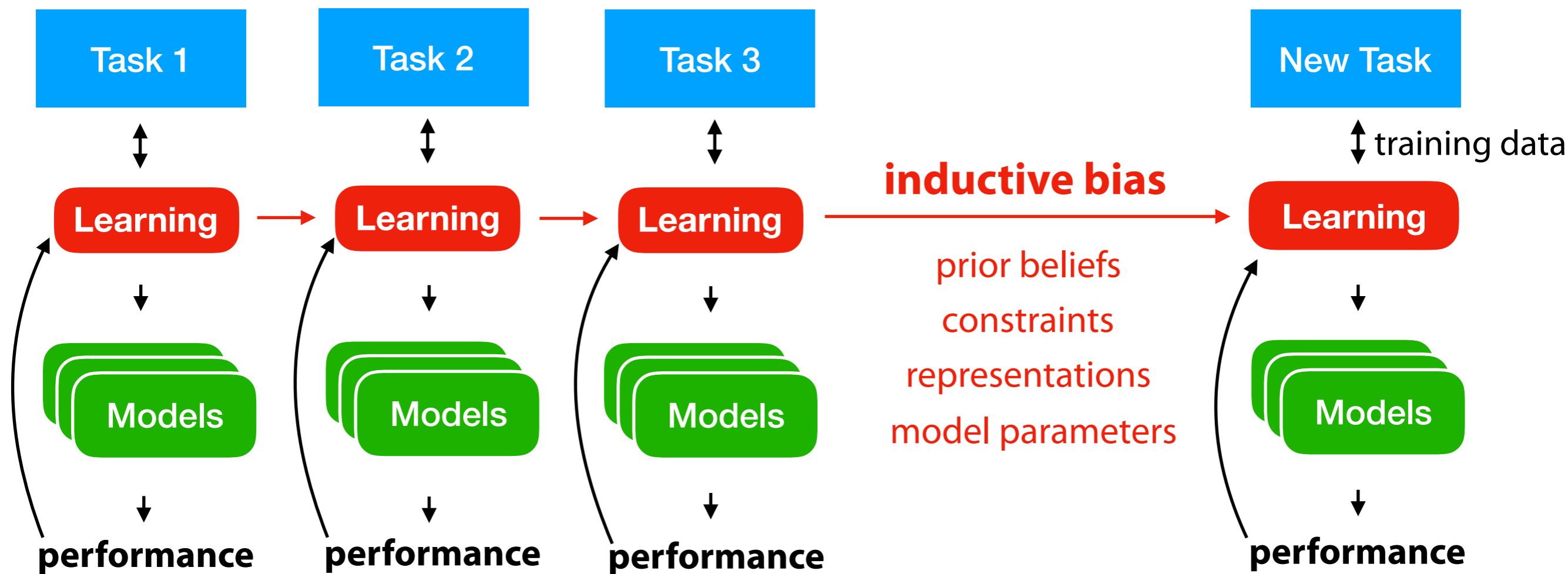
Why? Requires less trial-and-error, less data



Learning to learn

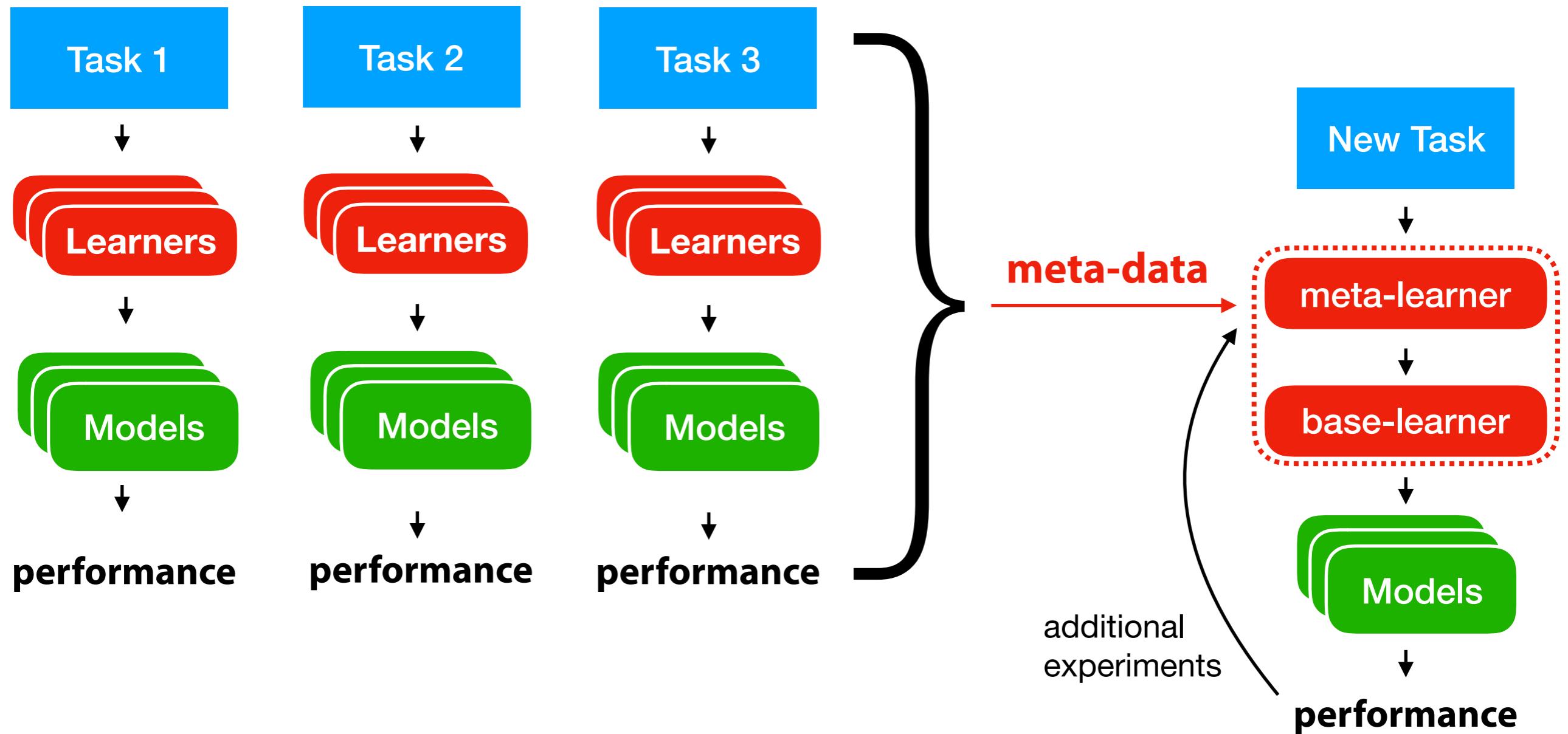
If prior tasks are *similar*, we can **transfer** prior knowledge to new tasks

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



Meta-learning

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



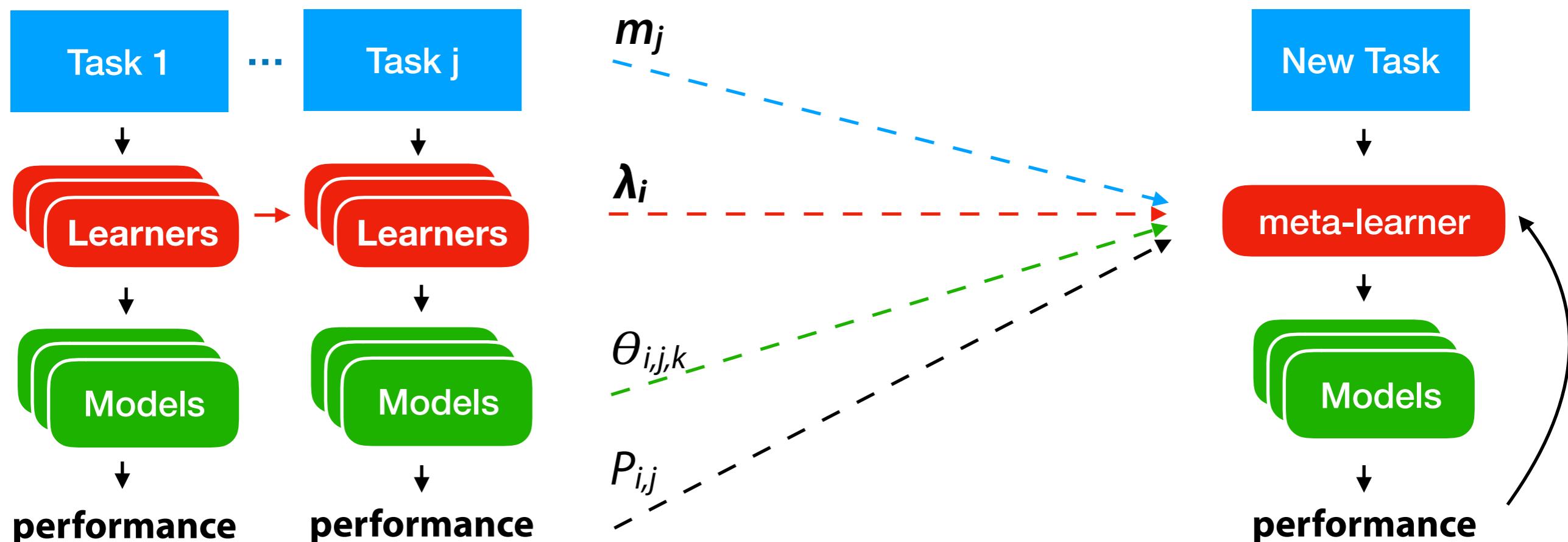
Meta-data

m_j Description of task j (meta-features)

λ_i Configuration i (architecture + hyperparameters)

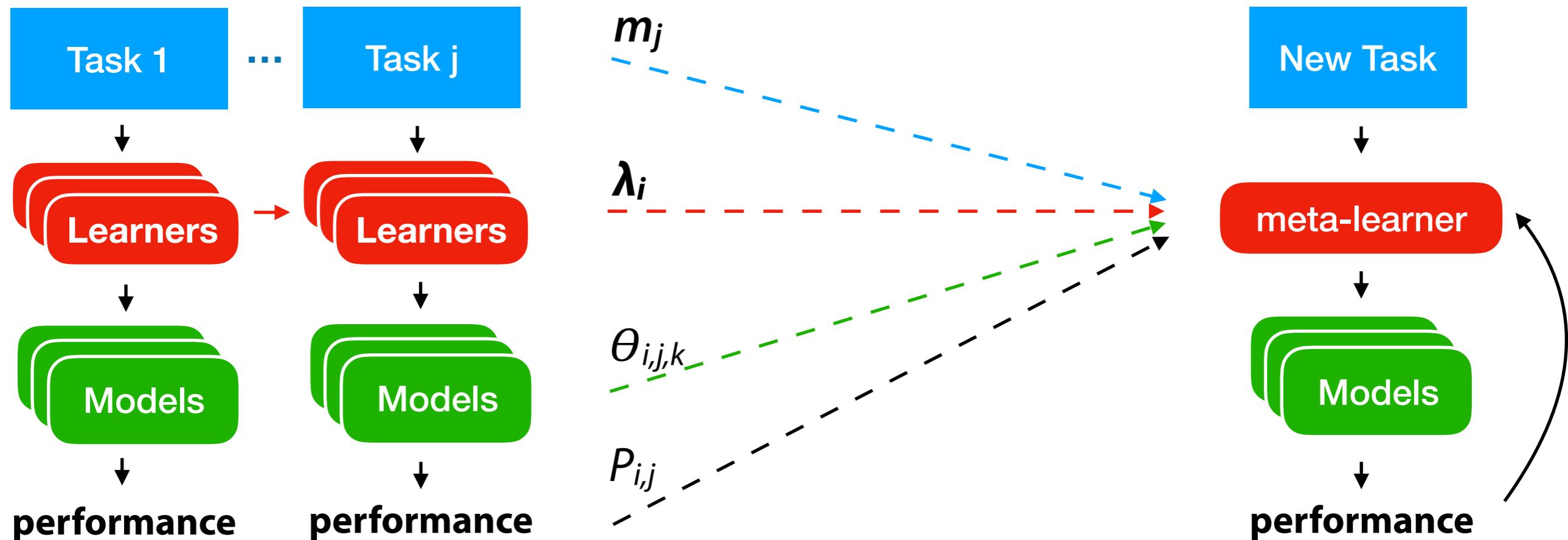
$\theta_{i,j,k}$ Model parameters (e.g. weights)

$P_{i,j}$ Performance estimate of model built with λ_i on task j



How?

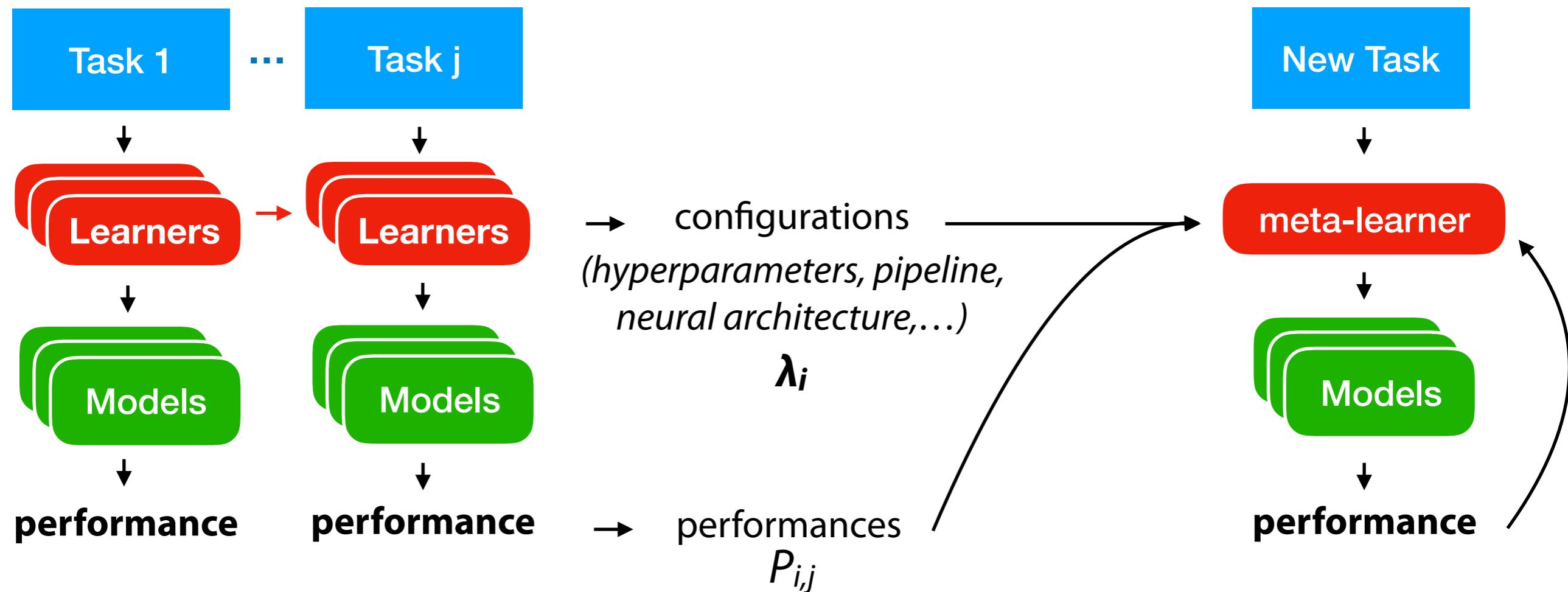
1. Learn from observing learning algorithms (for *any* task) $\lambda_i \ P_{i,j}$
2. Learn what may likely work (for *somewhat similar* tasks) $m_j \ \lambda_i \ P_{i,j}$
3. Learn from previous models (for *very similar* tasks) $\theta_{i,j,k} (m_j) \lambda_i \ P_{i,j}$



1. Learn from observing learning algorithms

Define, store and use meta-data:

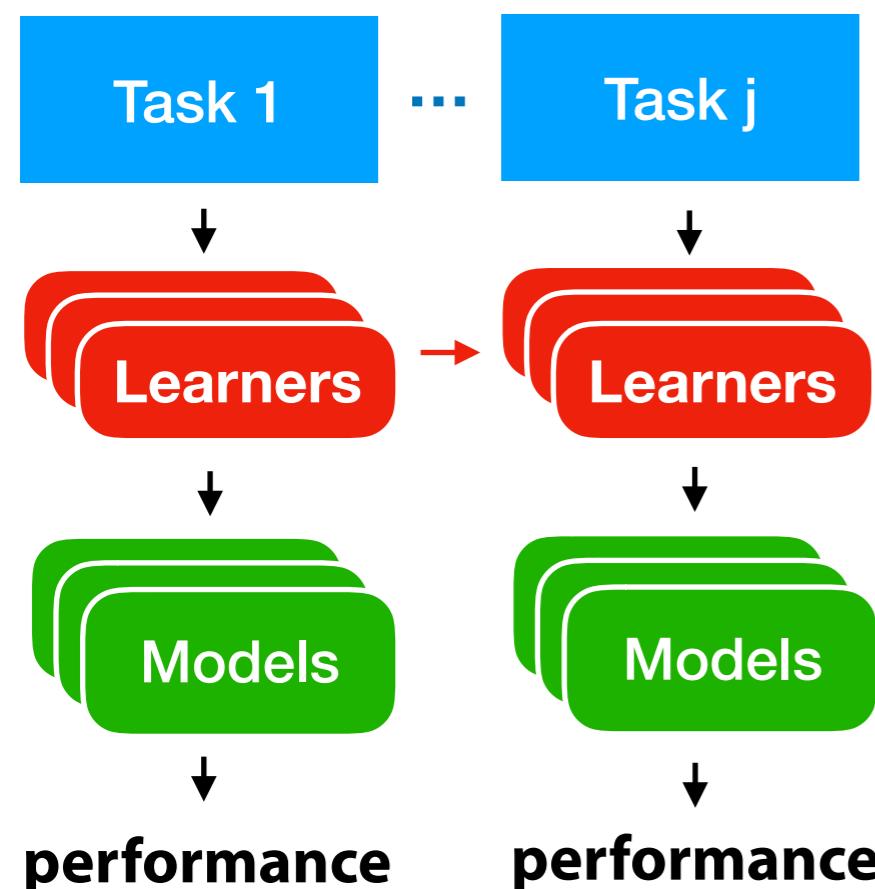
- **configurations**: settings that uniquely define the model
- **performance** (e.g. accuracy) on specific tasks



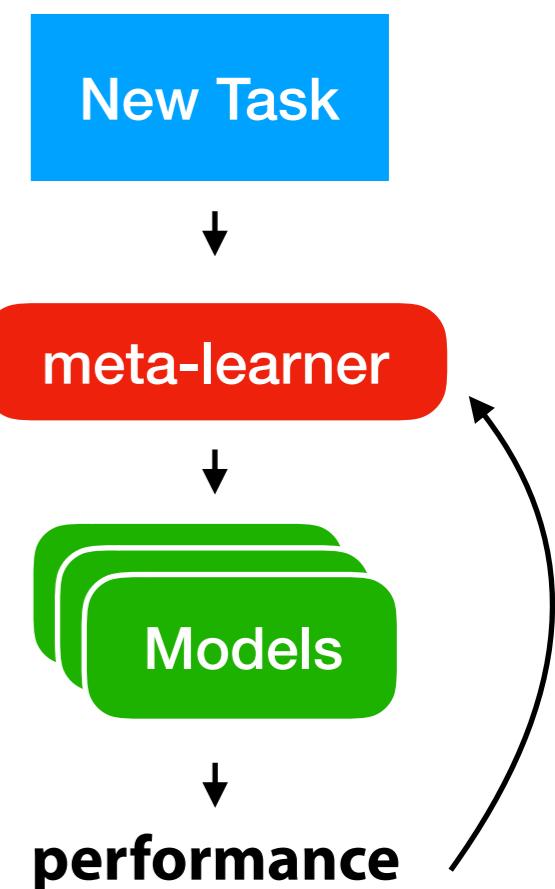
1. Learn from observing learning algorithms

Define, store and use meta-data:

- **configurations**: settings that uniquely define the model
- **performance** (e.g. accuracy) on specific tasks

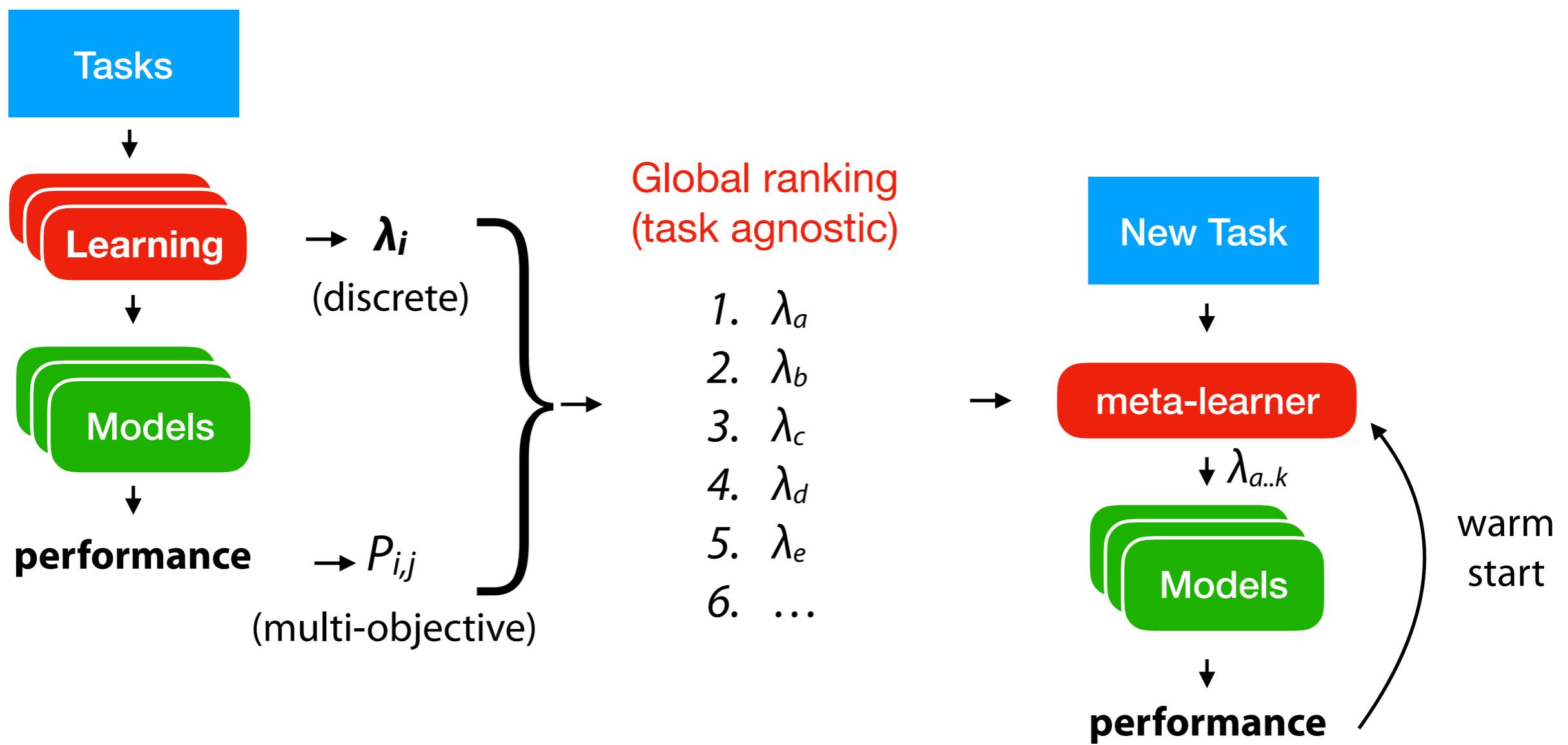


task	model config λ	score P
0	alg=SVM, C=0.1	0,876
0	alg>NN, lr=0.9	0,921
0	alg=RF, mtry=0.1	0,936
1	alg=SVM, C=0.2	0,674
1	alg=NN, lr=0.5	0,777
1	alg=RF, mtry=0.4	0,791



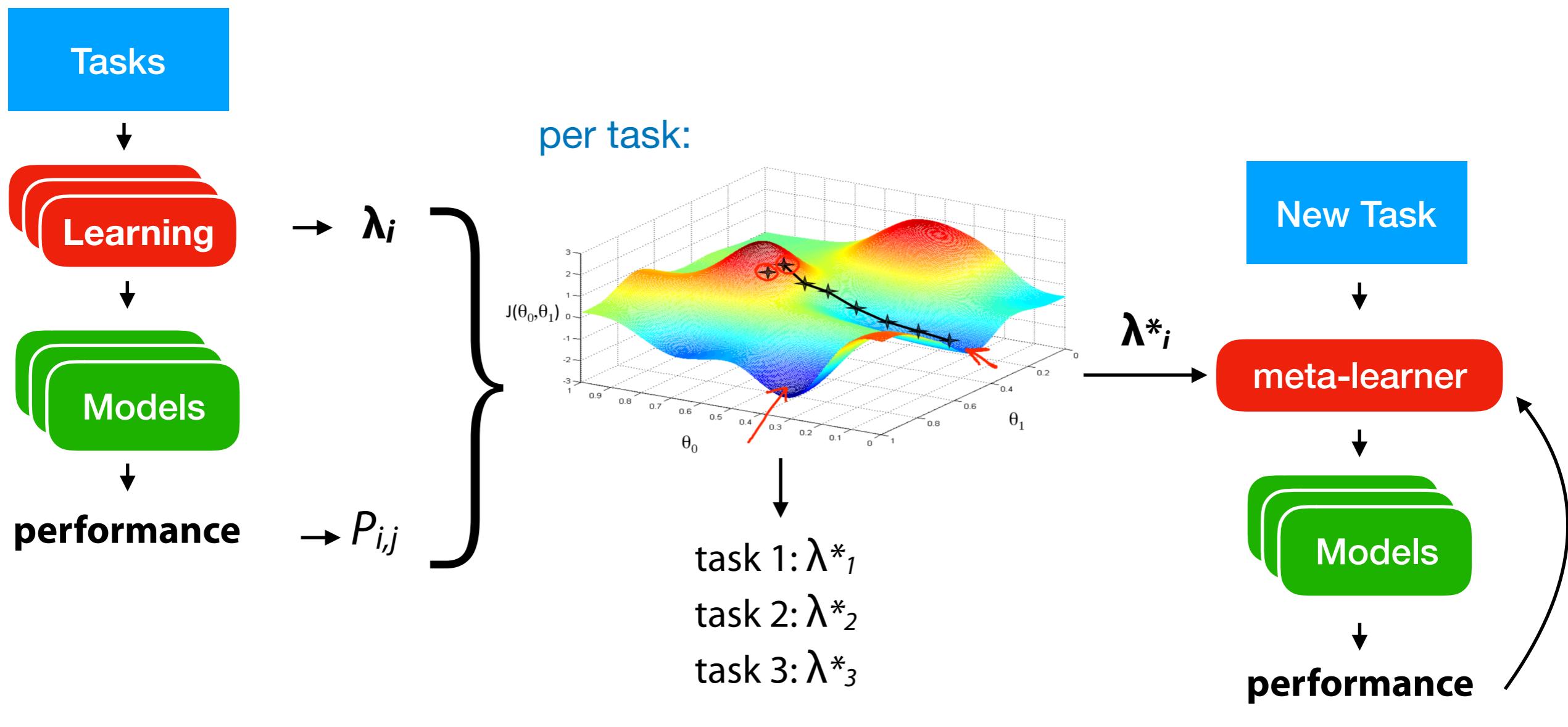
Rankings

- Build a *global (multi-objective) ranking*, recommend the top-K
- Can be used as a *warm start* for optimization techniques
 - E.g. Bayesian optimization, evolutionary techniques,...



Warm-starting with plugin estimators

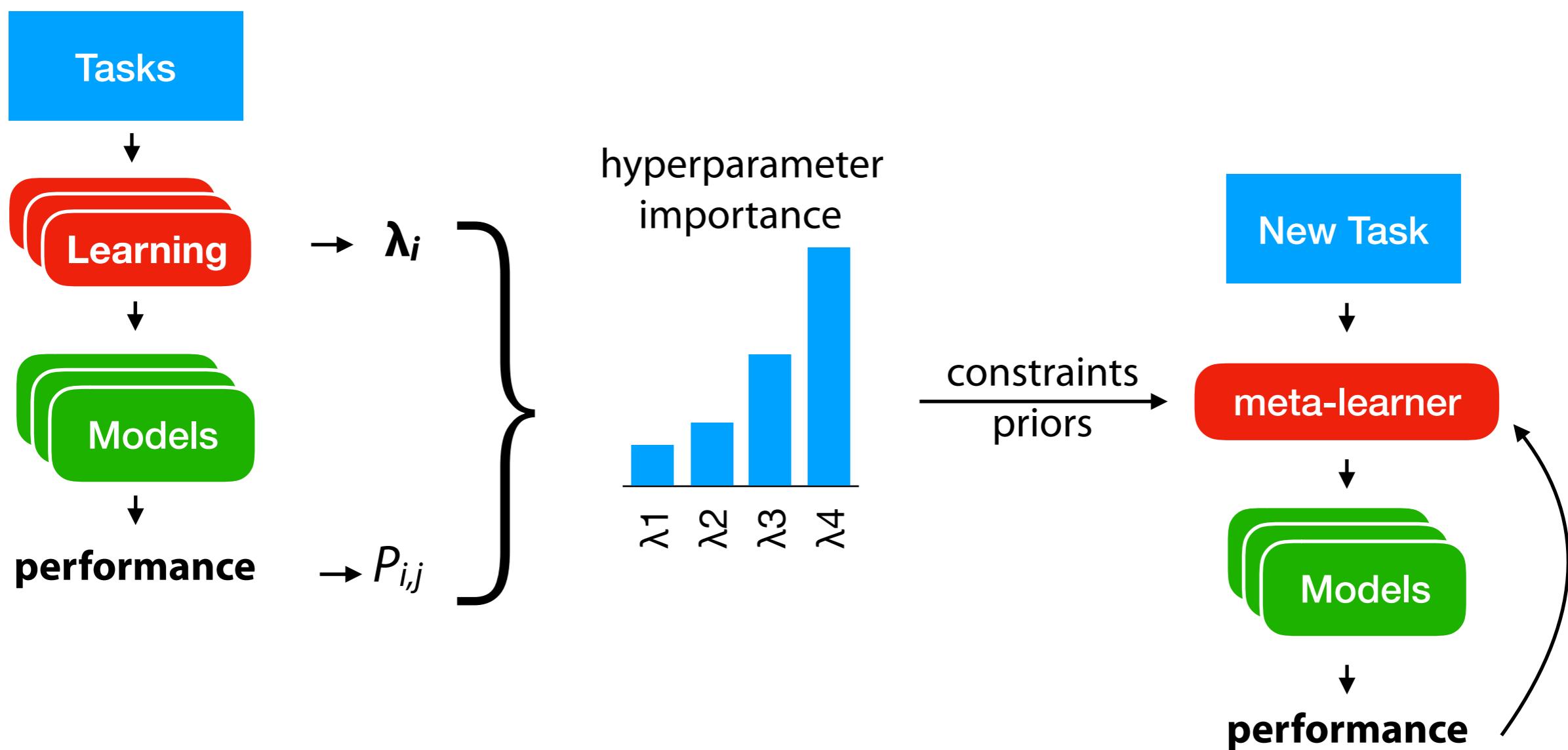
- What if prior configurations are not optimal?
- Per task, fit a differentiable plugin estimator on all evaluated configurations
- Do gradient descent to find optimized configurations, recommend those



Hyperparameter behavior

- **Functional ANOVA** ¹

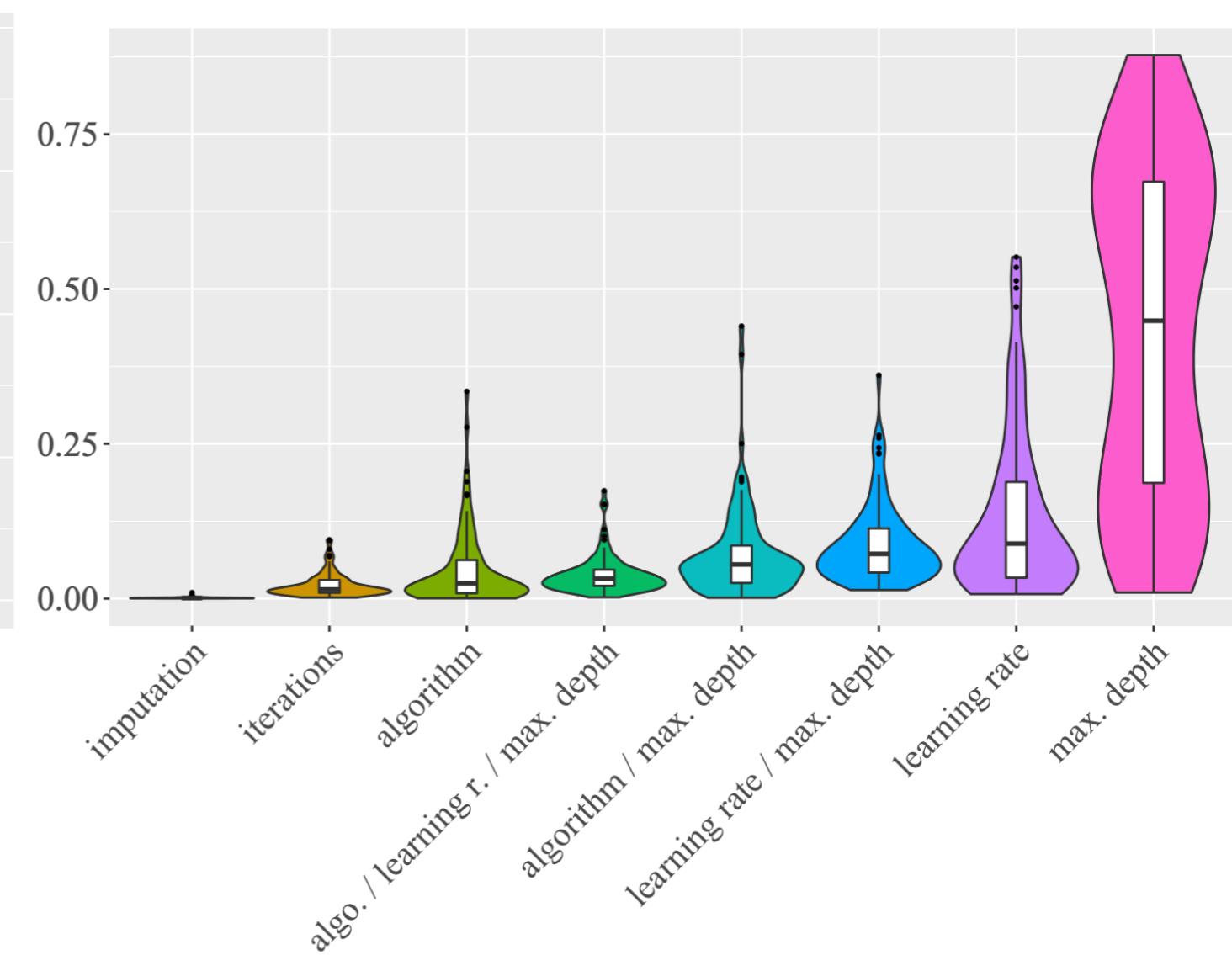
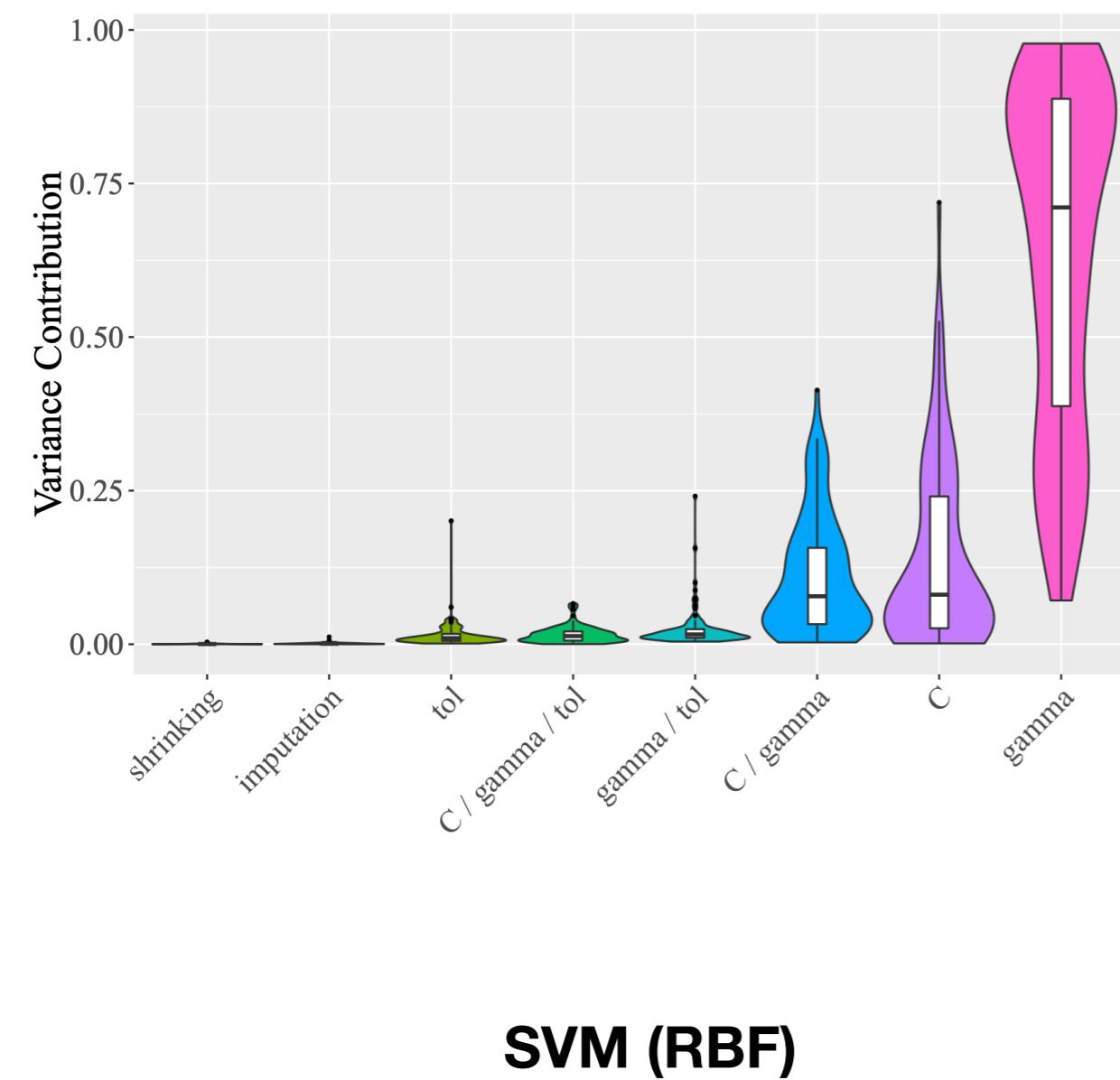
Select hyperparameters that cause variance in the evaluations.



Hyperparameter behavior

- **Functional ANOVA** ¹

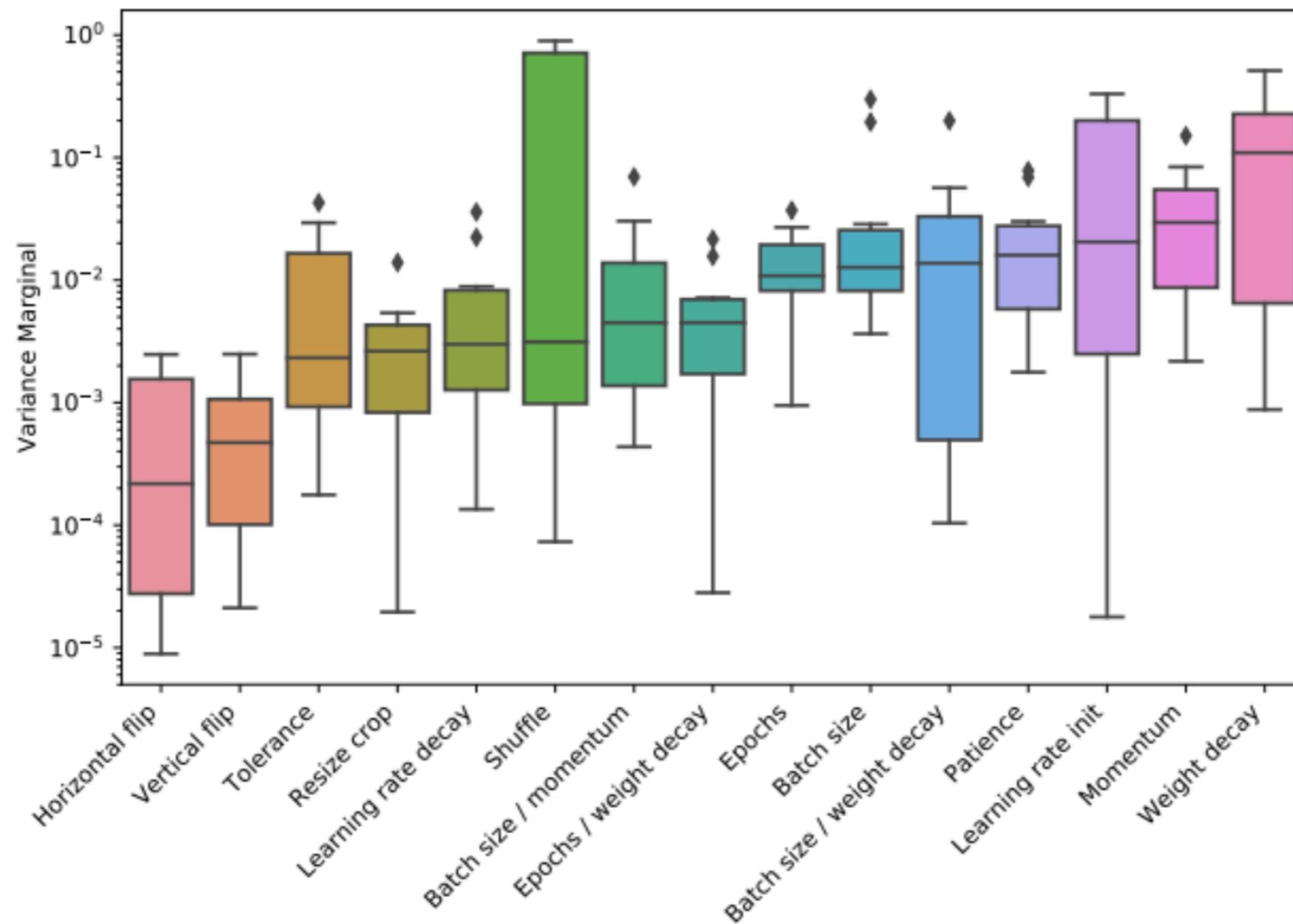
Select hyperparameters that cause variance in the evaluations.



Hyperparameter behavior

- **Functional ANOVA** ¹

Select hyperparameters that cause variance in the evaluations.

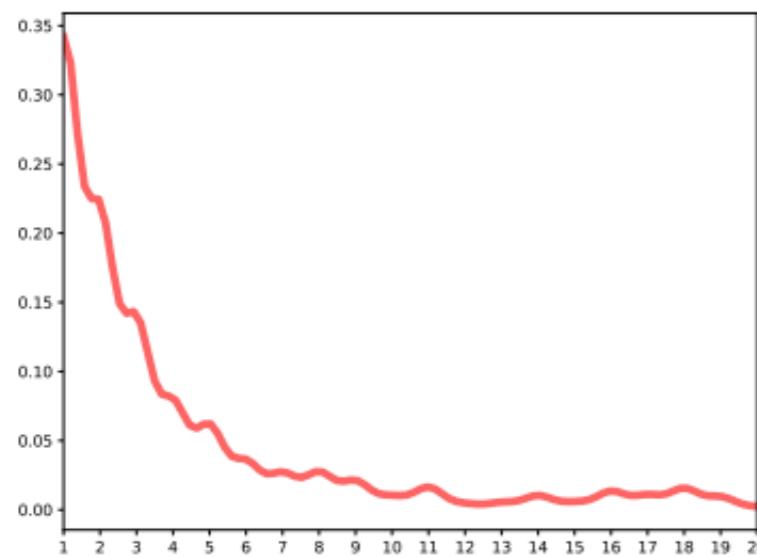


Hyperparameter behavior

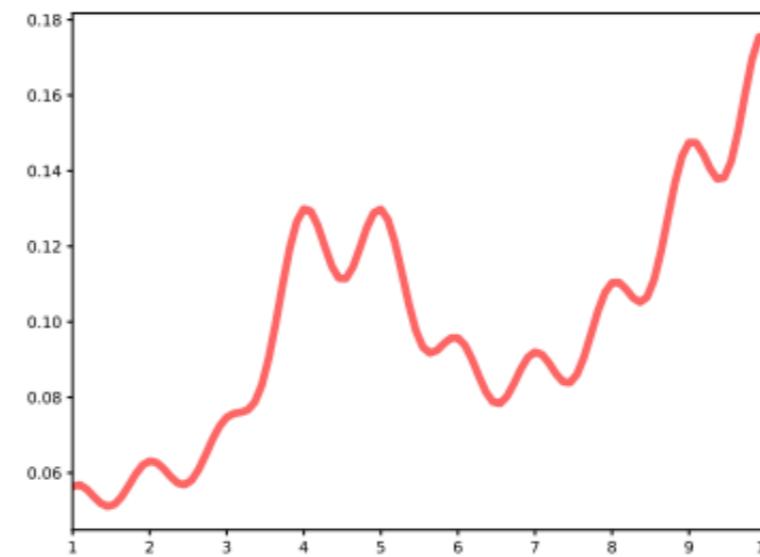
- **Functional ANOVA** ¹

Select hyperparameters that cause variance in the evaluations.

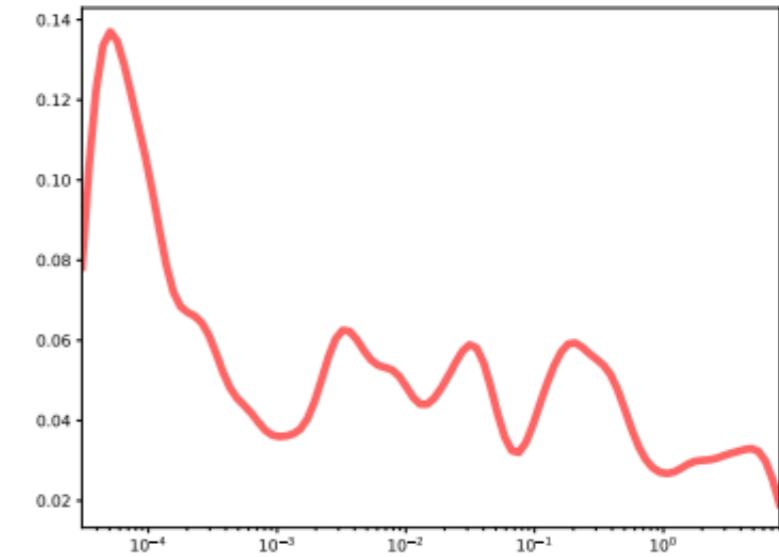
Learn priors for hyperparameter tuning (e.g. Bayesian optimization)



(a) RF: min. samples per leaf



(b) Adaboost: max. depth of tree



(c) SVM (RBF kernel): gamma

Priors (KDE) for the most important hyperparameters

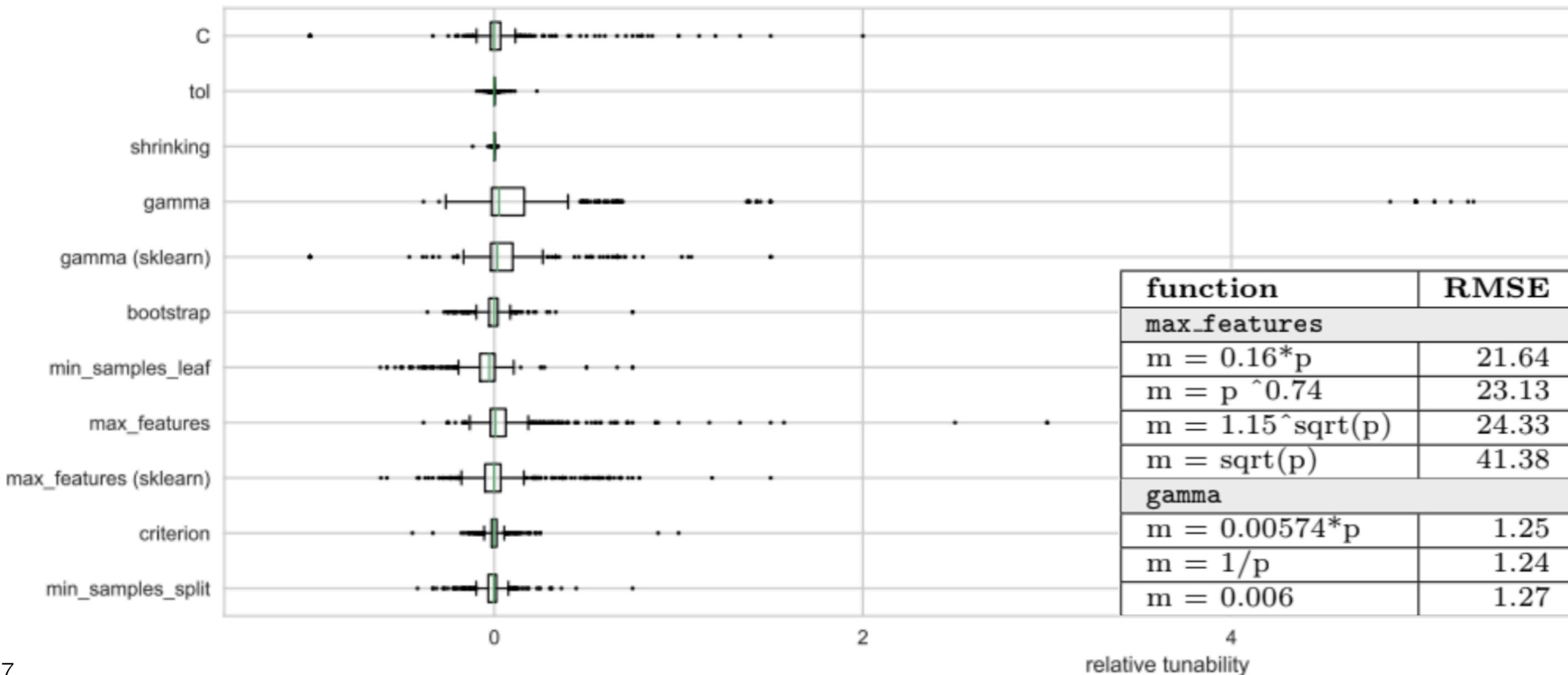
Hyperparameter behavior

- **Functional ANOVA** ¹

Select hyperparameters that cause variance in the evaluations.

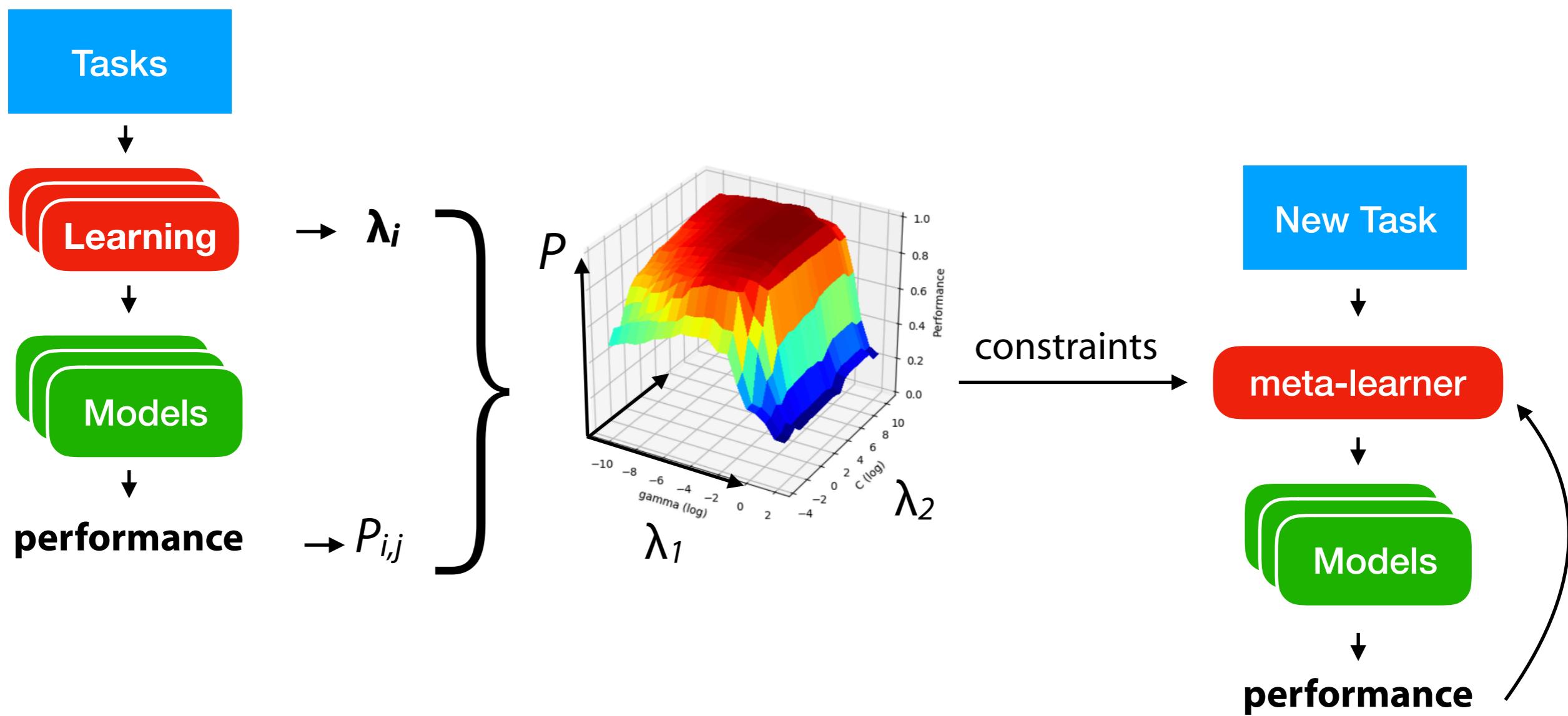
- **Tunability** ^{2,3,4}

Learn good defaults, measure improvement from tuning over defaults



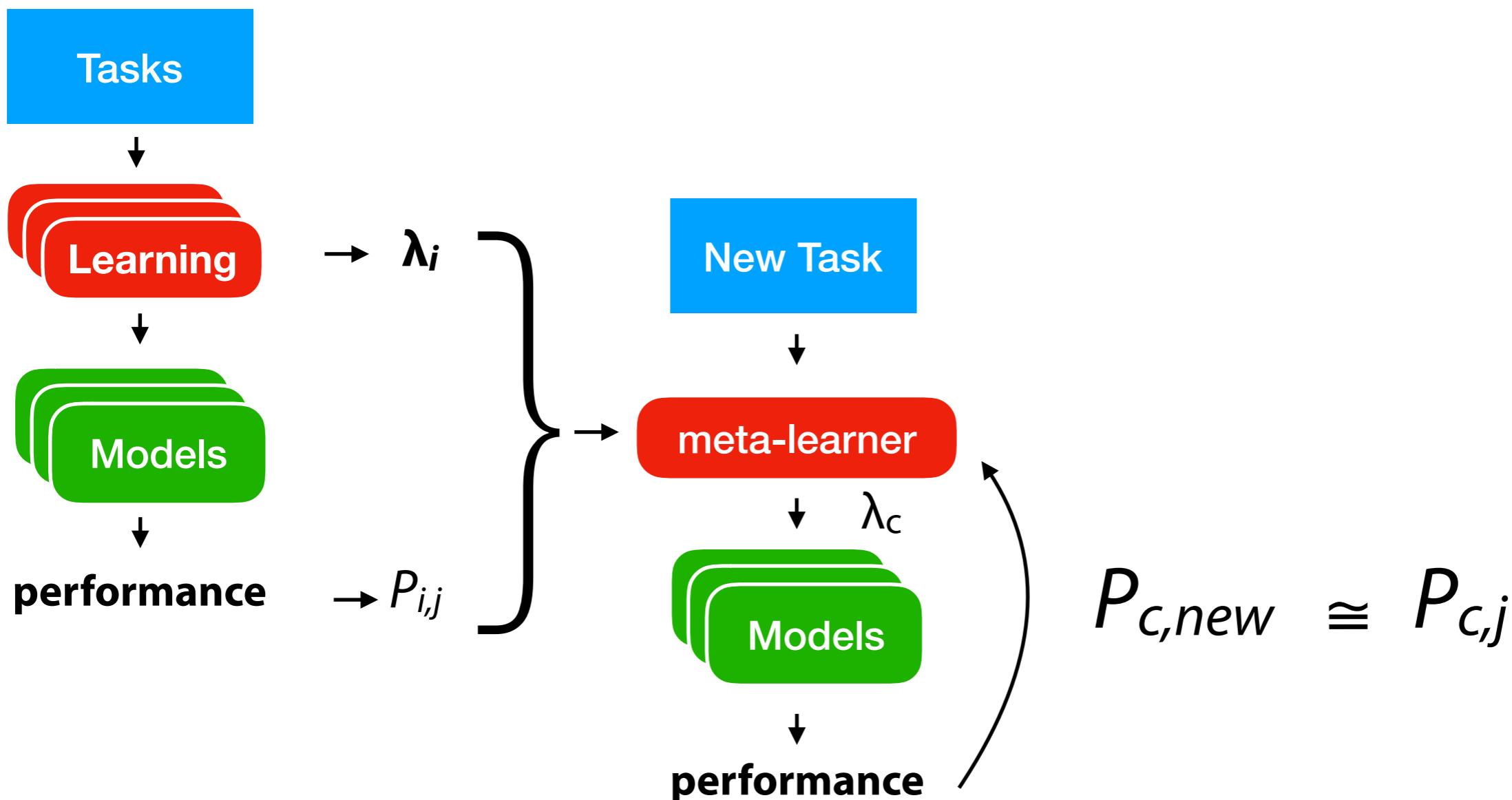
Hyperparameter behavior

- **Search space pruning**
Exclude regions yielding bad performance on (similar) tasks



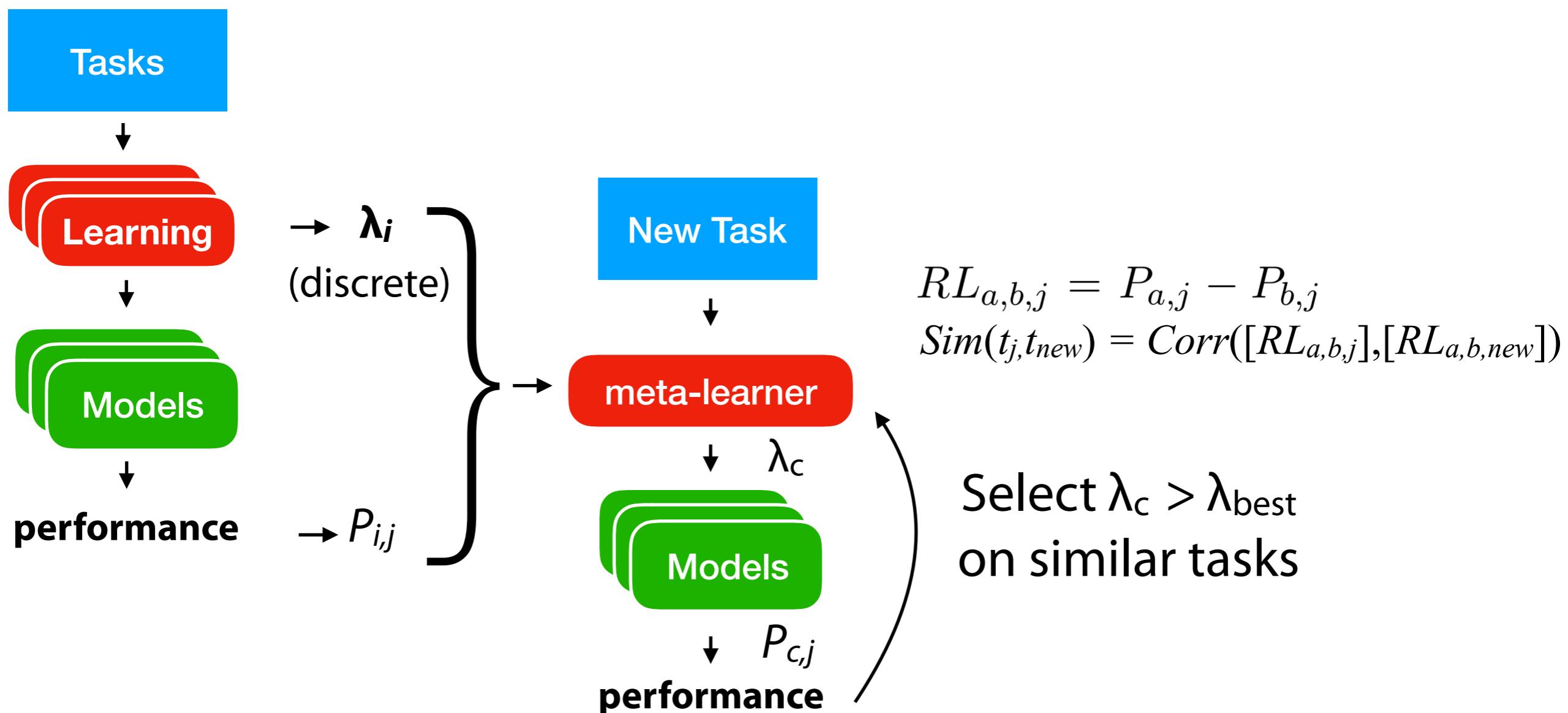
Learning task similarity

- Experiments on the *new task* can tell us how it is similar to *previous tasks*
- **Task are similar** if observed *performance* of configurations is similar
- Use this to recommend new configurations, run, repeat



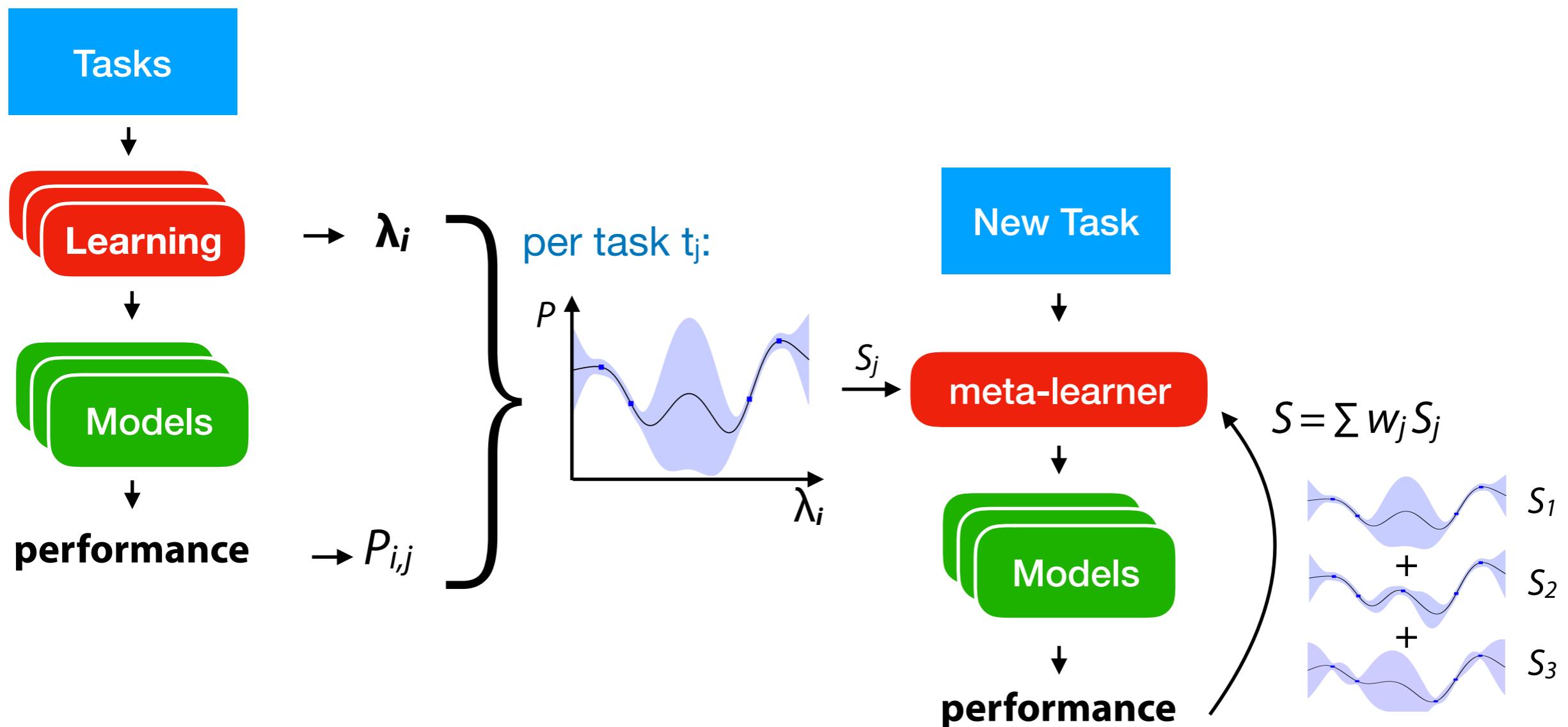
Active testing

- Learn task similarity while tuning configurations
- Tournament-style selection, warm-start with overall best config λ_{best}
- Next candidate λ_c : the one that beats current λ_{best} on similar tasks



Surrogate model transfer

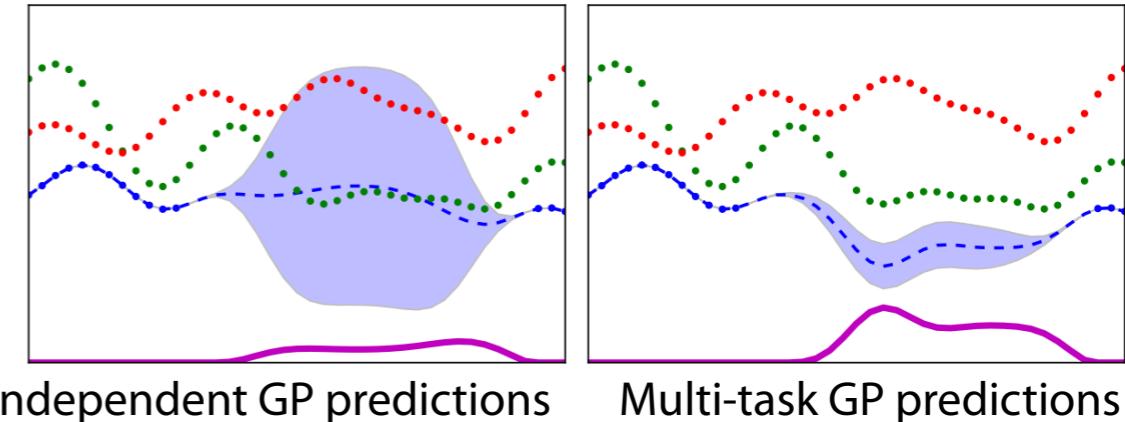
- If task j is *similar* to the new task, its surrogate model S_j will likely transfer well
- Sum up all S_j predictions, weighted by task similarity (as in active testing)¹
- Build combined Gaussian process, *weighted by current performance* on new task²



Multi-task Bayesian optimization

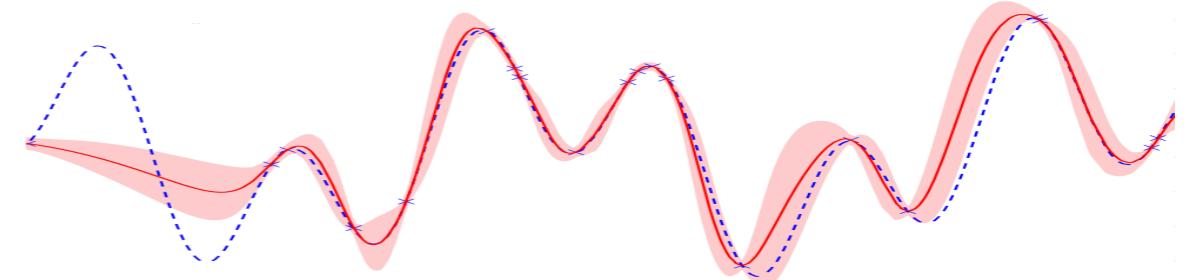
- **Multi-task Gaussian processes:** train surrogate model on t tasks simultaneously¹

- If tasks are similar: transfers useful info
- Not very scalable



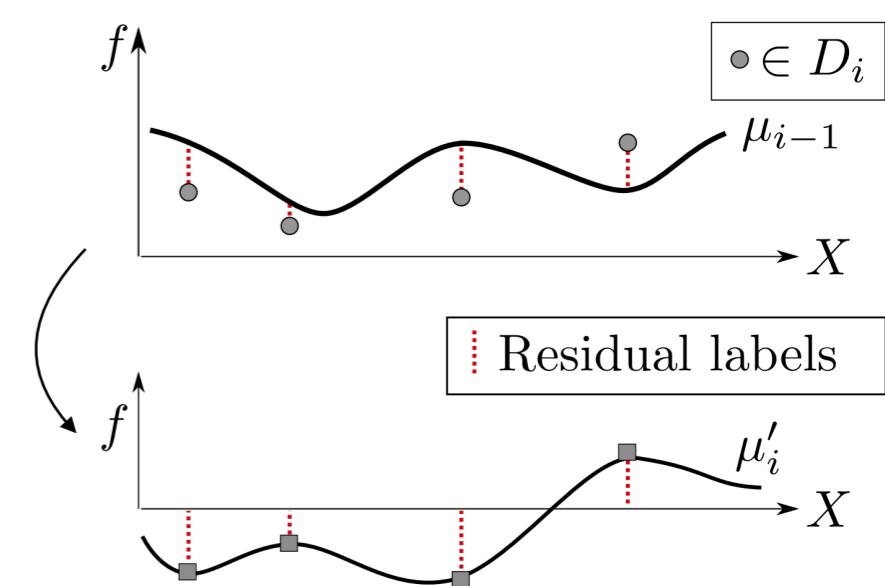
- **Bayesian Neural Networks** as surrogate model²

- Multi-task, more scalable



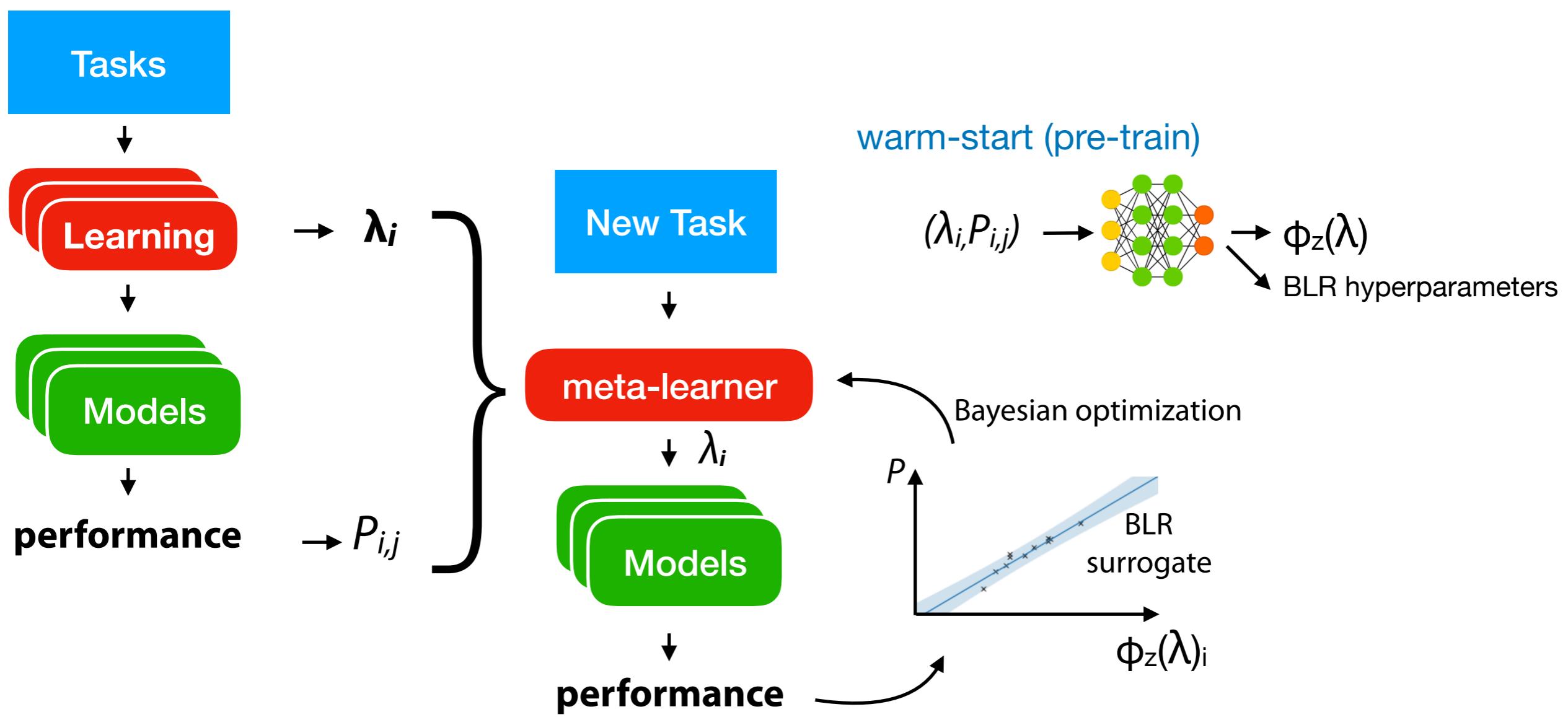
- **Stacking Gaussian Process regressors** (Google Vizier)³

- Continual learning (sequential similar tasks)
- Transfers a prior based on residuals of previous GP



More scalable variants

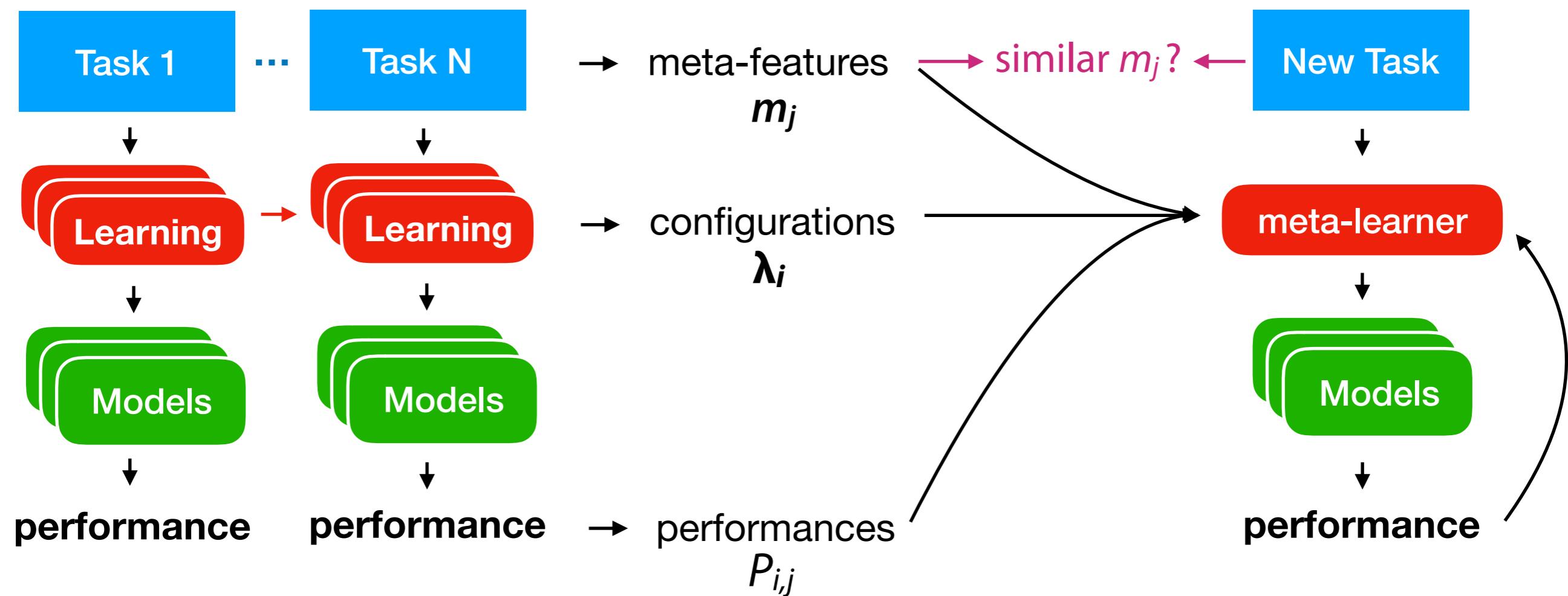
- Bayesian linear regression (BLR) surrogate model on every task
- Use neural net to learn a suitable basis expansion $\phi_z(\lambda)$ for all tasks
- Scales linearly in # observations, transfers info on configuration space



2. Learn what may likely work (for *partially similar* tasks)

Meta-features: measurable properties of the tasks

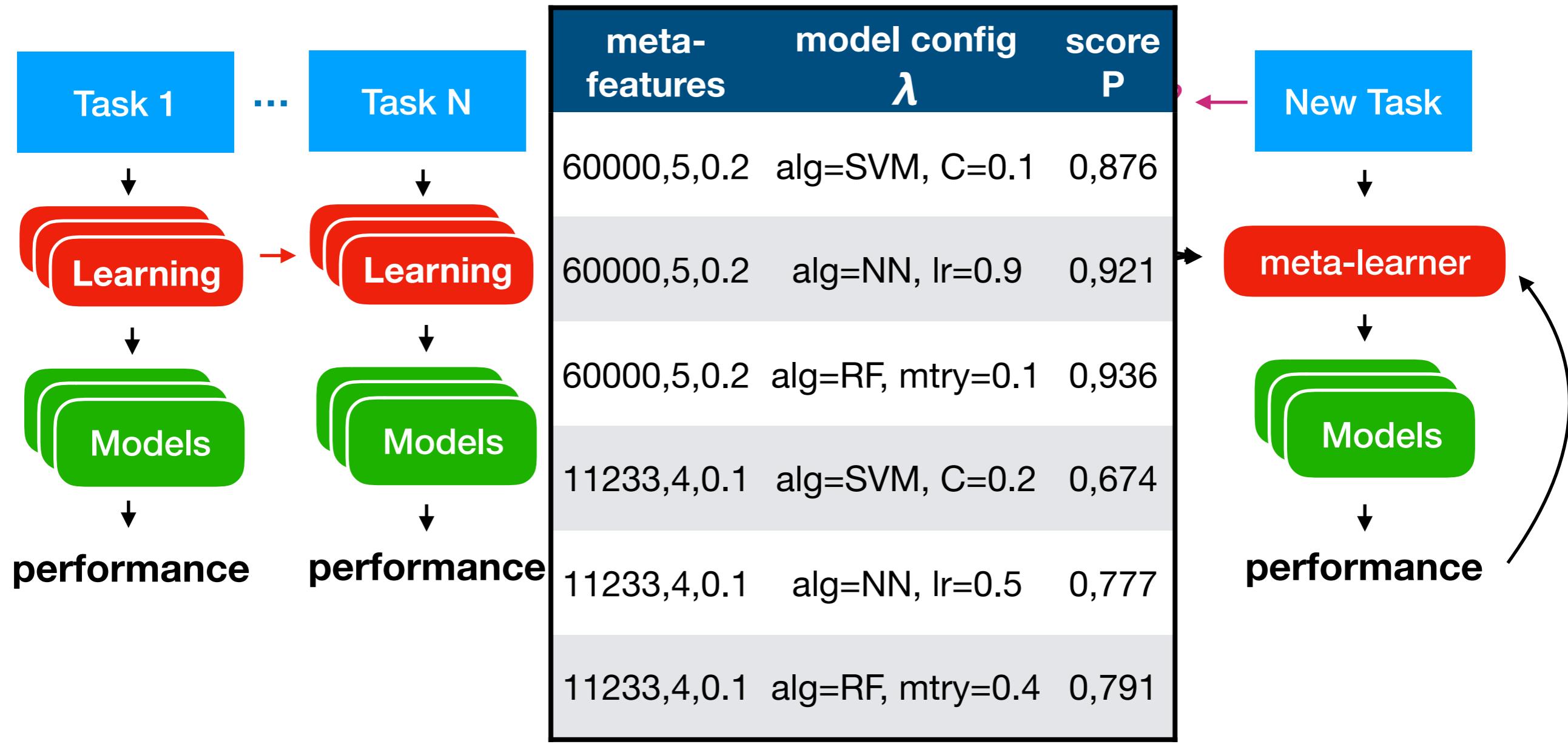
(number of instances and features, class imbalance, feature skewness,...)



2. Learn what may likely work (for *partially similar* tasks)

Meta-features: measurable properties of the tasks

(number of instances and features, class imbalance, feature skewness,...)



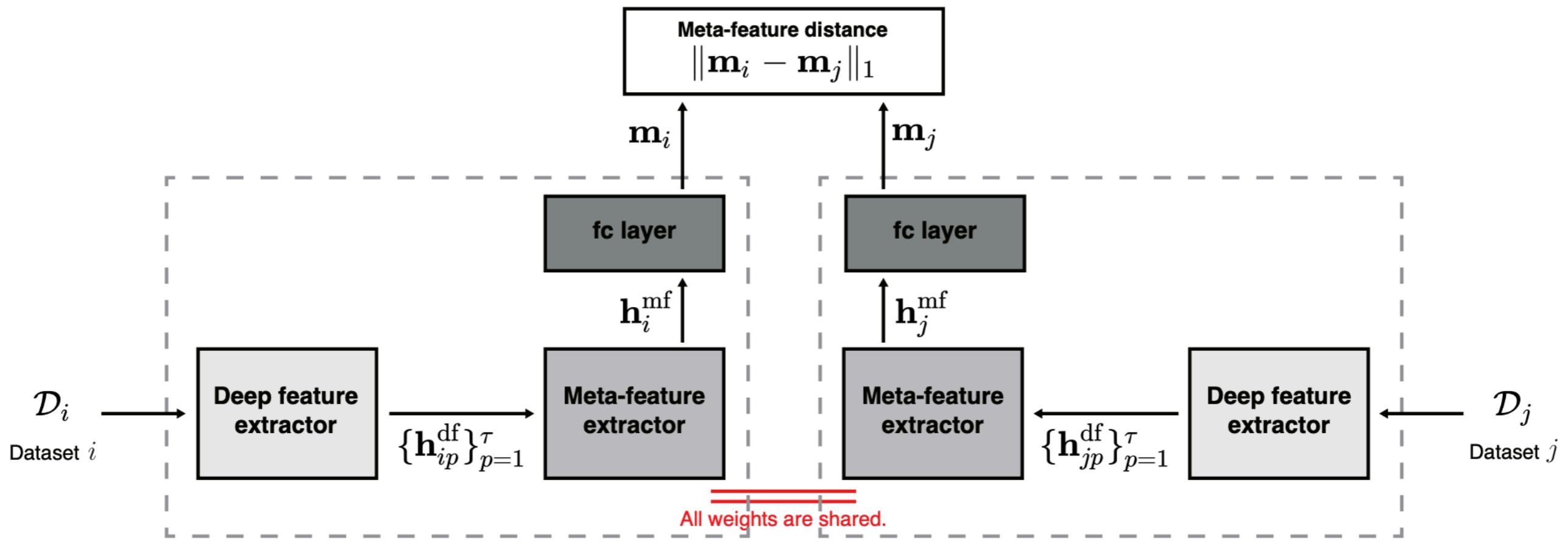
Meta-features

- **Hand-crafted (interpretable) meta-features¹**
 - **Number of** instances, features, classes, missing values, outliers,...
 - **Statistical:** skewness, kurtosis, correlation, covariance, sparsity, variance,...
 - **Information-theoretic:** class entropy, mutual information, noise-signal ratio,...
 - **Model-based:** properties of simple models trained on the task
 - **Landmarkers:** performance of fast algorithms trained on the task
 - Domain specific task properties
- Optimized (recommended) representations exist ²

Meta-features

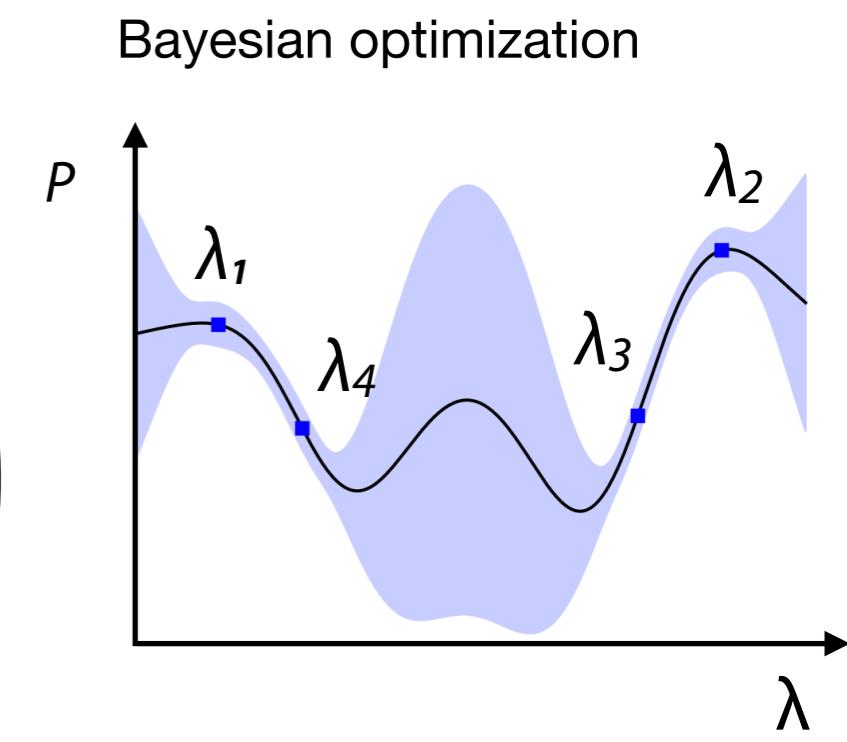
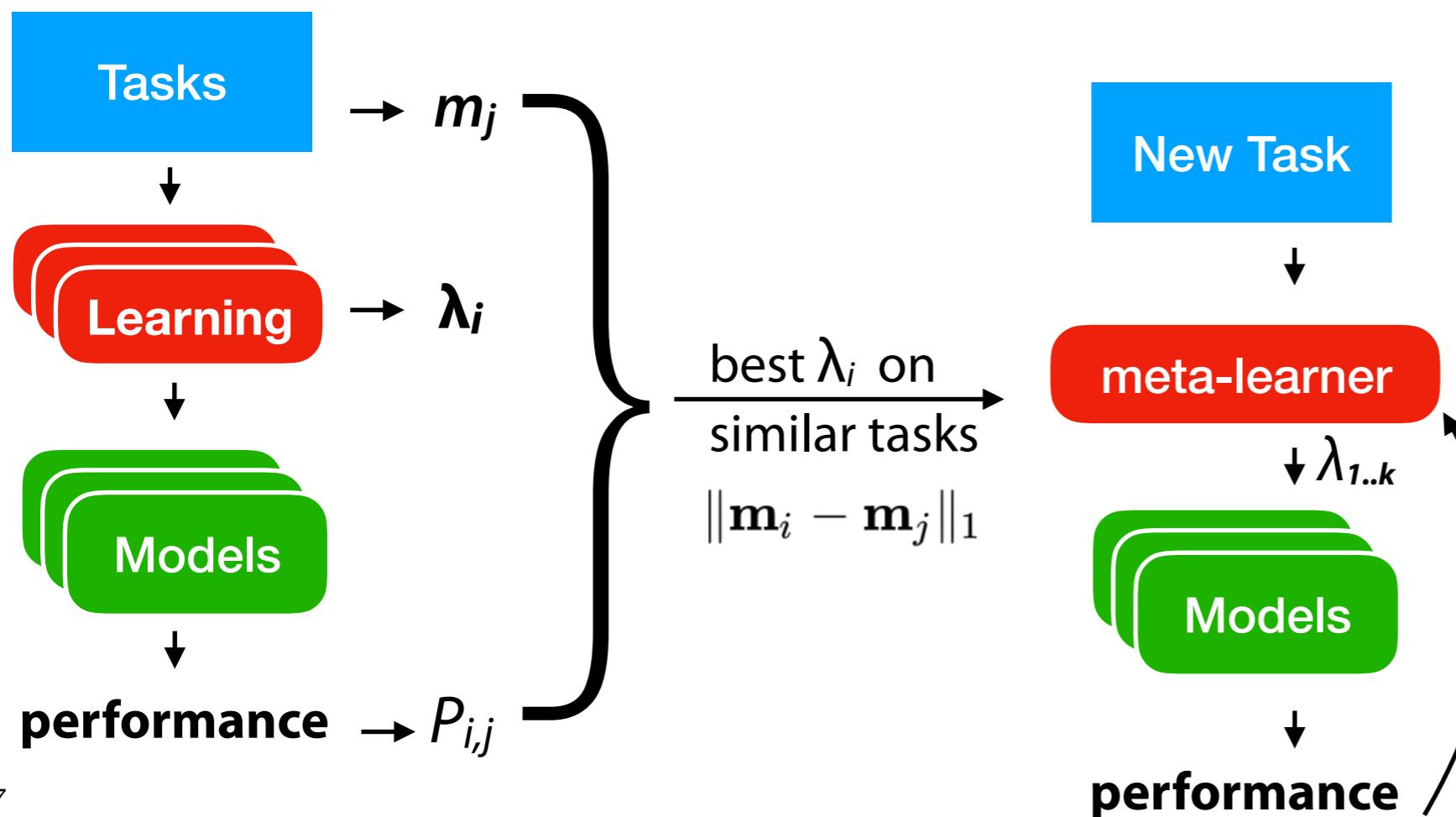
- **Learning a joint task representation**

- Deep metric learning: learn a representation h^{mf} using ground truth distance
 - Siamese Network: similar task, similar representation ¹
 - Feed through pretrained CNN, extract vector from weights ²
- Recommend neural architectures, or warm-start neural architecture search



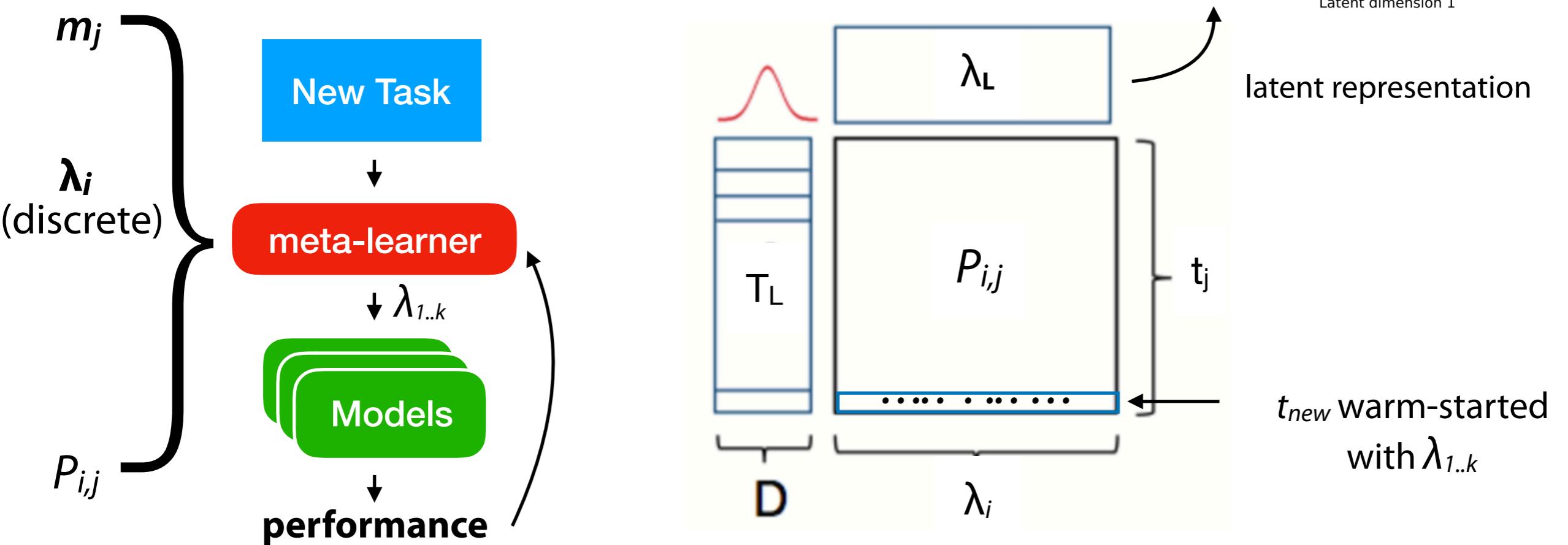
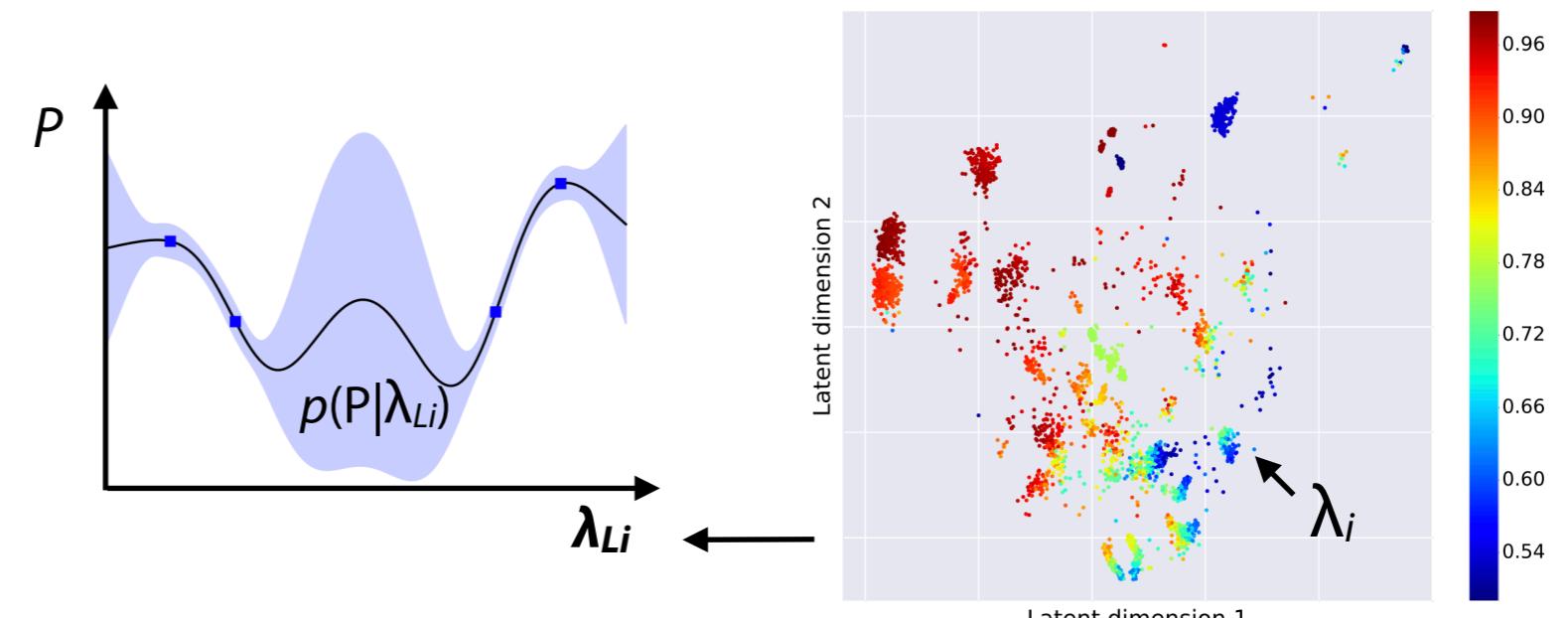
Warm-starting from similar tasks

- Find k most similar tasks, warm-start search with best λ_i
 - Auto-sklearn: Bayesian optimization (SMAC)
 - Meta-learning yield better models, faster
 - Winner of AutoML Challenges



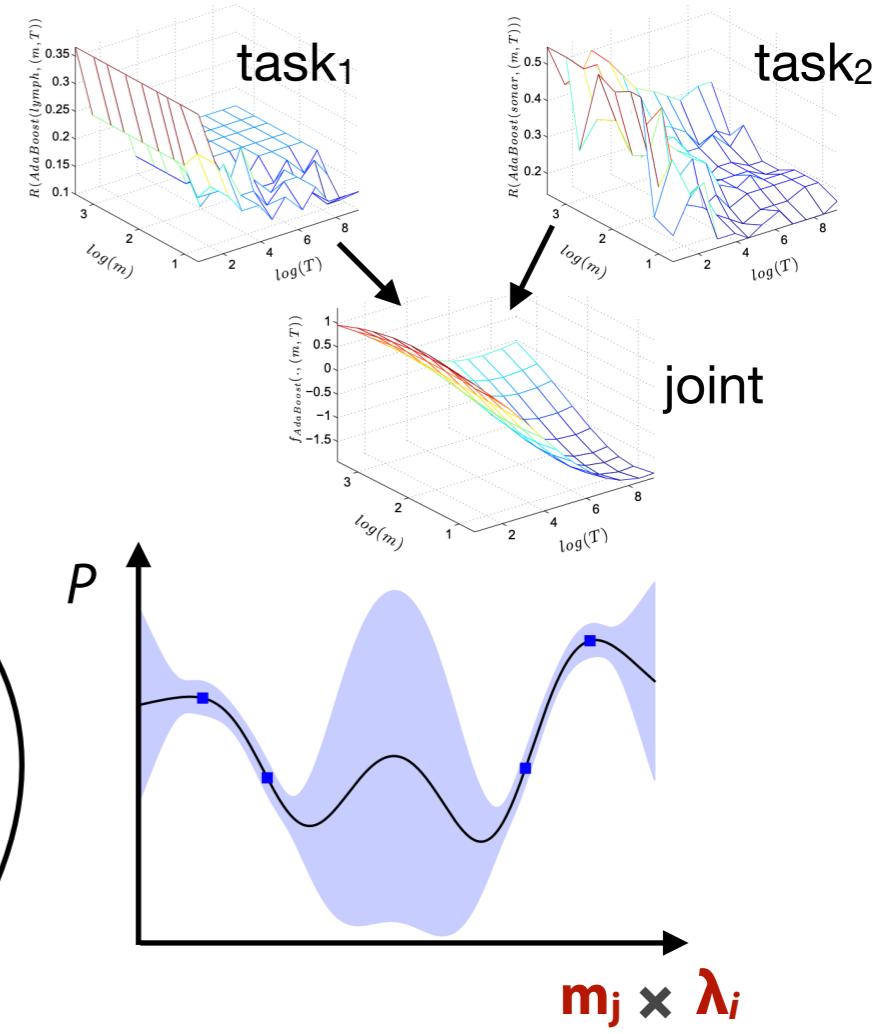
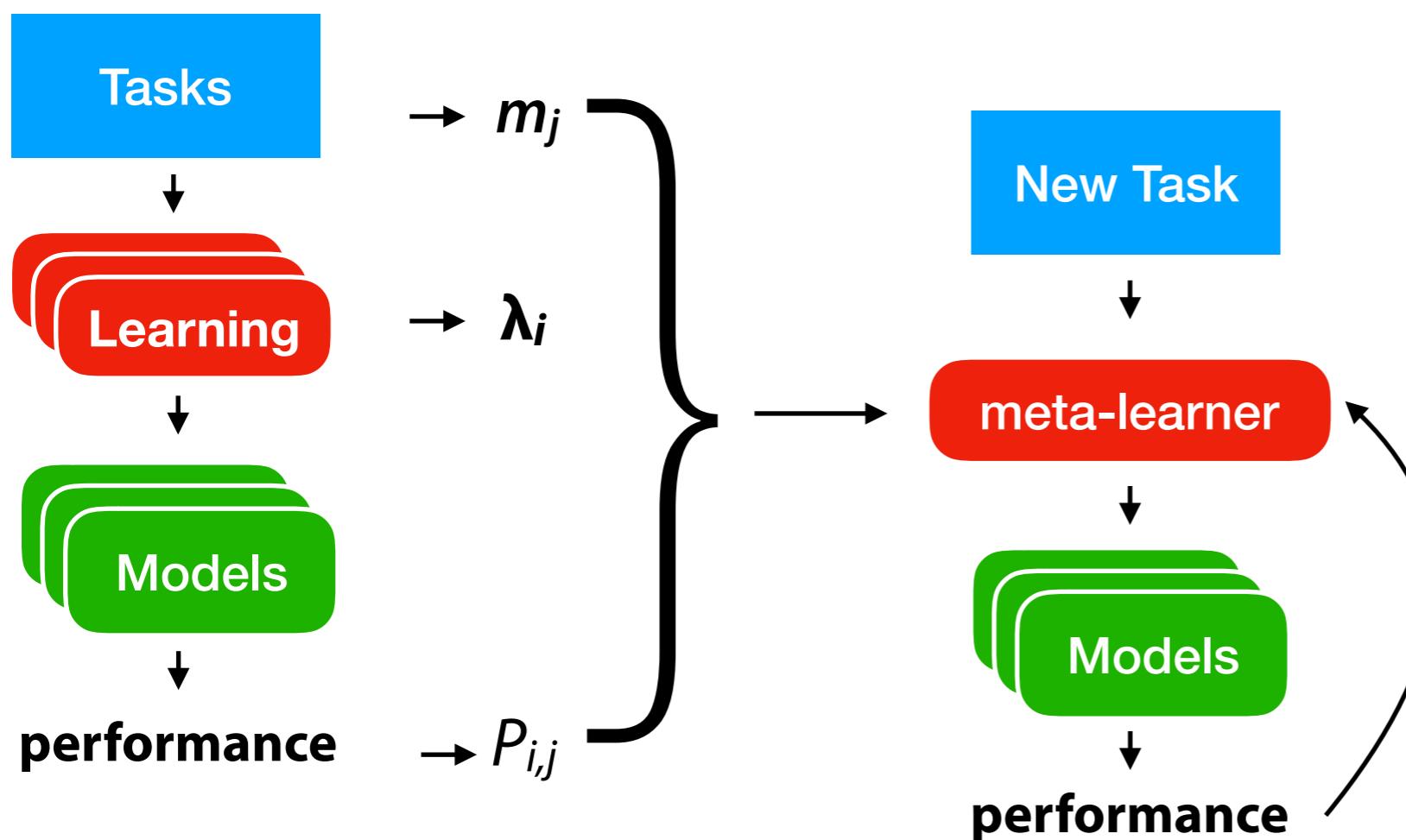
Warm-starting from similar tasks

- Collaborative filtering: configurations λ_i are ‘rated’ by tasks t_j
- Learn latent representation for tasks and configurations
- Use meta-features to warm-start on new task
- Returns probabilistic predictions for BayesOpt



Global surrogate models

- Train a task-independent surrogate model with meta-features in inputs
 - SCOT: Predict *ranking* of λ_i with surrogate ranking model + m_j . ¹
 - Predict $P_{i,j}$ with multilayer Perceptron surrogates + m_j . ²
 - Build joint GP surrogate model on most similar ($\|\mathbf{m}_i - \mathbf{m}_j\|_2$) tasks. ³
 - **Scalability is often an issue**



Meta-models

- Learn direct mapping between meta-features and $P_{i,j}$
 - Zero-shot meta-models: predict best λ_i given meta-features ¹



- Ranking models: return ranking $\lambda_{1..k}$ ²



- Predict which algorithms / configurations to consider / tune ³



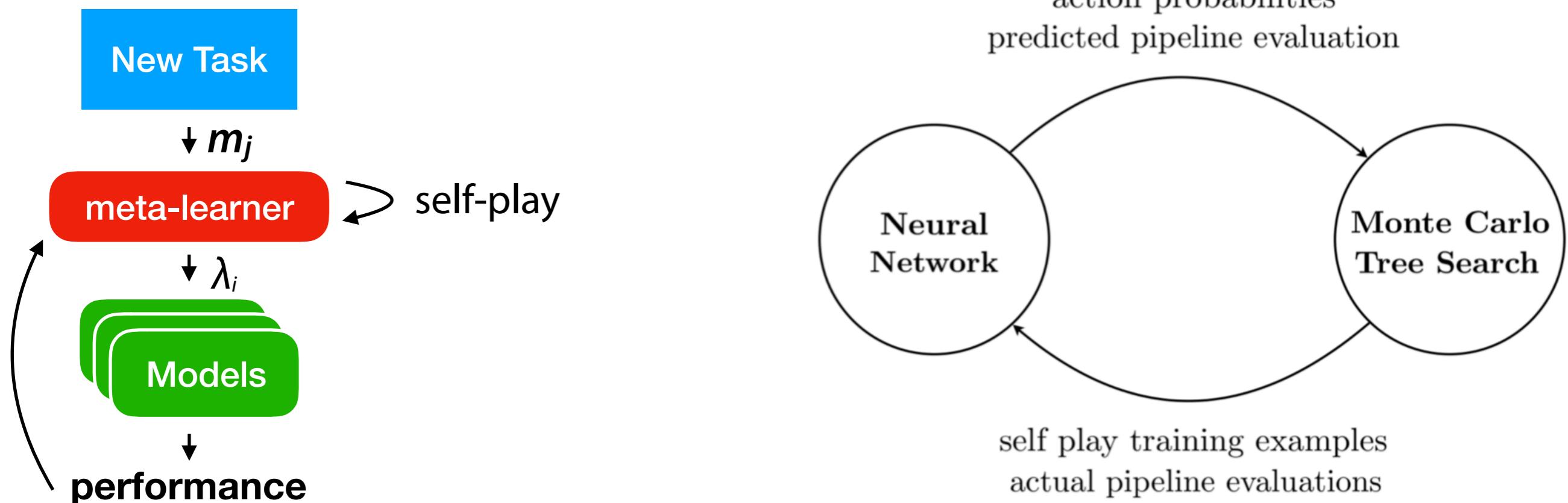
- Predict performance / runtime for given θ_i and task ⁴



- Can be integrated in larger AutoML systems: warm start, guide search,...

Learning to learn through self-play

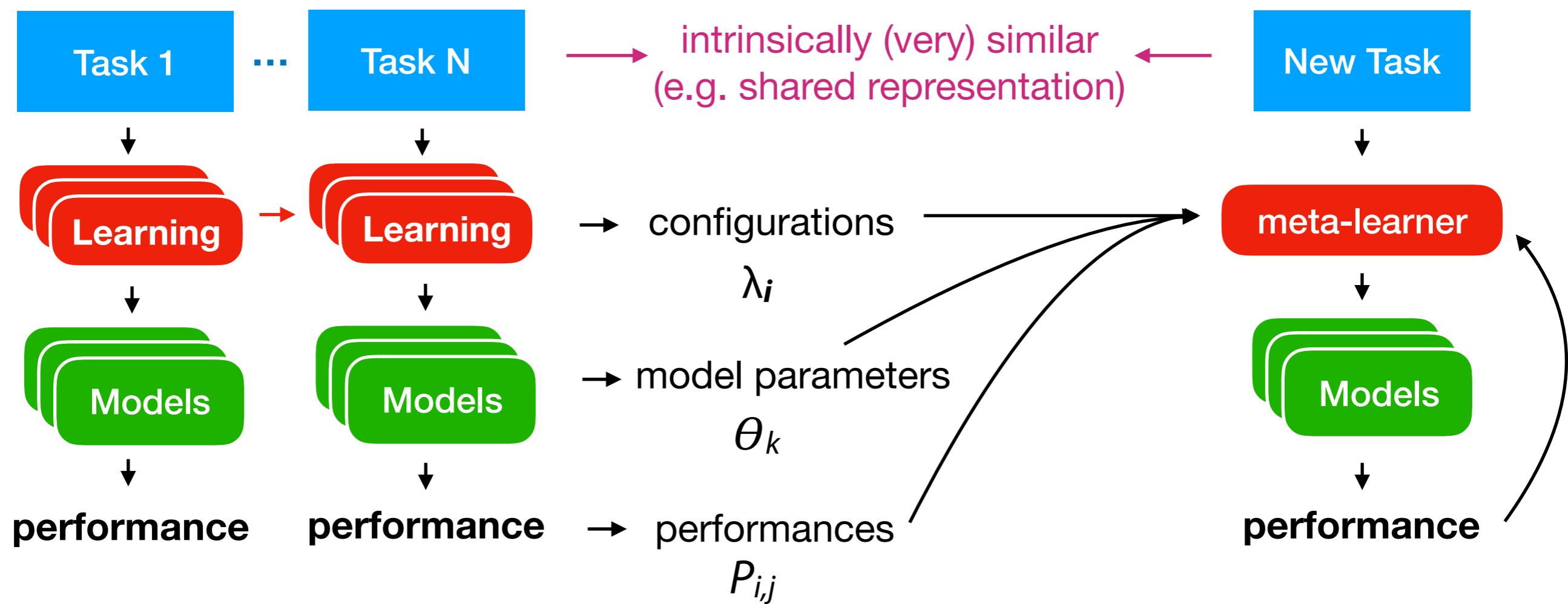
- Build pipelines by inserting, deleting, replacing components (actions)
- Neural network (LSTM) receives task meta-features, pipelines and evaluations
 - Predicts pipeline performance and action probabilities
- Monte Carlo Tree Search builds pipelines, runs simulations against LSTM



3. Learn from previous *models* (for *very similar* tasks)

Models trained on *intrinsically similar* tasks

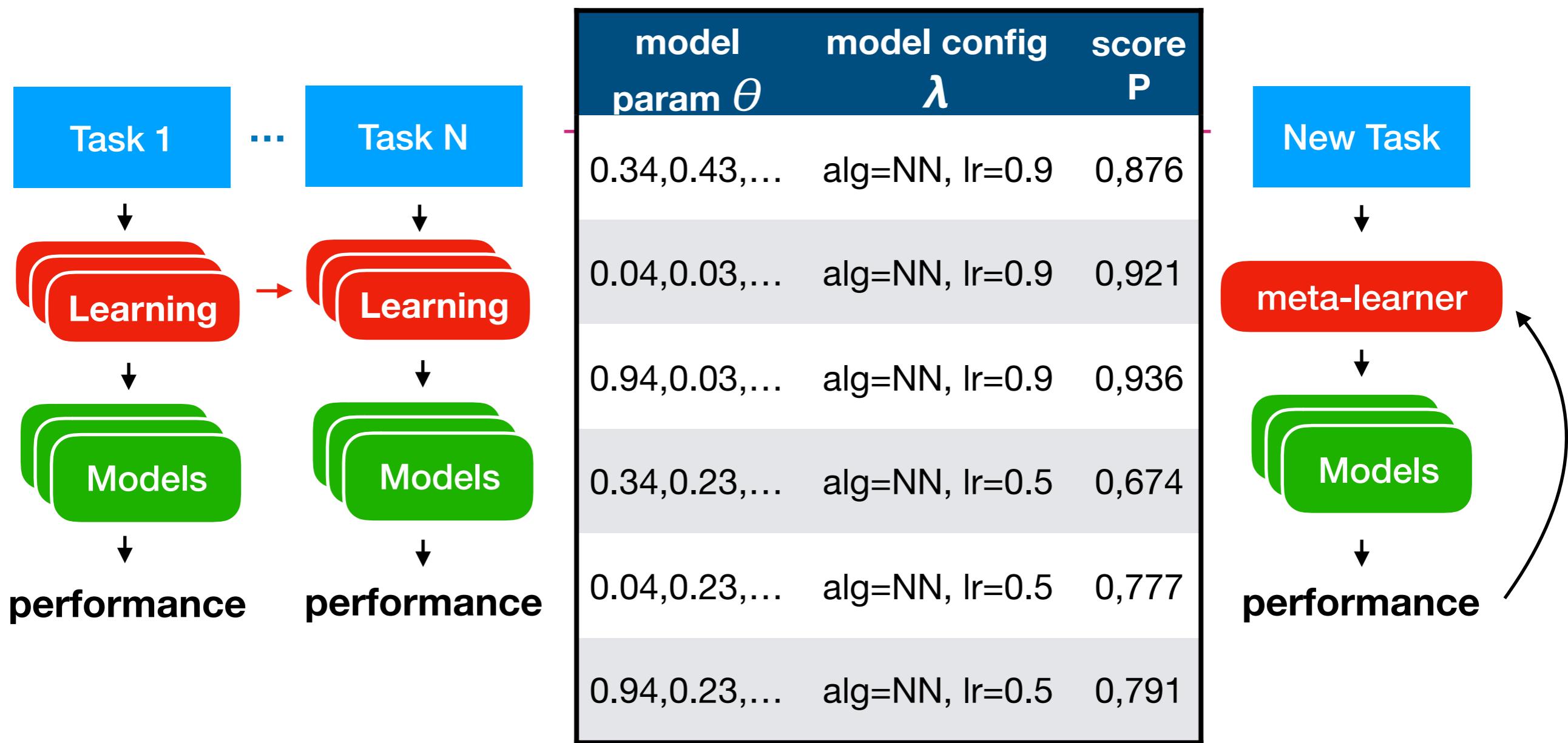
(model parameters, features,...)



3. Learn from previous *models* (for *very similar* tasks)

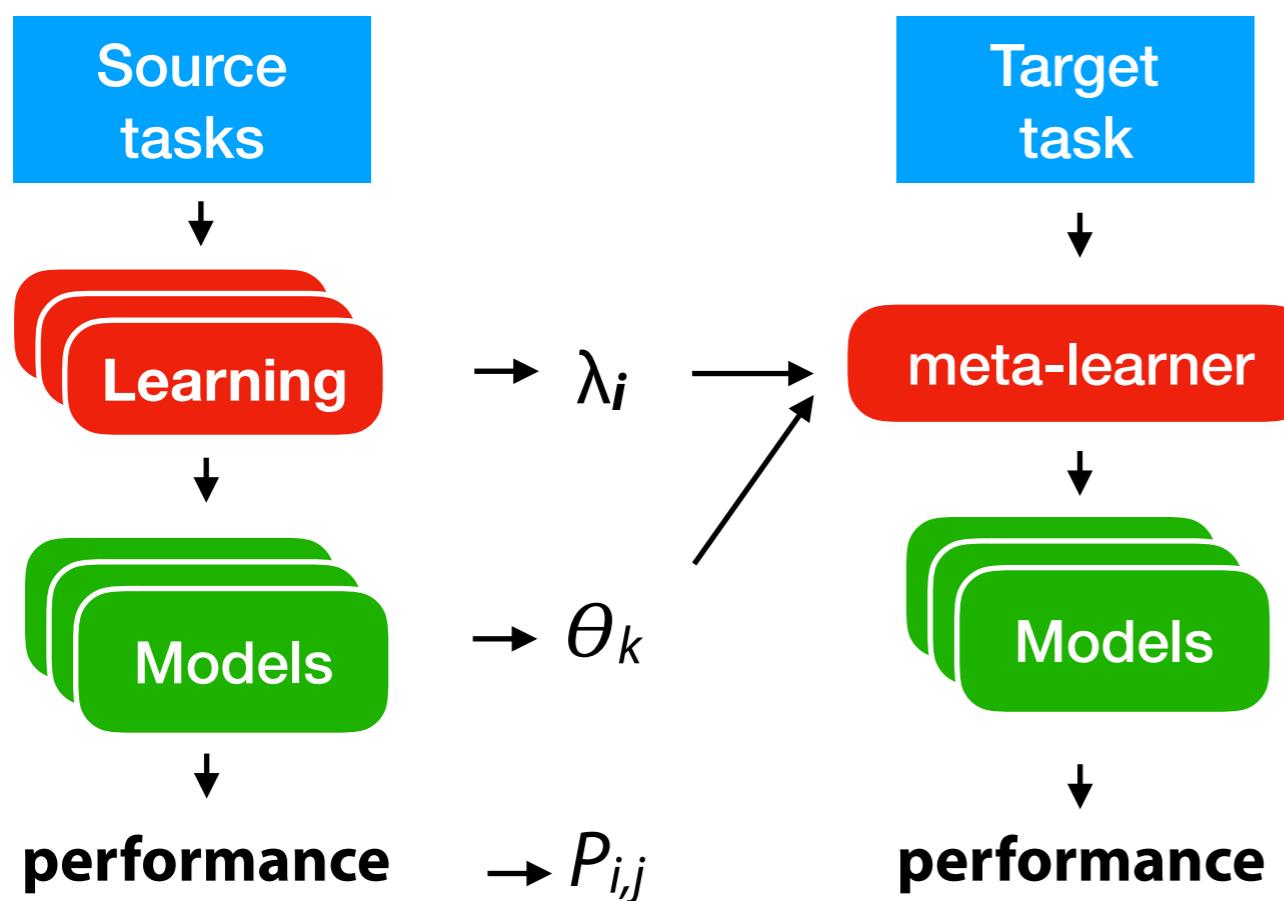
Models trained on *intrinsically similar* tasks

(model parameters, features,...)

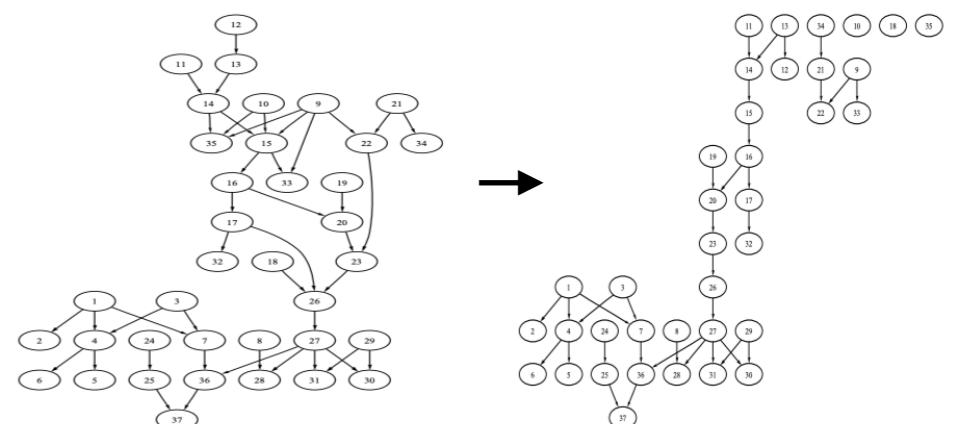


Transfer Learning

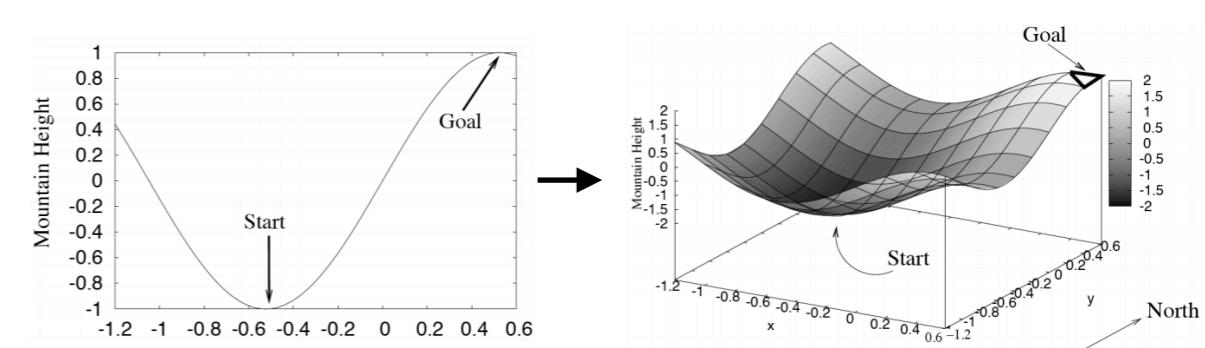
- Select source tasks, transfer trained models to similar target task ¹
- Use as starting point for tuning, or *freeze* certain aspects (e.g. structure)
 - Bayesian networks: start structure search from prior model ²
 - Reinforcement learning: start policy search from prior policy ³



Bayesian Network transfer



Reinforcement learning: 2D to 3D mountain car



Transfer learning

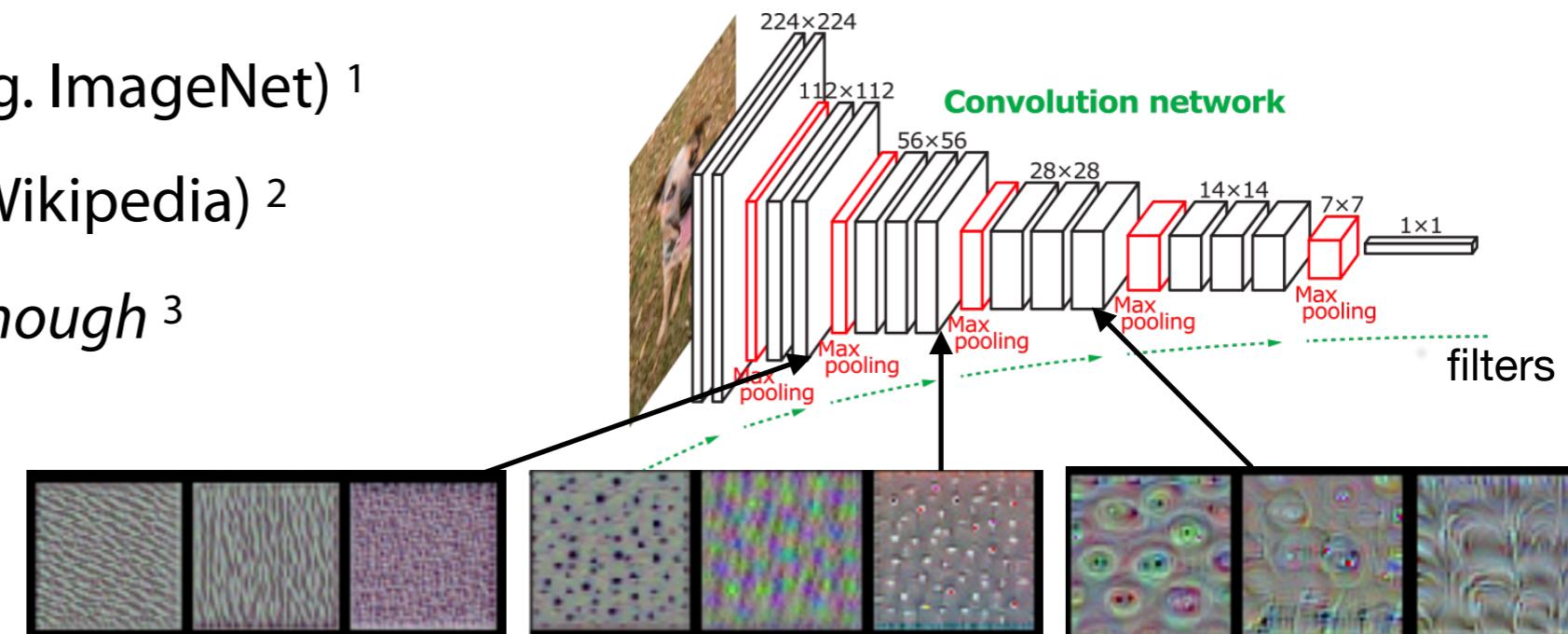
- For neural networks, both structure and weights can be transferred
- Features and initializations learned from:
 - Large image datasets (e.g. ImageNet) ¹
 - Large text corpora (e.g. Wikipedia) ²
- Fails if tasks are *not similar enough* ³

Source
tasks

Learning

pre-trained
convnet

performance



frozen new

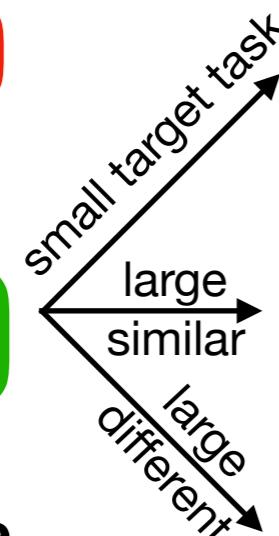
pre-trained new

frozen new

Feature extraction:
remove last layers, use output as features
if task is quite different, remove more layers

End-to-end tuning:
train from initialized weights

Fine-tuning:
unfreeze last layers, tune on new task



Learning to learn Neural networks

- End-to-end differentiable models afford many meta-learning opportunities
 - Transfer learning
 - Learning gradient descent procedures (weight update rules)
 - Few shot learning
 - Model-agnostic meta-learning (learning weight initializations)
 - Learning to reinforcement learn
 - ...

See part 3

Meta-data sharing building a shared memory

- What if we have a shared memory of all machine learning experiments?
- [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,...
 - Publish your own runs
 - Never ending learning
 - Benchmarks

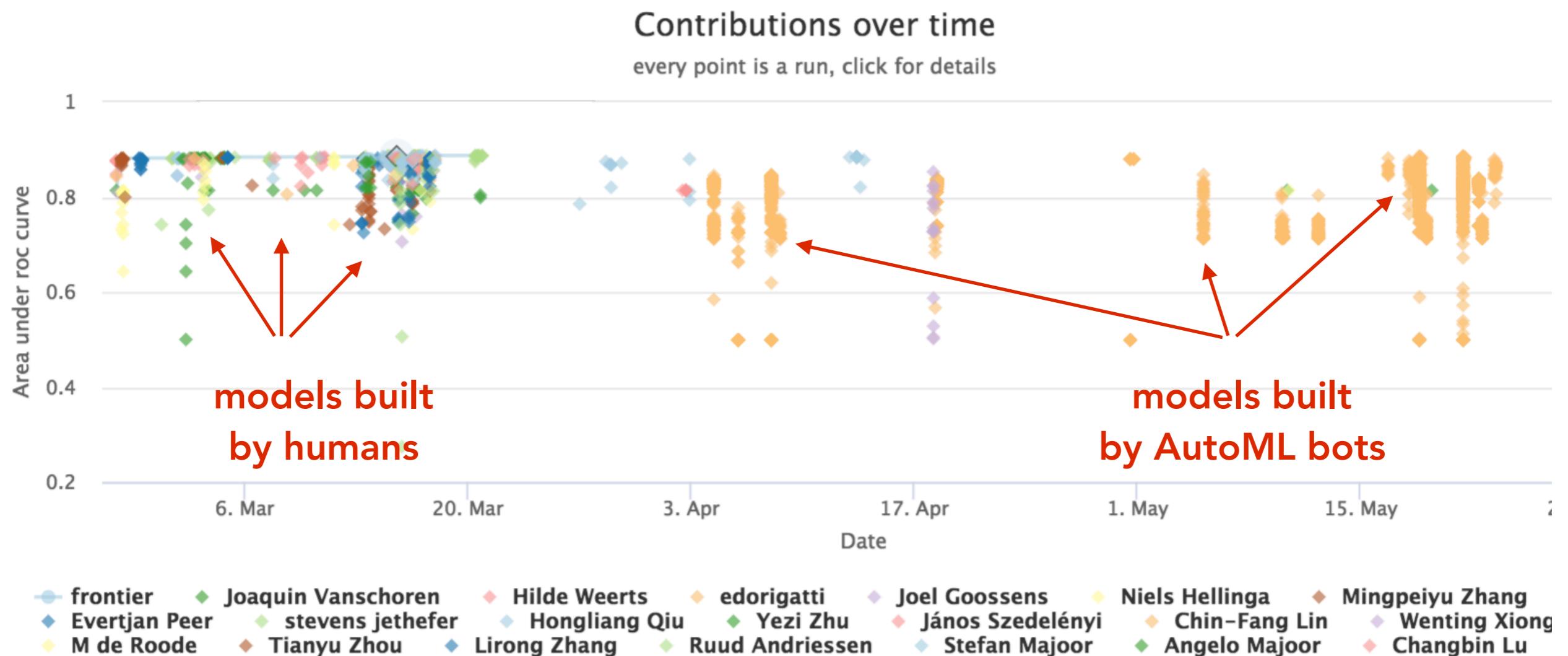
```
import openml as om1
from sklearn import tree

task = om1.tasks.get_task(14951)
clf = tree.ExtraTreeClassifier()
flow = om1.flows.sklearn_to_flow(clf)
run = om1.runs.run_flow_on_task(task, flow)
myrun = run.publish()
```

run *locally, share globally*

Meta-data sharing building a shared memory

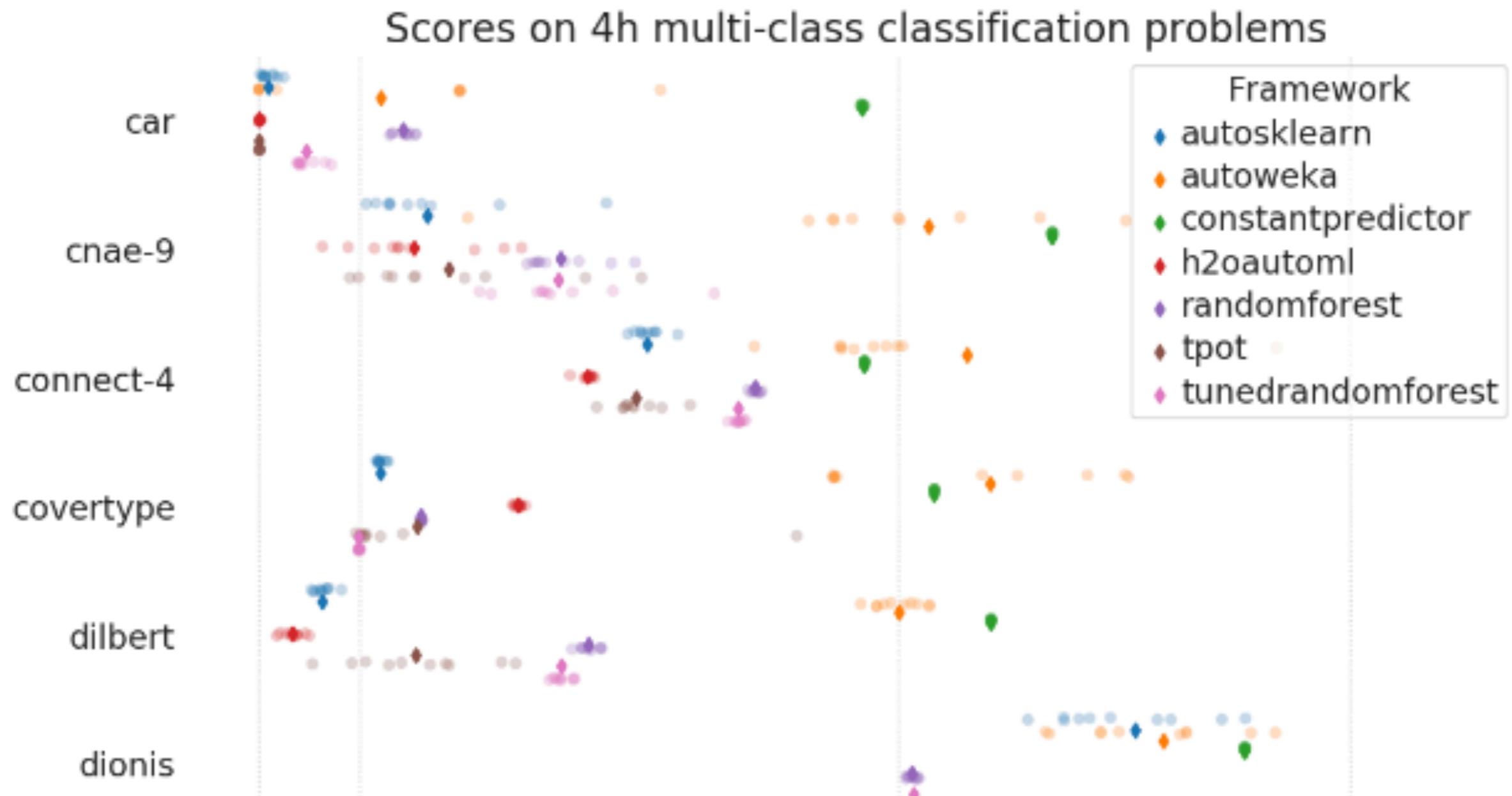
- Train and run AutoML bots on any OpenML dataset
- Never-ending learning (from robots and humans)



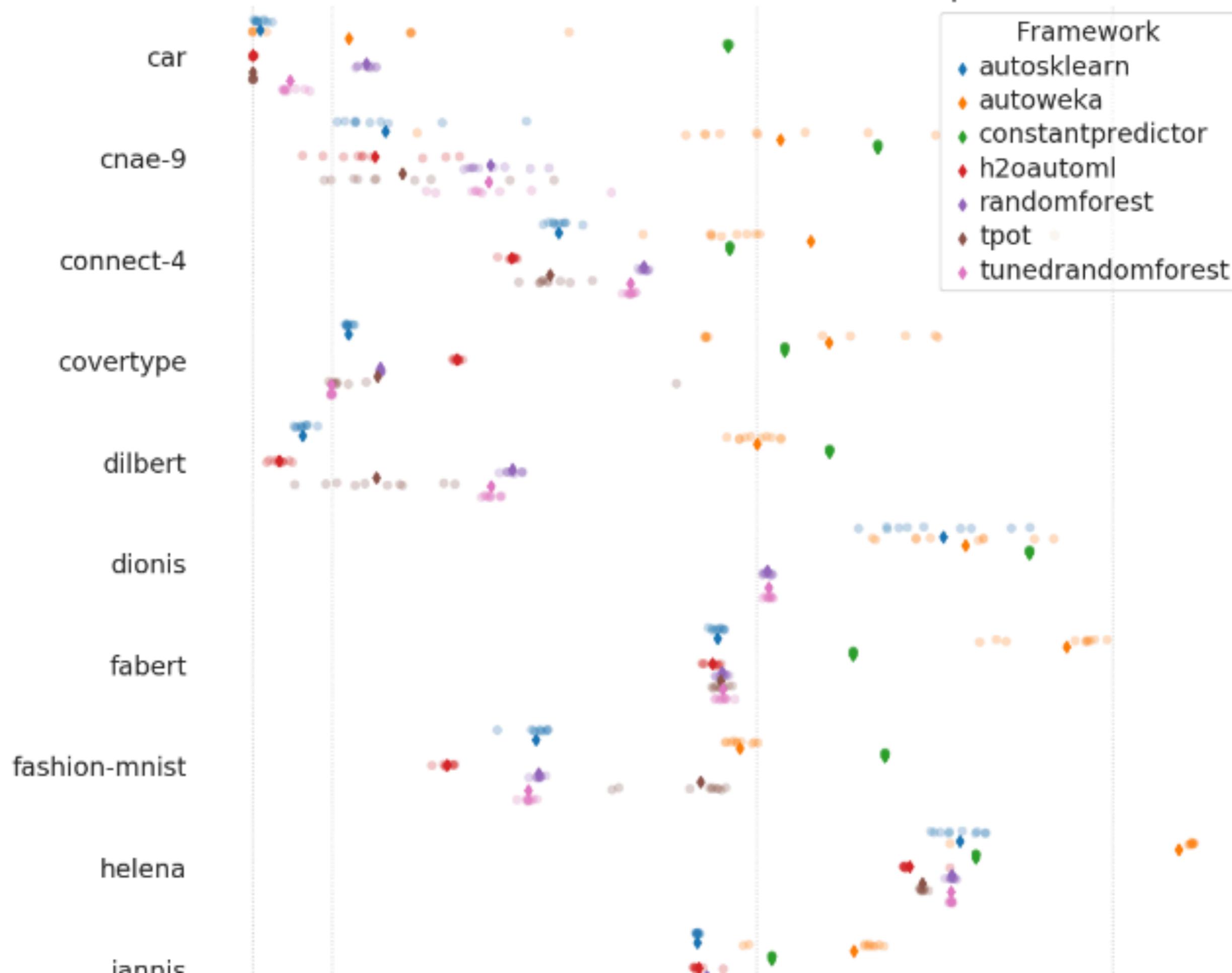
AutoML tools and benchmarks

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

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



Scores on 4h multi-class classification problems



AutoML open source tools

	Architecture search	Hyperparameter search	Improvements	Meta-learning
<u>Auto-WEKA</u>	Parameterized pipeline	Bayesian Optimization (RF)		
<u>auto-sklearn</u>	Parameterized pipeline	Bayesian Optimization (RF)	Ensembling	warm-start
BO-HB	Parameterized pipeline	Tree of Parzen Estimators	Ensembling, Hyperband	
<u>hyperopt-sklearn</u>	Parameterized pipeline	Tree of Parzen Estimators		
<u>TPOT</u>	Evolving pipelines (trees)	Genetic programming		
<u>GAMA</u>	Evolving pipelines (trees)	Asynchronous evolution	Ensembling, Hyperband	
<u>H2O AutoML</u>	Single algorithms	random search	Stacking	
<u>ML-Plan</u>	Planning-based pipeline	Hierarchical planner		
<u>OBOE</u>	Single algorithms	Low rank approximation	Ensembling	Runtime predictor

AutoML tools and benchmarks

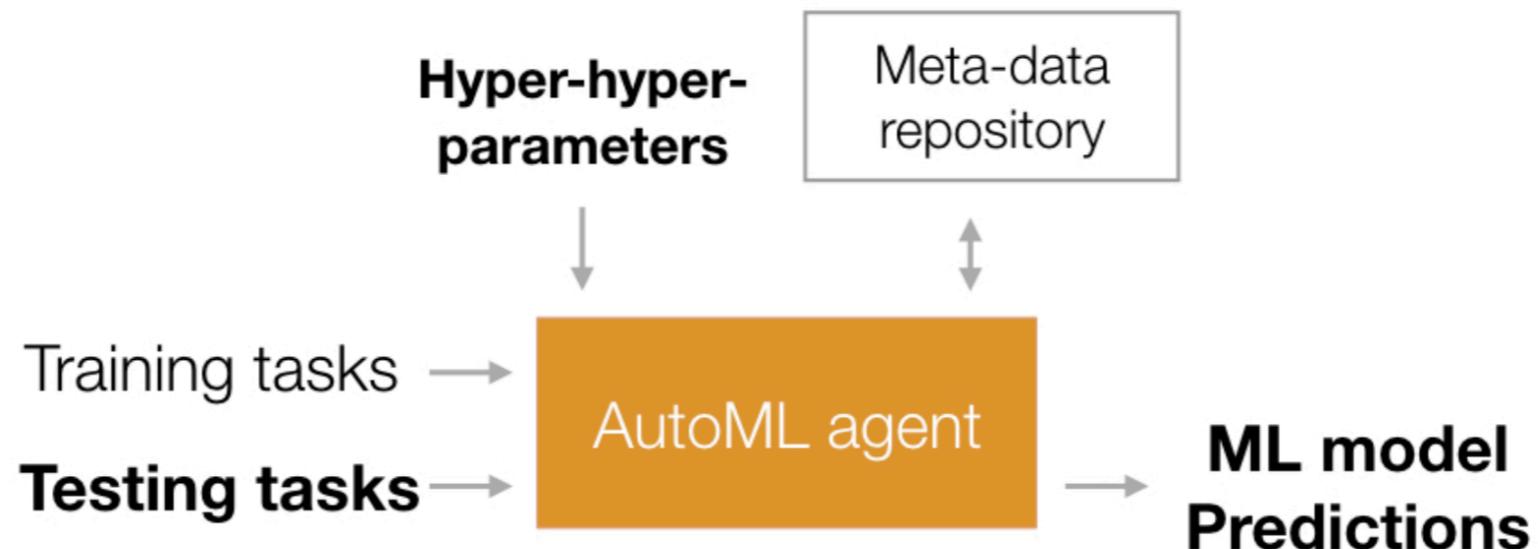
Observations so far:

- No AutoML system outperforms all others (Auto-WEKA is worse)
- Auto-sklearn, H2O, TPOT can outperform each other on given datasets
- Tuned RandomForests are a strong baseline
- High numbers of classes are a problem for most tools
- Unbalanced classes are a problem for most tools
- Little support for anytime prediction ...

AutoML Gym

Environment to train general-purpose AutoML algorithms

- Open benchmark, meta-learning, continual learning



Open PostDoc position!
<http://bit.ly/automl-gym>

Thank you!

Never stop learning



Part 3: *Deeper* into AutoML and meta-learning