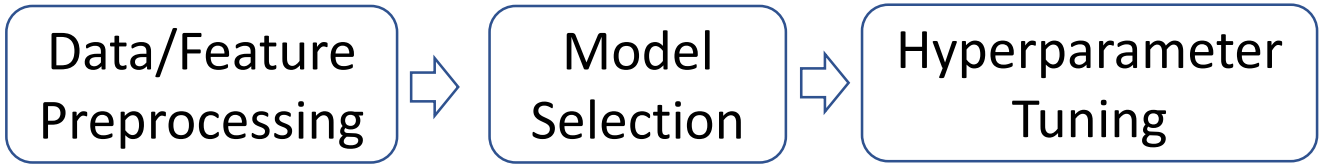
AutoML: A Perspective where Industry Meets Academy

Yaliang Li, Zhen Wang, Yuexiang Xie, Bolin Ding, and Ce Zhang

Alibaba Group, ETH Zürich

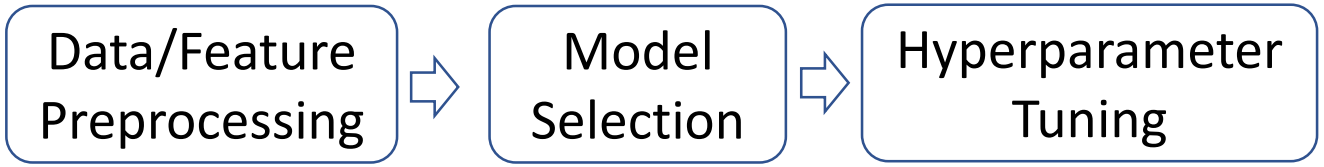
Machine Learning Pipeline



So **MANY** choices

- Which feature transformation?
- Which model architecture?
- Which hyperparameters?

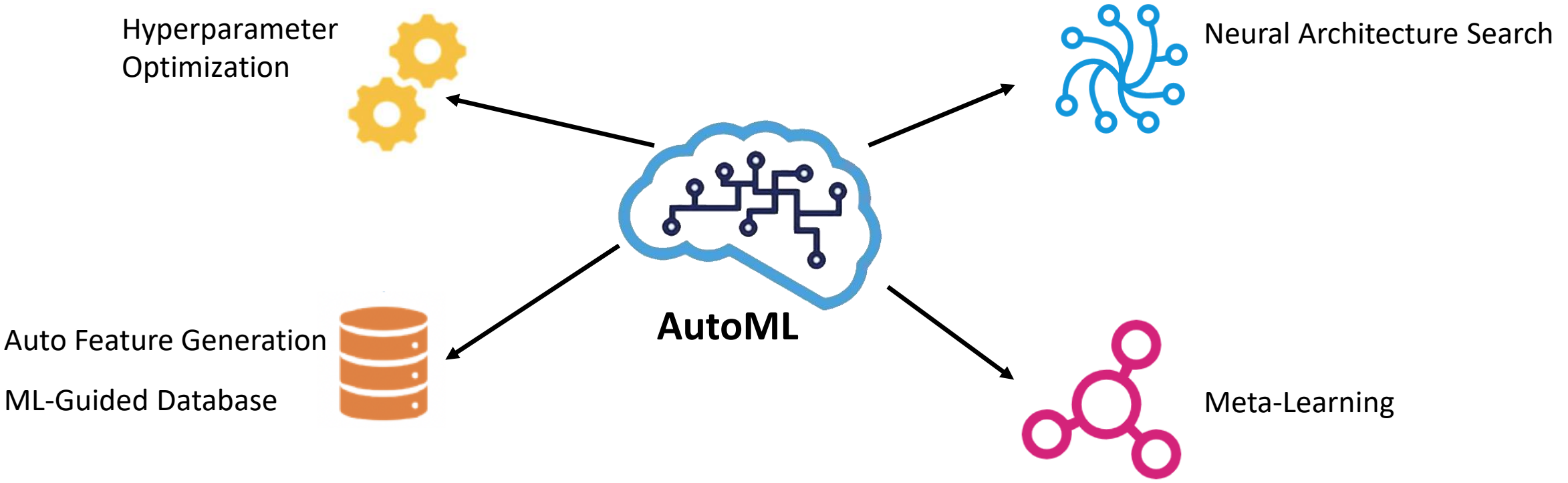
Machine Learning Pipeline



AutoML

- Auto Feature Generation
- Neural Architecture Search
- Hyperparameters Optimization
- Meta Learning

Automated Machine Learning



AutoML: How to automate the process of applying machine learning components to various real-world tasks?

Automated Machine Learning

Inductive bias (prior α): how we represent data, which kinds of models to be considered, how to tune hyper-parameter, how to transfer knowledge across tasks, etc…

Traditional ML:

$$\theta^* = argmax_\theta \, P(D|\theta) \cdot P(\theta|\alpha)$$

where α is given by experts

AutoML : bilevel optimization

$$\theta^* = argmax_\theta \, P(D|\theta) \cdot P(\theta|\alpha^*)$$

where $\alpha^* = argmax_\alpha \, f(D', \theta^*)$

$D' = D$ for HPO,

D' is validation data for NAS, AutoFeature,

D' is data of other tasks for Meta-learning.

Tutorial Schedule

Yaliang Li, Background and Overview of AutoML
Hyperparameter Optimization

Zhen Wang, Neural Architecture Search
Meta-Learning

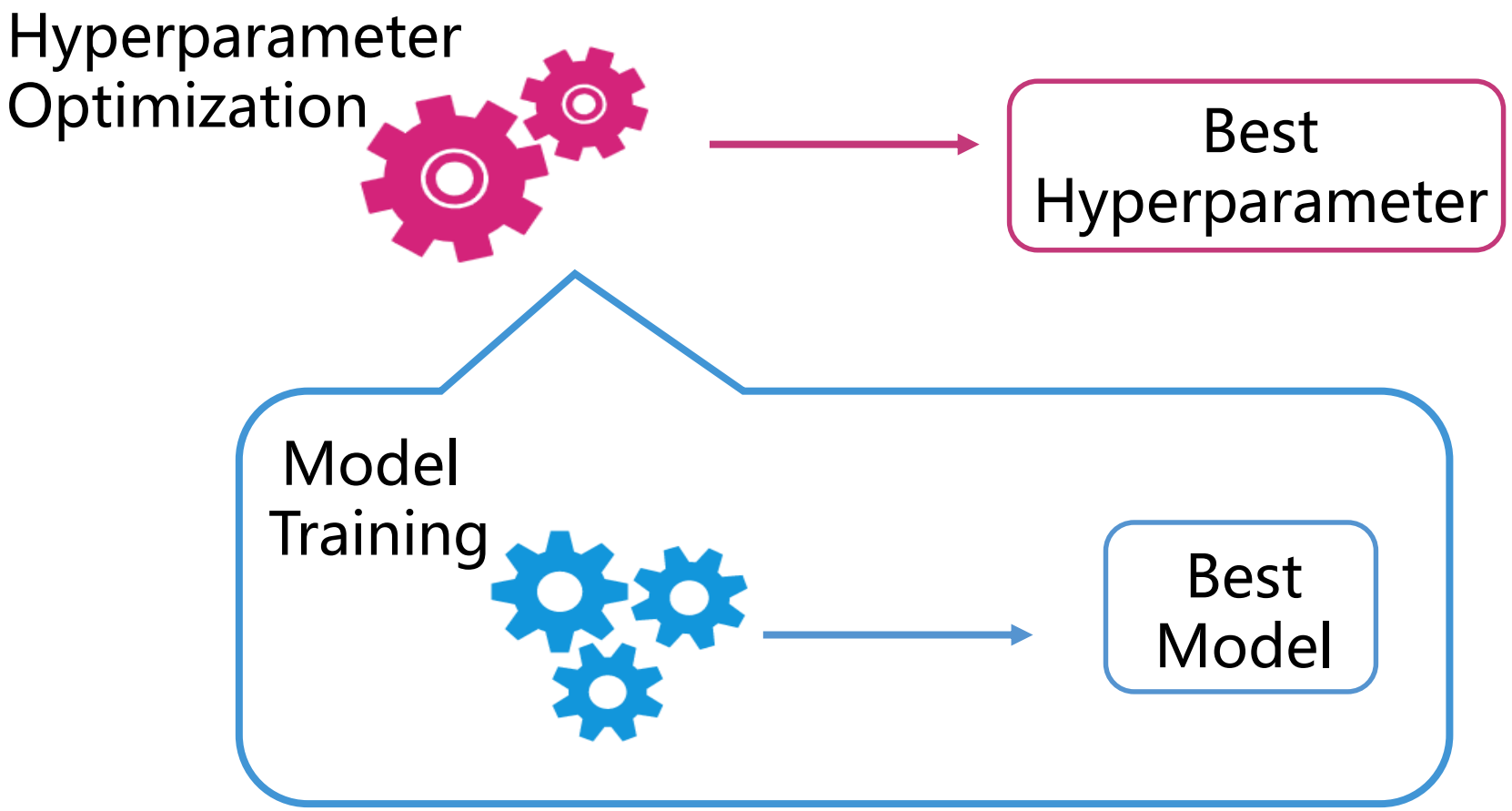
Yuexiang Xie, Automatic Feature Generation

Ce Zhang, VolcanoML: End-to-End AutoML via
Scalable Search Space Decomposition

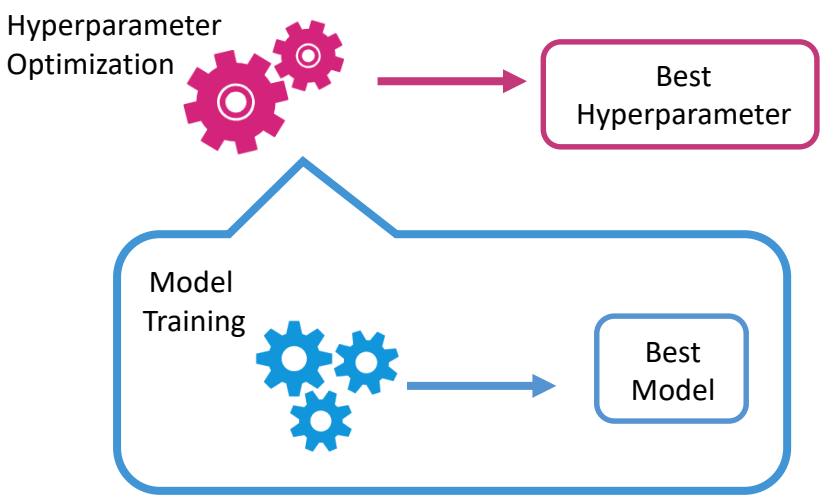
Bolin Ding, Machine Learning Guided Database

Hyperparameter Optimization (HPO)

Hyperparameter Optimization

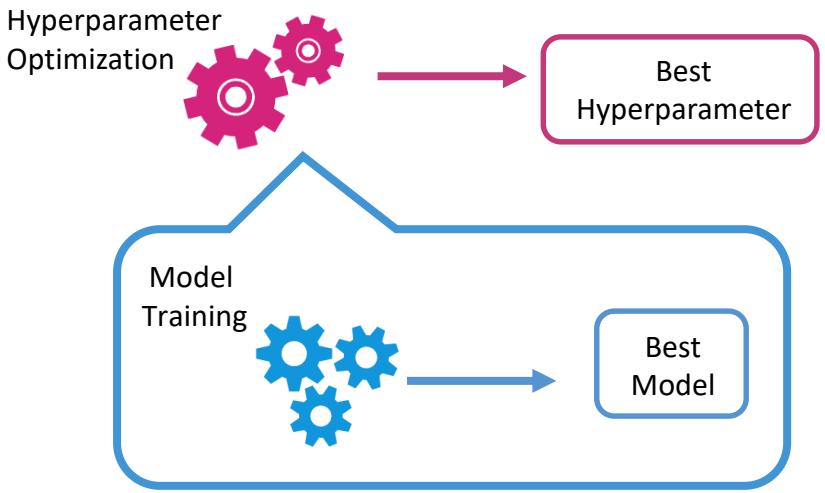


Hyperparameter Configuration v.s. Schedule



- Hyperparameter **configuration** search methods find a **fixed** hyperparameter setting to maximize the model performance.
- Hyperparameter **schedule** search methods seek a **dynamic** hyperparameter schedule in the model training process.

Hyperparameter Optimization



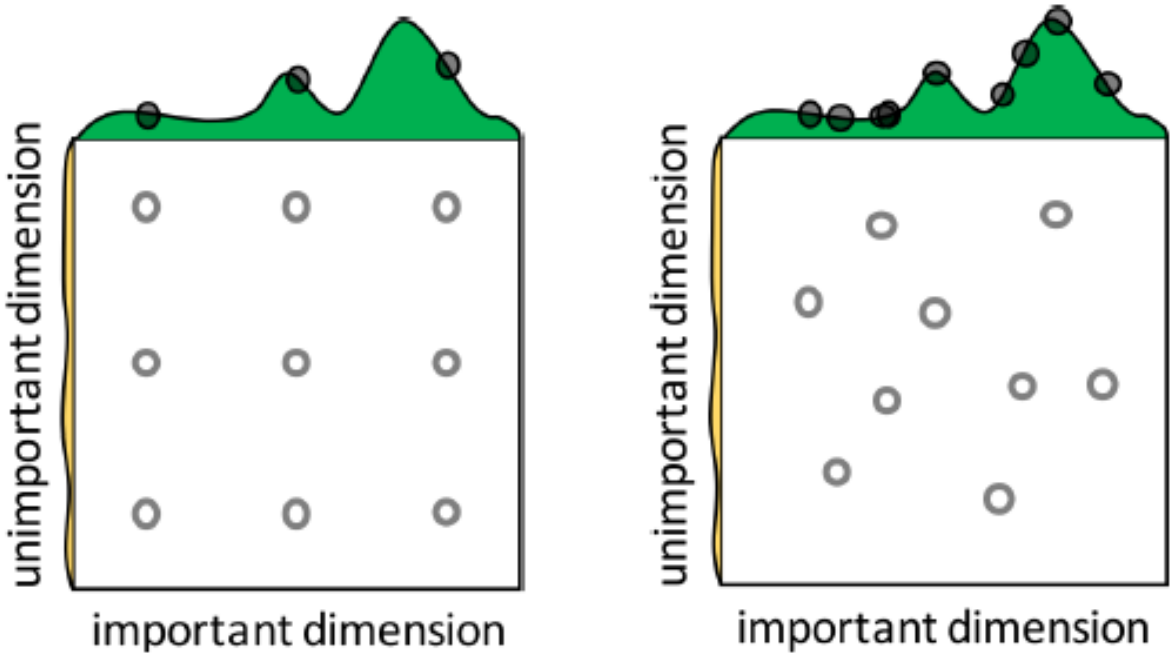
❑ Hyperparameter Configuration

- Random search, Grid Search
- Successive-halving, Hyperband
- Bayesian optimization

❑ Hyperparameter Schedule

- Population-based training
- Hypergradient

Search Methods



(a) Grid search. (b) Random search.

Image source: Bergstra & Bengio. *JMLR,* 2012.

Successive-Halving

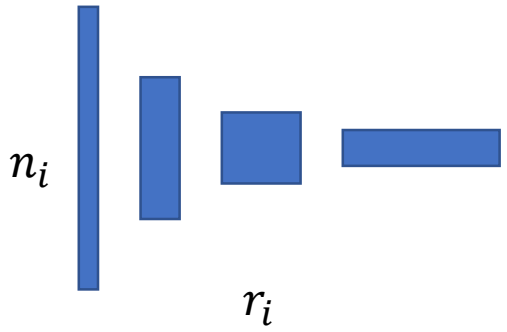
- Uniformly allocate *a budget* to a set of hyperparameter configurations
- Evaluate the performance of all configurations
- Throw out the worst half

Repeat until one configuration remains

Non-stochastic best arm identification and hyperparameter optimization. 2016.

Hyperband

- Successive-Halving needs to determine the number of configurations (i.e., n)
- Outer loop
  - Grid search for different n
- Inner loop
  - Successive-Halving for given n configs
  - s.t. at least one config is trained for R



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

Hyperband: A novel bandit-based approach to hyperparameter optimization. *JMLR,* 2018.

Bayesian Optimization

Given some tried {hyperparameter, performance} pairs, which hyperparameter should be the next one to try?

Bayesian Optimization

Given some tried {hyperparameter, performance} pairs, which hyperparameter should be the next one to try?

Independence assumption

Follow a certain distribution

Bayesian Optimization

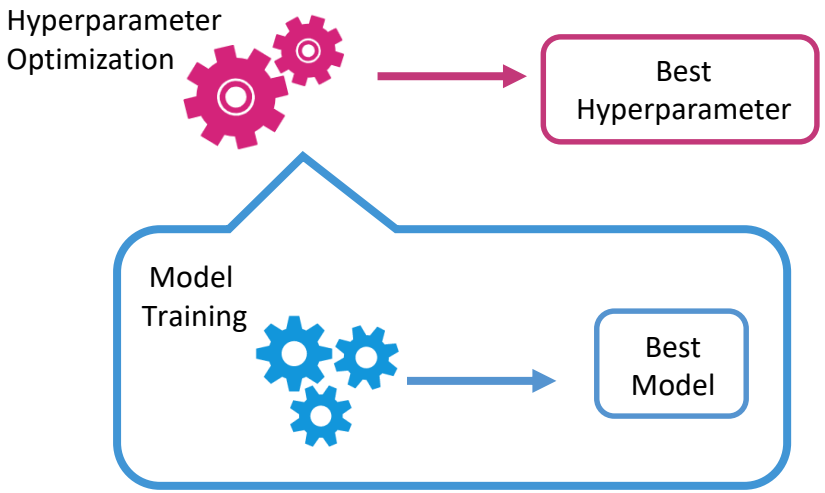
Bayesian Optimization

Fit a probabilistic function f(x) to model {x=hyperparameter, f(x)=performance}

- Function f(x) isn't required to be convex, differentiable
- Rich theoretical results: convergence, sync v.s. async, various model choices

- Exploration-exploitation trade-off
- Costly

Hyperparameter Optimization



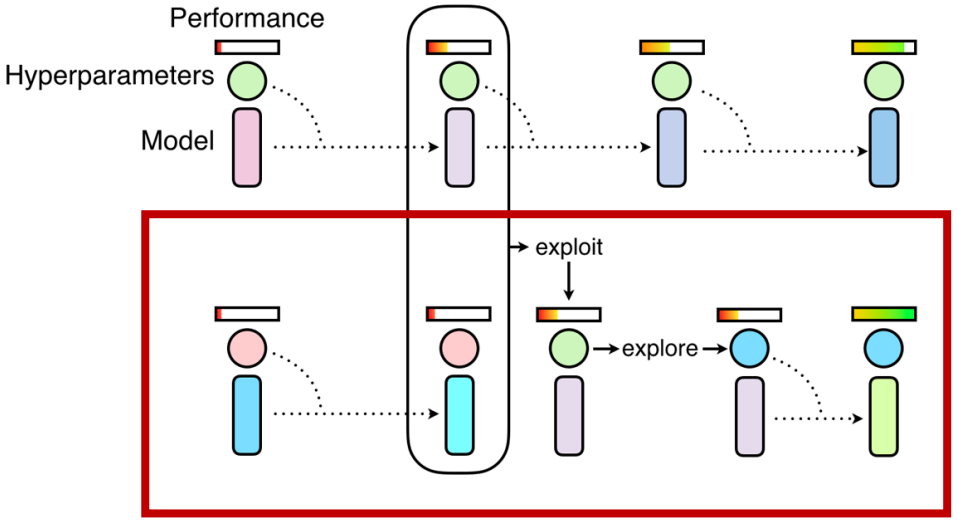
❑ Hyperparameter Configuration

- Random search, Grid Search
- Successive-halving, Hyperband
- Bayesian optimization

❑ Hyperparameter Schedule

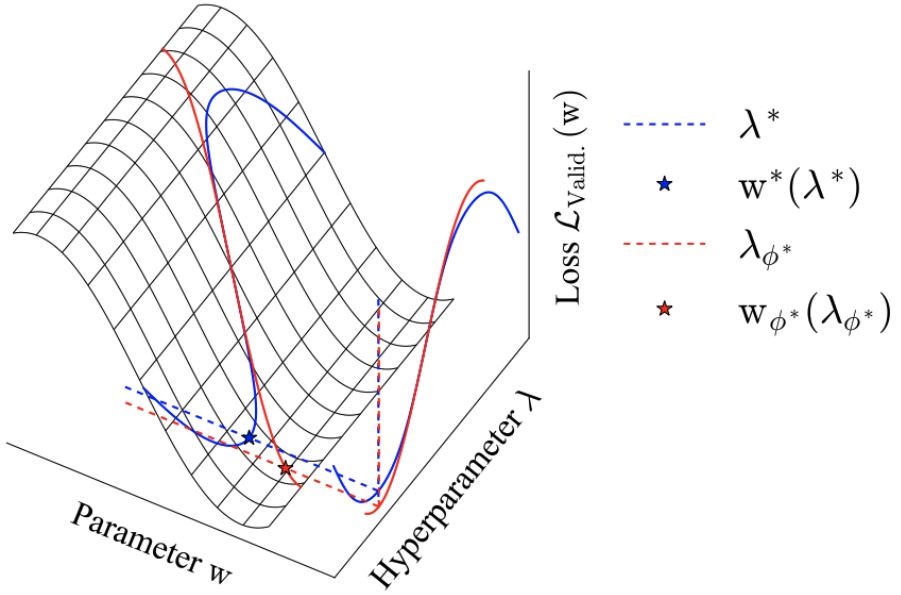
- Population-based training
- Hypergradient

Hyperparameter Schedule



Population-based training

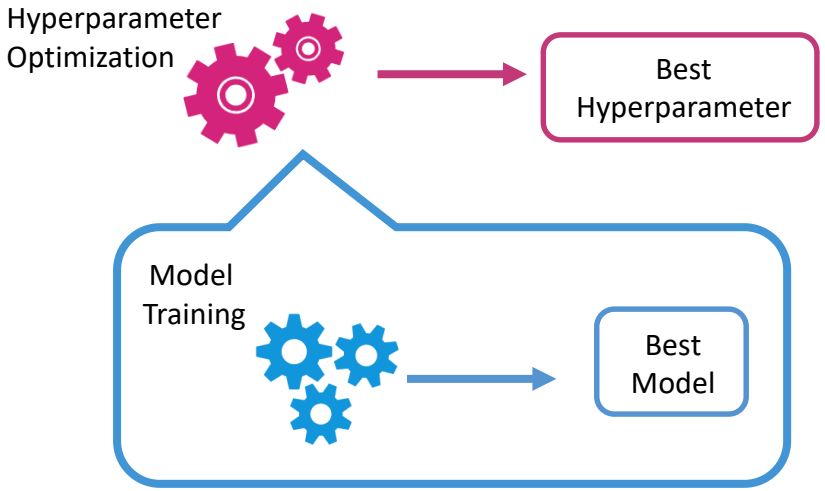
A generalized framework for population based training. *KDD*, 2019.

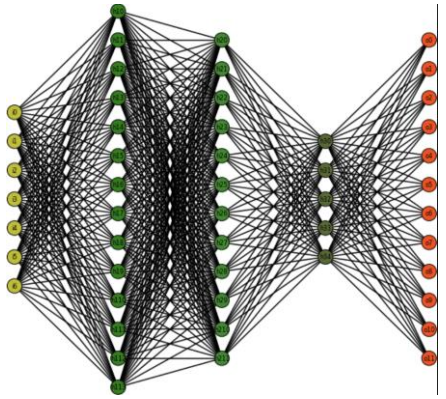


Hypergradient

Self-tuning networks: Bilevel optimization of hyperparameters using structured best-response functions. *ICLR*, 2019.

Practical Challenge (1)

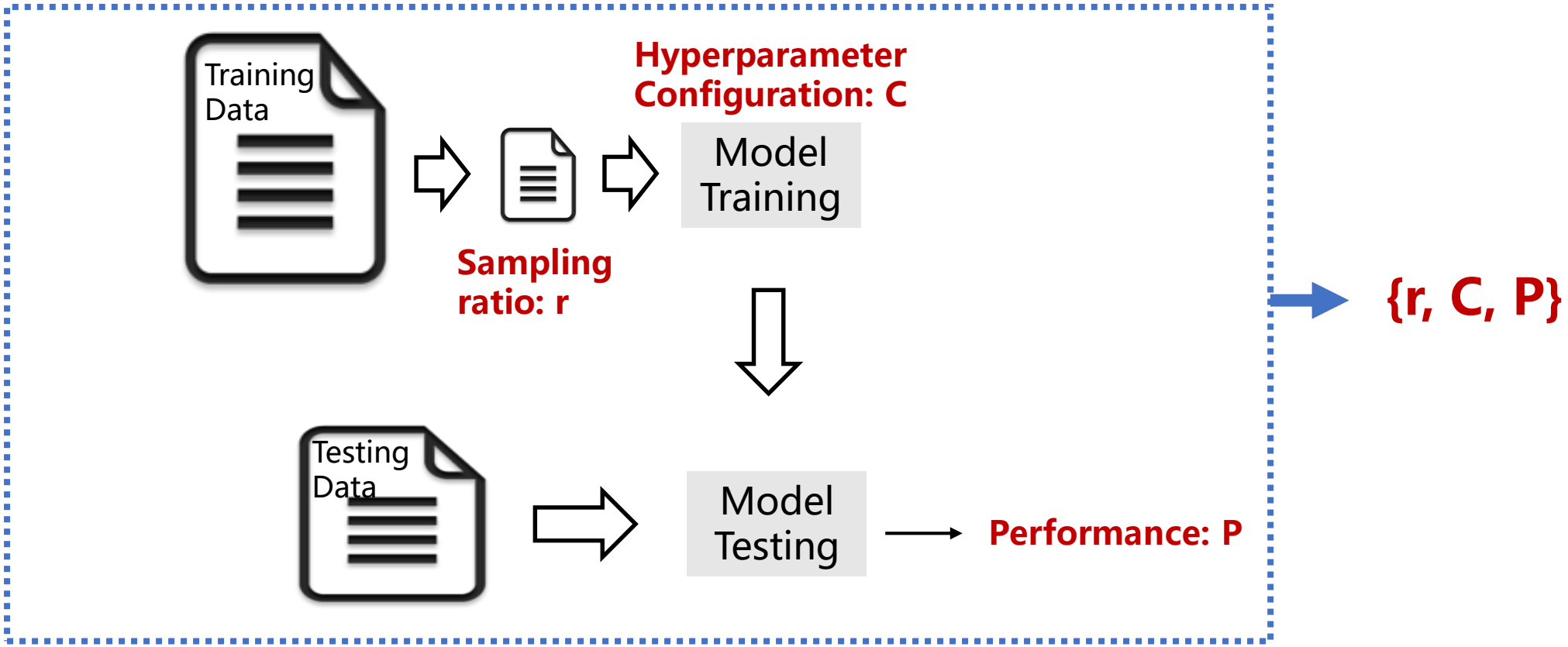




Model Size

Data Size

ABC: Sampling



Efficient Identification of Approximate Best Configuration of Training in Large Datasets. *AAAI*, 2019.

A New Method: εGE

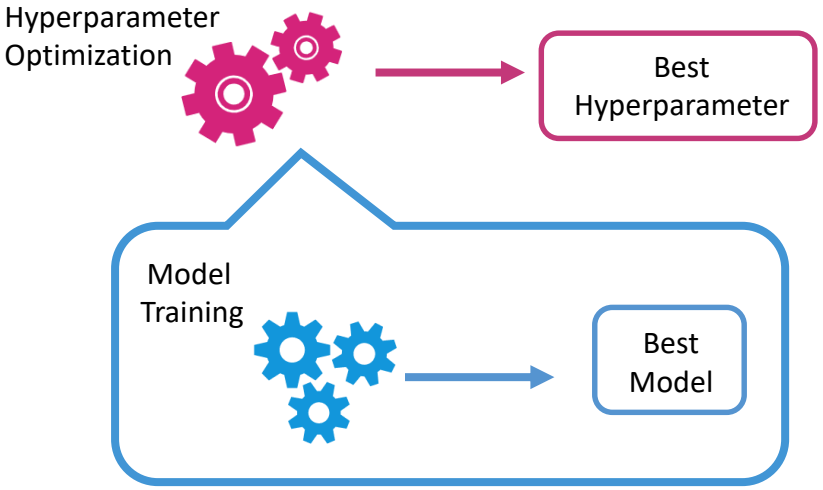


Illustration of Hyperparameter optimization

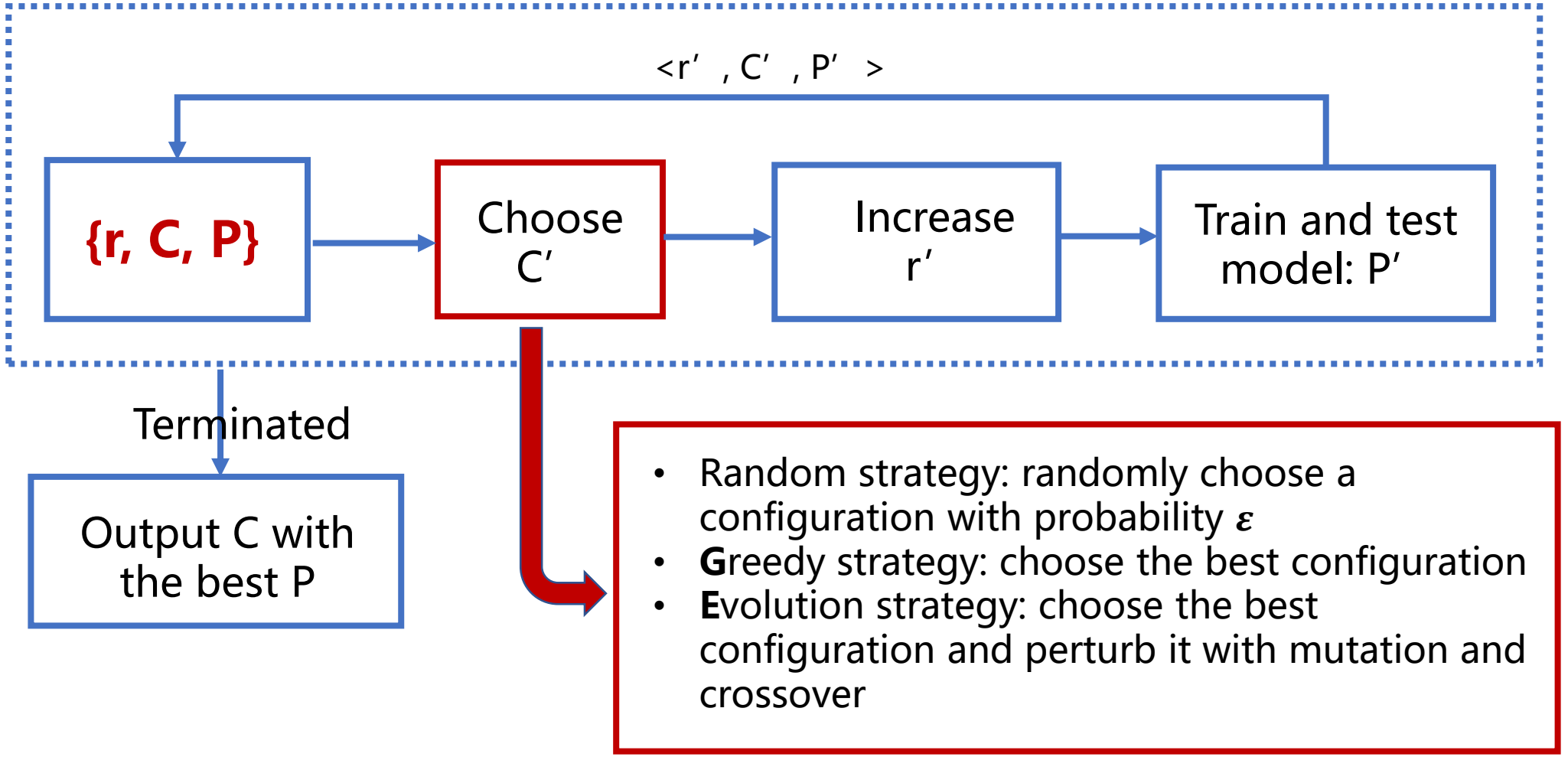
Each category of hyperparameter optimization methods has its advantages and disadvantages. Can we **adaptively** combine them and utilize their advantages for different tasks?

❑ Existing methods

- Search-strategy based: Successive-halving, Hyperband, etc.
- Evolutionary algorithm: Population Based Training, etc.
- Bayesian optimization

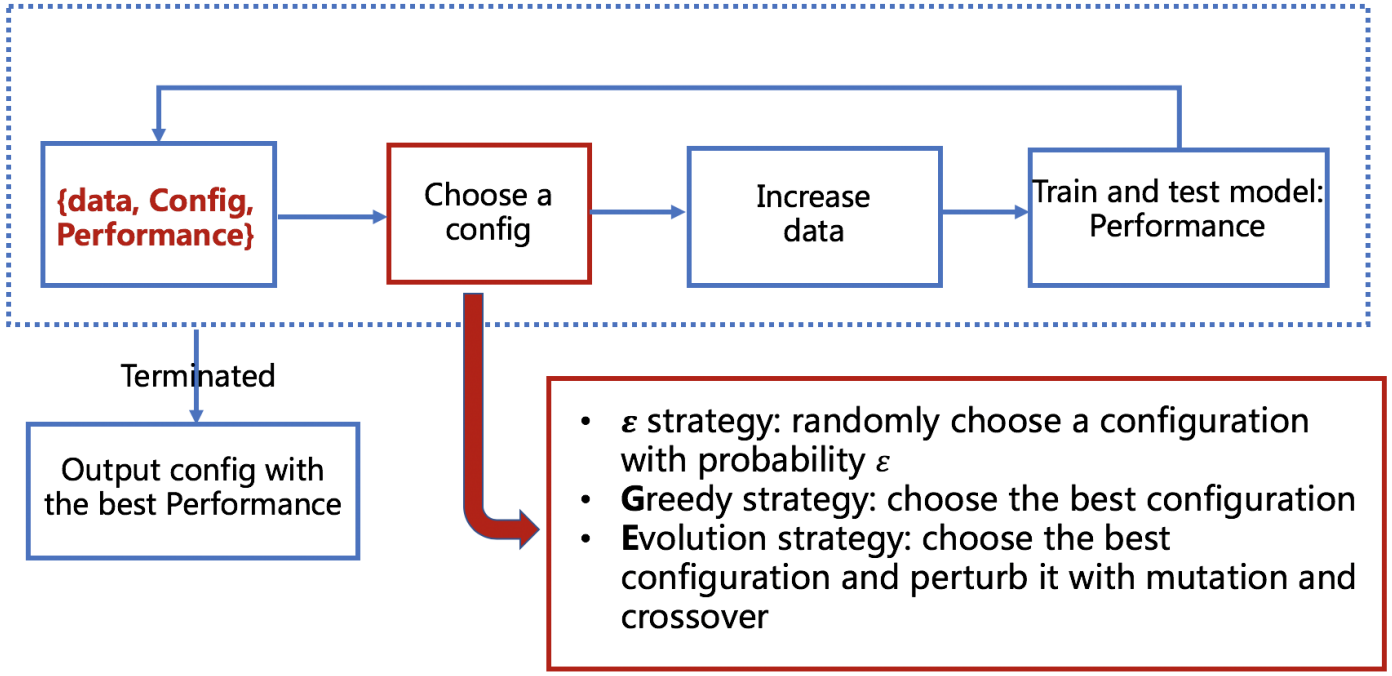
Hyperparameter Recommendation for Machine Learning Method. *Patent*, 2019.

A New Method: εGE



HPO: Sampling method-εGE

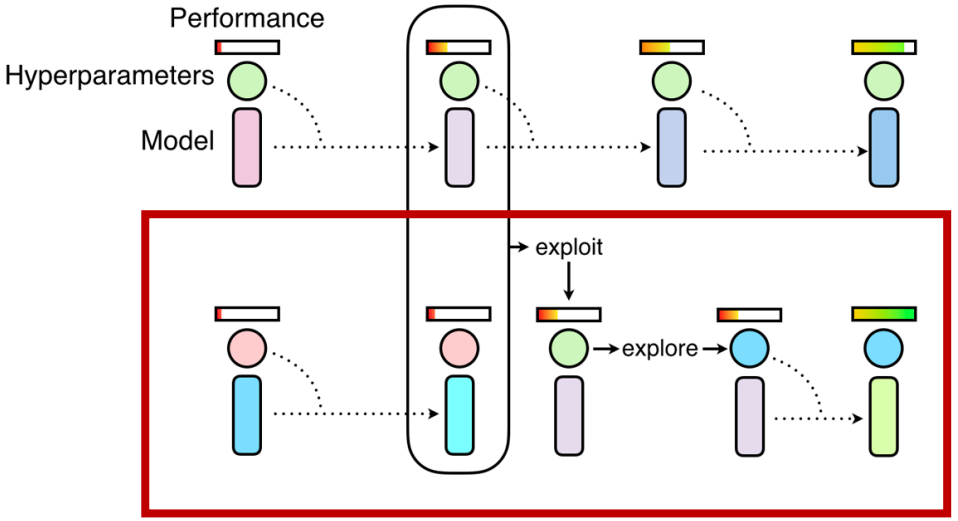
The task-adaptively combination of different hyperparameter optimization methods leads to faster solutions!



- A soft version of Hyperband
- Evolutionary operation
- A simplified version of Bayesian optimization (i.e., local smoothness assumption)

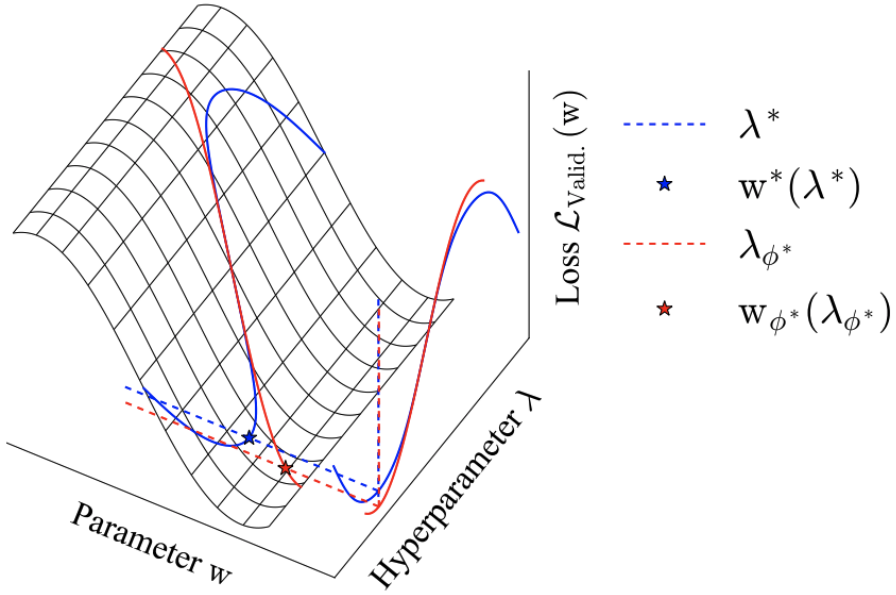
Hyperparameter Recommendation for Machine Learning Method. *Patent*, 2019.

Practical Challenge (2)



Mutation-driven **global** search

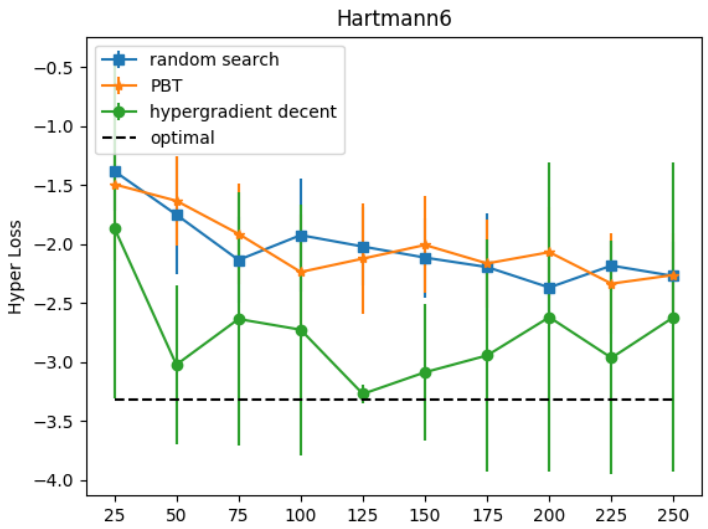
PBT, KDD2019



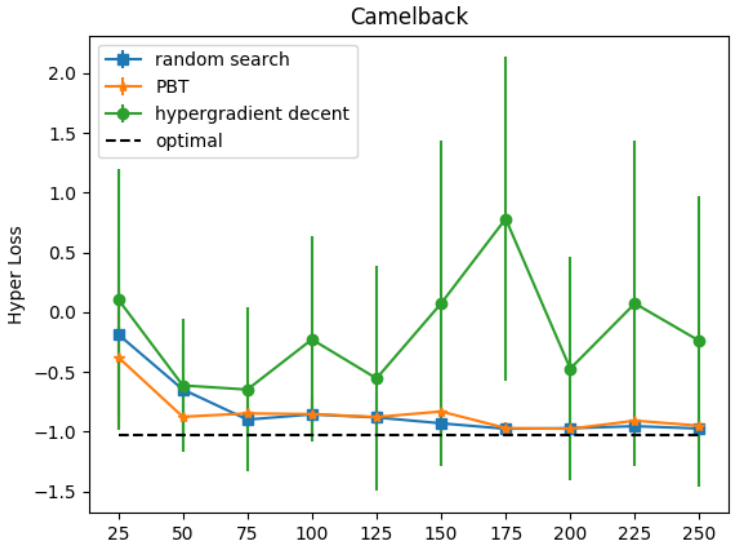
Hypergradient-guided **local** search

STN, ICLR2019

Hyperparameter Schedule



A smooth optimization problem



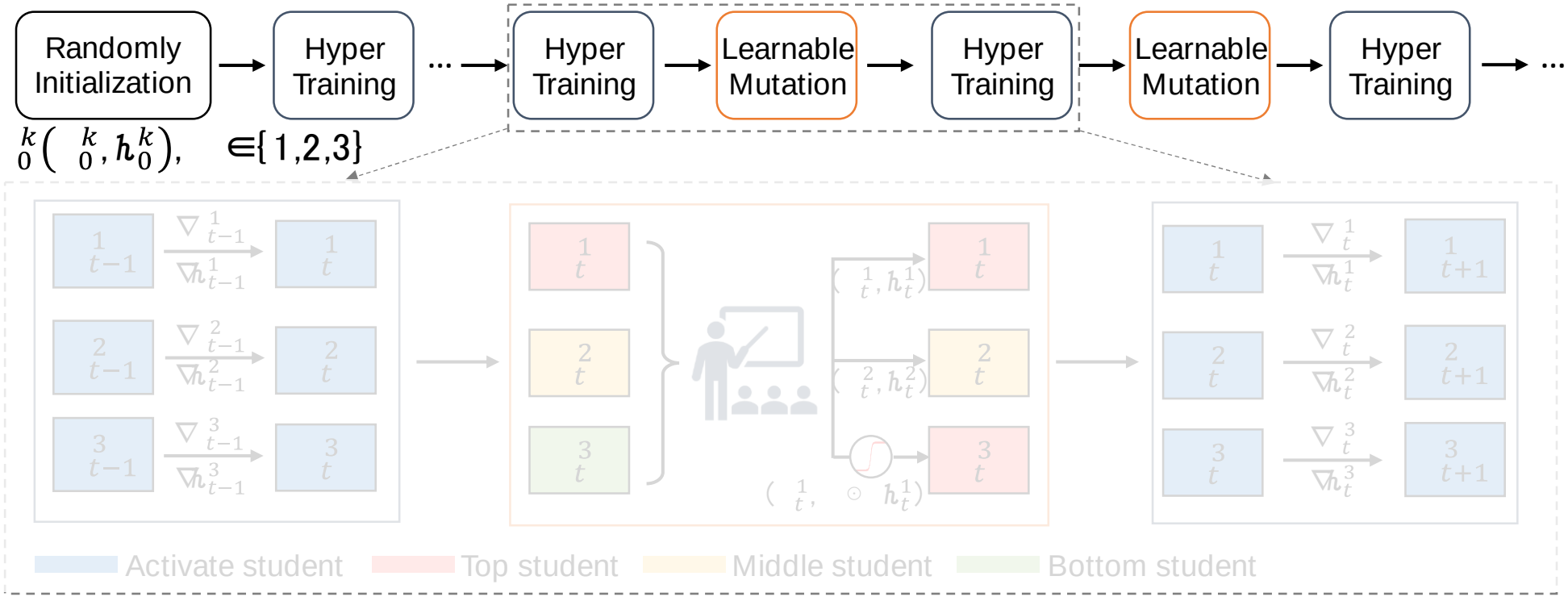
A non-smooth optimization problem

Trade-off between Evolutionary algorithm (PBT) and Hyper-gradient based method:

- Hyper-gradient based method performs better than PBT on the smooth optimization problems.
- Hyper-gradient based method performs worse than PBT on the cases of many local minima (non-smooth).

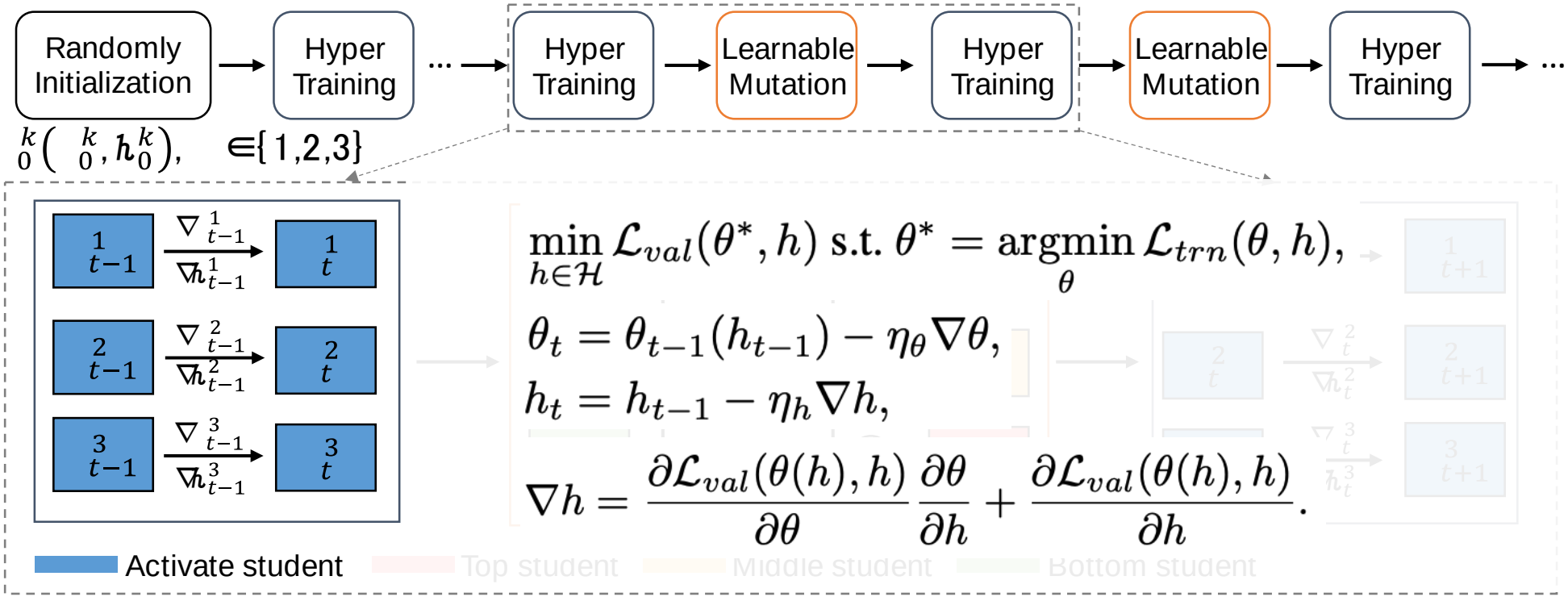
How to learn a good trade-off between the global search and local search?

HyperMutaion (HPM)



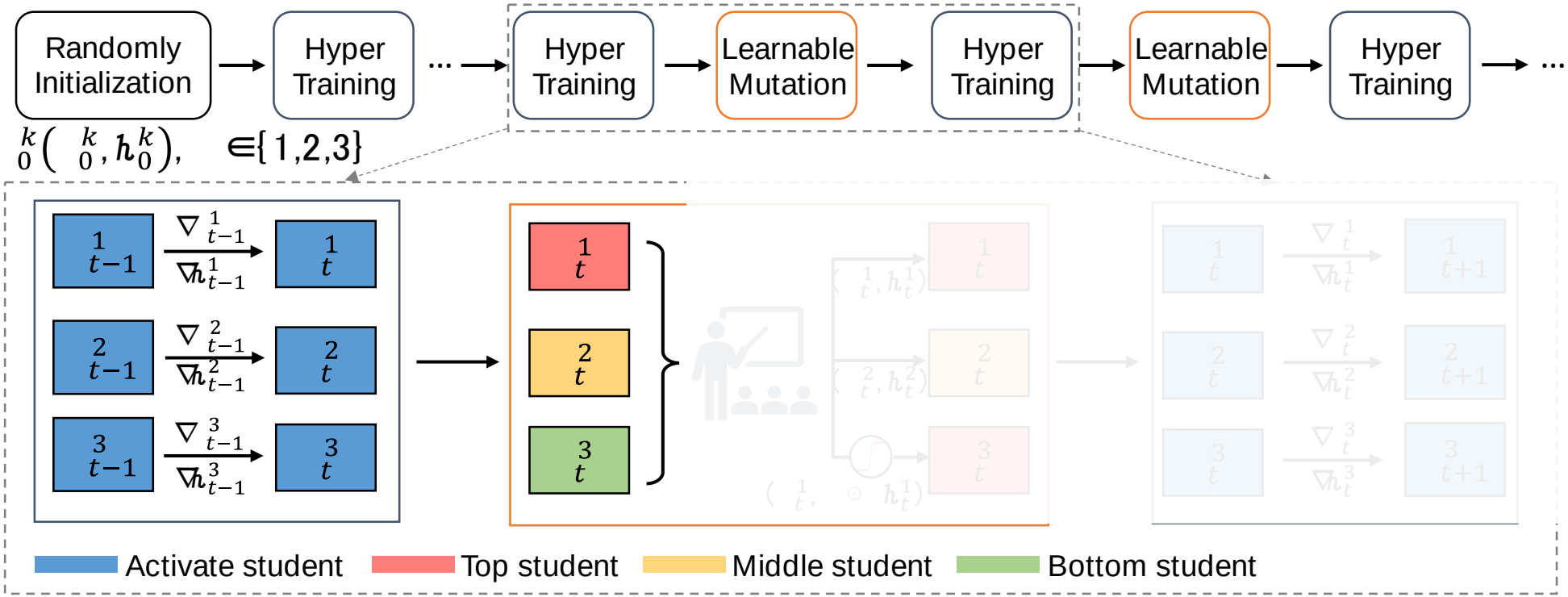
Learning to Mutate with Hypergradient Guided Population. *NeurIPS*, 2020.

Hypertraining: a joint optimization over θ and h



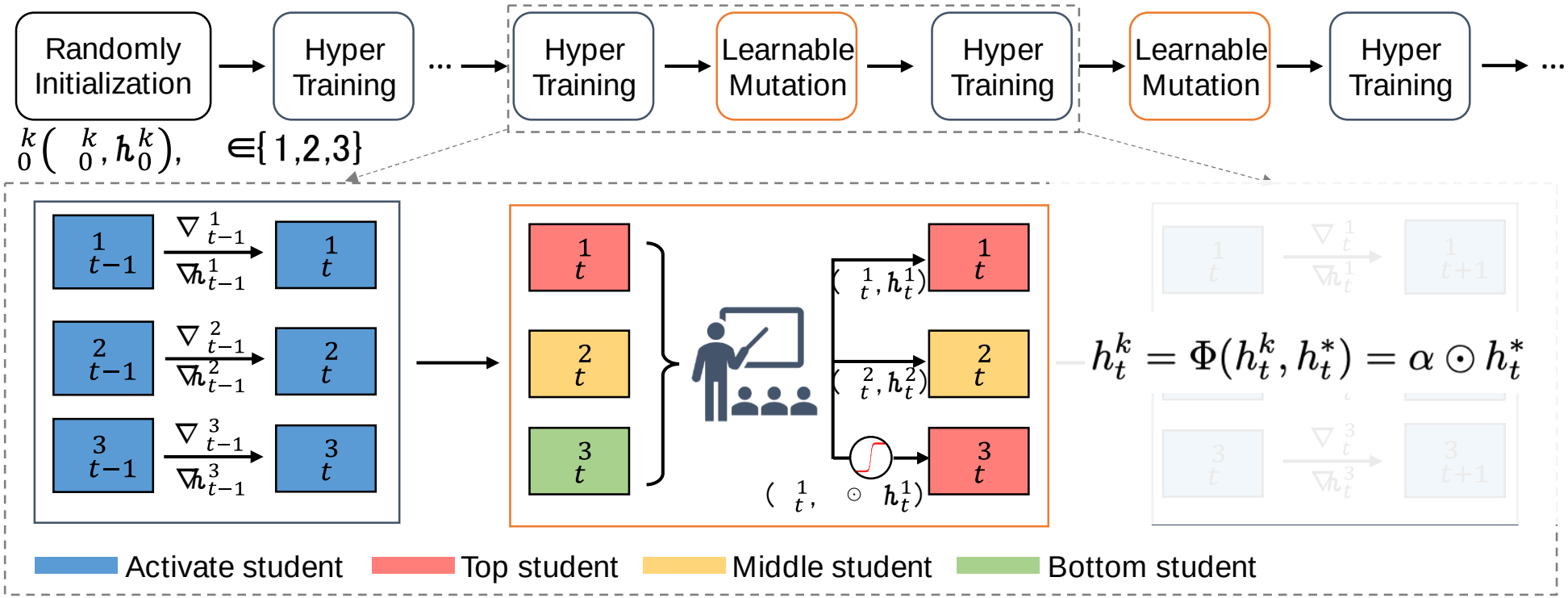
Learning to Mutate with Hypergradient Guided Population. *NeurIPS*, 2020.

Exploit by a truncation selection



Learning to Mutate with Hypergradient Guided Population. *NeurIPS*, 2020.

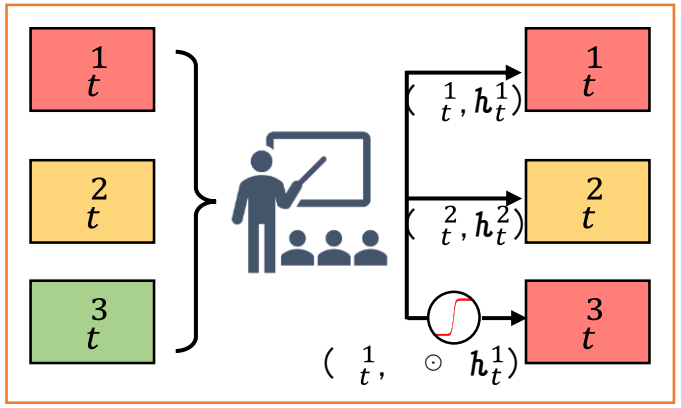
Explore by the learnable mutation



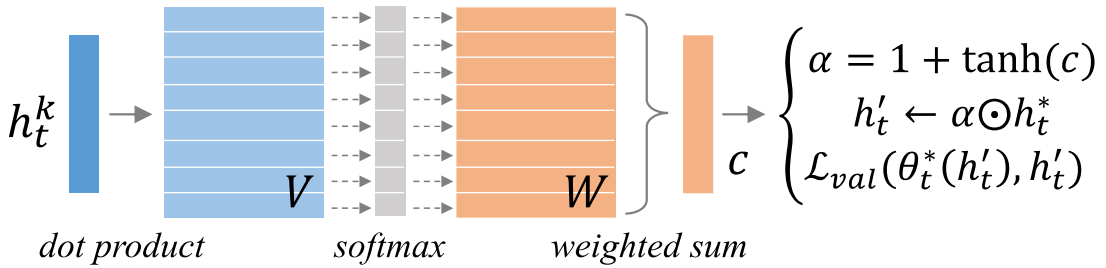
Learning to Mutate with Hypergradient Guided Population. *NeurIPS*, 2020.

Learning mutations with a teacher network

- Student-teaching schema

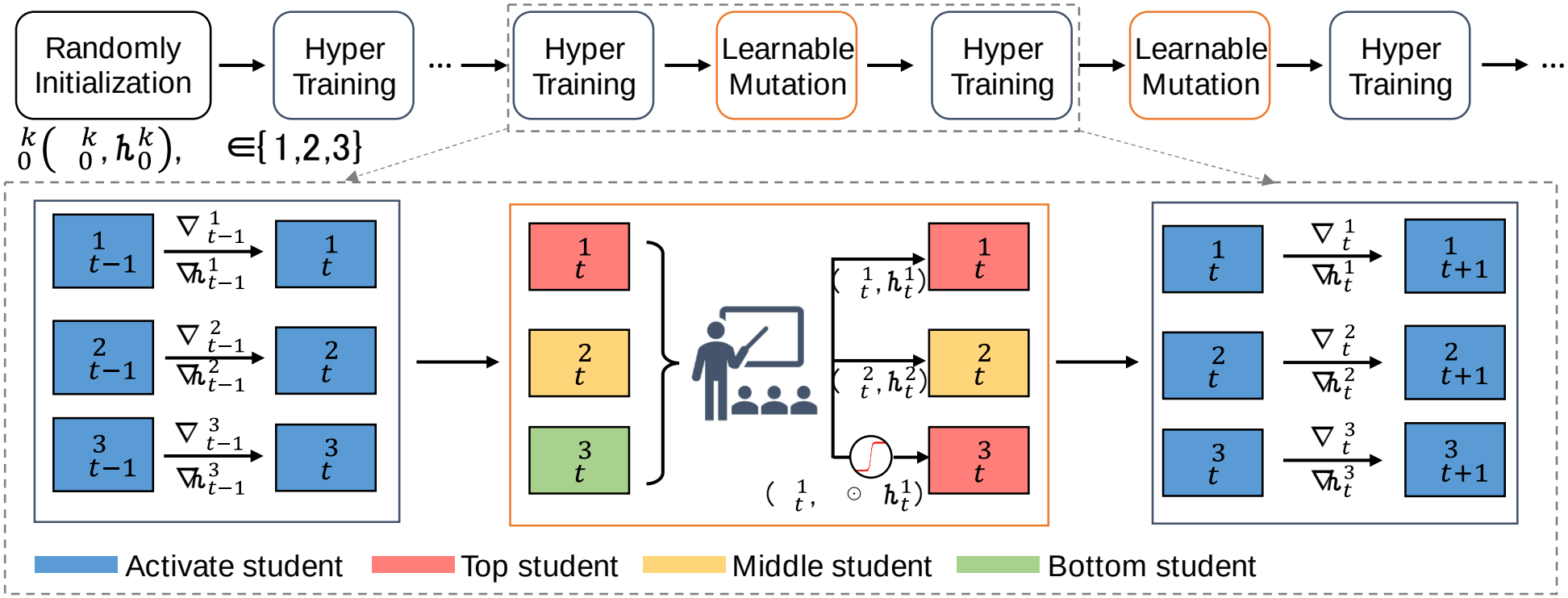


- Teacher model with attention networks

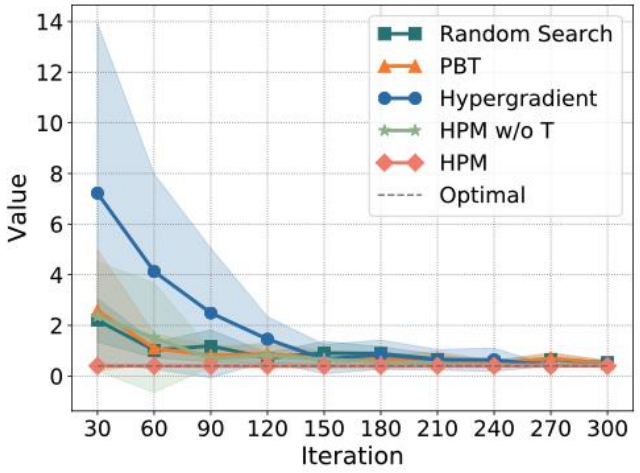


$$\alpha = g_\phi(h_t^k) = 1 + \tanh(c), c = W\,\mathrm{softmax}(V^{\mathrm{T}} h_t^k),$$

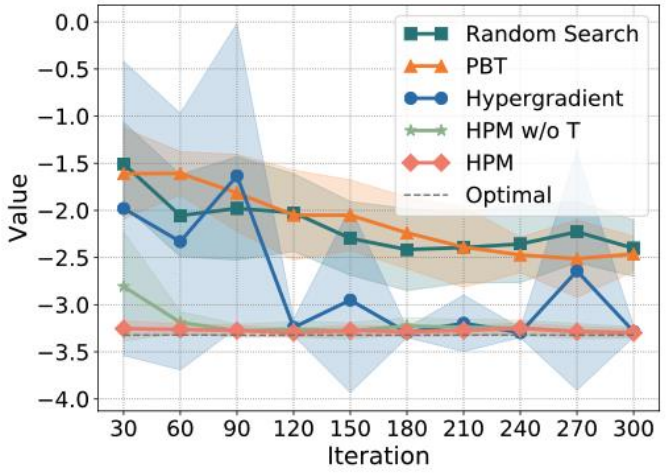
Continue hypertraining after exploit & explore



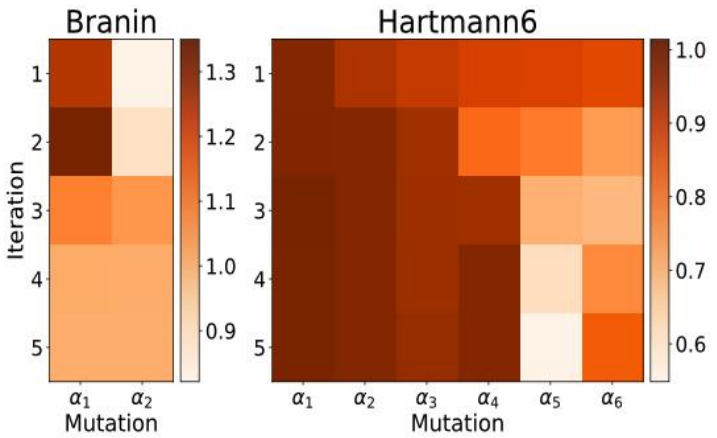
Experiments on test functions



(a) Branin



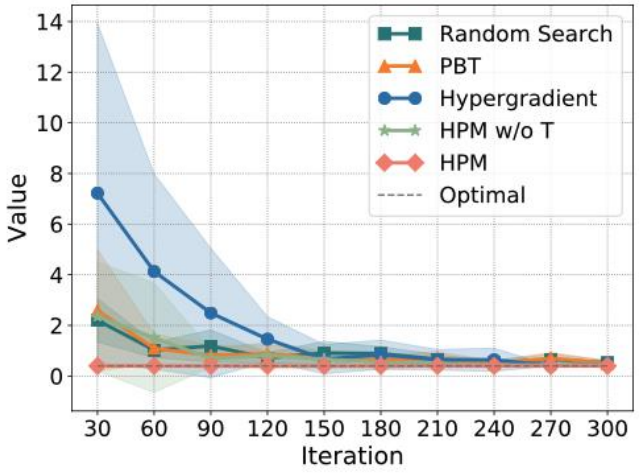
(b) Hartmann6D



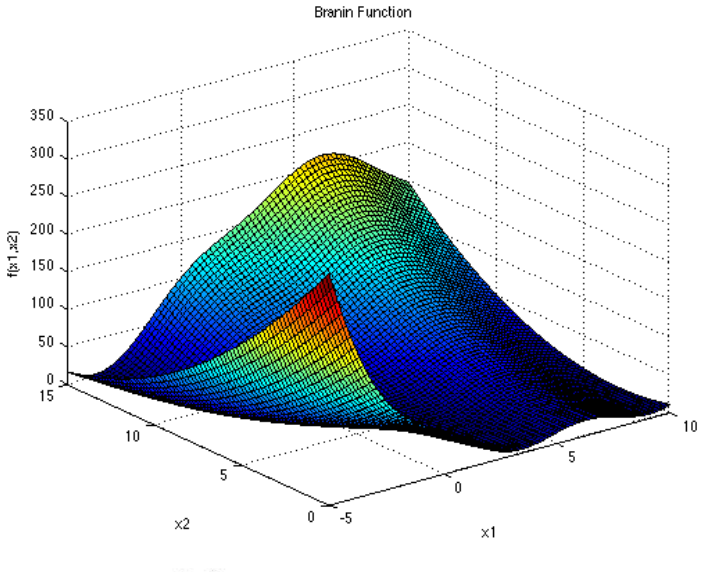
(c) Learned Mutations

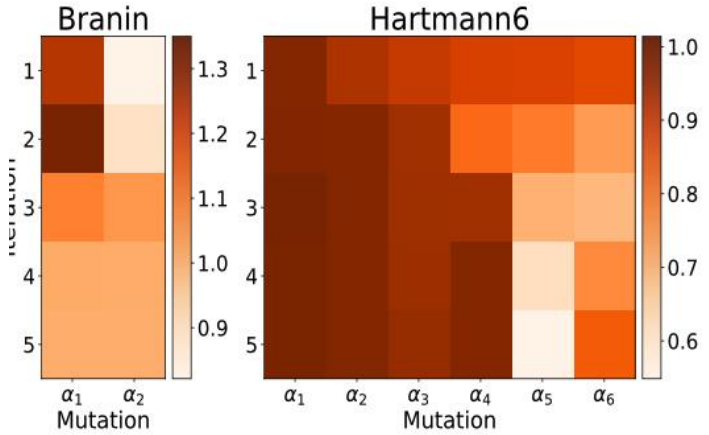
Figure: (a)-(b) The mean performance computed by different methods along with the standard deviation over 10 trials, in terms of different given budget of iterations. (c) The average mutation values learned by HPM over 10 trials. In each trial, HPM runs 30 iterations in total with a population size of 5, resulting in 6 training steps and 5 mutations.

Experiments on test functions



(a) Branin





(c) Learned Mutations

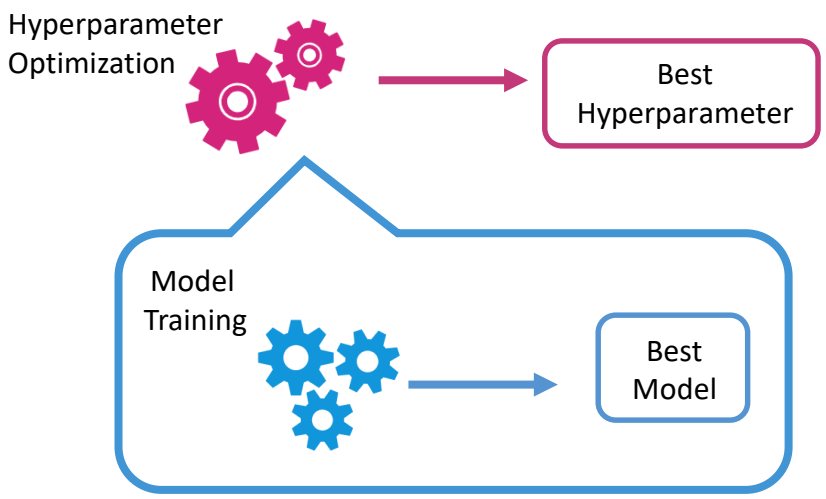
Figure: (a)-(b) The mean performance computed by different methods along with the standard deviation over 10 trials, in terms of different given budget of iterations. (c) The average mutation values learned by HPM over 10 trials. In each trial, HPM runs 30 iterations in total with a population size of 5, resulting in 6 training steps and 5 mutations.

Experiments on benchmark datasets

Table 1: Performance comparison for the image classification task on the CIFAR10 dataset by validation/test loss and the language modeling task on the PTB corpus dataset by perplexity (PPL).

	Method	CIFAR10		PTB	
		Val Loss	Test Loss	Val PPL	Test PPL
Fixed	Grid Search	0.7940	0.8090	97.32	94.58
	Random Search	0.9210	0.7520	84.81	81.86
	Bayesian Optimization	0.6360	0.6510	72.13	69.29
	Hyperband [20]	0.7156	0.7491	71.25	68.39
Schedule	PBT [15]	0.6253	0.6437	72.07	69.33
	STN [23]	0.5892	0.5878	71.49	68.29
	HPM w/o T	0.5724	0.5802	73.18	70.48
	HPM	**0.5636**	**0.5649**	**70.49**	**67.88**

Takeaways

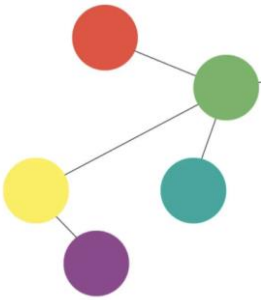


- ❑ Hyperparameter Configuration
  - Random search, Grid Search
  - Successive-halving, Hyperband
  - Bayesian optimization
- ❑ Hyperparameter Schedule
  - Population-based training
  - Hypergradient
  - HyperMutation (HPM)

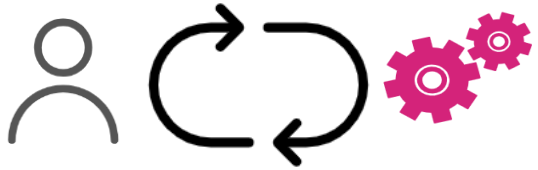
Future Directions

- Faster, Green
  - HPO via Meta-Learning

- HPO for a specific domain
  - a group of algorithm, e.g. Graph-related



- Interactive, Human-in-the-loop



Neural Architecture Search (NAS)

Neural Architecture Search

❑ What is neural architecture search (NAS)?

- To find the optimal topology and/or size configuration for the neural network.
  - E.g., select a filter from $\{\text{CNN}_{3\times3}, \text{CNN}_{5\times5}, \text{DilatedCNN}_{5\times5}\}$.
  - E.g., determine the depth and width of a neural network.

❑ Why NAS?

- Architecture matters a lot on the performance!
- The choices cannot be exhausted.
- Useful prior knowledge, e.g., the invariance possessed by the task, has been exploited.

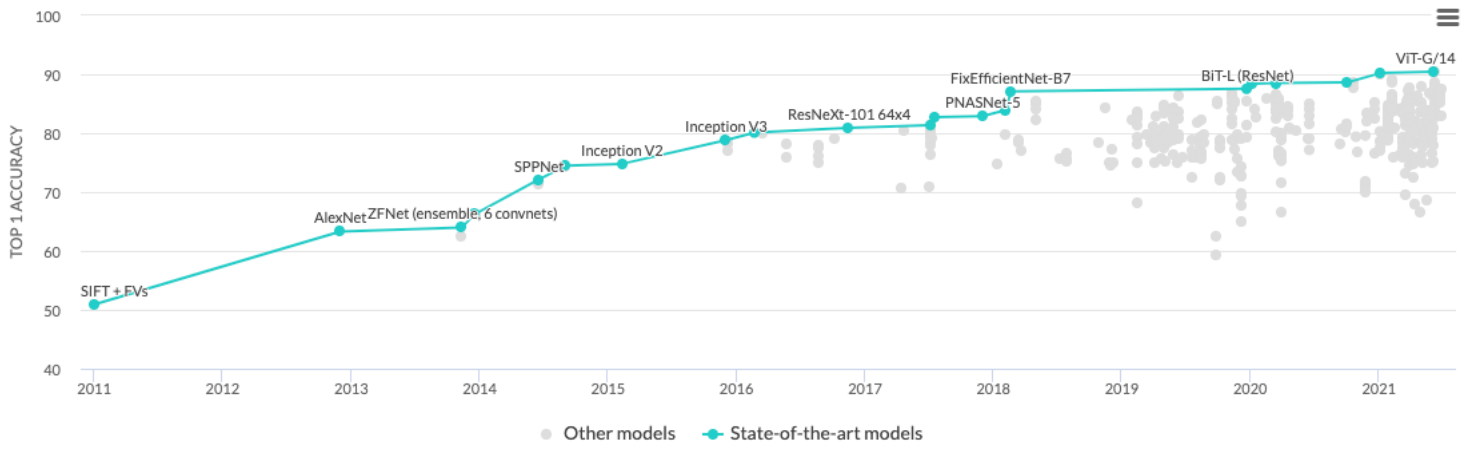


Figure: Image classification on ImageNet (source: https://paperswithcode.com/sota/image-classification-on-imagenet).

Elements of NAS

- ❑ Search space
  - All the possible configurations.
  - E.g., filter size, activation functions, depth, etc.
- ❑ Search strategy
  - How to utilize experience?
  - How to propose new configuration to try?
  - E.g., RL, ES, and differentiable search.
- ❑ Performance estimation strategy
  - How to evaluate a configuration?
  - E.g., standard training and surrogate objective.

And the Theme of NAS

Exploitation v.s. Exploration

❑ Search space

- All the possible configurations.
- E.g., filter size, activation functions, depth, etc.

Incorporating prior knowledge reduces search space but makes it constrained to some extent, e.g., Inception-v2/3 → stacked cells [Zoph et al. 2018].

❑ Search strategy

- How to utilize experience?
- How to propose new configuration to try?
- E.g., RL, ES, and differentiable search.

Instead of asymptotic regret, practitioners balance the exploitation and exploration to achieve best solution under *a given finite horizon*.

❑ Performance estimation strategy

- How to evaluate a configuration?
- E.g., standard training and surrogate objective.

Standard training&validation is expensive but accurate.

The proposed surrogate objectives are efficient but less correlated.

Pioneer Works of NAS

- ❑ Search space
  - Consider both CNN and RNN cells.
  - The configuration of each layer can be determined respectively.
- ❑ Search strategy
  - RL with the policy parameterized by a RNN.
- ❑ Performance estimation strategy
  - Standard train&validation

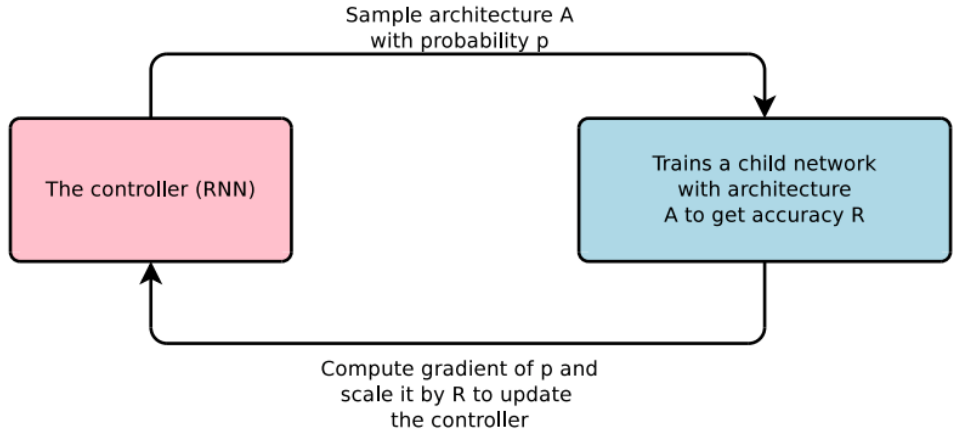


Figure: An overview o the trial-and-error process of NAS.

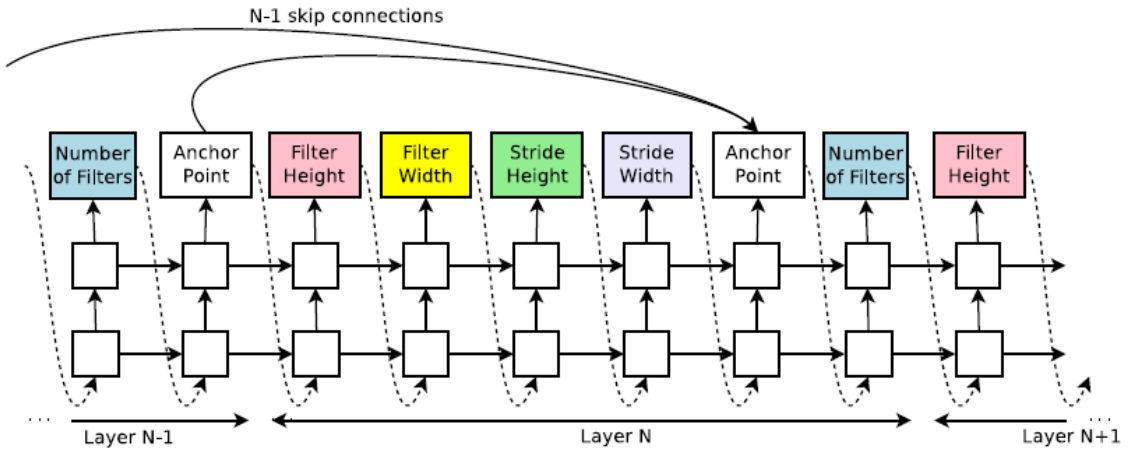


Figure: How the controller (i.e., a RNN) samples a CNN with skip connection.

Neural Architecture Search with Reinforcement Learning. *ICLR*, 2017, https://arxiv.org/pdf/1611.01578.pdf

Pioneer Works of NAS

- ❑ Search space
  - Consider both CNN and RNN cells.
  - The configuration of each layer can be determined respectively.
- ❑ Search strategy
  - RL with the policy parameterized by a RNN.
- ❑ Performance estimation strategy
  - Standard train&validation

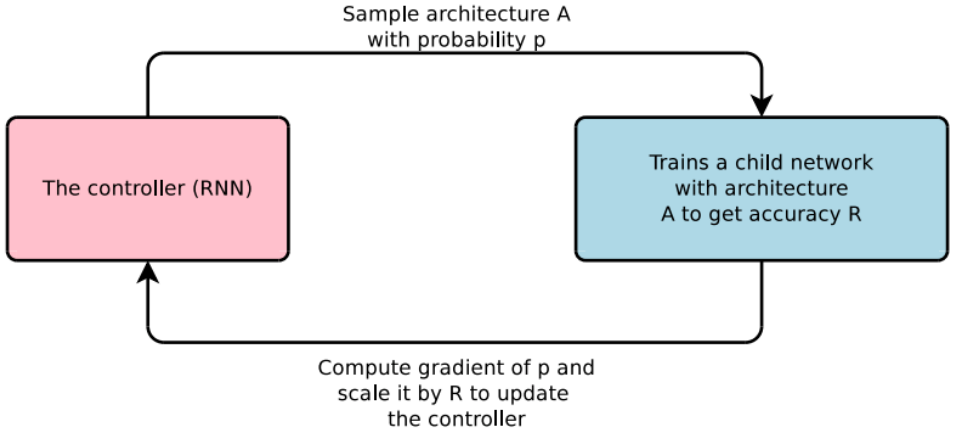


Figure: An overview o the trial-and-error process of NAS.

- ❑ Unfold the gain of NAS 😄 and also its pain ☹
  - Searched CNN and RNN cells achieve competitive performances against manually designed architectures on CIFAR-10 and PTB respectively.
  - Searched architecture can be transferred to other tasks.
  - Trained *12,800* models in total on *800* GPUs.

Neural Architecture Search with Reinforcement Learning. *ICLR*, 2017, https://arxiv.org/pdf/1611.01578.pdf

Weight Sharing for One-shot NAS

❑ Weight sharing

- Represent NAS's search space using a single DAG.
- An architecture can be realized by taking a subgraph.
- E.g., deducing a RNN cell as follow:

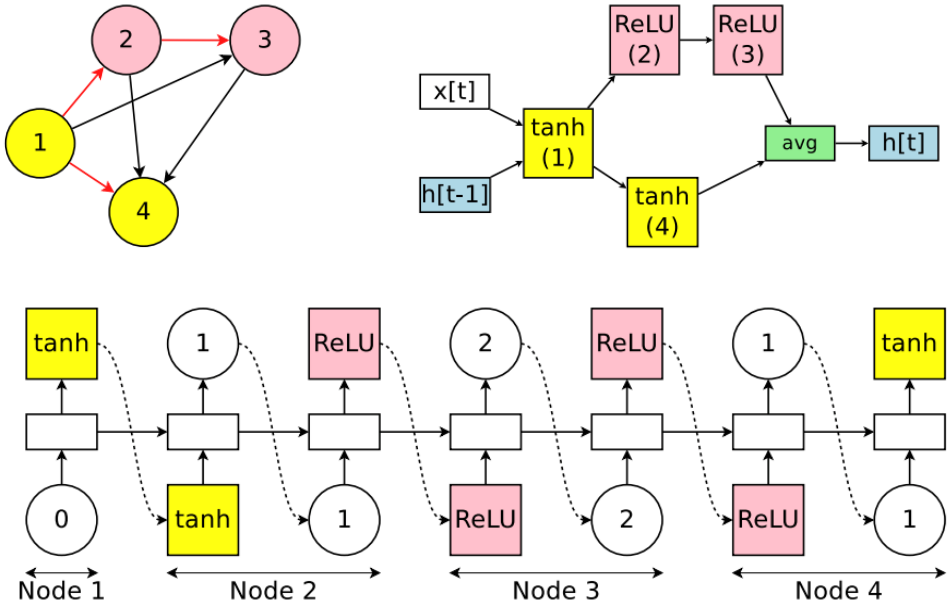


Figure: How a RNN cell (i.e., highlighted subgraph) inherits the shared parameters.

❑ One-shot NAS

- Each architecture (i.e., subgraph) is evaluated by inheriting the shared parameters.
- Shared parameters are trained with sampled architecture.
- Parameters and the controller are updated alternatively.

❑ Advantage and concern

- ENAS [Pham and Guan et al., 2018] uses 10h of one GTX1080Ti, which is *1000x faster* than [Zoph et al., 2017].
- *Does the performance of a stand-alone training correlate with that of one-shot NAS* [Bender et al., 2018, Zhang et al., 2020]*?*

Efficient Neural Architecture Search via Parameter Sharing. *ICML*, 2018, http://proceedings.mlr.press/v80/pham18a/pham18a.pdf

Differentiable NAS

❑ Continuous relaxation

- Each edge denotes a mixture of ops in $\mathrm{O} = \{\mathrm{CNN}_{3\times3}, \mathrm{DilatedCNN}_{3\times3}, \mathrm{Zero}, \mathrm{Identity}, \dots\}$.
- For each edge (i, j), they parameterize the weights of ops by architecture parameter $\alpha^{(i,j)}$.
- Suppose the tensor at node_i is x, then the tensor propagated to node_j will be:

$$\bar{o}^{(i,j)}(x) = \sum_{o\in\mathcal{O}} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o'\in\mathcal{O}} \exp(\alpha_{o'}^{(i,j)})} o(x)$$

❑ Differentiable learning

- Formulated as a bilevel optimization problem:

$$\begin{aligned} \min_{\alpha} \quad & \mathcal{L}_{val}(w^*(\alpha), \alpha) \\ \text{s.t.} \quad & w^*(\alpha) = \mathrm{argmin}_w \ \mathcal{L}_{train}(w, \alpha) \end{aligned}$$

- Regarded as a Stackelberg game
  - Architecture parameters as leader
  - Model parameters as follower

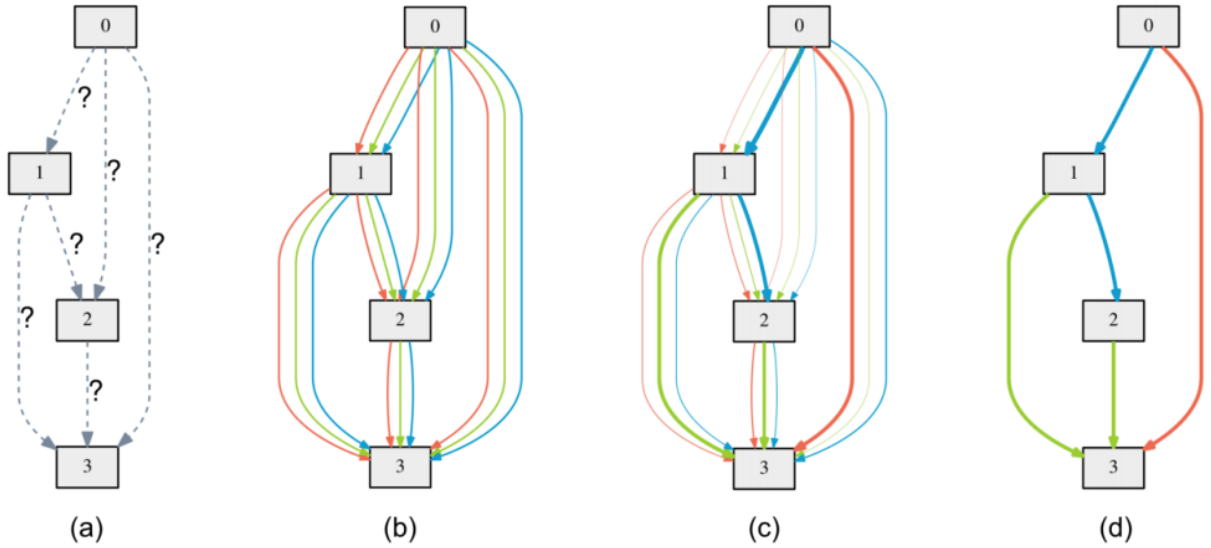


Figure: An overview of DARTS. (a) Operations are initially unknown. (b) Continuous relaxation. (c) architecture parameters are optimized jointly. (d) Inducing the final architecture.

DARTS: Differentiable Architecture Search. *ICLR*, 2019, https://arxiv.org/pdf/1806.09055.pdf

Differentiable NAS

❑ Differentiable learning (contd')

$$\min_{\alpha} \quad \mathcal{L}_{val}(w^*(\alpha), \alpha)$$
$$\text{s.t.} \quad w^*(\alpha) = \text{argmin}_w \ \mathcal{L}_{train}(w, \alpha)$$

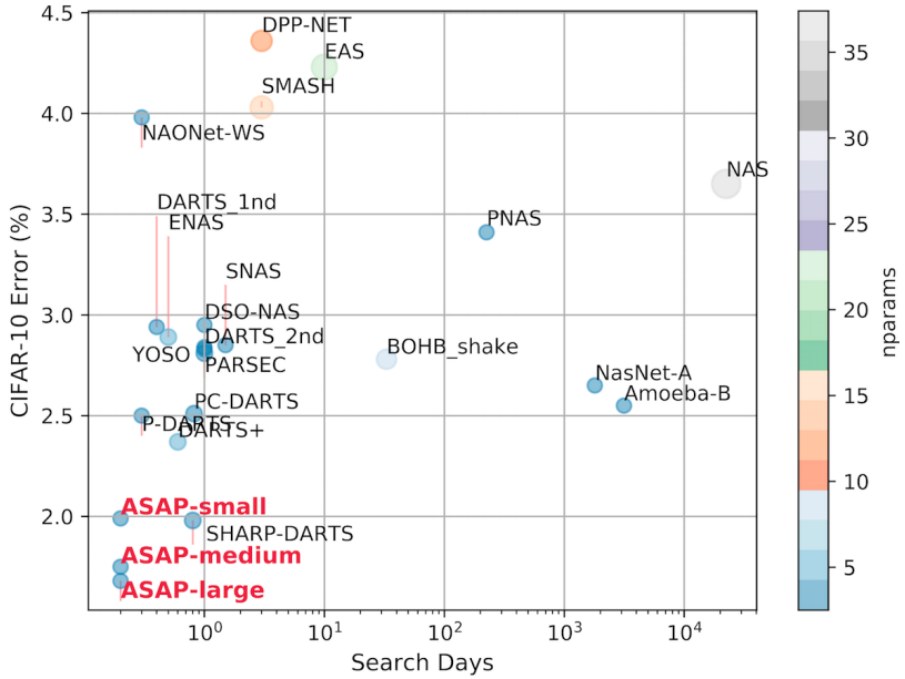
- No way to estimate the $\nabla_\alpha L_{val}(w^*(\alpha), \alpha)$ exactly.
- DARTS approximates the gradient by looking ahead one-step for ω like meta-learning.
- It is further simplified by treating the parameters equally [Li et al., 2021].

❑ Deriving discrete architecture

- Retain the top-k strongest predecessors for each node j where strength of (i, j) is defined as: $\underset{o \in O}{\text{argmax}} \frac{\exp\{\alpha_o^{(i,j)}\}}{\sum_{o' \in O} \exp\{\alpha_{o'}^{(i,j)}\}}$.
- Replace each edge by the most likely op: $o^{(i,j)} = \underset{o \in O}{\text{argmax}}\, \alpha_o^{(i,j)}$

❑ Improve DARTS by annealing and pruning

$$\Phi_o(\boldsymbol{\alpha}^{(i,j)}; T) = \frac{\exp\left\{\frac{\boldsymbol{\alpha}_o^{(i,j)}}{T}\right\}}{\sum_{o' \in \mathcal{O}} \exp\left\{\frac{\boldsymbol{\alpha}_{o'}^{(i,j)}}{T}\right\}}$$



ASAP: Architecture Search, Anneal and Prune. *AISTATS*, 2020, http://proceedings.mlr.press/v108/noy20a/noy20a.pdf

Dealing with Scalability Issue

- Horrible memory occupation of one-shot NAS
  - The supergraph cannot fit into GPU memory for large datasets.
  - Usually search architecture on CIFAR-10 and transfer to ImageNet.

Figure: ProxylessNAS directly optimizes neural architecture on target task and hardware.

- Binarized architecture
  - Transform real-valued path weights to binary gates.
  - Only one path is active in memory at runtime.

```
y_hard = tf.cast(tf.equal(y, tf.reduce_max(y, 0, keep_dims=True)),
                 y.dtype)
y_soft = tf.stop_gradient(y_hard - y) + y
```

Figure: Note the straight-through estimator (STE) trick.

ProxylessNAS: Direct Neural Architecture Search of Target Task And Hardware. *ICLR*, 2019, https://arxiv.org/pdf/1812.00332.pdf

Rethinking the Search Space of NAS

- ❑ Explore less constrained search spaces [Xie et al. 19]
  - Consider stochastic network generator, e.g., ER, BA, and WS.
  - All yield >73% mean accuracy on ImageNet with a low variance!
  - Presented graph damage ablation.

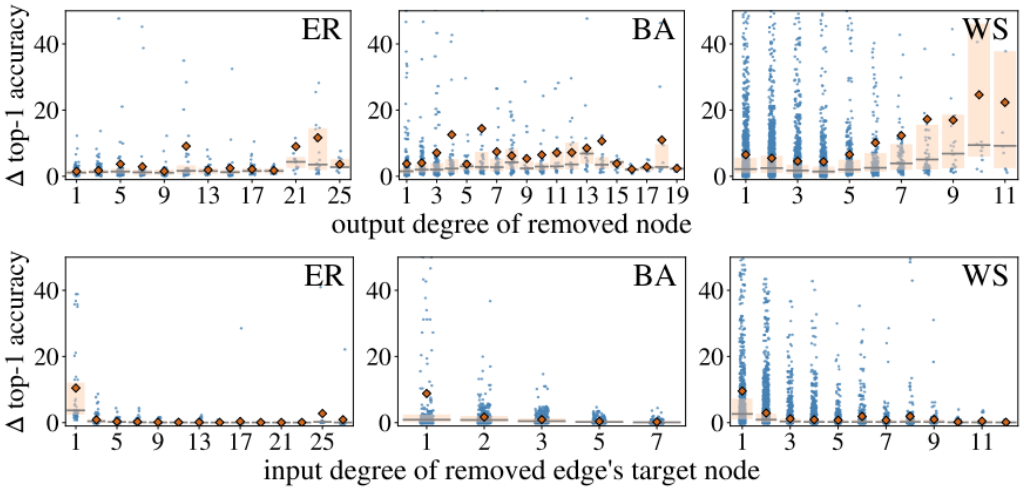


Figure: randomly remove one node/edge.

- ❑ Design search space [Radosavovic et al. 20]
  - Evaluate a search space by its error distribution.
  - Input a search space and output a refined one.

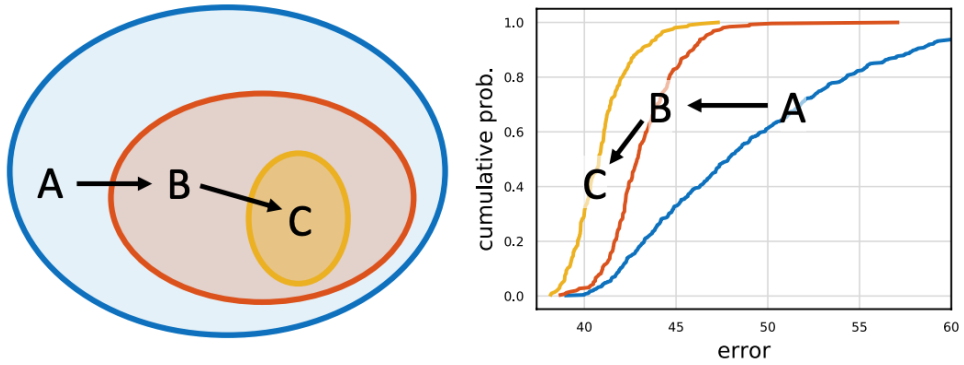


Figure: two steps of refinement with the error distribution constantly improved.

Rethinking the Search Space of NAS

- From the view of graph structure [You et al. 20a]
  - From DAG to relational graph.
  - Sweet spots are consistent across different datasets and architectures.

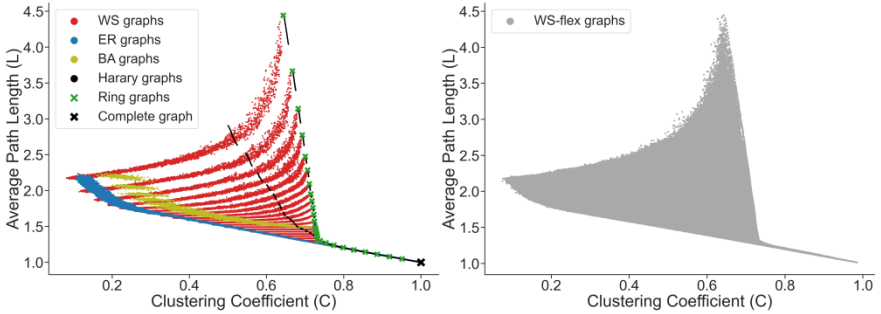


Figure: proposed WS-flex provides a larger search space.

Graph Structure of Neural Networks. *ICML*, 2020, http://proceedings.mlr.press/v119/you20b/you20b.pdf

Rethinking the Search Space of NAS

- From the view of graph structure [You et al. 20a]
  - From DAG to relational graph.
  - Sweet spots are consistent across different datasets and

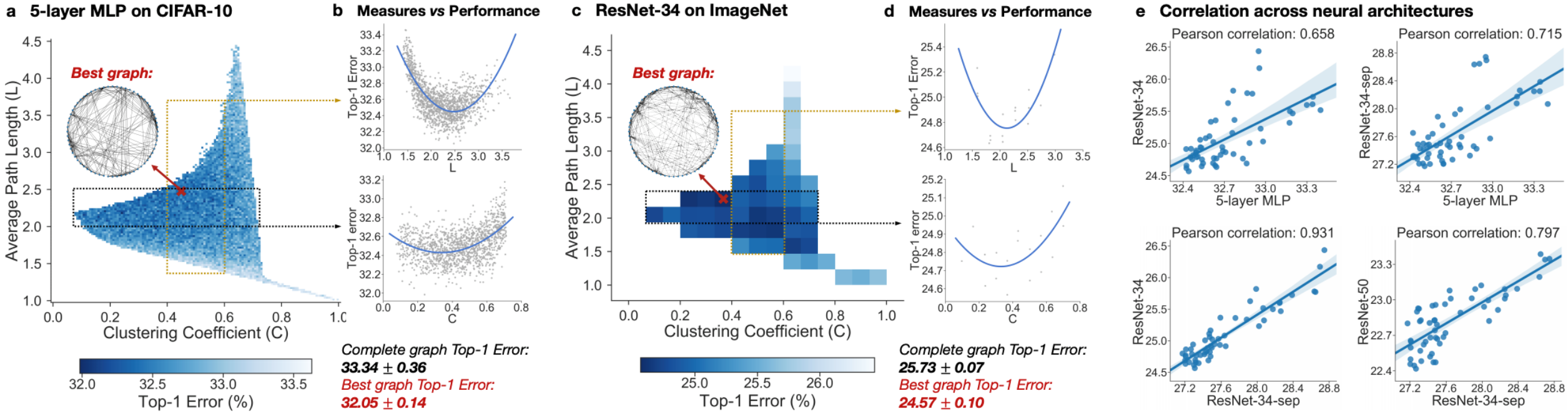


Figure: Key results.

Graph Structure of Neural Networks. *ICML*, 2020, http://proceedings.mlr.press/v119/you20b/you20b.pdf

Size Search Space

- Model scaling
  - Keep the architecture but adjust the size:
    - Depth L
    - Width C
    - And resolution H, W
  - Maximize the performance w.r.t. the size.

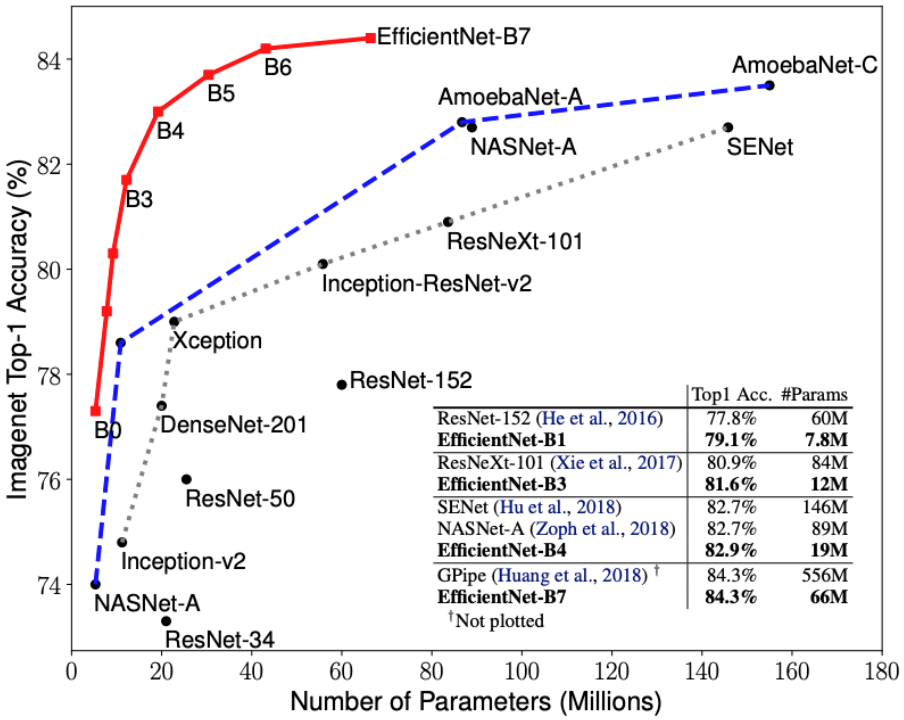
- EfficientNet [Tan et al. 19]

$$\max_{d,w,r} \quad Accuracy\big(\mathcal{N}(d,w,r)\big)$$

$$s.t. \quad \mathcal{N}(d,w,r) = \bigodot_{i=1\ldots s} \hat{\mathcal{F}}_i^{d\cdot\hat{L}_i}\big(X_{\langle r\cdot\hat{H}_i, r\cdot\hat{W}_i, w\cdot\hat{C}_i\rangle}\big)$$

$$\text{Memory}(\mathcal{N}) \le \text{target\_memory}$$

$$\text{FLOPS}(\mathcal{N}) \le \text{target\_flops}$$



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.1%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.6%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.9%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.3%**	**66M**

†Not plotted

Figure: Model size v.s. ImageNet accuracy.

- Compound scaling method: $d = \alpha^\phi,\ \omega = \beta^\phi,\ r = \gamma^\phi$ where $\alpha \times \beta^2 \times \gamma^2 \approx 2,\ \alpha \ge 1,\ \beta \ge 1,\ \gamma \ge 1.$
- Step1: Fix $\phi = 1$ and do a small grid search for α, β, γ. Step2: Fix α, β, γ and scale up ϕ.

From CNN/RNN to GNN

- Uniqueness in search space
  - More dimensions of choices:
    - Micro: mainly aggregation and combine functions.
    - Macro: how node embeddings in each layer produce the final one.
  - Nodes are not independent, so how about in a node-wise manner?

$$\mathbf{m}_i^{(l)} = \mathrm{AGG}^{(l)}\left(\left\{a_{ij}^{(l)}\mathbf{W}^{(l)}\mathbf{h}_i^{(l)}, \forall j \in \mathcal{N}(i)\right\}\right)$$
$$\mathbf{h}_i^{(l+1)} = \sigma\left(\mathrm{COMBINE}^{(l)}\left[\mathbf{m}_i^{(l)}, \mathbf{h}_i^{(l)}\right]\right),$$

Figure: General message passing.

- Challenges of weight-sharing one-shot NAS
  - Different options lead to quite different output statistics [Zhou et al. 19].
- Transfer across datasets and tasks [You et al. 20b]
  - Collect 32 (diverse) tasks.
  - Use anchor models to calculate task similarities.

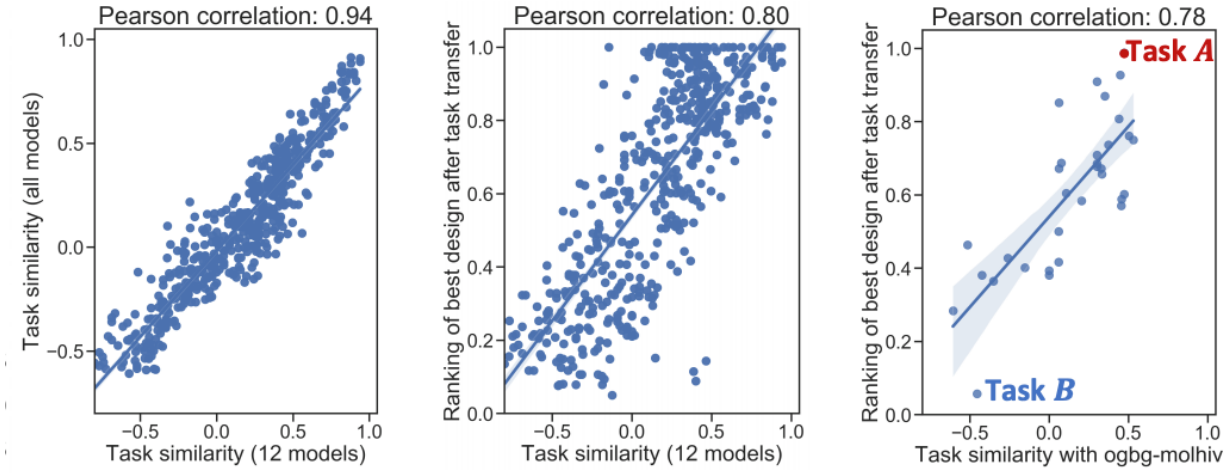
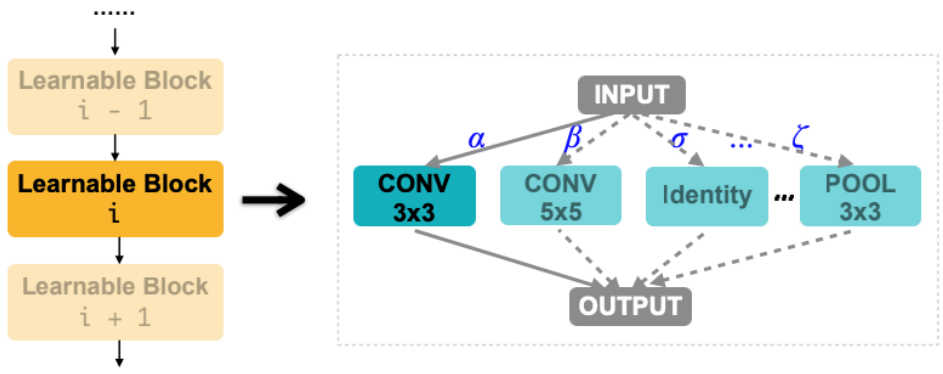


Figure: Comparing the correlations.

Beyond Accuracy: Efficiency and Robustness

❑ Making latency differentiable [Cai et al. 19]



$$\mathbb{E}[\text{Latency}] = \alpha \times F(\text{conv\_3x3}) + \beta \times F(\text{conv\_5x5}) + \sigma \times F(\text{identity}) + \ldots\ldots + \zeta \times F(\text{pool\_3x3})$$

$$\mathbb{E}[\text{latency}] = \sum_i \mathbb{E}[\text{latency}_i]$$

$$Loss = Loss_{CE} + \lambda_1 ||w||_2^2 + \lambda_2 \mathbb{E}[\text{latency}]$$

Figure: Introducing latency regularization loss.

❑ Searching robust architecture [Guo et al. 20]

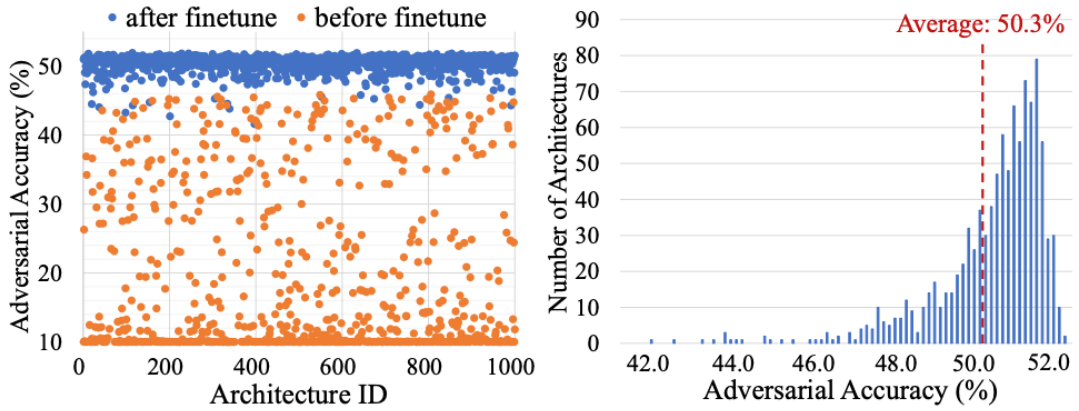


Figure: Performance of 1k sampled architecture.

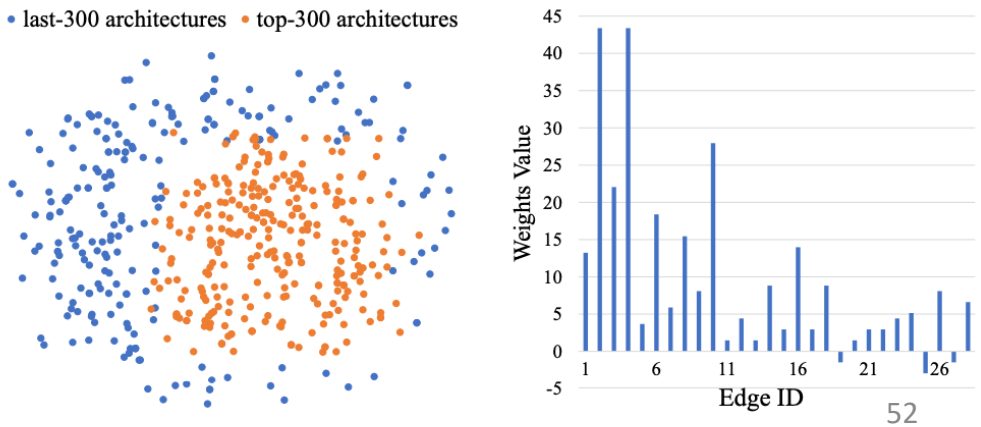
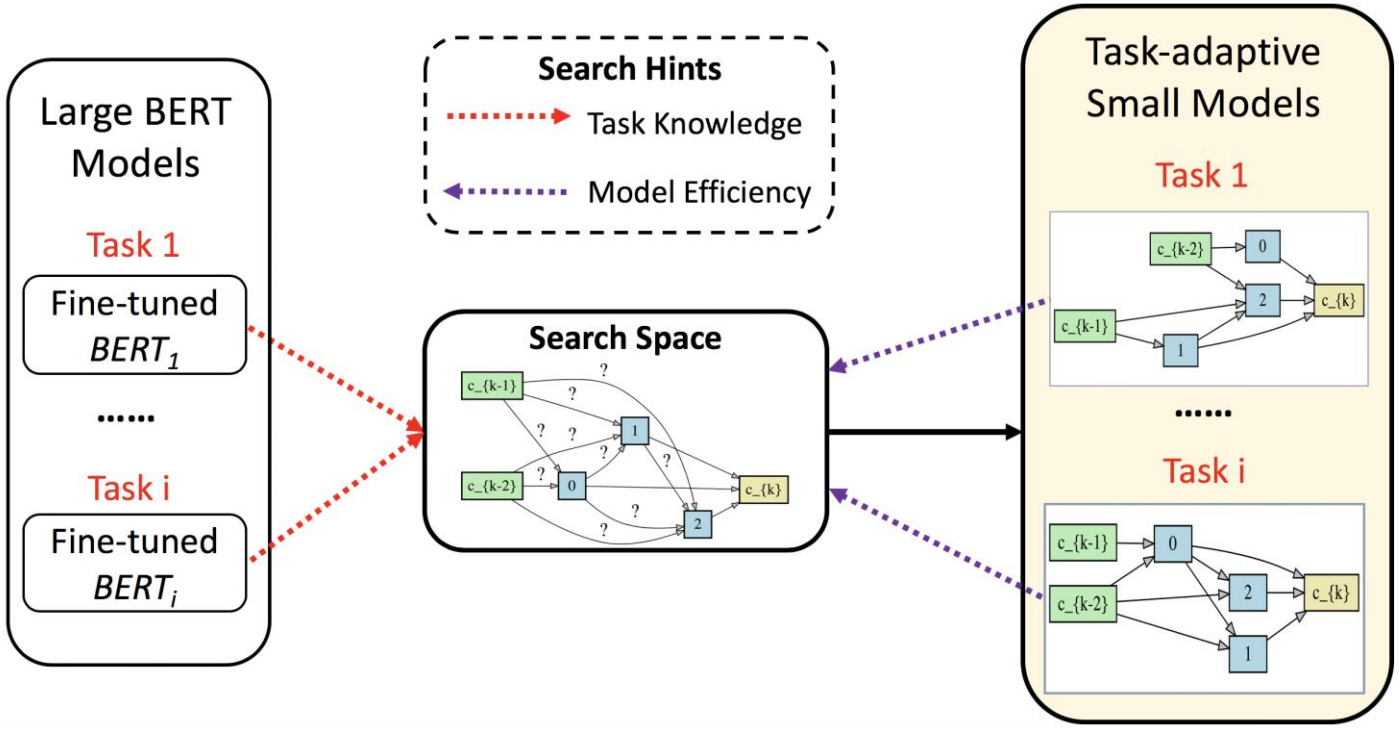


Figure: Analysis of top 300 robust v.s. non-robust architectures.

Beyond Accuracy: Compressed Model Search

- Pre-trained language model such as BERT achieves great performance on various tasks, but it is difficult to be deployed to real-time applications.
- Can we **task-adaptively** compresses original BERT for different tasks?



Method	Averaged Performance	Inference Speed
BERT	82.5	1x
BERT-PKD	80.6	1.9x
DistillBERT	76.8	3.0x
TinyBERT	80.6	9.4x
AdaBERT	**80.1**	**12.7x ~ 29.3x**

The proposed AdaBERT achieves significant speedup in inference time while maintaining comparable performance compared to uncompressed model.

AdaBERT: Task-Adaptive BERT Compression with D-NAS. *IJCAI*, 2020, https://arxiv.org/abs/2001.04246

Beyond Accuracy: Compressed Model Search

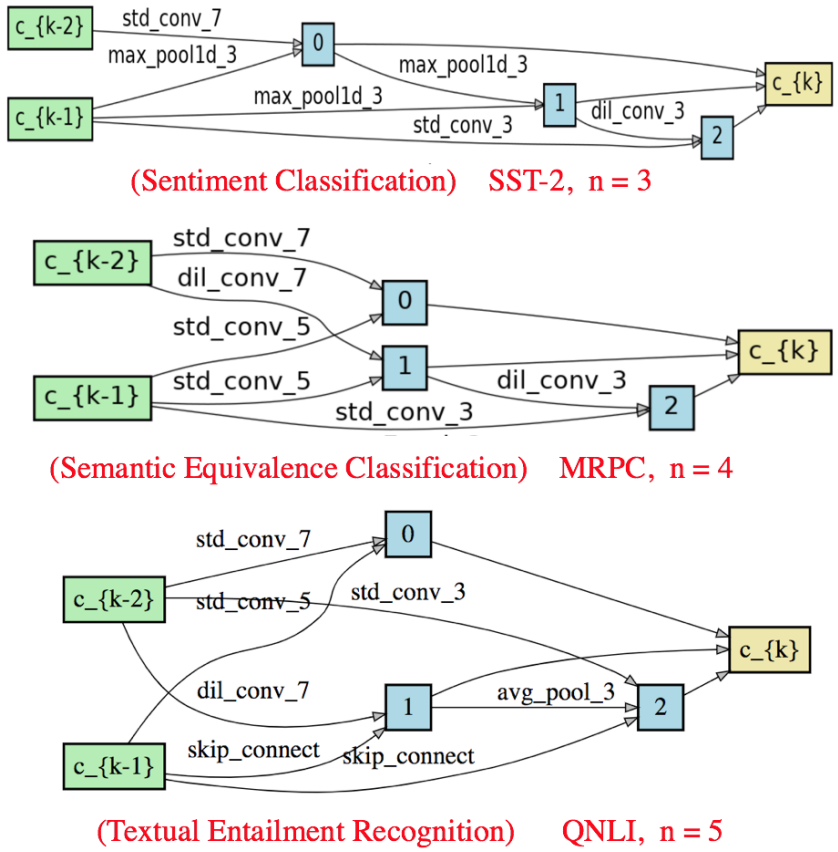


Figure: Searched structures of compressed models for different tasks

Models \ Tasks	SST-2	MRPC	QQP	MNLI	QNLI	RTE
SST-2	**91.9**	78.1	58.6	63.7	74.1	53.8
MRPC	81.5	**84.7**	68.9	75.7	82.2	60.3
QQP	81.9	84.1	**70.5**	76.2	82.5	60.5
MNLI	82.1	81.5	66.8	**80.4**	86.1	63.2
QNLI	81.6	82.3	67.7	79.1	**87.2**	62.9
RTE	82.9	81.1	66.5	79.6	86.0	**64.1**

Table: Performance of searched structures across different tasks

These results demonstrate that the proposed AdaBERT compresses original BERT adaptively for different downstream tasks.

AdaBERT: Task-Adaptive BERT Compression with D-NAS, *IJCAI*, 2020, https://arxiv.org/abs/2001.04246

NAS Benchmarks

- NAS-Bench-101 [Ying et al. 19]
  - Provides a lookup table for the 423k architectures.
  - Including their train/valid/test accuracies, number of parameters, and training time.
- NATS-Bench [Dong et al. 21]
  - Search space considers both size and topology factors.

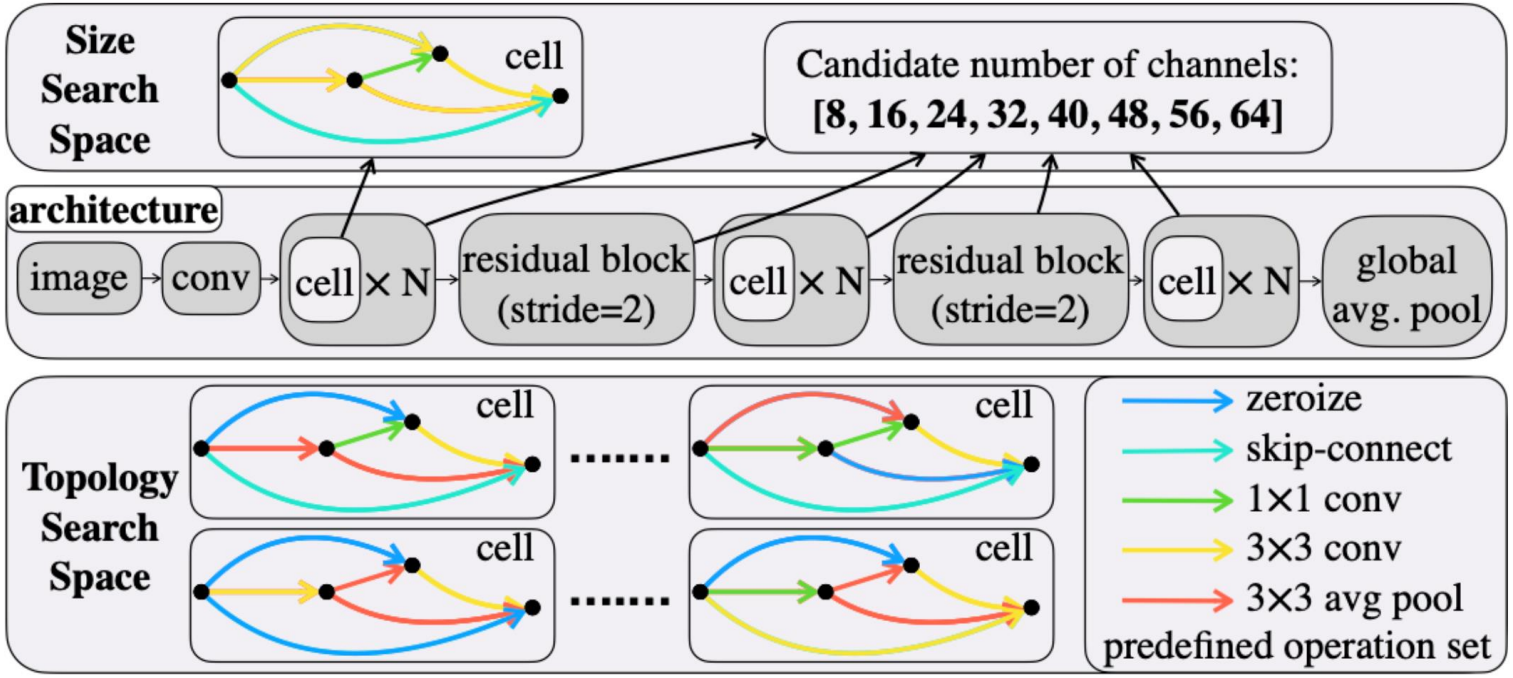


Figure: The search space of NATS-Bench.

NAS Benchmarks

- ❑ NAS-Bench-101 [Ying et al. 19]
  - Provides a lookup table for the 423k architectures.
  - Including their train/valid/test accuracies, number of parameters, and training time.
- ❑ NATS-Bench [Dong et al. 21]
  - Search space considers both size and topology factors.

	#Unique Architectures	#Datasets	Diagnostic Information	Search Space
NAS-Bench-101	423K	1	X	topology
St in NATS-Bench	6.5K	3	fine-grained information	topology
Ss in NATS-Bench	32.8K	3	fine-grained information	size

Figure: Comparison the benchmarks.

Takeaways

❑ Search space

- Layer by layer
- Pre-defined restricted design space
- Pre-defined size

→

- Repeated normal&reduction cell
- Search for design space
- Also search for optimal size

❑ Search strategy

- Trial-and-error, e.g., RL and ES

→

- One-shot NAS
- Differentiable (+sampling ops)

❑ Performance estimation strategy

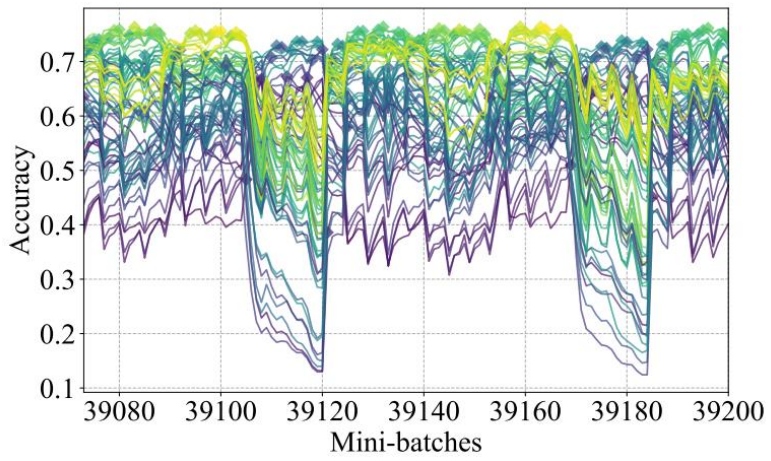
- Stand training&validation
- Single objective

→

- With weight-sharing
- Multiple objectives

Future Directions

- ❑ Reduce the variance of one-shot NAS
  - The interference between child models is a main factor [Zhang et al. 2020].
  - E.g., sharing unless some condition(s) are satisfied.



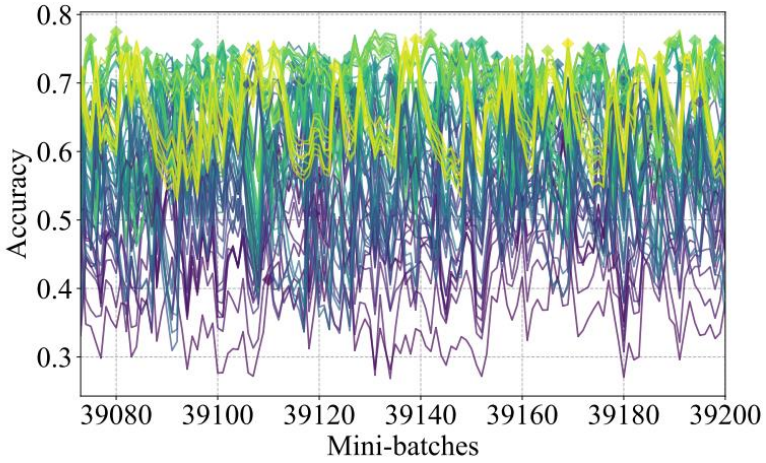
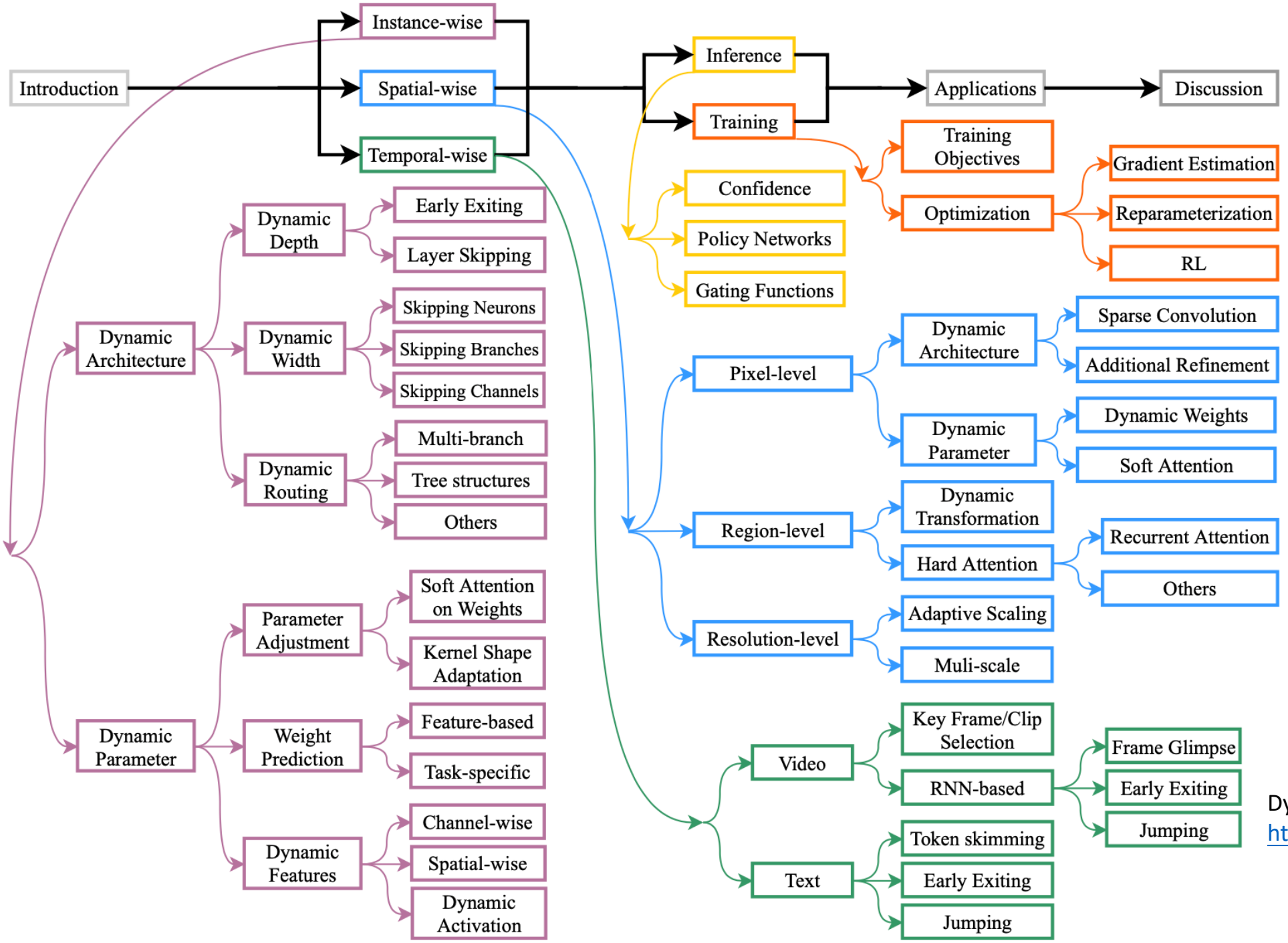


Figure: Validation performance of each child model during the last 120 steps.

- ❑ Select the truly useful architecture
  - The magnitude of architecture parameters does not necessarily indicate how much the operation contributes to the supernet's performance [Wang and Cheng et al. 2021].



- ❑ Beyond NAS: From static to dynamic neural architecture.
  - Fine-grained tuning.
  - Mainly focusing on CNNs and efficiency issue now.

Dynamic Neural Networks: A Survey, https://arxiv.org/pdf/2102.04906.pdf

References of NAS

- [Zoph et al. 2017] Neural Architecture Search with Reinforcement Learning. ICLR. 2017.
- [Zoph et al. 2018] Learning Transferable Architectures for Scalable Image Recognition. CVPR. 2018.
- [Bender et al. 2018] Understanding and Simplifying One-Shot Architecture Search. ICML. 2018.
- [Zhang et al. 2020] Deeper Insights into Weight Sharing in Neural Architecture Search. arXiv. 2020.
- [Li et al. 2021] Geometry-aware Gradient Algorithms for Neural Architecture Search. ICLR. 2021.
- [Xie et al. 2019] Exploring Randomly Wired Neural Networks for Image Recognition. ICCV. 2019.
- [Radosavovic et al. 2020] Designing Network Design Spaces. CVPR. 2020.
- [You et al. 2020a] Graph Structure of Neural Networks. ICML. 2020a.
- [Zhou et al. 2019] Auto-GNN: Neural Architecture Search of Graph Neural Networks. Arxiv. 2019.
- [You et al. 2020b] Design Space for Graph Neural Networks. NeurIPS. 2020b.
- [Cai et al. 2019] ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. ICLR. 2019.
- [Guo et al. 2020] When NAS Meets Robustness: In Search of Robust Architectures against Adversarial Attacks. CVPR. 2020.
- [Ying et al. 2019] NAS-Bench-101: Towards Reproducible Neural Architecture Search. ICML. 2019.
- [Dong et al. 2021] NATS-Bench: Benchmarking NAS Algorithms for Architecture Topology and Size. TPAMI. 2021.
- [Tan et al. 2019] EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. ICML. 2019.
- [Wang and Cheng et al. 2021] Rethinking Architecture Selection in Differentiable NAS. ICLR. 2021.

Meta-Learning

Meta-learning

❑ What is meta-learning?

- Training on a meta-dataset consisting of many datasets, where each is a different task.
- Extract prior knowledge from it that accelerates the learning of new tasks.

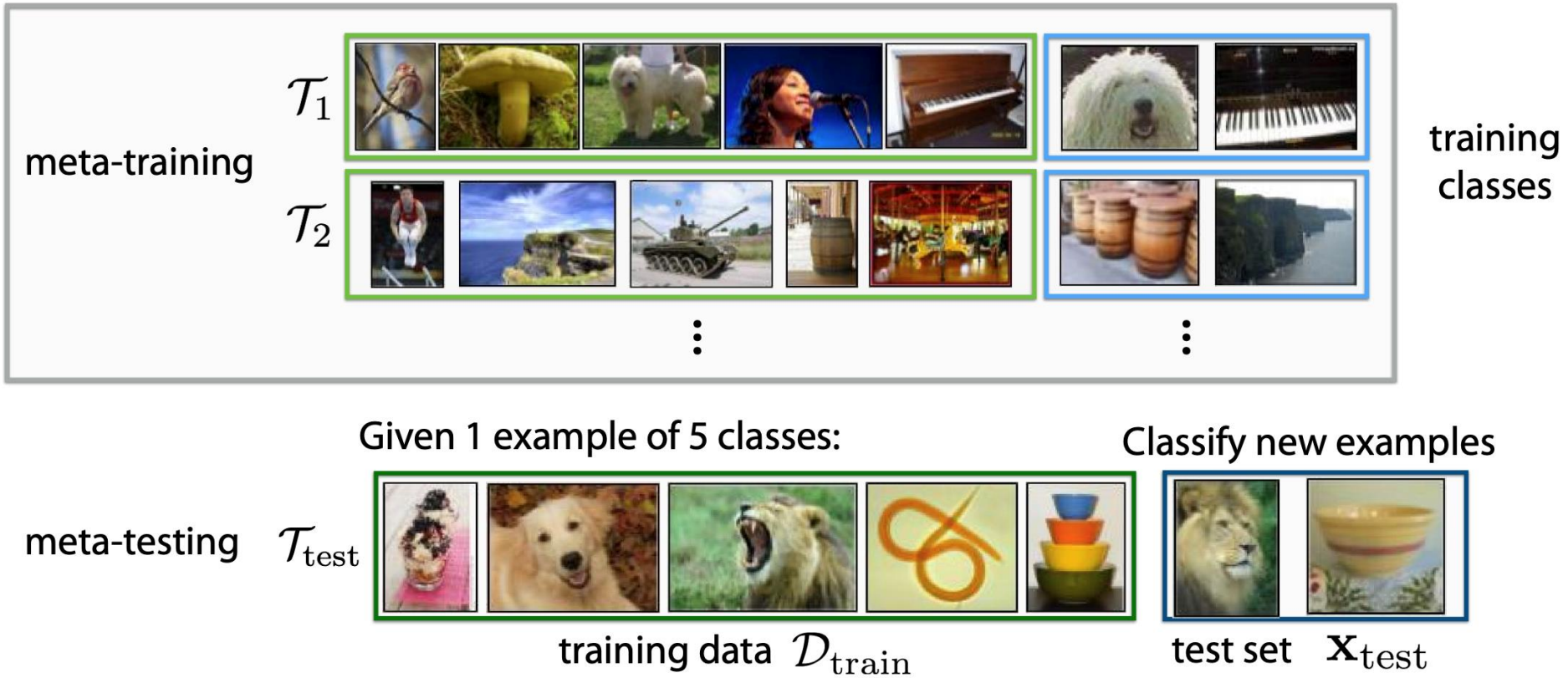


Figure: Example of how meta-learning works (source: https://cs330.stanford.edu/slides/cs330_metalearning_bbox_2020.pdf).

When Meta-learning Meets AutoML

- AutoML as a service
  - What if users do not have a large dataset for training a deep model?
  - What if users want to quickly learn a new task?

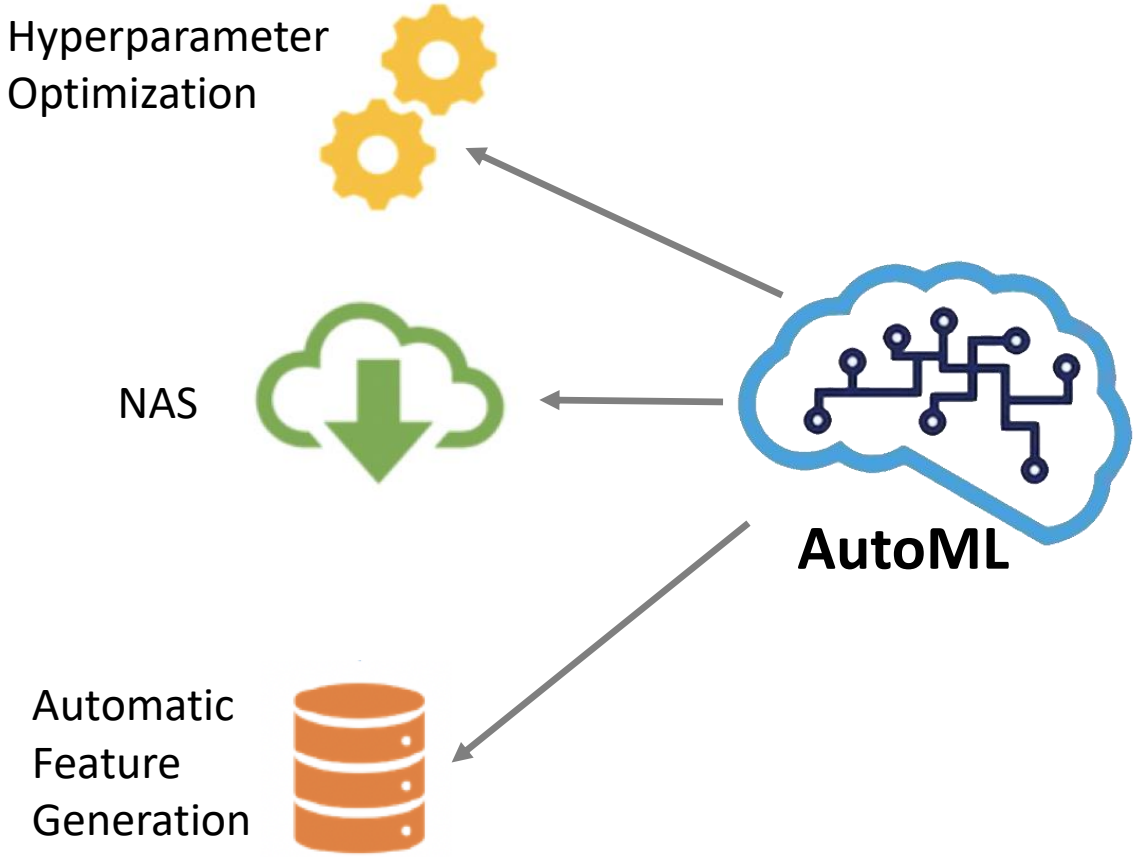
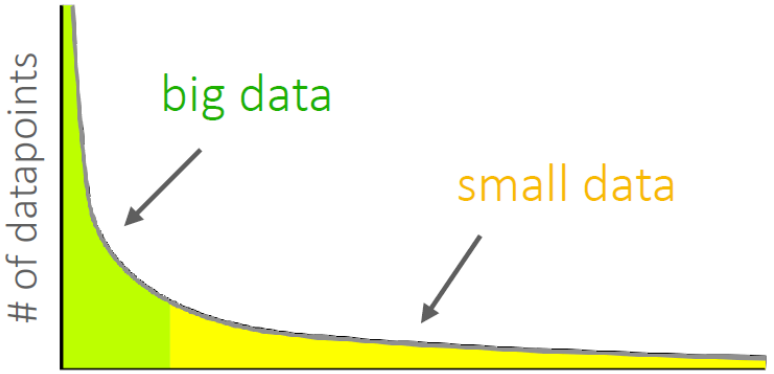


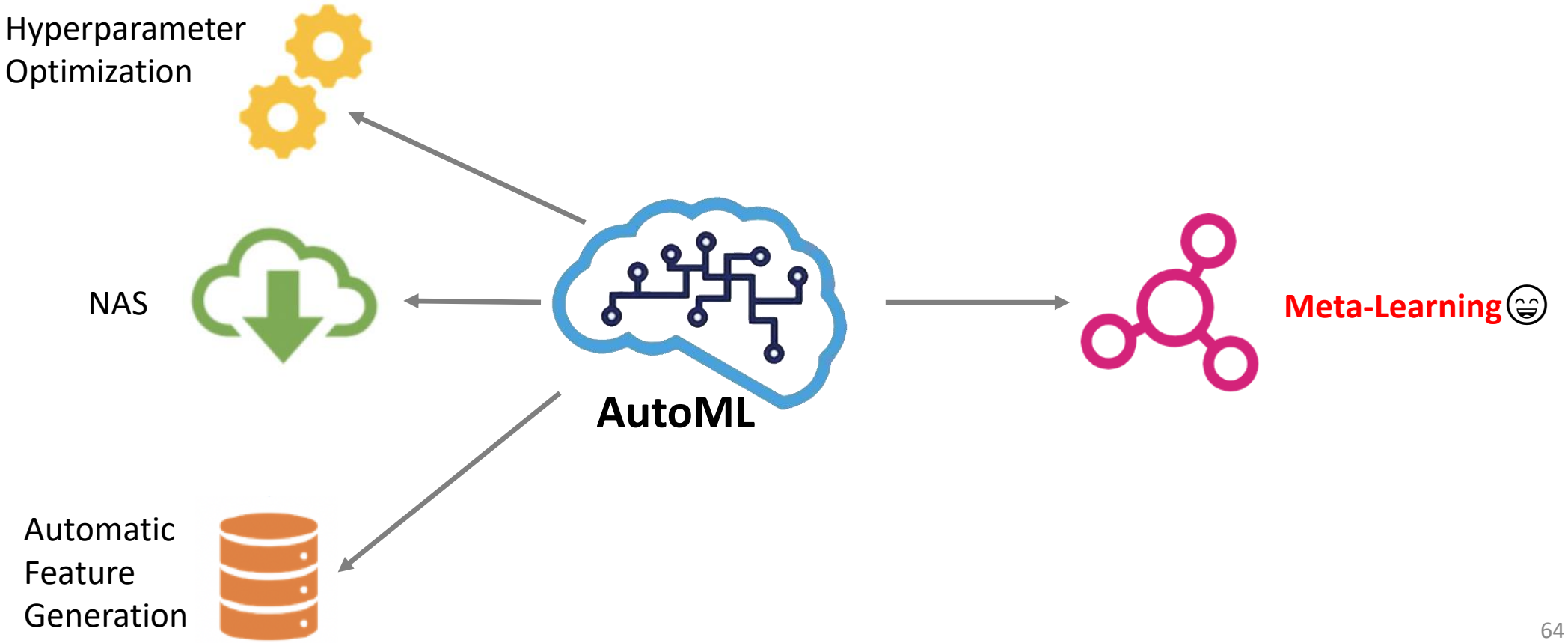
Figure: The distribution of the scales of datasets. (source: https://cs330.stanford.edu/).



When Meta-learning Meets AutoML

❑ AutoML as a service

- Assume different tasks share some common principles.
- Can we exploit the cumulated experience?



Meta-learning Basics

- Exploit the meta-dataset
  - Conventional ML: $\arg\max_{\phi} \log p(\phi|\mathcal{D})$
  - Meta-learning: $\arg\max_{\phi} \log p(\phi|\mathcal{D}, \mathcal{D}_{\text{meta-train}})$
- Replace the meta-dataset by meta-parameters
  - Sufficient to represent the meta-dataset.

learn *meta-parameters* θ: $p(\theta|\mathcal{D}_{\text{meta-train}})$

whatever we need to know about $\mathcal{D}_{\text{meta-train}}$ to solve new tasks

this is the meta-learning problem

assume $\phi \perp\!\!\!\perp \mathcal{D}_{\text{meta-train}}|\theta$

$$\log p(\phi|\mathcal{D}, \mathcal{D}_{\text{meta-train}}) = \log \int_{\Theta} p(\phi|\mathcal{D}, \theta) p(\theta|\mathcal{D}_{\text{meta-train}}) d\theta$$

$$\approx \log p(\phi|\mathcal{D}, \theta^\star) + \log p(\theta^\star|\mathcal{D}_{\text{meta-train}})$$

$$\arg\max_{\phi} \log p(\phi|\mathcal{D}, \mathcal{D}_{\text{meta-train}}) \approx \arg\max_{\phi} \log p(\phi|\mathcal{D}, \theta^\star)$$

this is the adaptation problem

$$\theta^\star = \arg\max_{\theta} \log p(\theta|\mathcal{D}_{\text{meta-train}})$$

Optimization-based Meta-learning

- Adaptation problem
  - Acquire ϕ_i via optimization $\phi_i = argmax_\phi \log p(D_i^{tr}|\phi) + \log p(\phi|\theta)$.
  - θ serves as a prior.
- Which form of prior to take?
  - Initialization and fine-tuning!

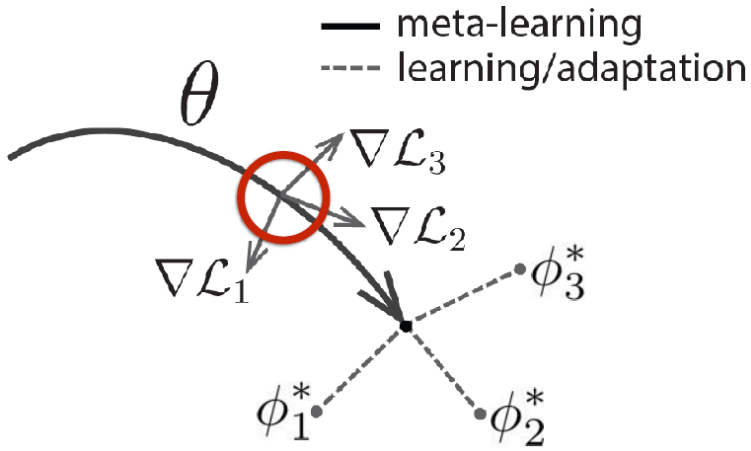


Figure: Illustrating the idea of optimization-based meta-learning (source: https://arxiv.org/pdf/1703.03400.pdf).

Fine-tuning $\phi \leftarrow \theta - \alpha\nabla_\theta\mathcal{L}(\theta, \mathcal{D}^{\text{tr}})$

pre-trained parameters; training data for new task; (typically for many gradient steps)

Meta-learning $\min_\theta \sum_{\text{task } i} \mathcal{L}(\theta - \alpha\nabla_\theta\mathcal{L}(\theta, \mathcal{D}_i^{\text{tr}}), \mathcal{D}_i^{\text{ts}})$

$$g_{\text{MAML}} = \bar{g}_2 - \alpha\bar{H}_2\bar{g}_1 - \alpha\bar{H}_1\bar{g}_2 + O(\alpha^2)$$
$$g_{\text{FOMAML}} = g_2 = \bar{g}_2 - \alpha\bar{H}_2\bar{g}_1 + O(\alpha^2)$$
$$g_{\text{Reptile}} = g_1 + g_2 = \bar{g}_1 + \bar{g}_2 - \alpha\bar{H}_2\bar{g}_1 + O(\alpha^2)$$

Where $g_i = \frac{\partial}{\partial\theta_i}L(\theta_i, D^{tr})$, $\bar{g}_i = \frac{\partial}{\partial\theta_1}L(\theta_i, D^{tr})$,

$\bar{H}_i = \frac{\partial}{\partial\theta_1}\bar{g}_i$ (Hessian w.r.t. θ_1)

Optimization-based Meta-learning

- ❑ Probabilistic interpretation
  - Maximize a posterior (MAP) with θ as the prior.
- ❑ MAML [Finn et al. 17] approximates hierarchical Bayesian inference!
  - Gradient descent with early stop = MAP inference under Gaussian prior with mean at initial parameters.
  - Other forms, e.g.,

$$\phi \leftarrow \min_{\phi'} \mathcal{L}(\phi', \mathcal{D}^{\text{tr}}) + \frac{\lambda}{2}||\theta - \phi'||^2$$

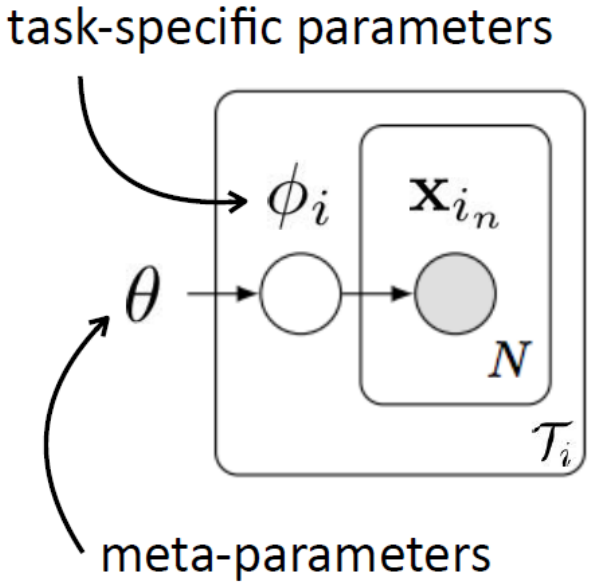


Figure: Probabilistic interpretation of optimization-based meta-learning (source: https://cs330.stanford.edu/).

😄 Model-agnostic
😄 Maximally expressive with sufficiently deep neural networks
☹ Typically requires second-order computation/memory intensive

Model-based Meta-learning

❑ Adaptation problem

- From solving optimization problem to black-box adaptation $\phi_i = f_\theta(D_i^{tr}) = argmax_\phi \log p(\phi|D_i^{tr}, \theta)$
- Train a neural networks to represent $p(\phi_i|D_i^{tr}, \theta)$
  - E.g., RNN, Neural Turing Machine, memory-augmented NN [Santoro et al. 16], etc.

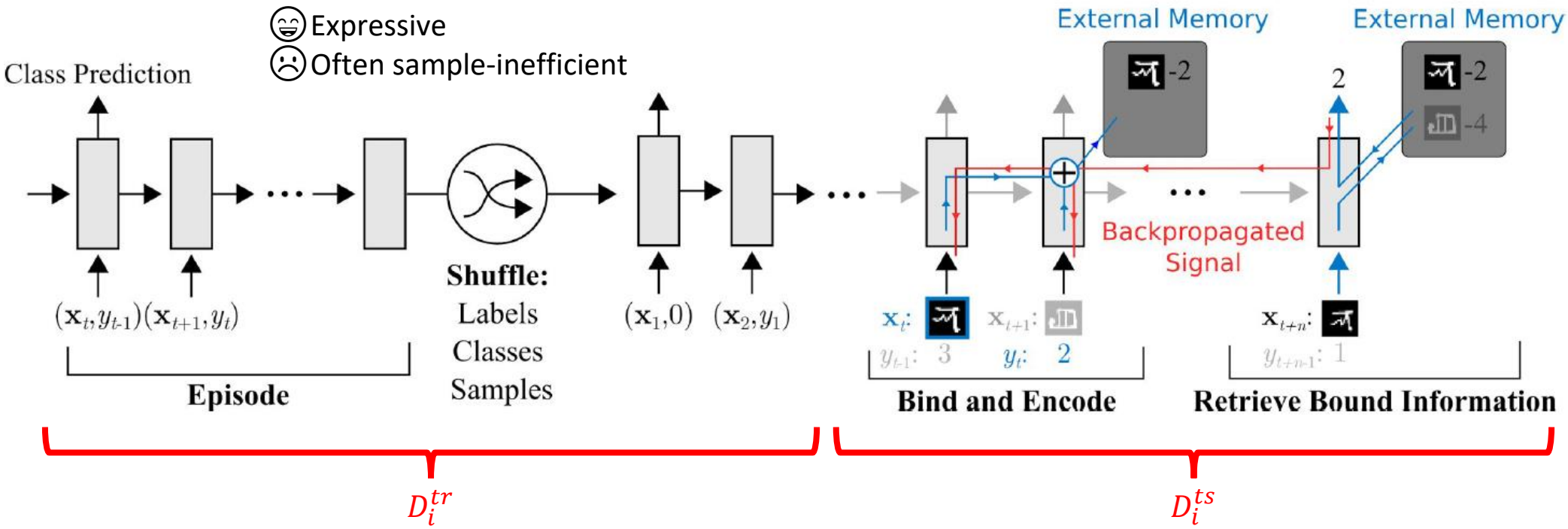


Figure: Memory-augmented neural networks (source: https://proceedings.mlr.press/v48/santoro16.pdf).

Metric-based Meta-learning

- Use Non-parametric learner

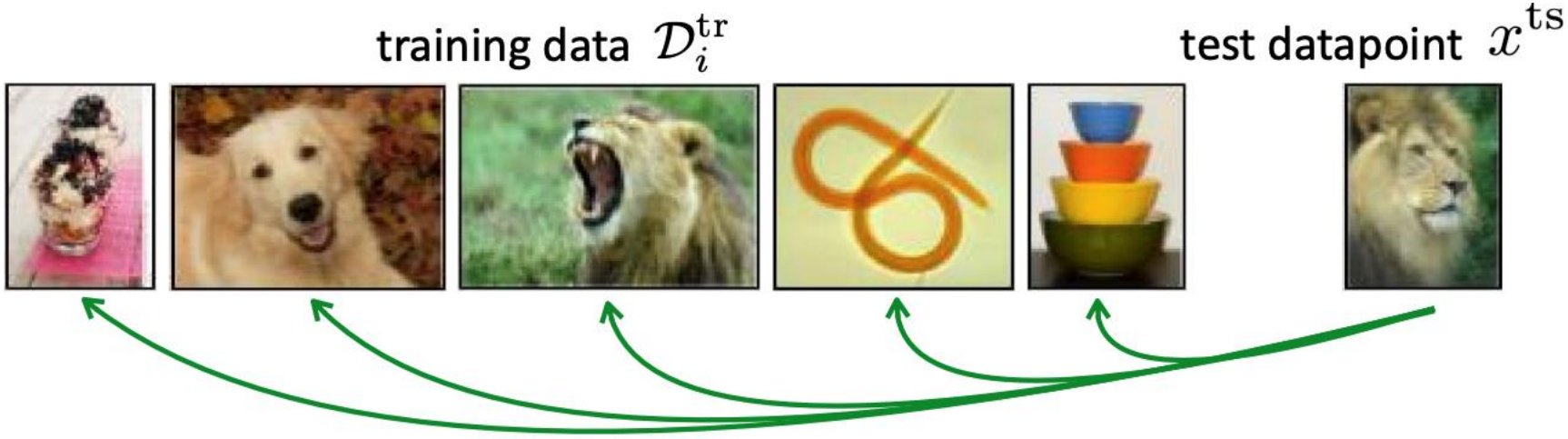


Figure: The idea of metric-based meta-learning (source: https://cs330.stanford.edu/slides/cs330_nonparametric_2020.pdf).

Compare test image with training images

In what space do you compare? With what distance metric?

~~pixel space, l_2 distance?~~

Learn to compare using meta-training data!

😄 Entirely feedforward
😄 Easy to optimize
☹ Harder to generalize to varying k-ways (especially for very large k)

Metric-based Meta-learning

- Use Siamese neural networks

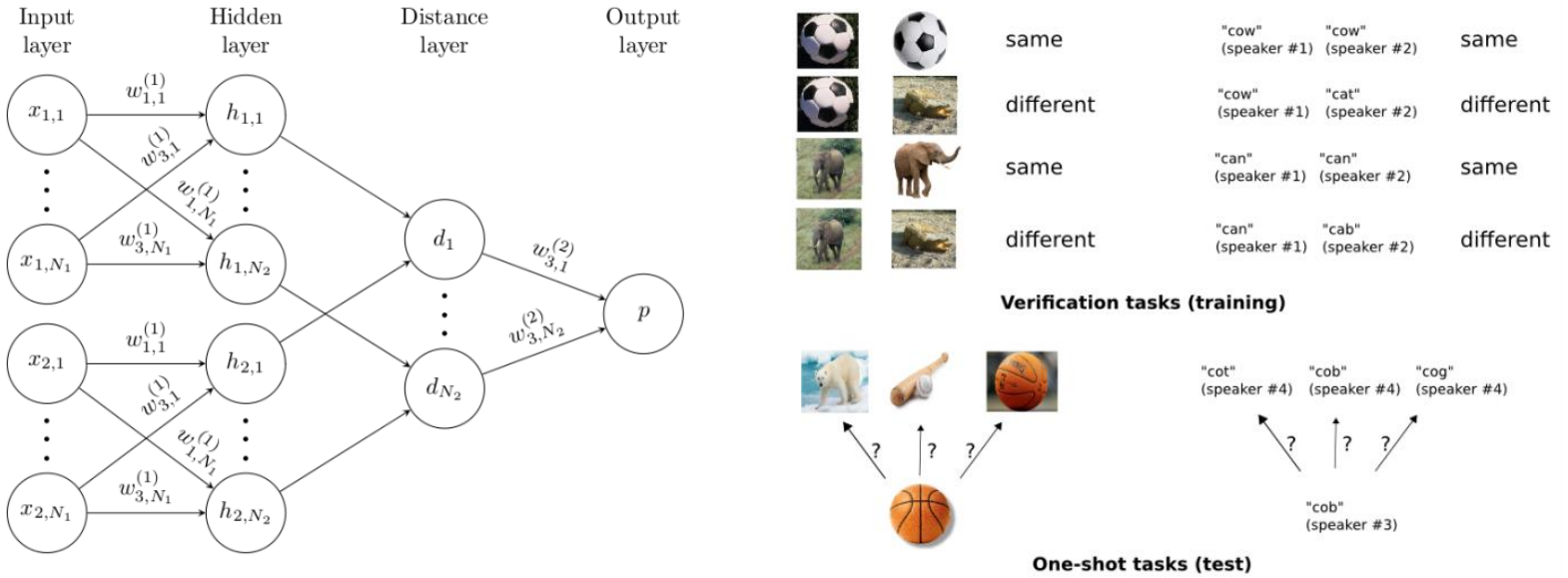


Figure: Architecture of Siamese neural networks and its application to one-shot learning.

- Meta-training: binary classification.
- Meta-test: k-way classification.

Siamese Neural Networks for One-shot Image Recognition. *ICML*, 2015, https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf

Metric-based Meta-learning

- Match the train&test phases by Matching networks
  - Fix the mismatch between meta-training and meta-test.
  - Map a (support) set $S = \{(x_i, y_i)\}$ to a classifier:

$$P(\hat{y}|\hat{x}, S) = \sum_{i=1}^{k} a(\hat{x}, x_i) y_i$$

    - The attention mechanism $a(\cdot,\cdot)$ fully specifies the classifier.

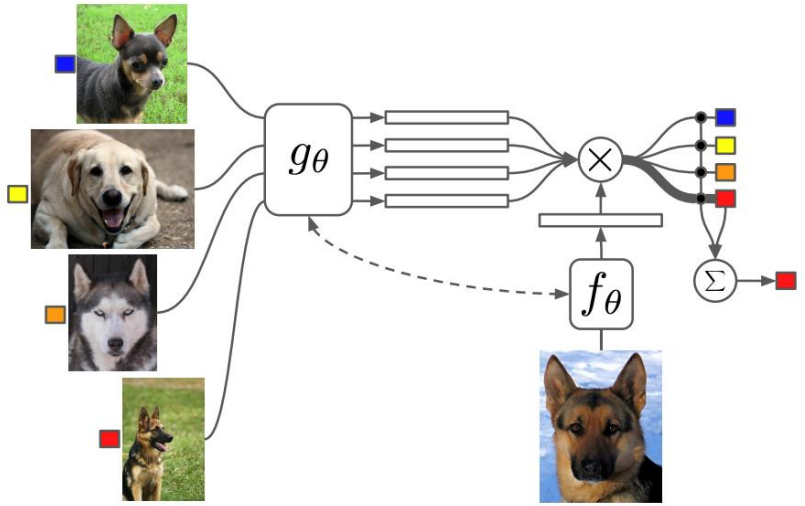


Figure: Architecture of Matching network.

Matching Networks for One Shot Learning. *NeurIPS*, 2016, https://arxiv.org/abs/2001.00745

GPT-3: meta-learning as pre-training

- ❑ What's the meta-dataset?
  - Crawled text corpora.
  - D_i^{tr}: sequence of characters, D_i^{ts}: the following sequence of characters.
- ❑ What's the meta-learning problem?
  - Put different tasks all in the form of text.
  - Thus trained on language generation tasks.
- ❑ What's the extracted prior knowledge?
  - A "Transformer" model as the initialization.

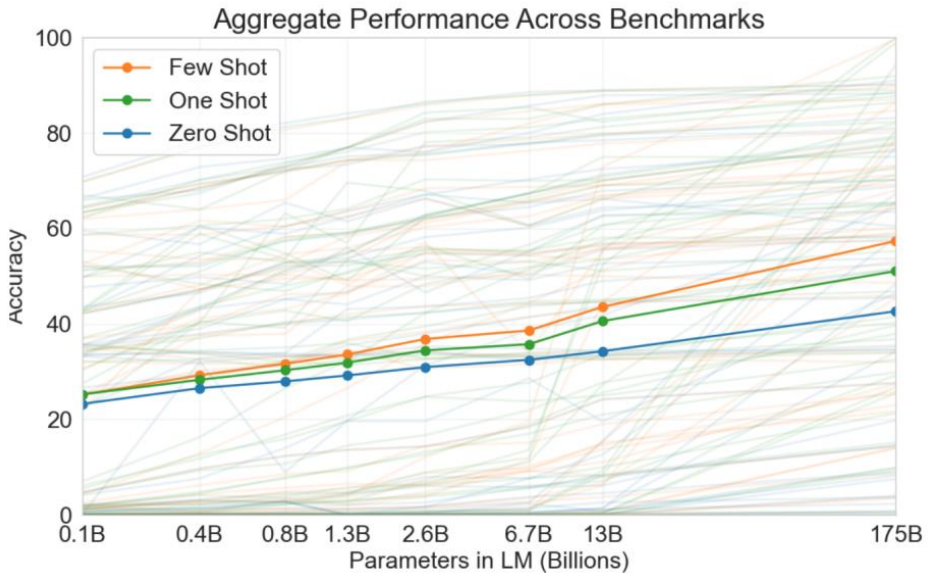


Figure: The model is far from perfect (source: https://github.com/shreyashankar/gpt3-sandbox/blob/master/docs/priming.md).

Generalization v.s. Customization

- Key assumption of meta-learning
  - Meta-training and meta-test tasks are drawn i.i.d. from the same task distribution.
  - E.g., Omniglot:
    - 1623 characters from 50 different alphabets.
    - 20 instances for each character.

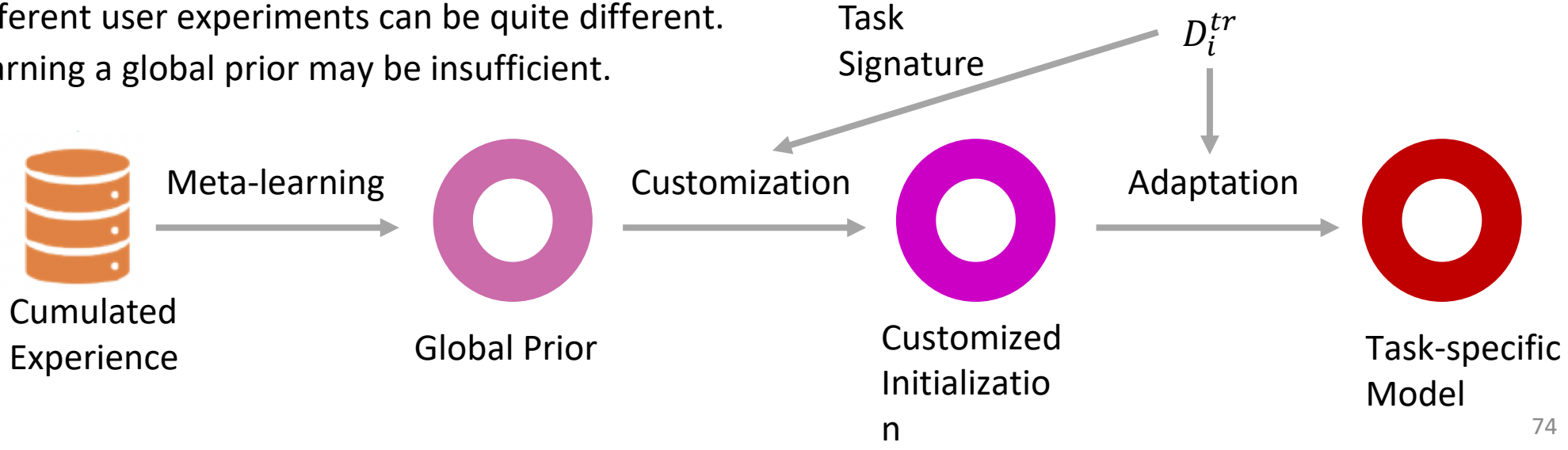
Figure: Characters of different alphabets (source: https://omniglot.com/).

Generalization v.s. Customization

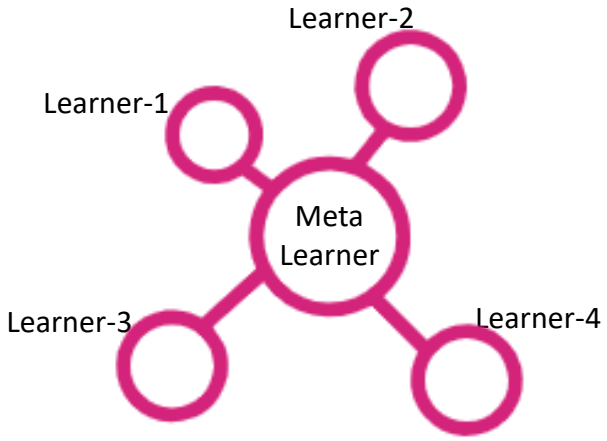
- Key assumption of meta-learning
  - Meta-training and meta-test tasks are drawn i.i.d. from the same task distribution.
  - E.g., Omniglot:
    - 1623 characters from 50 different alphabets.
    - 20 instances for each character.

☹ Can NOT be strictly satisfied!

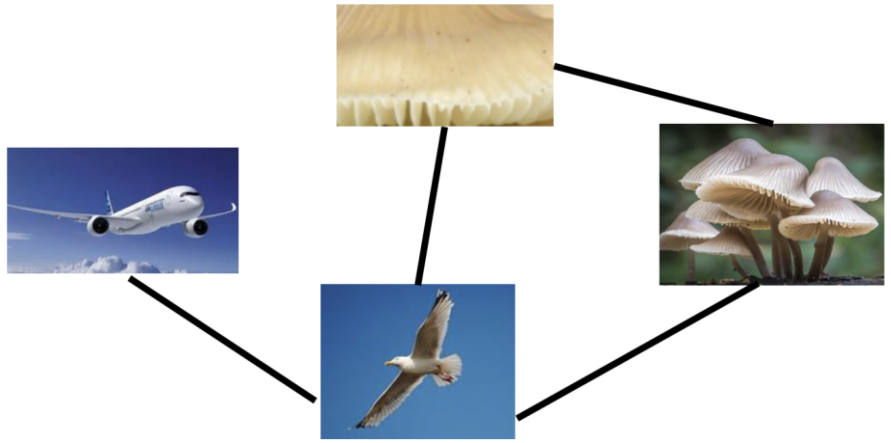
- Experience cumulated on the cloud
  - Different user experiments can be quite different.
  - Learning a global prior may be insufficient.



Relational Meta-Learning



Most Meta-learning methods don't capture the relations among tasks/learners.



Algorithms	Data: Bird	Data: Texture	Data: Aircraft	Data: Fungi
MAML	$53.94 \pm 1.45\%$	$31.66 \pm 1.31\%$	$51.37 \pm 1.38\%$	$42.12 \pm 1.36\%$
MetaSGD	$55.58 \pm 1.43\%$	$32.38 \pm 1.32\%$	$52.99 \pm 1.36\%$	$41.74 \pm 1.34\%$
MT-Net	$58.72 \pm 1.43\%$	$32.80 \pm 1.35\%$	$47.72 \pm 1.46\%$	$43.11 \pm 1.42\%$
MUMOMAML	$56.82 \pm 1.49\%$	$33.81 \pm 1.36\%$	$53.14 \pm 1.39\%$	$42.22 \pm 1.40\%$
HSML	$60.98 \pm 1.50\%$	$35.01 \pm 1.36\%$	$57.38 \pm 1.40\%$	$44.02 \pm 1.39\%$
ProtoNet	$54.11 \pm 1.38\%$	$32.52 \pm 1.28\%$	$50.63 \pm 1.35\%$	$41.05 \pm 1.37\%$
TADAM	$56.58 \pm 1.34\%$	$33.34 \pm 1.27\%$	$53.24 \pm 1.33\%$	$43.06 \pm 1.33\%$
ARML	$\mathbf{62.33 \pm 1.47\%}$	$\mathbf{35.65 \pm 1.40\%}$	$\mathbf{58.56 \pm 1.41\%}$	$\mathbf{44.82 \pm 1.38\%}$

The proposed relational meta-learning method can capture the relations among different tasks, which enhances the effectiveness of meta-learners.

Automated Relational Meta-learning, *ICLR*, 2020, https://arxiv.org/abs/2001.00745

Summary and Future Directions

- How to utilize existing experience---meta-learning
  - Learn a meta-parameter, so that we can quickly transfer to new task.
  - Optimization-based, model-based, metric-based
- What if tasks are heterogeneous?
  - Trade-off between generalization v.s. customization
- Use meta-learning for improving real-world services
  - AutoML as a service has cumulated a lot of experience.
  - Learning tasks on different domains and/or with different models share some intrinsic patterns of machine learning.
  - What kinds of features are transferable? How to represent a task, a model, and a objective?

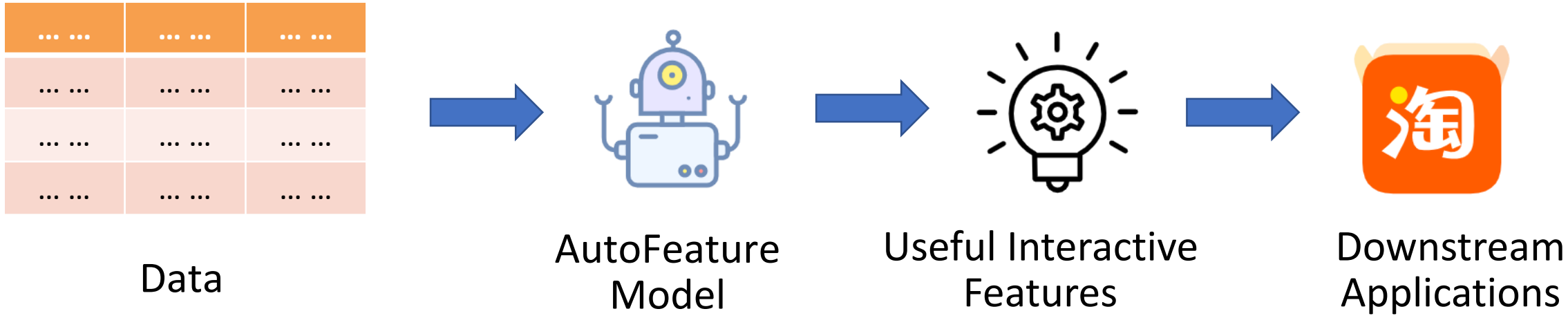
References of Meta-learning

[Santoro et al. 2016] Meta-Learning with Memory-Augmented Neural Networks. ICML. 2016.
[Finn et al. 2017] Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. ICML. 2017.
[Nichol et al. 2018] On First-Order Meta-Learning Algorithm. arXiv. 2018.
[Kock et al. 2015] Siamese Neural Networks for One-shot Image Recognition. ICML. 2015.
[Vinyals et al. 2016] Matching Networks for One Shot Learning. NeurIPS. 2016.

Auto Feature Generation 

Automatic Feature Generation

- In practice, many data scientists search for useful interactive features in a trial-and-error manner, which has occupied a lot of their workloads.
- Therefore, automatic feature generation (AutoFeature), as one major topic of automated machine learning (AutoML), has received a lot of attention from both academia and industry.



Automatic Feature Generation

Feature Interpretability

- Industries such as healthcare and finance need interpretability
- Can be applied to train lightweight models for real-time requirement

Search Efficiency

- The number of possible interactive features is too large to be traversed
  ($O(2^m)$ for m original features)

Automatic Feature Generation

- The related works on automatic feature generation can be roughly divided into two categories:
  - DNN-based methods
  - Search-based methods

DNN-based methods design specific neural architectures to express the interactions among different features.

- Implicit feature generation
- One-shot training course
- Lack of interpretable rules for feature interactions

Search-based methods focus on designing different search strategies that prune as much of the candidates to be evaluated as possible, while aiming to keep the most useful interactive features.

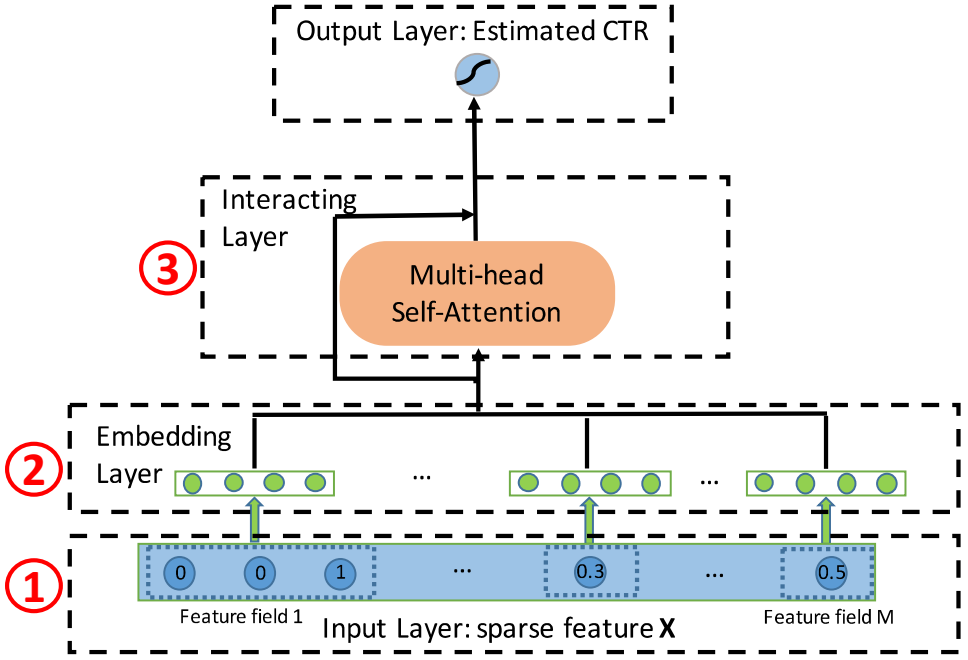
- Explicit feature generation
- Trial-and-error training manner
- Need lots of time and computing resource

AutoInt

- Map the original features into low-dimensional feature space and model the high-order feature interactions via self-attention.

① **Input Layer:**
Each feature field is represented as an one-hot vector (for categorical feature) or a scalar value (for numerical feature).

② **Embedding Layer:**
To transform the sparse and high-dimension features into a low-dimensional feature space via a learnable embedding matrix.



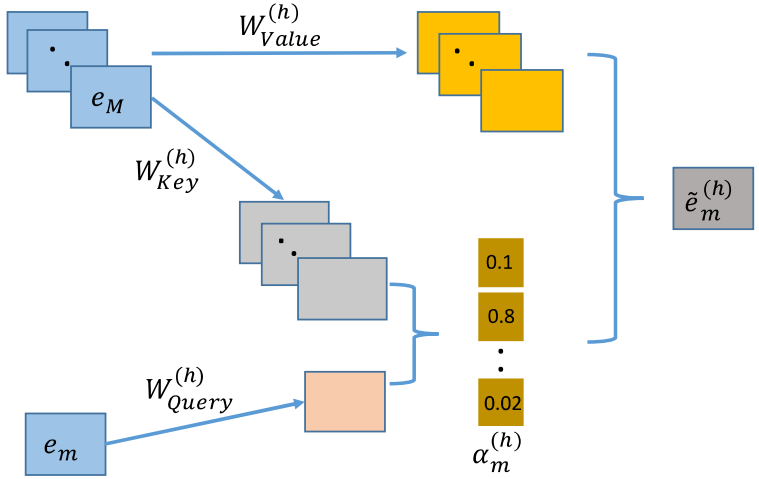
Overview of the proposed model AutoInt.

AutoInt: Automatic Feature Interaction Learning via Self-Attentive Neural Networks. *CIKM,* 2019.

AutoInt

- Map the original features into low-dimensional feature space and model the high-order feature interactions via self-attention.

③ Interacting Layer:
The multi-head key-value attention mechanism is adopted to capture the interactions between different features.

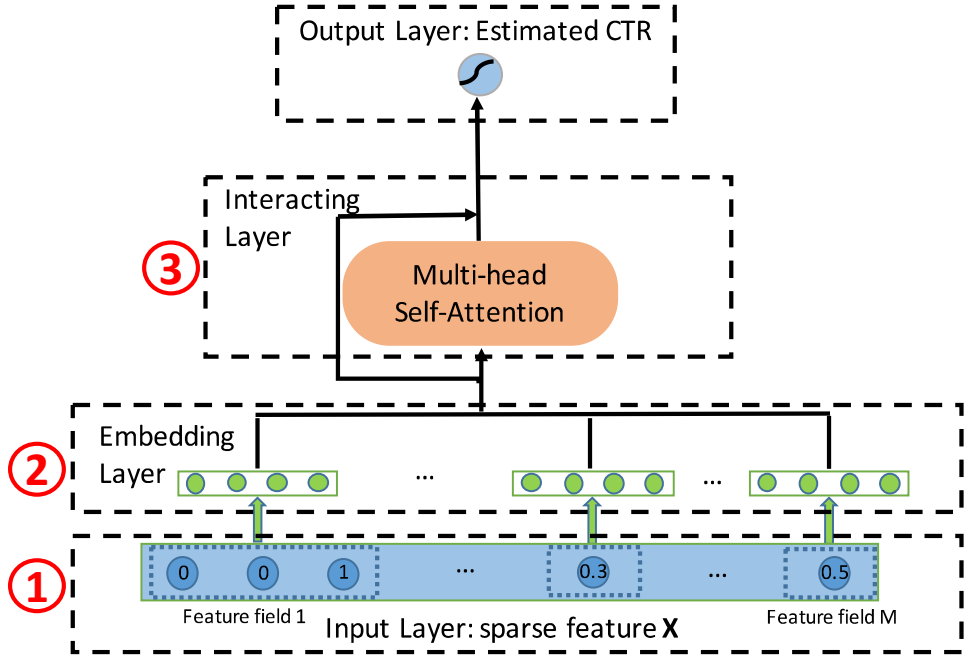


The architecture of interacting layer.

$$\alpha_{m,k}^{(h)} = \frac{\exp(\psi^{(h)}(\mathbf{e}_m, \mathbf{e}_k))}{\sum_{l=1}^{M} \exp(\psi^{(h)}(\mathbf{e}_m, \mathbf{e}_l))},$$

$$\psi^{(h)}(\mathbf{e}_m, \mathbf{e}_k) = \langle \mathbf{W}_{Query}^{(h)} \mathbf{e}_m, \mathbf{W}_{Key}^{(h)} \mathbf{e}_k \rangle,$$

$$\tilde{\mathbf{e}}_m^{(h)} = \sum_{k=1}^{M} \alpha_{m,k}^{(h)} (\mathbf{W}_{Value}^{(h)} \mathbf{e}_k),$$

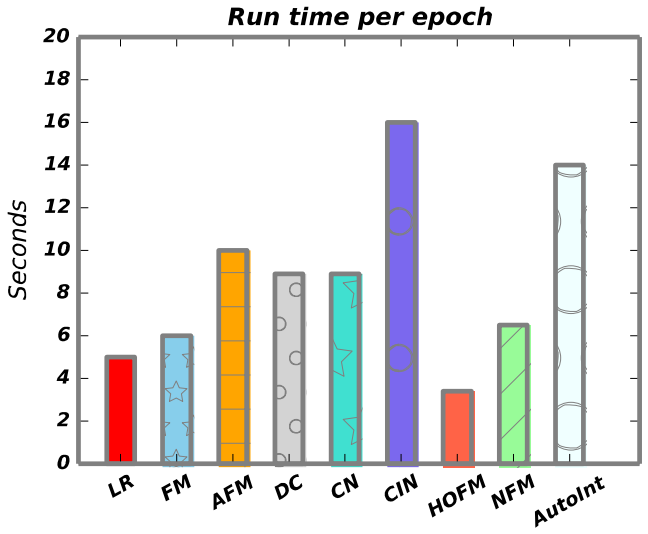


Overview of the proposed model AutoInt.

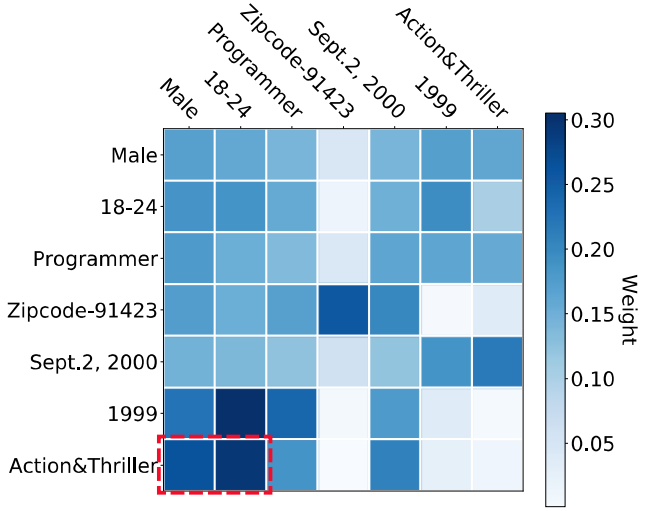
AutoInt: Automatic Feature Interaction Learning via Self-Attentive Neural Networks. *CIKM,* 2019.

AutoInt

- Experimental results on four real-world datasets show the advantages of AutoInt
  - Performance comparison in offline AUC evaluation for click-through rate (CTR) prediction
  - Efficiency comparison
  - Explainable recommendations



Efficiency comparison on *MovieLens-1M*.



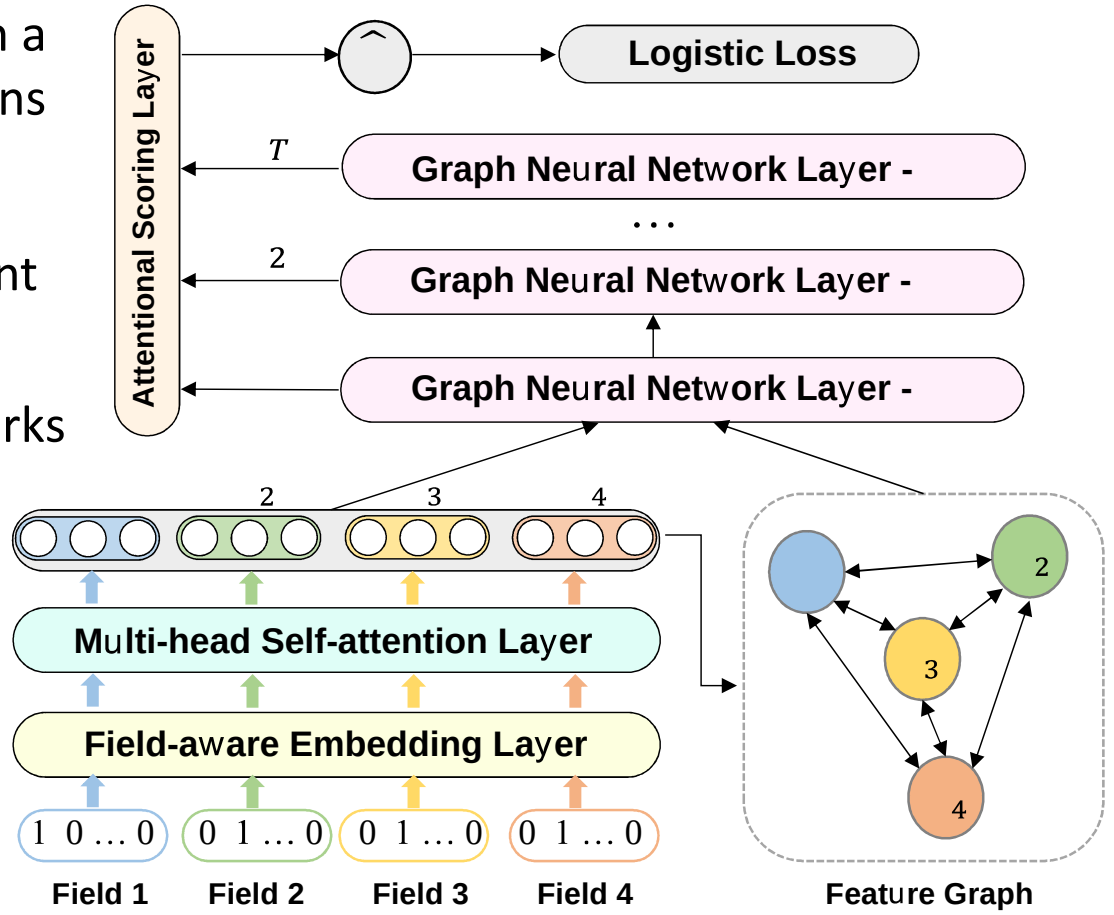
An instance of attention weights for feature interactions on *MovieLens-1M*.

AutoInt: Automatic Feature Interaction Learning via Self-Attentive Neural Networks. *CIKM,* 2019.

Fi-GNN

- Fi-GNN proposes to represent multi-field features in a graph structure, and captures the feature interactions through node representation learning in the graph.

- Feature interaction via a graph view: nodes represent features and edges denote their interactions
- Model feature interactions via Graph Neural Networks (GNN)
- Attentional scoring for predictions



Overview of the proposed Fi-GNN.

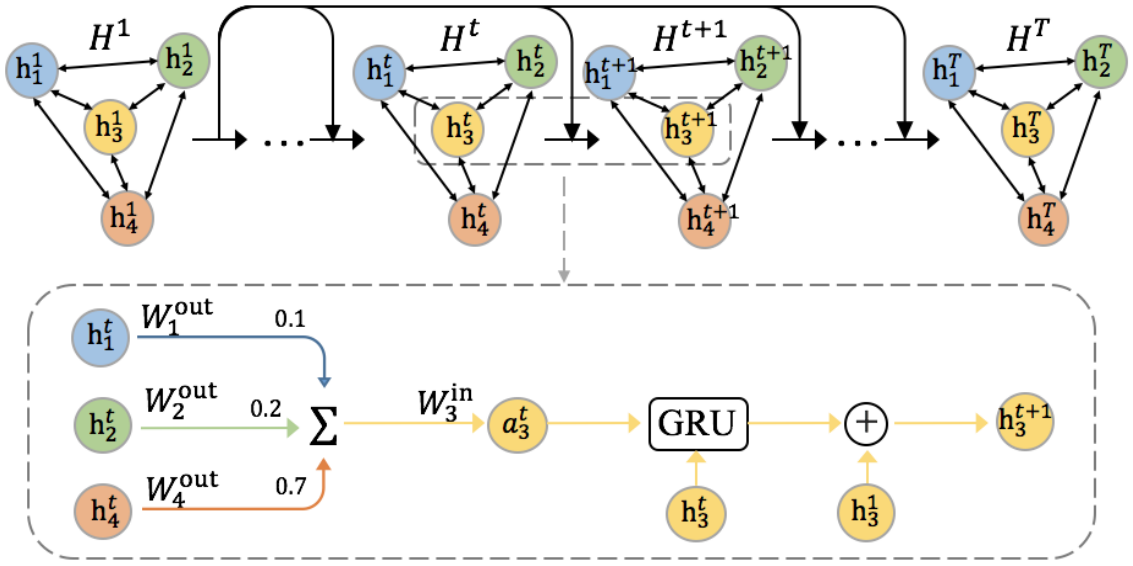
Fi-GNN: Modeling Feature Interactions via Graph Neural Networks for CTR Prediction. *CIKM,* 2019.

Fi-GNN

- Feature interaction in Fi-GNN: The nodes interact with neighbors and update their states in a recurrent fashion.

Feature Graph:
The edge weights reflect the importance of interactions between the connected nodes (features), which are learned via an attention mechanism.

Node Aggregation:
The node aggregates the transformed information from neighbors and update its state according to the aggregated information and history via GRU and residual connection.



Feature interaction in Fi-GNN.

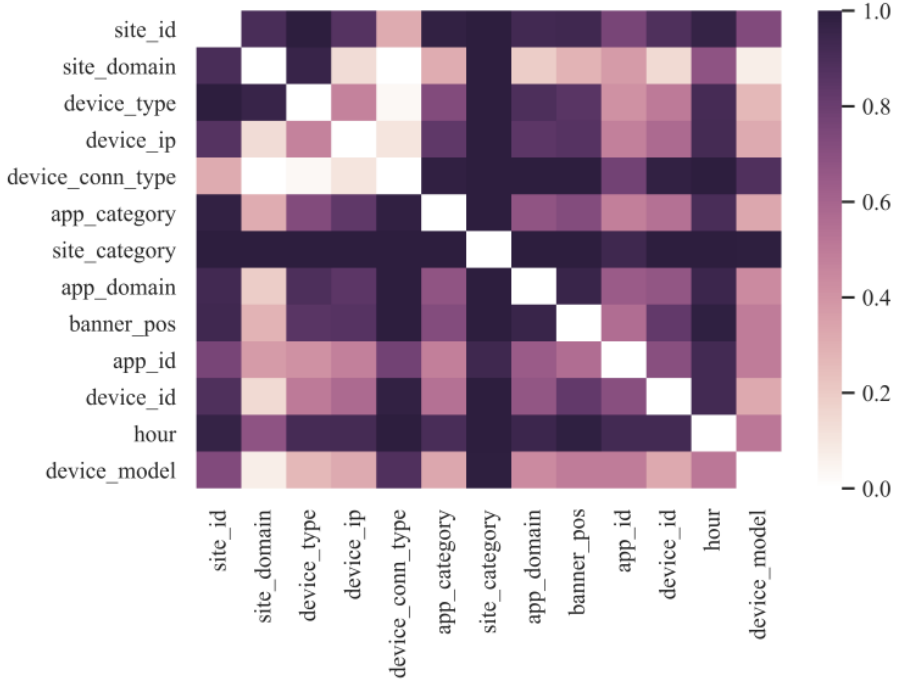
Fi-GNN: Modeling Feature Interactions via Graph Neural Networks for CTR Prediction. *CIKM,* 2019.

Fi-GNN

- Taking advantage of the strong representative power of graphs, Fi-GNN captures high-order feature interaction in an efficient way.
- Fi-GNN also provides good model explanations for CTR prediction.

Model Type	Model	Criteo			
		AUC	RI-AUC	Logloss	RI-Logloss
First-order	LR	0.7820	3.00%	0.4695	5.43%
Second-order	FM [23]	0.7836	2.80%	0.4700	5.55%
	AFM[34]	0.7938	1.54%	0.4584	2.94%
High-order	DeepCrossing [25]	0.8009	0.66%	0.4513	1.35%
	NFM [8]	0.7957	1.57%	0.4562	2.45%
	CrossNet [31]	0.7907	1.92%	0.4591	3.10%
	CIN [15]	0.8009	0.63%	0.4517	1.44%
	Fi-GNN (ours)	**0.8062**	0.00%	**0.4453**	0.00%

Performance comparison.



Heat map of attentional edge weights.

Fi-GNN: Modeling Feature Interactions via Graph Neural Networks for CTR Prediction. *CIKM,* 2019.

Automatic Feature Generation

- The related works on automatic feature generation can be roughly divided into two categories:
  - DNN-based methods
  - Search-based methods

DNN-based methods design specific neural architectures to express the interactions among different features.

- Implicit feature generation
- One-shot training course
- Lack of interpretable rules for feature interactions

Search-based methods focus on designing different search strategies that prune as much of the candidates to be evaluated as possible, while aiming to keep the most useful interactive features.

- Explicit feature generation
- Trial-and-error training manner
- Need lots of time and computing resource

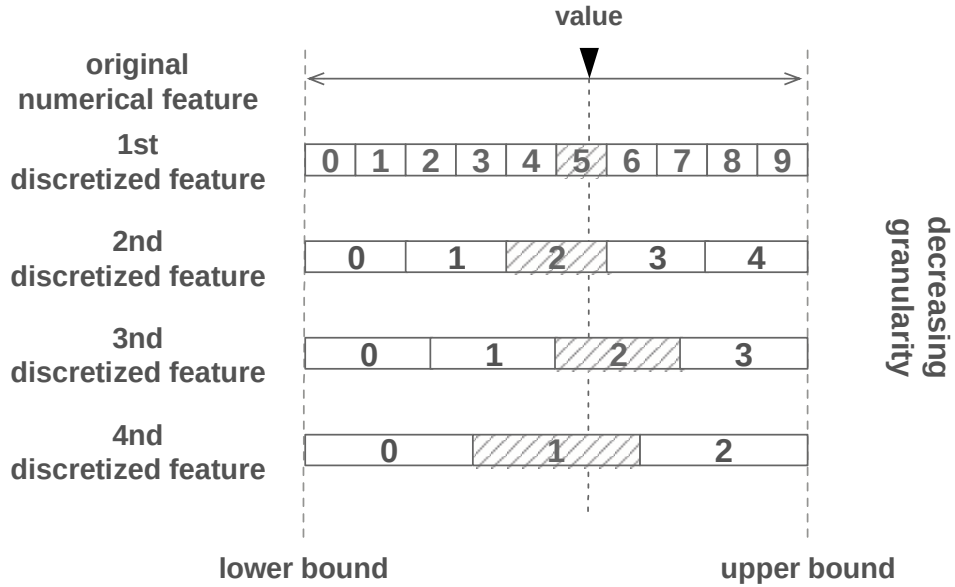
AutoCross

❑ AutoCross searches useful feature interactions in the high-order interactive feature space by incrementally constructing local optimal feature set

- Multi-granularity discretization
- Greedy & beam search
- Field-wise logistic regression
- Successive mini-batch gradient descent

Multi-granularity discretization:

- For automatic discretization, each numerical feature is discretized into several categorical features with different granularities.



An illustration of multi-granularity discretization.

AutoCross: Automatic Feature Crossing for Tabular Data in Real-World Applications. *KDD,* 2019.

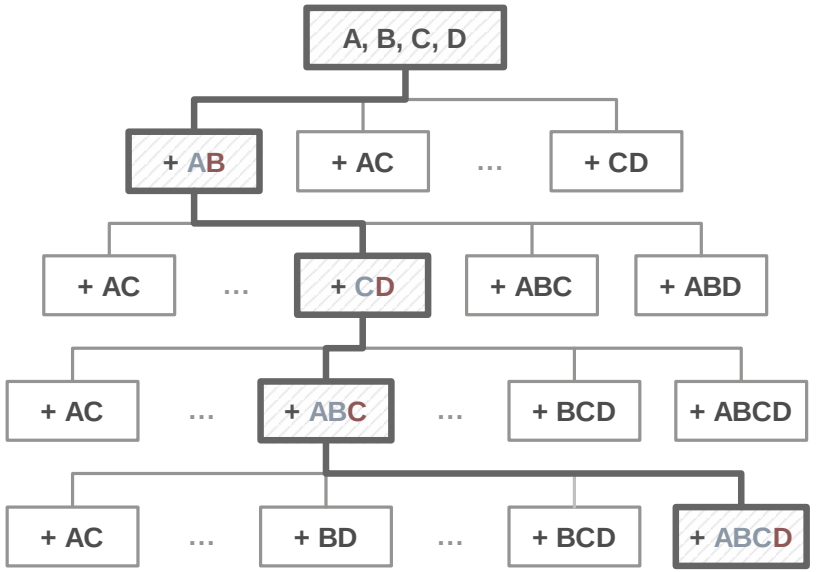
AutoCross

❑ AutoCross searches useful feature interactions in the high-order interactive feature space by incrementally constructing local optimal feature set

- Multi-granularity discretization
- Greedy & beam search
- Field-wise logistic regression
- Successive mini-batch gradient descent

Greedy & beam search:

- Tree-structured space with the original features as the root.
- The children are generated by added one pair-wise crossing to the parent.
- Only the most promising child will be expanded during the search



An illustration of the search space and beam search strategy.

AutoCross: Automatic Feature Crossing for Tabular Data in Real-World Applications. *KDD,* 2019.

AutoCross

❑ AutoCross searches useful feature interactions in the high-order interactive feature space by incrementally constructing local optimal feature set

- Multi-granularity discretization
- Greedy & beam search
- Field-wise logistic regression
- Successive mini-batch gradient descent

Field-wise logistic regression :

- For each node, the weights of the newly added interactive features are updated during training, while other weights are inherited from the parent and fixed.

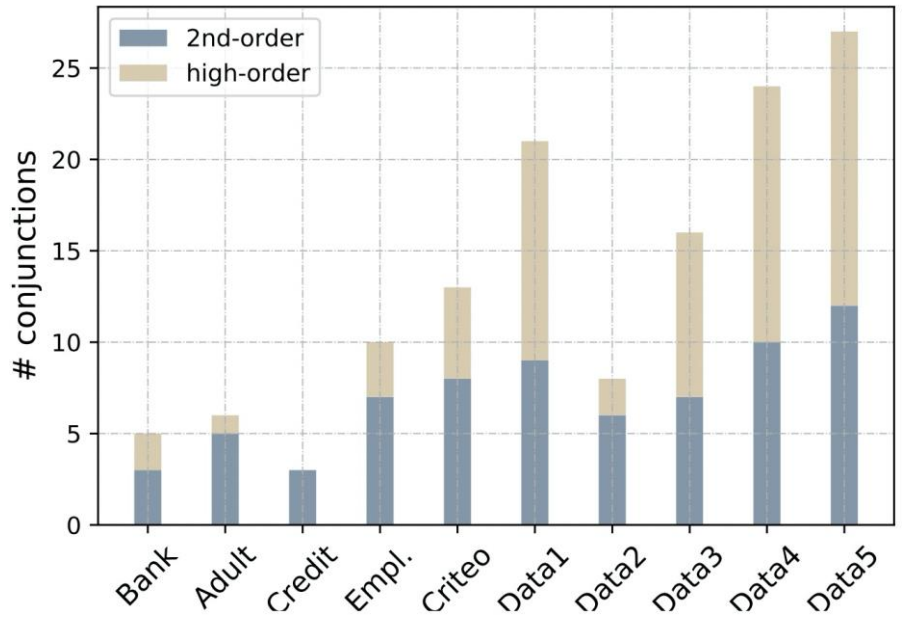
Successive mini-batch gradient descent :

- The data are split into several blocks, and gradually added into the training process along with narrowing the candidate interactive features.

AutoCross: Automatic Feature Crossing for Tabular Data in Real-World Applications. *KDD,* 2019.

AutoCross

- The advantages of AutoCross:
  - Explicit high-order feature generation
  - Fast inference
  - Interpretability



The number of second/high-order interactive features.

Benchmark Datasets					
Method	Bank	Adult	Credit	Employee	Criteo
AC+LR	0.00048	0.00048	0.00062	0.00073	0.00156
AC+W&D	0.01697	0.01493	0.00974	0.02807	0.02698
Deep	0.01413	0.01142	0.00726	0.02166	0.01941
xDeepFM	0.08828	0.05522	0.04466	0.06467	0.18985
Real-World Business Datasets					
Method	Data1	Data2	Data3	Data4	Data5
AC+LR	0.00367	0.00111	0.00185	0.00393	0.00279
AC+W&D	0.03537	0.01706	0.04042	0.02434	0.02582
Deep	0.02616	0.01348	0.03150	0.01414	0.01406
xDeepFM	0.32435	0.11415	0.40746	0.12467	0.13235

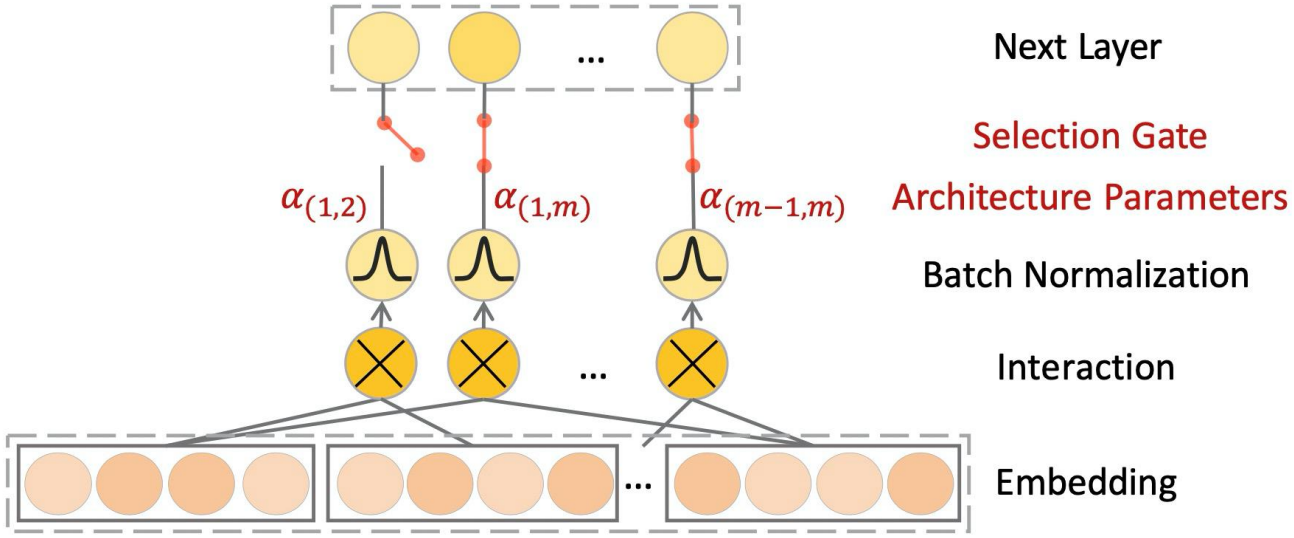
Inference latency comparison.

AutoCross: Automatic Feature Crossing for Tabular Data in Real-World Applications. *KDD,* 2019.

AutoFIS

❑ AutoFIS automatically identifies important feature interactions for Factorization Models (FM).

- *Search Stage*: Learn the relative importance of each feature interaction via architecture parameters within one full training process.
- *Re-train Stage*: Remove the unimportance interactions and re-train the resulting neural networks.



Overview of AutoFIS.

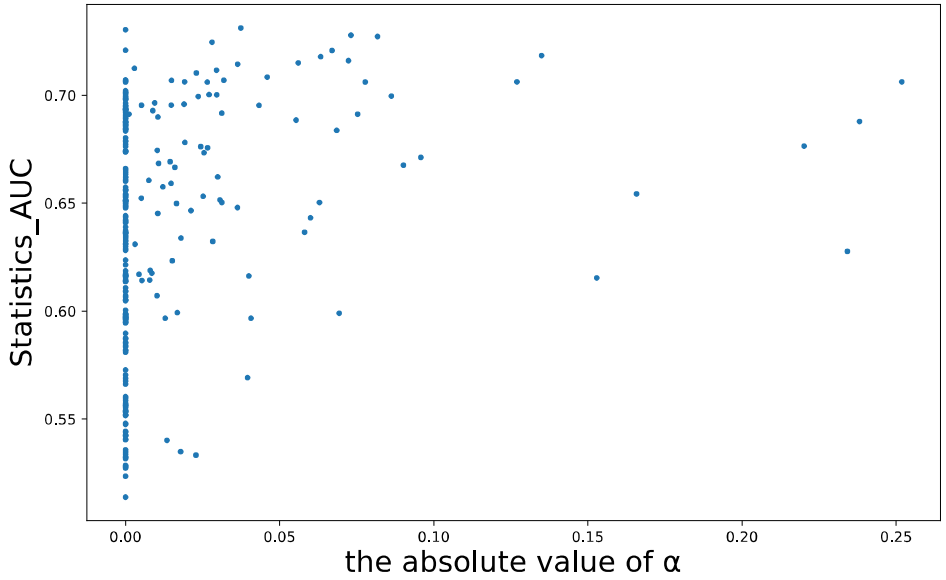
AutoFIS: Automatic Feature Interaction Selection in Factorization Models for Click-Through Rate Prediction. *KDD,* 2020.

AutoFIS

- Experiments on large-scale datasets demonstrate that AutoFIS can improve various FM based models in CTR prediction tasks.

Model	AUC	log loss	top	ReI. Impr
FM	0.8880	0.08881	100%	0
FwFM	0.8897	0.08826	100%	0.19%
AFM	0.8915	0.08772	100%	0.39%
FFM	0.8921	0.08816	100%	0.46%
DeepFM	0.8948	0.08735	100%	0.77%
AutoFM(2nd)	0.8944*	0.08665*	37%	0.72%
AutoDeepFM(2nd)	**0.8979***	**0.08560***	**15%**	1.11%

Performance comparison.



Correlations between the architecture parameters α and AUC.

AutoFIS: Automatic Feature Interaction Selection in Factorization Models for Click-Through Rate Prediction. *KDD,* 2020.

FIVES

- To possess both feature interpretability and search efficiency, the proposed method FIVES formulates the task of interactive feature generation as searching for edges on the defined feature graph.

1. Search Strategy

PROPOSITION 1. *Let X_1, X_2 and Y be Bernoulli random variables with a joint conditional probability mass function, $p_{x_1,x_2|y} := \mathbb{P}(X_1 = x_1; X_2 = x_2 \mid Y = y)$ such that $x_1, x_2, y \in \{0, 1\}$. Suppose further that mutual information between X_i and Y satisfies $\mathcal{I}(X_i; Y) < L$ where $i \in \{1, 2\}$ and L is a non-negative constant. If X_1 and X_2 are weakly correlated given $y \in \{0, 1\}$, that is, $\left|\frac{Cov(X_1,X_2|Y=y)}{\sigma_{X_1|Y=y}\sigma_{X_2|Y=y}}\right| \leq \rho$, we have*

$$\mathcal{I}(X_1X_2; Y) < 2L + \log(2\rho^2 + 1). \quad (1)$$

Theoretical support for the search strategy.

- This proposition states that informative interactive features unlikely come from the uninformative lower-order ones.
- The theory motivates the bottom-up search strategy in FIVES: Searching for a group of informative k-order features from the interactions between original features and the group of $(k-1)$-order features.

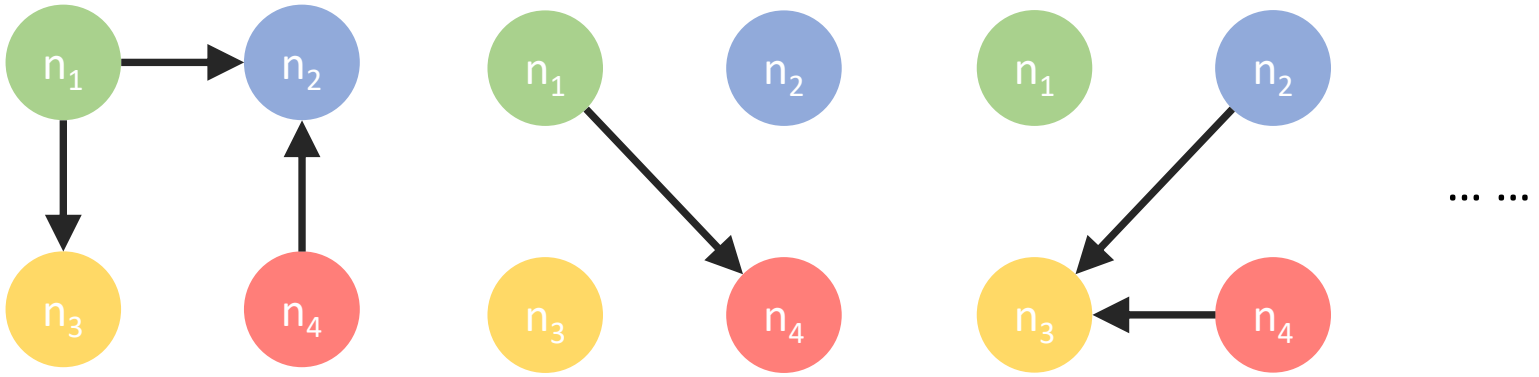
FIVES: Feature Interaction Via Edge Search for Large-Scale Tabular Data. *KDD,* 2021.

FIVES

❑ To possess both feature interpretability and search efficiency, the proposed method FIVES formulates the task of interactive feature generation as searching for edges on the defined feature graph.

2. Feature Graph

- To instantiate the proposed search strategy, the original features are conceptually regarded as a *feature graph* and their interactions are modeled by a designed GNN.
- Each node n_i corresponds to a feature f_i. Each edge $e_{i,j}$ indicates an interaction between n_i and n_j.



The constructed feature graph to represent high-order feature interactions.

FIVES: Feature Interaction Via Edge Search for Large-Scale Tabular Data. *KDD,* 2021.

FIVES

- To possess both feature interpretability and search efficiency, the proposed method FIVES formulates the task of interactive feature generation as searching for edges on the defined feature graph.

2. Feature Graph

- The feature graph consists of K subgraphs to represent high-order interactive feature. Each subgraph indicates a layer-wise interaction between features, represented by an adjacency matrix $A^{(k)} \in \{0,1\}^{m \times m}$. The graph convolutional operator for aggregation are defined as:

$$n_i^{(k)} = p_i^{(k)} \odot n_i^{(k-1)}, \qquad \text{where } p_i^{(k)} = \text{MEAN}_{j|A_{i,j}^{(k)}=1} \left\{ W_j n_j^{(0)} \right\} \tag{1}$$

- The node representation at k-th layer corresponds to the generated features:

$$n_i^{(k)} = \text{MEAN}_{j|A_{i,j}^{(k)}=1} \left\{ W_j n_j^{(0)} \right\} \odot n_i^{(k-1)} \approx \text{MEAN}_{(c_1,\ldots,c_k)|A_{i,c_j}^{(j)}=1, j=1,\ldots,k} \{ f_{c_1} \otimes \ldots \otimes f_{c_k} \otimes f_i \} \tag{2}$$

FIVES: Feature Interaction Via Edge Search for Large-Scale Tabular Data. *KDD*, 2021.

FIVES

- To possess both feature interpretability and search efficiency, the proposed method FIVES formulates the task of interactive feature generation as searching for edges on the defined feature graph.

3. Differentiable Edge Search

- The task of generating useful interactive features is equivalent to learning an optimal adjacency tensor A, so-called *edge search*.

$$\begin{aligned} & \min_{\boldsymbol{A}} \mathcal{L}(\mathcal{D}_{\text{val}}|A, \Theta(A)) \\ \text{s.t.} \quad & \Theta(A) = \arg\min_{\Theta} \mathcal{L}\left(\mathcal{D}_{\text{train}}|A, \Theta\right) \end{aligned} \tag{3}$$

- To make the optimization more efficient, A is regraded as Bernoulli random variables parameterized by $H \in [0,1]^{K\times m\times m}$, and a soft $A^{(k)}$ is allowed to be used for propagation at the k-th layer.

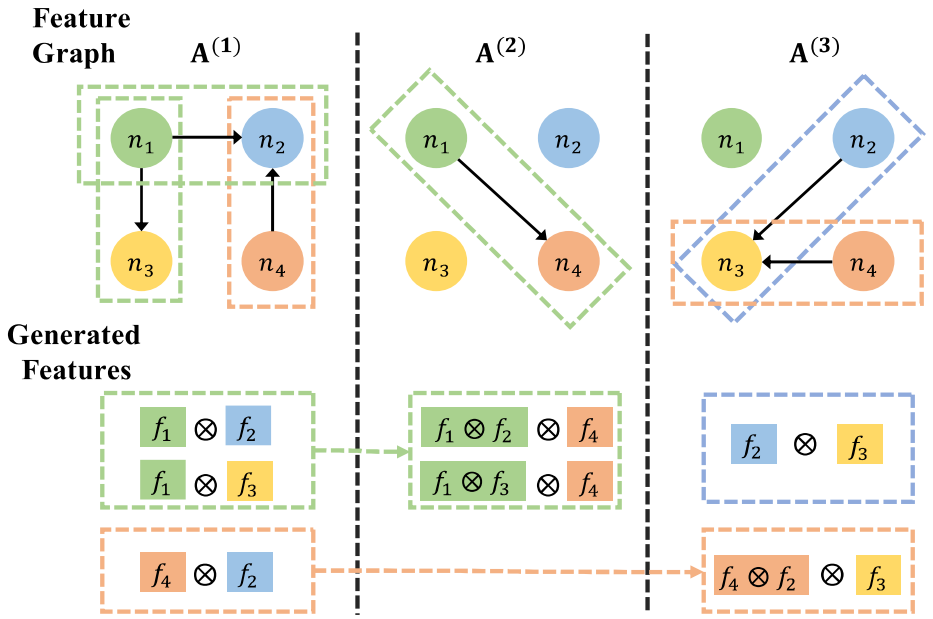
FIVES: Feature Interaction Via Edge Search for Large-Scale Tabular Data. *KDD,* 2021.

FIVES

❑ To possess both feature interpretability and search efficiency, the proposed method FIVES formulates the task of interactive feature generation as searching for edges on the defined feature graph.

4. Interactive Feature Derivation

- The learned adjacency tensor can explicitly indicate which interactive features are useful.
- One can inductively derive useful high-order interactive features by specify layer-wise thresholds for binarizing the learned A.
- FIVES serves as a feature generator for lightweight models to meet the requirement of inference speed.

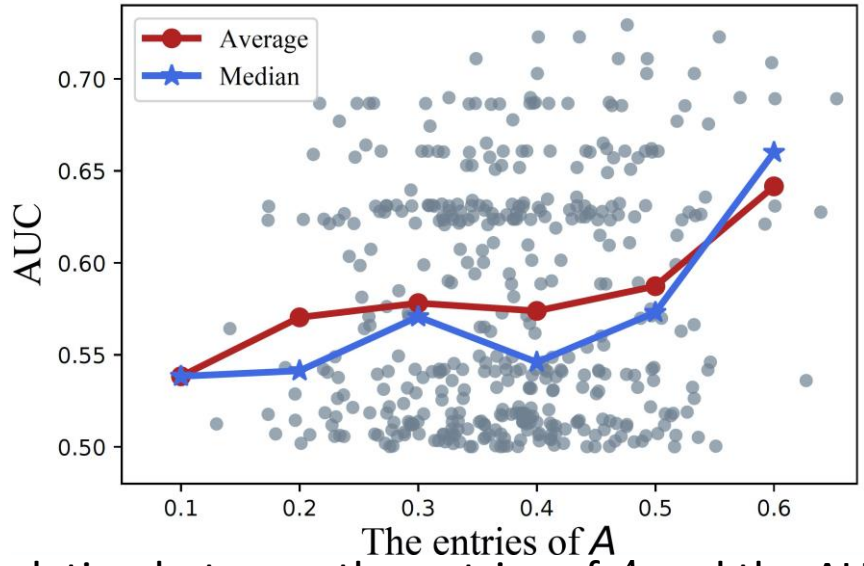


An example of interactive feature derivation.

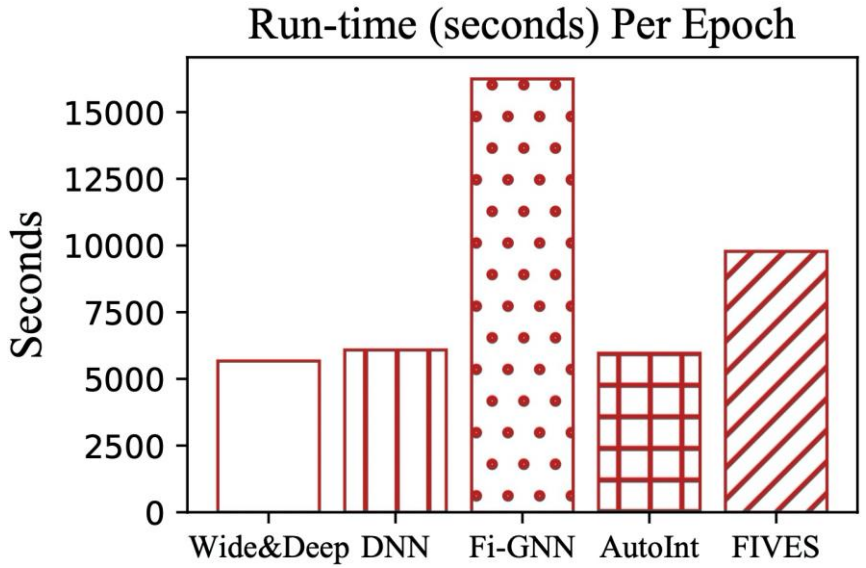
FIVES: Feature Interaction Via Edge Search for Large-Scale Tabular Data. *KDD*, 2021.

FIVES

- Extensive experiments on five public datasets and two business datasets confirm that FIVES can generate useful interactive features.
  - FIVES as a predictive model for downstream tasks, such as CTR prediction
  - FIVES as the feature generator for lightweight models to meet the requirement of inference speed



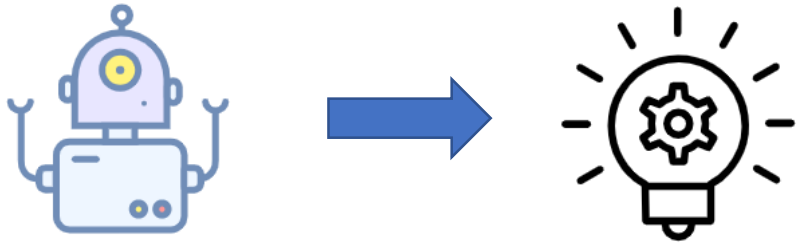
Correlation between the entries of A and the AUC of the corresponding indicated feature.



Efficiency comparisons.

FIVES: Feature Interaction Via Edge Search for Large-Scale Tabular Data. *KDD*, 2021.

Takeaways



AutoFeature Model

Useful Interactive Features

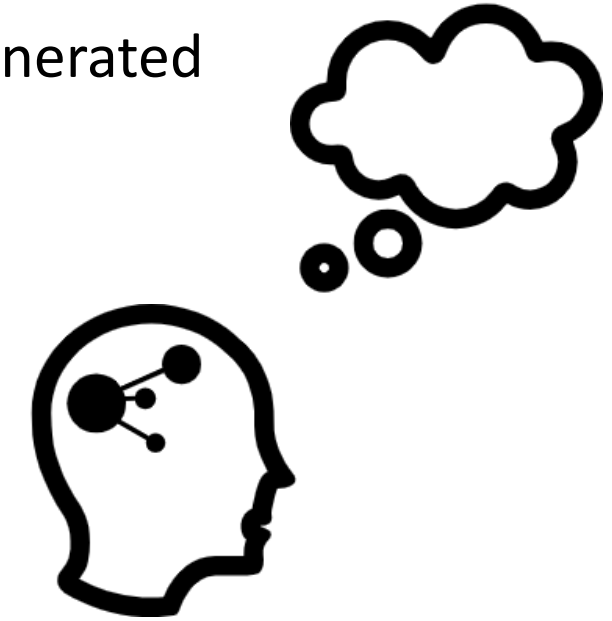
✓ Feature Interpretability
✓ Search Efficiency

DNN-based methods
- Implicit feature generation
- One-shot training course
- Lack of interpretable rules for feature interactions

Search-based methods
- Explicit feature generation
- Trial-and-error training manner
- Need lots of time and computing resource

Future Directions

- How to introduce human experience as prior knowledge for AutoFeature?
- Causal features or spurious correlations?
- How to balance the trade-off between the usefulness of generated features and the completeness of them?



References of AutoFeature

[1] AutoInt: Automatic Feature Interaction Learning via Self-Attentive Neural Networks. In CIKM 2019.

[2] Fi-GNN: Modeling Feature Interactions via Graph Neural Networks for CTR Prediction. In CIKM 2019.

[3] AutoCross: Automatic Feature Crossing for Tabular Data in Real-World Applications. In KDD 2019.

[4] AutoFIS: Automatic Feature Interaction Selection in Factorization Models for Click-Through Rate Prediction. In KDD 2020.

[5] FIVES: Feature Interaction Via Edge Search for Large-Scale Tabular Data. In KDD 2021.

VolcanoML: End-to-End AutoML via Scalable Search Space Decomposition

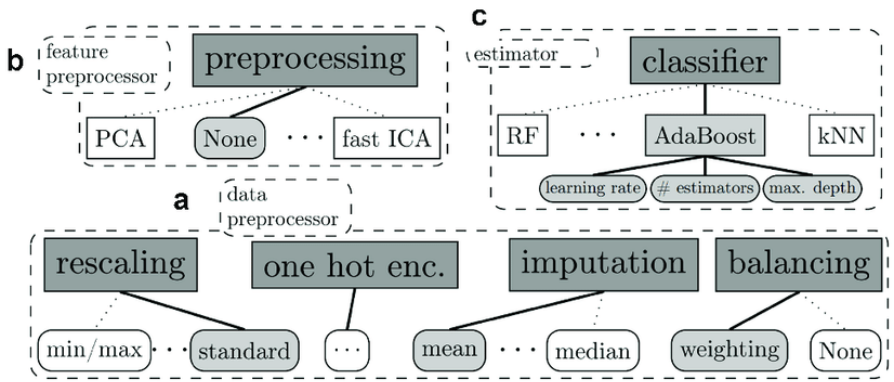
Two Complications of AutoML going E2E

AutoML

$$\alpha^* = \underset{\alpha}{\text{argmax}}\, f(D', \theta_\alpha^*)$$
$$\text{s.t, } \theta_\alpha^* = \underset{\theta}{\text{argmax}} P(D|\theta)P(\theta|\alpha)$$

Two Complications

1. α is not a homogenous space, it is rather heterogenous



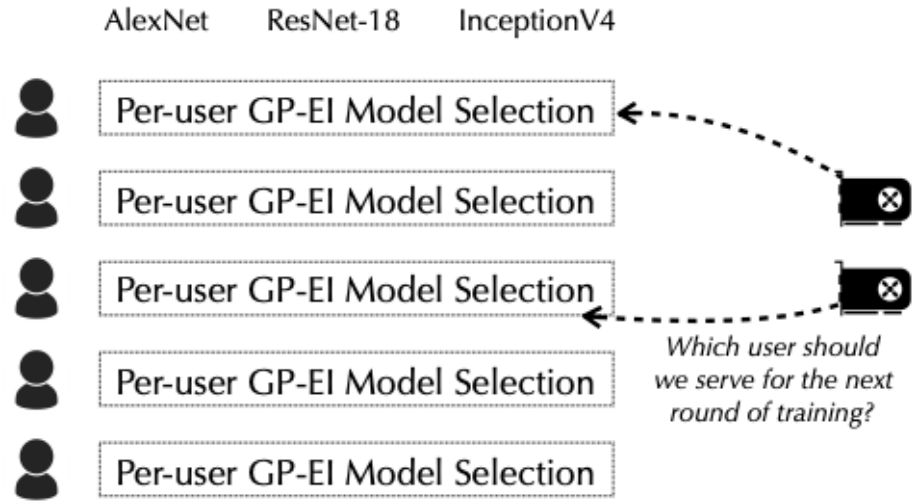
(auto-sklearn)

$\alpha \in Feature \times HP \times Model$

Personal perspectives, from our past experiences

- **VolcanoML: Speeding up End-to-End AutoML via Scalable Search Space Decomposition**. ***VLDB*** 2021.
- **AutoML from Service Provider's Perspective: Multi-device, Multi-tenant Model Selection with GP-EI**. ***AISTATS*** 2019.
- **Ease.ml: Towards Multi-tenant Resource Sharing for Machine Learning Workloads**. ***VLDB*** 2018.

2. From single-tenant to multi-tenant scenarios

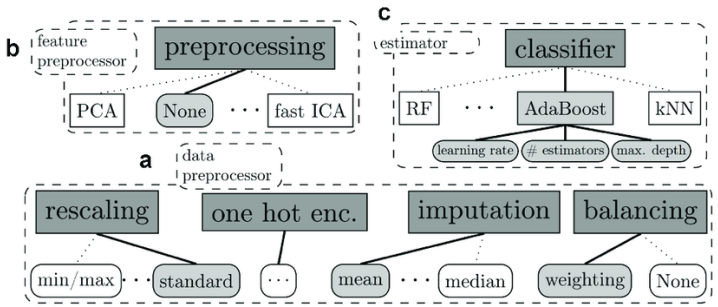


Disclaimer

This segment of the tutorial is more opinioned and closer to our own experience than previous segments

It is less about how much we know about these two problems, but more about discussing some observations and preliminary explorations to show you what we don’t know and a “cry for help”.

Heterogenous Search Space



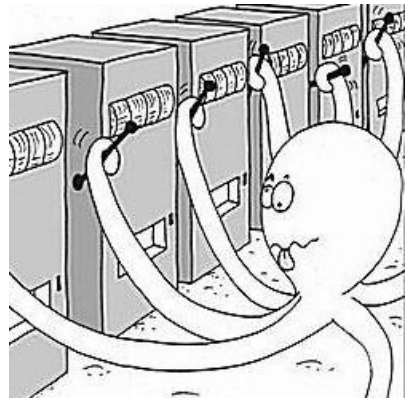
- ***Feature* × *HP* × *Model***
- ***A strong baseline***: Treat the heterogenous space as a single joint space.
  - Model it with a single Bayesian optimization problem, a single genetic algorithm, or a single hyperband problem
  - Good? Very powerful approach, yet simple.
  - Could be improved?
    - "The curse of dimensionality": often it is not easy to scale up when the dimensionality of the space is high.
    - Heterogeneity in algorithm: Different subspaces might benefit from different algorithms.
- ***Can we do better?***

Heterogenous Search Space

- ***Different ways to conduct search. Let's take for example the space*** $\alpha \in \boldsymbol{X} \times \boldsymbol{Y}$
- ***Strategy 1. Joint***
  - Treating the space $\boldsymbol{X} \times \boldsymbol{Y}$ as a single search space
  - (If you are doing BO) Create a surrogate model M to approximate $f(\alpha)$
  - Use M to select $\overline{\boldsymbol{\alpha}}$
  - Evaluate $f(\overline{\boldsymbol{\alpha}})$ and update the surrogate model M
- One can implement such a strategy using methods beyond BO.

Heterogenous Search Space

- ***Different ways to conduct search. Let's take for example the space*** $\boldsymbol{\alpha \in X \times Y}$
- ***Strategy 2. Conditioning***
  - Idea: decompose $\boldsymbol{X \times Y}$ into multiple subspaces, e.g., one for each value of $\boldsymbol{X}$
  - $\min_{x,y} f(x,y) \Rightarrow \min_{x \in X} \min_{y} g_x(y)$
  - Then treating each $x \in X$ as a subproblem $\min_{y} g_x(y)$
  - Can be modeled as a Multi-armed bandit problem – each arm corresponds to a possible value of $x \in X$, playing an arm means optimizing $\min_{y} g_x(y)$ one step



- For example, think about X as Algorithm and Y as Feature – For each Algorithm, search for the best feature, and pick the best Algorithm

Heterogenous Search Space

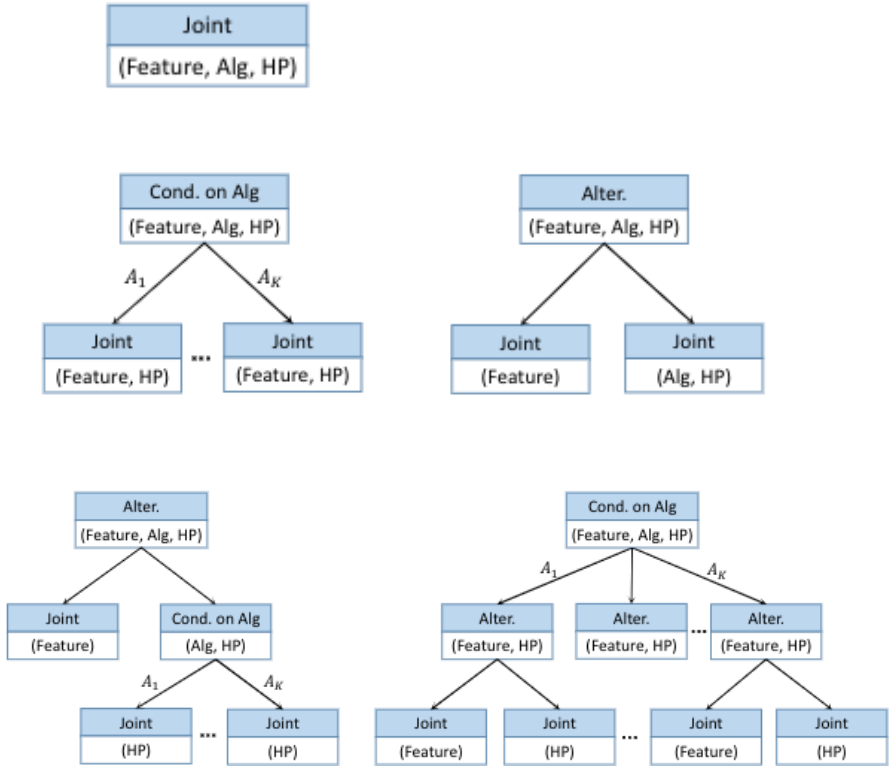
- ***Different ways to conduct search. Let's take for example the space*** $\boldsymbol{\alpha \in X \times Y}$
- ***Strategy 3. Alternating***
  - Idea: decompose $\boldsymbol{X \times Y}$ into two subspaces, $\boldsymbol{X}$ and $\boldsymbol{Y}$
  - Solve two problems alternatively:
    - $\min_x g_{\bar{y}}(x)$, where $\bar{y}$ is the current best value for subspace Y
    - $\min_y g_{\bar{x}}(y)$, where $\bar{x}$ is the current best value for subspace X
  - Each subproblem can be solved either jointly or via some conditioning strategy
  - At each iteration, pick the subproblem with the largest expected improvement
- For example, think about X as Feature and Y as HP – Alternating the process of search for feature and search for HP

Heterogenous Search Space

- ***Different ways to conduct search***
- ***Strategy 1. Joint***
  - ***Pros:*** Simple, works well when dimensionality is low
  - ***Cons:*** Might suffer when the dimensionality is high
- ***Strategy 2. Conditioning***
  - ***Pros:*** Effective when some dimension is categorical variable with small cardinality
  - ***Cons:*** Might not be applicable to other scenarios.
- ***Strategy 3. Alternating***
  - ***Pros:*** Very effective in reducing dimensions
  - ***Cons:*** Assuming conditional independence of two subspaces

Heterogenous Search Space

- A single search space can be decomposed in different ways.



Dataset	Plan 1	Plan 2	Plan 3	Plan 4	Plan 5
puma8NH	0.8275	**0.8312**	0.8271	0.8280	0.8303
kin8nm	0.8808	0.8886	0.8886	0.8654	**0.8910**
cpu_cmall	0.9122	0.9126	0.9126	0.9027	**0.9127**
puma32H	0.8849	0.8864	0.8848	0.8835	**0.8894**
cpu_act	0.9303	**0.9315**	0.9305	0.9302	0.9309
bank32nh	0.7896	0.7889	0.7838	0.7891	**0.7957**
mc1	0.8796	0.8904	0.8722	0.8721	**0.8975**
delta_elevators	0.8763	0.8760	0.8779	0.8766	**0.8790**
jm1	0.6718	**0.6721**	0.6581	0.6473	0.6692
pendigits	0.9932	0.9936	0.9929	**0.9945**	0.9937
delta_ailerons	0.9235	0.9240	0.9242	0.9225	**0.9259**
wind	0.8587	0.8589	0.8566	0.8583	**0.8593**
satimage	0.8961	0.8954	0.8965	0.8946	**0.8981**
optdigits	**0.9889**	**0.9889**	0.9883	**0.9889**	**0.9889**
phoneme	0.8799	0.8832	0.8808	0.8791	**0.8866**
spambase	0.9401	**0.9406**	0.9379	0.9387	0.9386
abalone	**0.6688**	0.6679	0.6618	0.6614	0.6680
mammography	0.8740	0.8783	0.8577	0.8755	**0.8787**
waveform	0.8948	**0.8961**	0.8900	0.8835	0.8952
pollen	0.4934	**0.5013**	0.5012	**0.5013**	**0.5013**
Average Rank	3.28	2.33	3.80	3.98	**1.63**

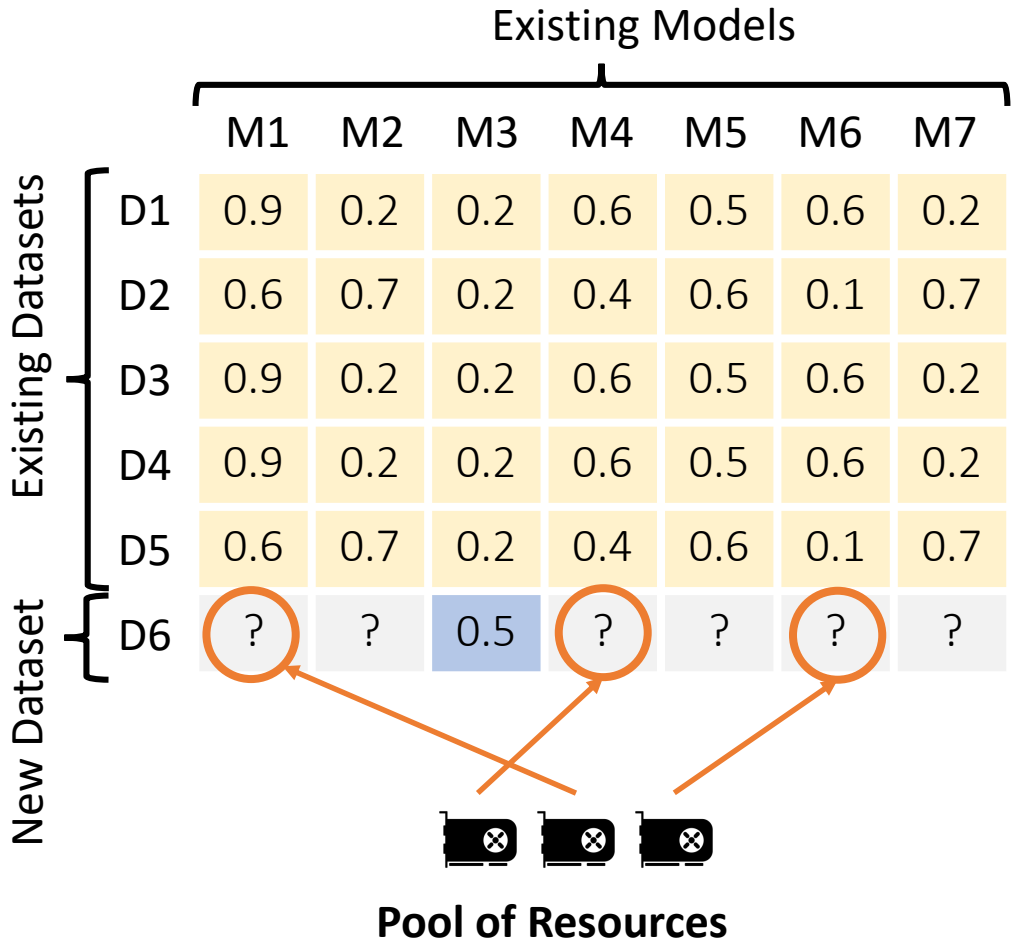
Different plans have different performance

Potentially, can learn to decompose given a target workload

Heterogenous Search Space

- ***Moving Forward***
  - Build up a suite of different building blocks – what is the unified framework to talk about different search algorithms?
  - How to automatically construct search space decomposition?
  - How to automatically conduct building block selection? AutoML for AutoML?

AutoML: From Single-tenant to Multi-tenant

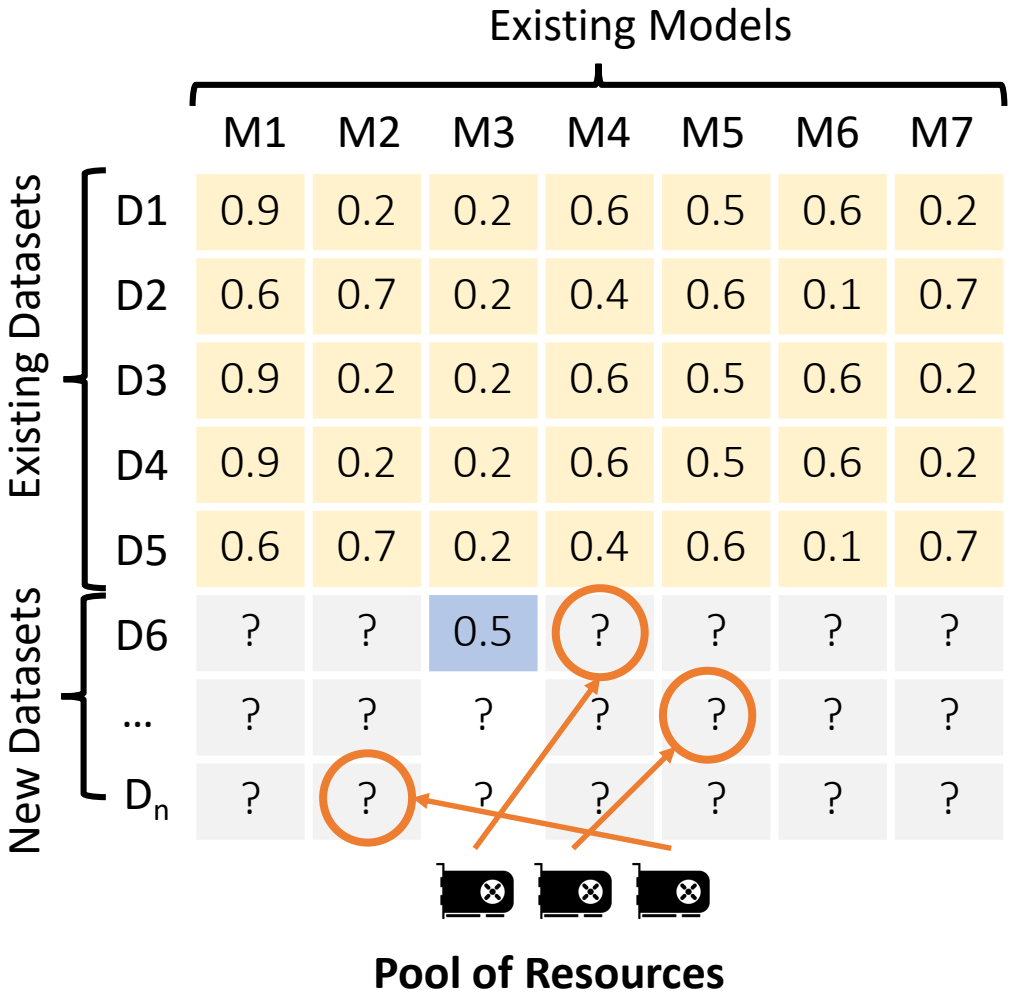


Single-tenant Scenario: One target dataset

What if multiple users running their own AutoML workload over a shared infrastructure?

Interesting problem especially when AutoML as a service becomes more and more popular.

AutoML: From Single-tenant to Multi-tenant

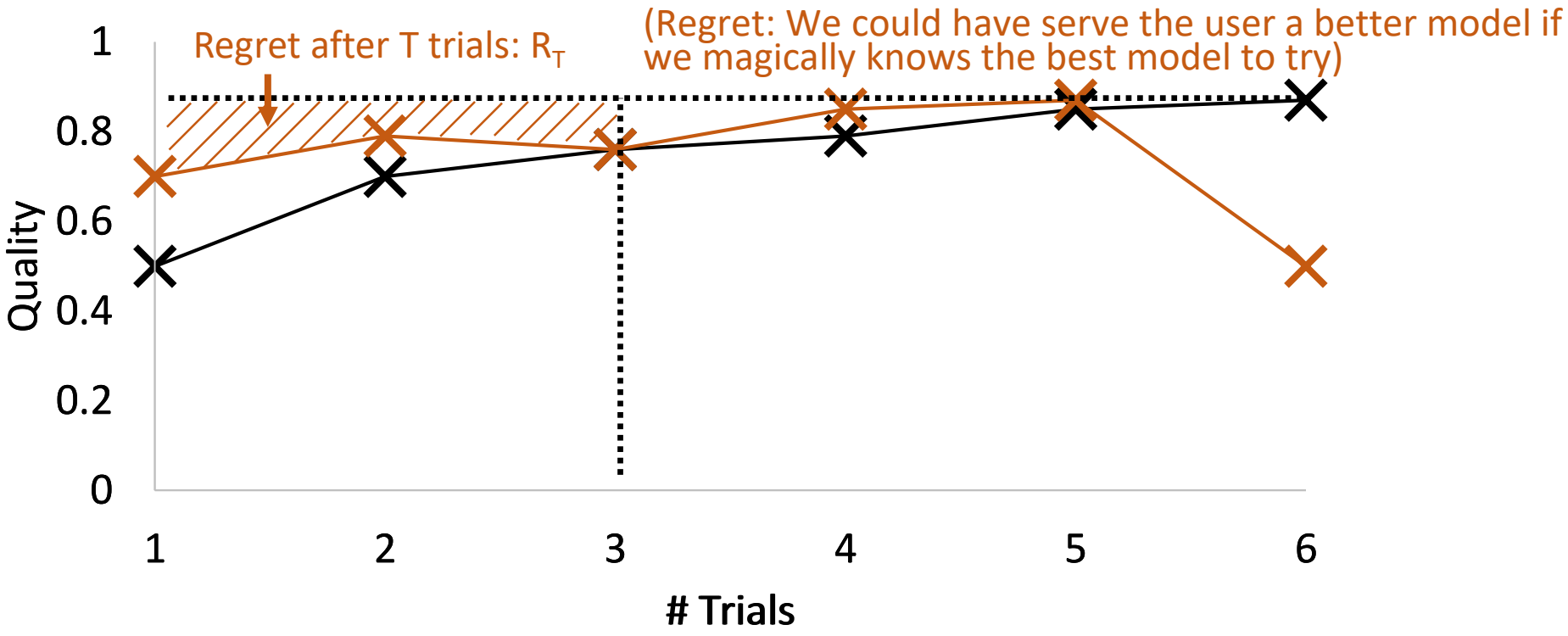


How to balance resource allocations to different users?

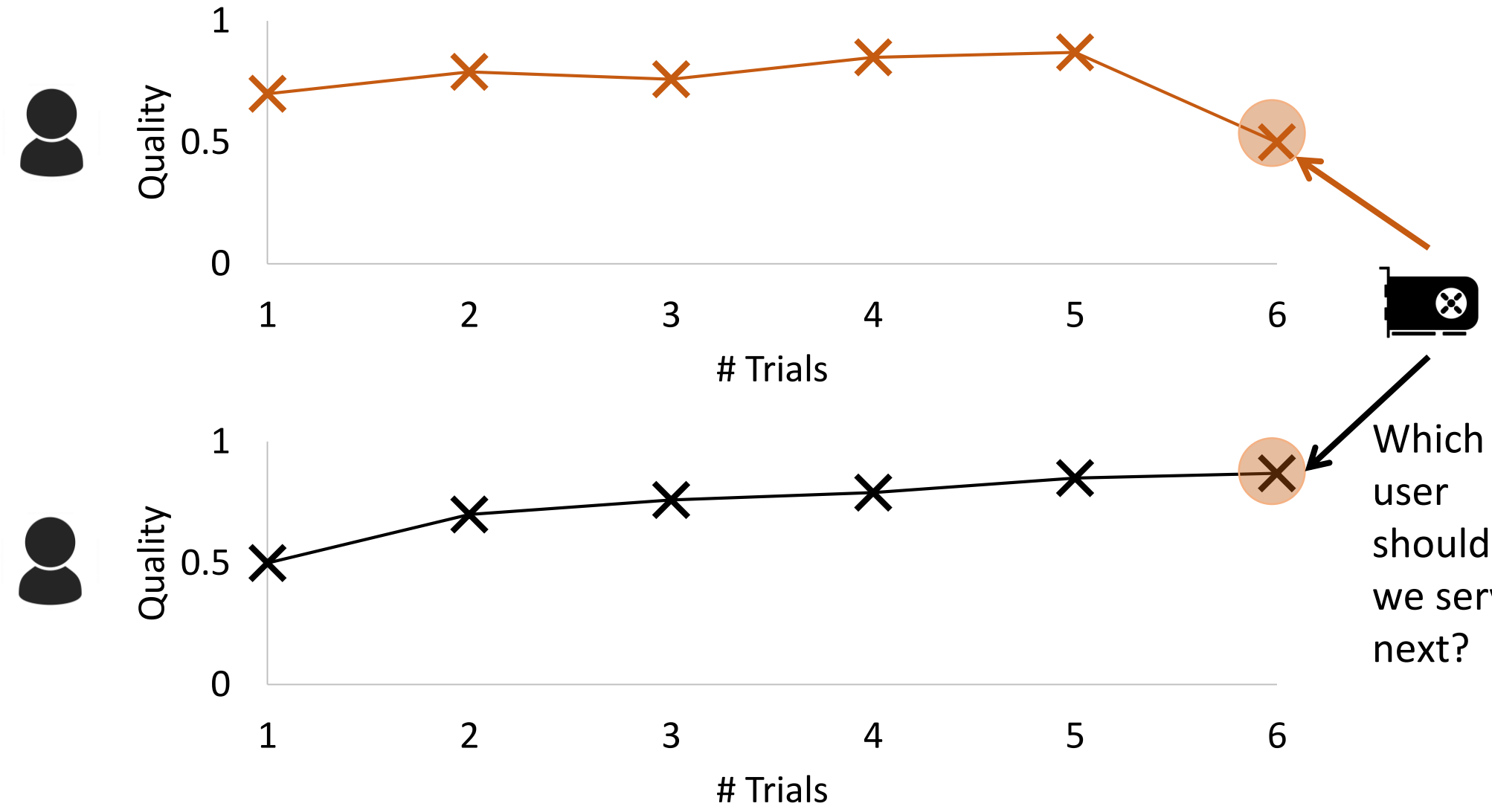
AutoML: From Single-tenant to Multi-tenant

- ***Regret: A Single User's Unhappiness***

Decisions	Quality
M1	0.5
M2	0.7
M3	0.76
M4	0.79
M5	0.85
M6	0.87



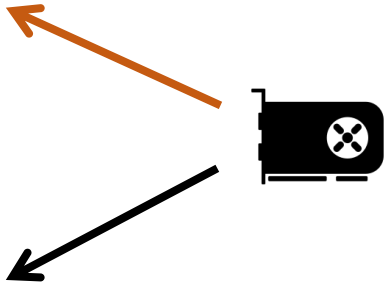
AutoML: From Single-tenant to Multi-tenant



AutoML: From Single-tenant to Multi-tenant

User 1: [0.99] [0.99]

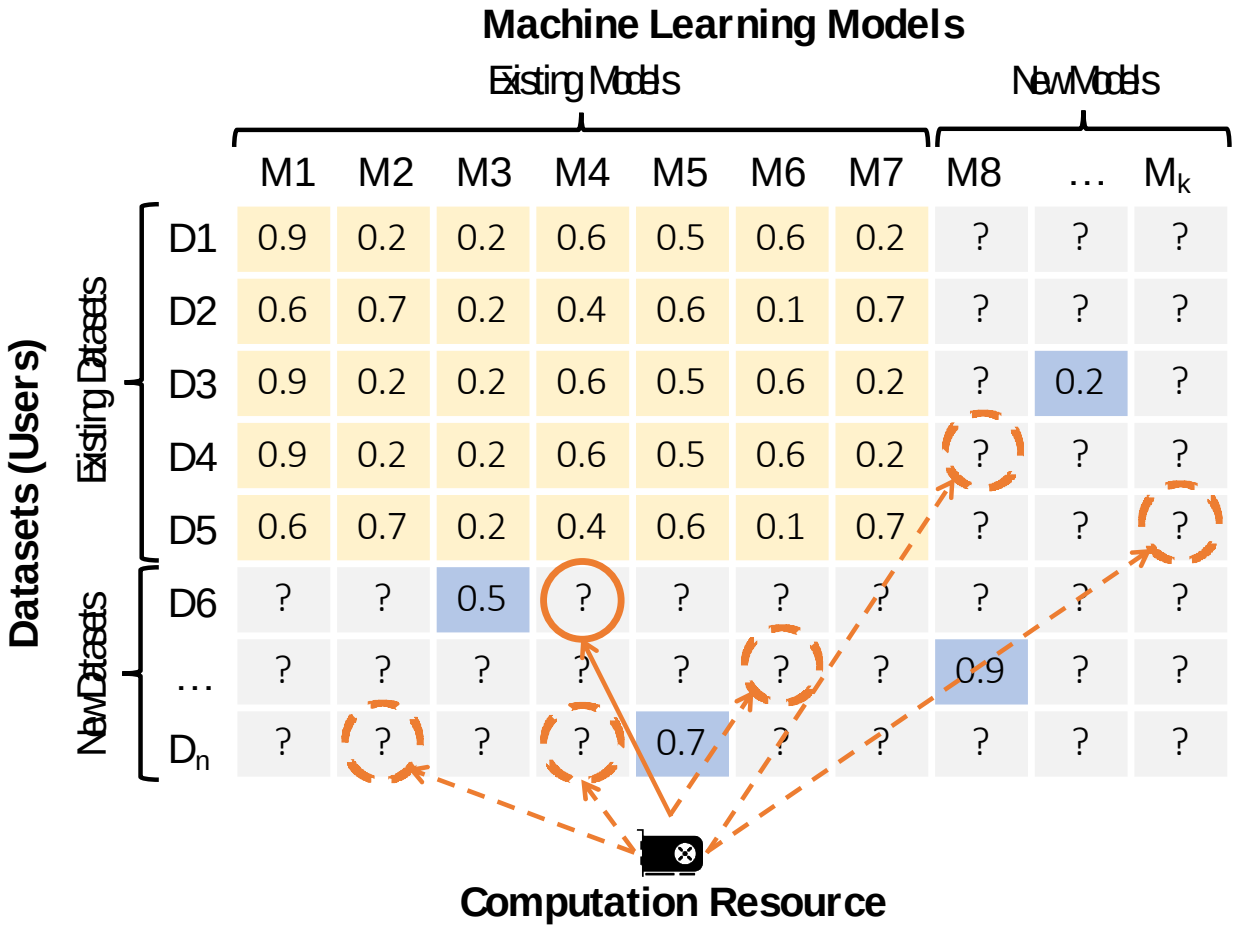
User 2: [0.10] [0.35]



Extreme Case: User 1 is not worth serving any more

How about more general case?

AutoML: From Single-tenant to Multi-tenant

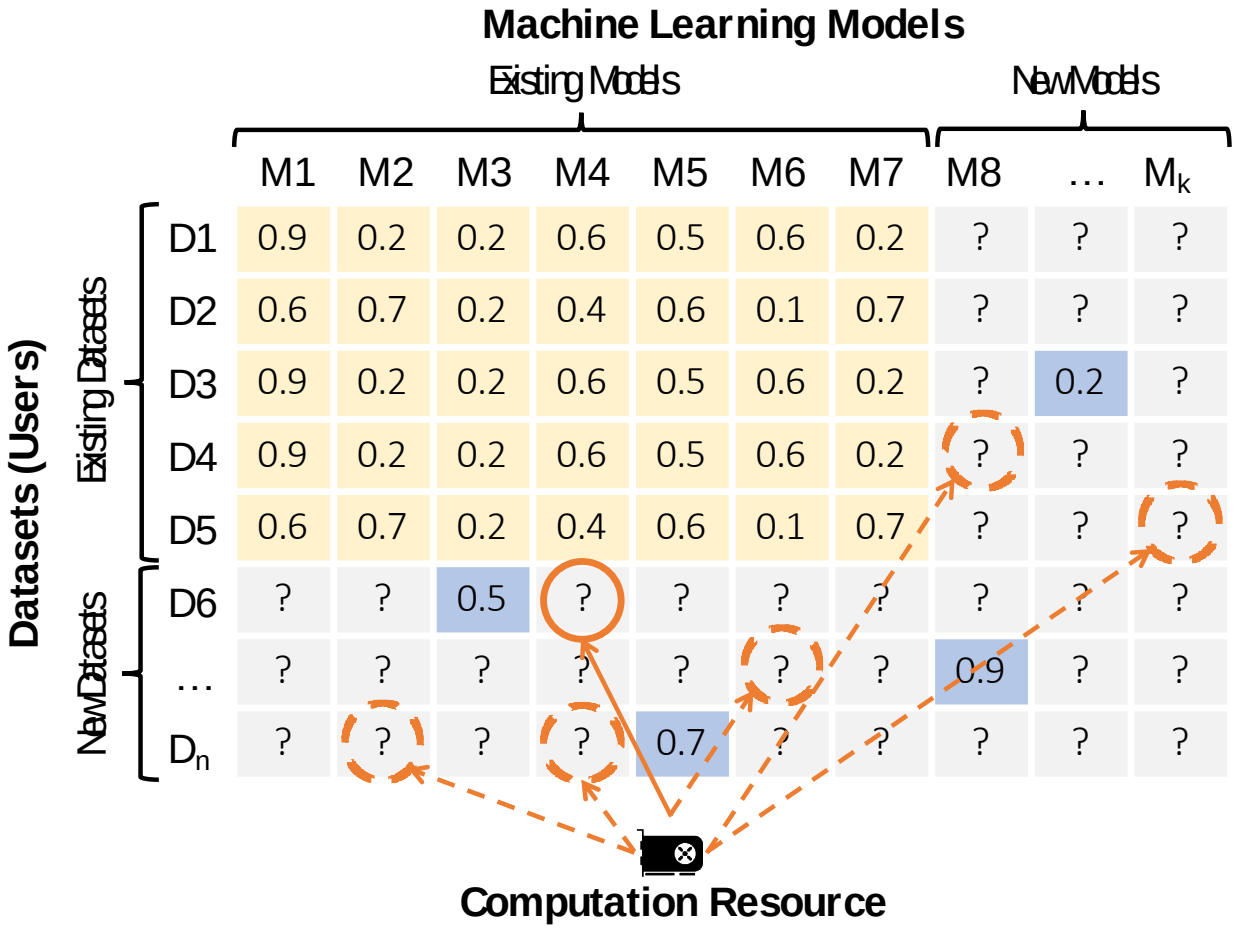


01 *Each user runs their own GP-EI model selection*

02 *Serve the user with highest expected improvement.*

Informal Theorem. *If the performance of all models is a linear combination of a finite, shared set of hidden Gaussian variables, the global regret converges to 0 with rate O(1 / runtime).*

AutoML: From Single-tenant to Multi-tenant



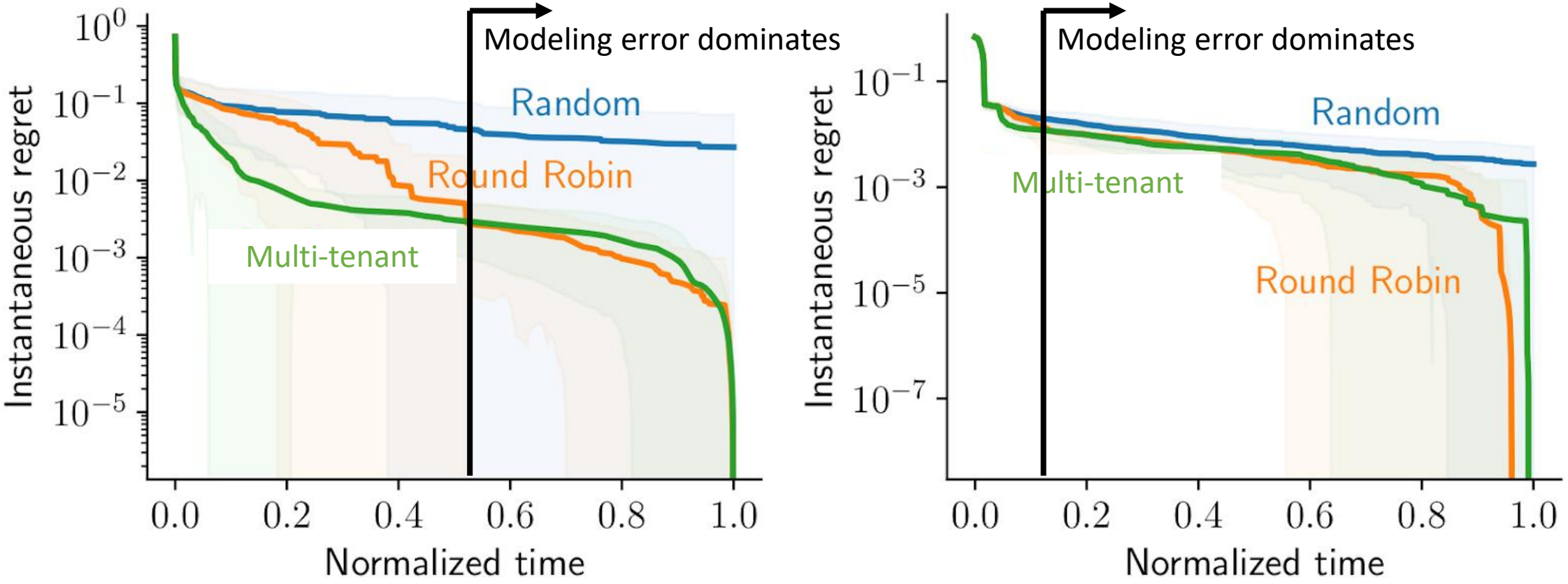
01 *Each user runs their own GP-UCB algorithm*

02 *Serve the user with a factor that is very similar to expected improvement (directly comparing each user's UCB does not work, for obvious reason)*

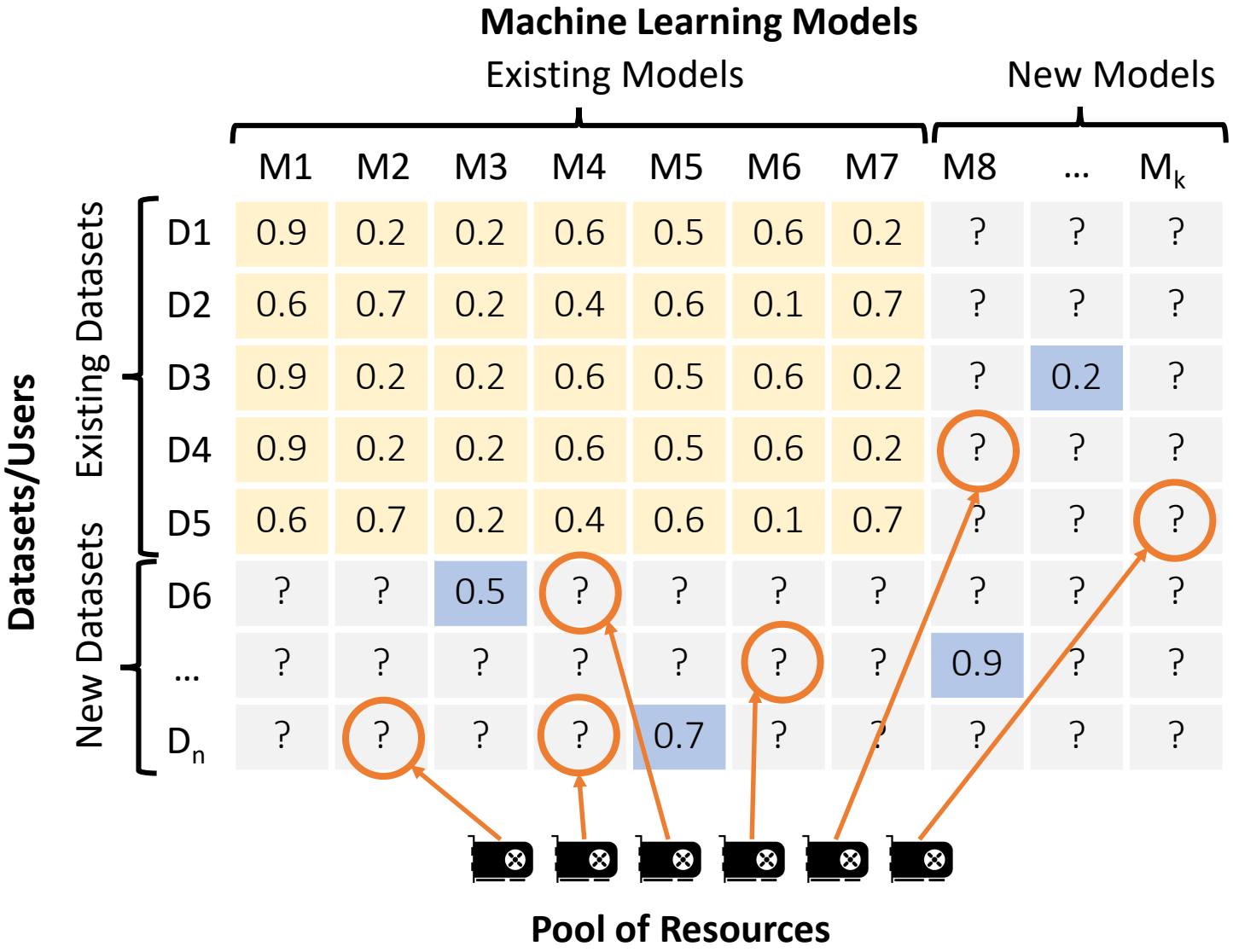
In this case, the total regret is bounded (up to some constant) by

$$n^{3/2}\sqrt{\beta^* T \sum_{i=1}^{n} \log(|T(i)|)} \leq \underbrace{n^{3/2}\sqrt{\beta^* T \log\left(\frac{T}{n}\right)}}_{\text{the regret for RR, see (1)}}.$$

AutoML: From Single-tenant to Multi-tenant



AutoML: From Single-tenant to Multi-tenant



Need some special care on the diversity: don't put all GPUs on a single user.

Theorem. *Near linear speed up with respect to the number of devices when # devices << # users.*

AutoML: From Single-tenant to Multi-tenant

- ***Moving Forward***
  - In my opinion, it is exciting future direction to try to understand resource allocation and scheduling for AutoML workloads
  - What's the unified way to talk about and think about different AutoML workloads, e.g., those we have been talking about over the last two hours
  - Fairness? Efficiency? How should we aggregate unhappiness from multiple users?

Two Complications of AutoML going E2E

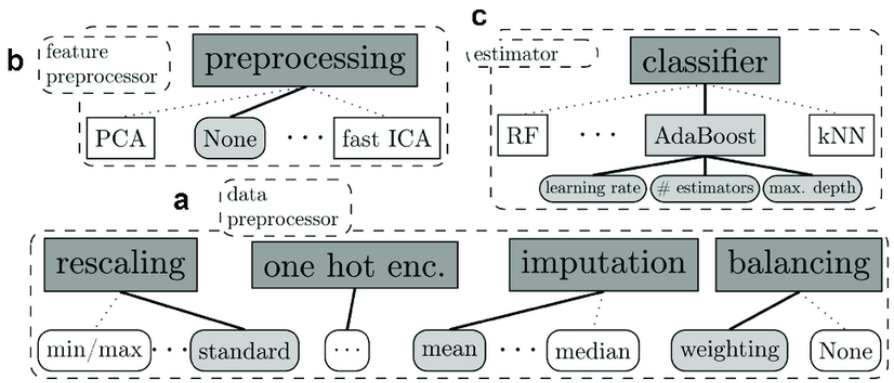
AutoML

$$\alpha^* = \underset{\alpha}{\text{argmax}}\, f(D', \theta_\alpha^*)$$
$$\text{s.t, } \theta_\alpha^* = \underset{\theta}{\text{argmax}} P(D|\theta)P(\theta|\alpha)$$

A lot of challenges and exciting opportunities when bring AutoML to and end-to-end production scenario!

Two Complications

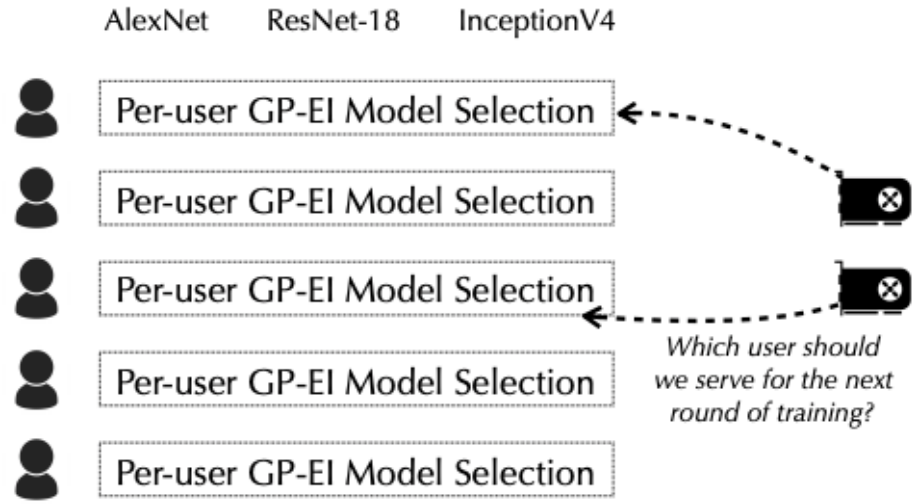
1. α is not a homogenous space, it is rather heterogenous



(auto-sklearn)

$$\alpha \in Feature \times HP \times Model$$

2. From single-tenant to multi-tenant scenarios

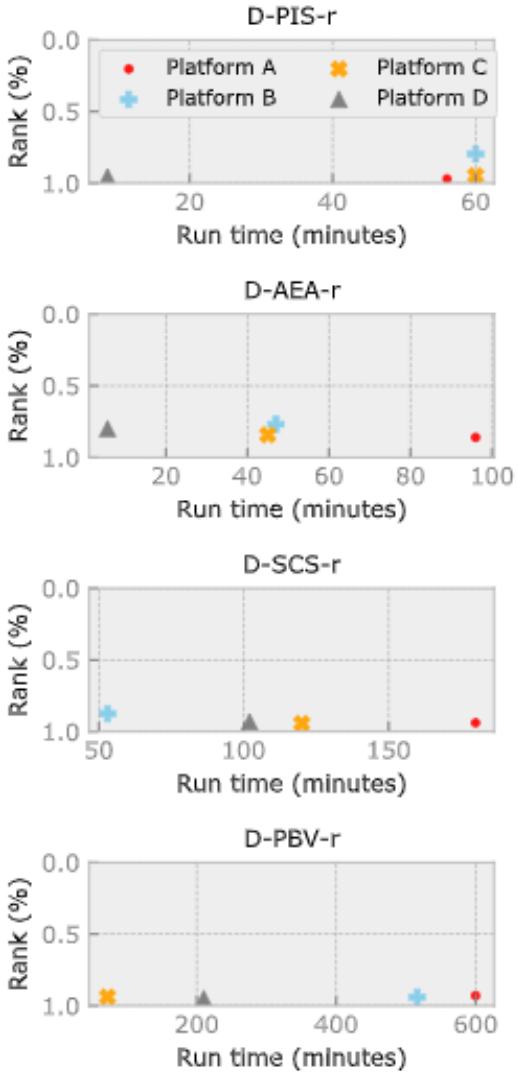


AutoML: A Small Personal Remark

ML today is now a Data Problem

- For many tasks, given the raw features from Kaggle, most AutoML platforms rank in the bottom 50%.
- It is the data that we need to improve, and knowledge that we need to integrate, to build better ML applications.
- To improve data, we need to first understand them.

Moving from a Model-driven development to a Data-driven development.



MLBench

VLDB (2018)
http://www.vldb.org/pvldb/vol11/p1220-liu.pdf

ML-Guided Database

Where DB Meets ML

- Human involved in research/engineering/analyzing/administrating:
  - Building and maintaining indexes
  - Query optimization
  - Physical design tuning
  - Optimizing view materialization

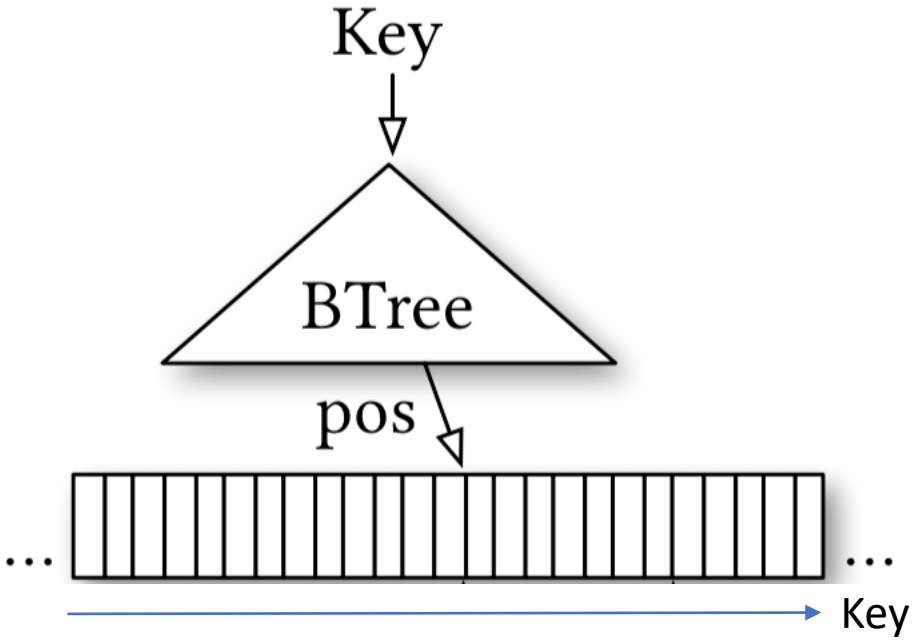
- Learning to automatically designing/optimizing/tuning?

Where DB Meets ML: Learning to Index

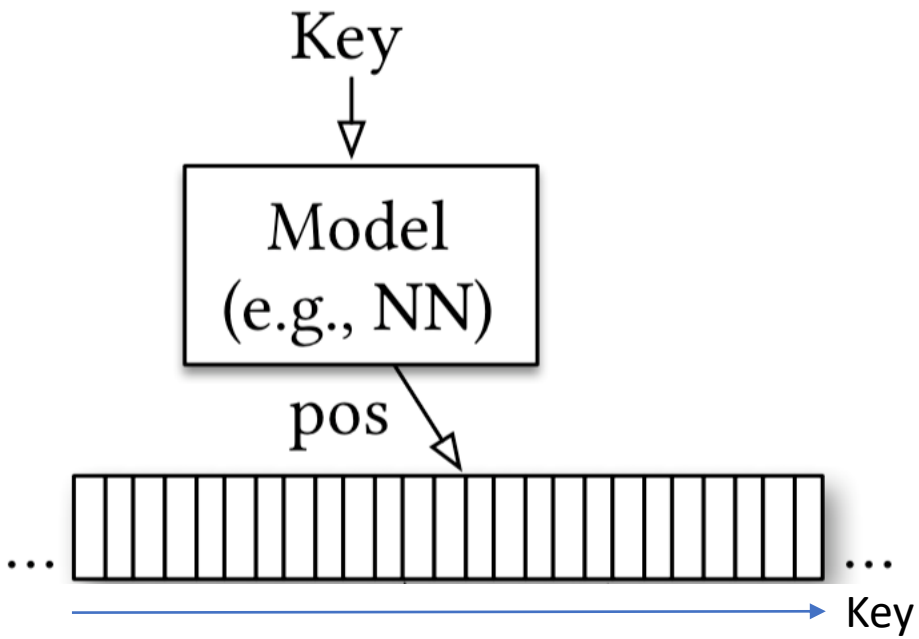
- Human involved in research/engineering/analyzing/administrating:
  - **Building and maintaining indexes**
  - Query optimization
  - Physical design tuning
  - Optimizing view materialization

- **Learning to automatically designing/optimizing/tuning?**

B-Tree Index from Learning Perspective



Input: Key
Output: Position
B-Tree Index: position = B-tree(Key)



Input: Key
Output: Position
Learned Index: position = function(Key)

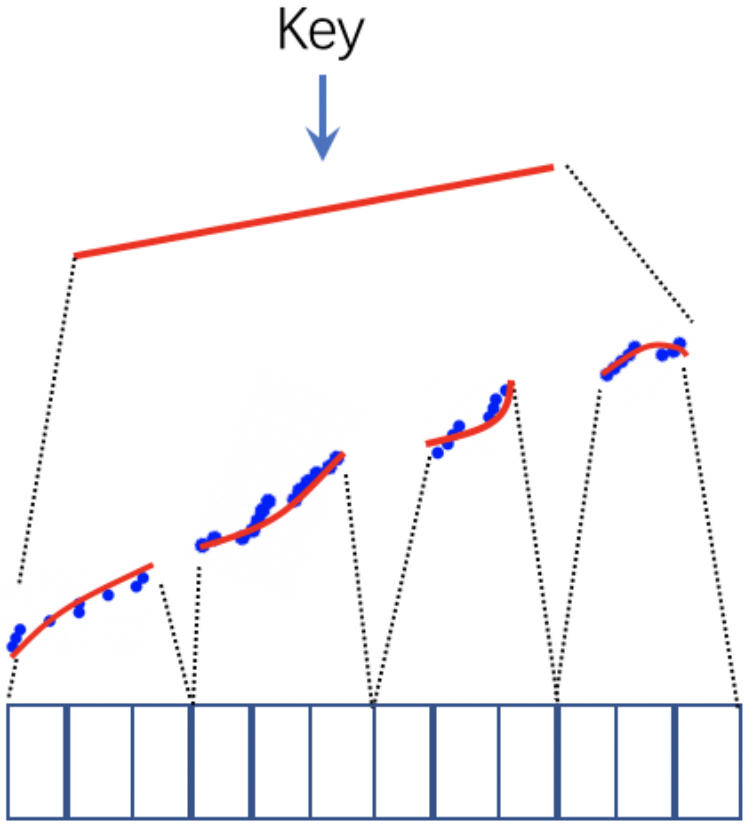
[Image source] Kraska et al., The case for learned index structures. *SIGMOD*, 2018

Why **Learning** Index from Data?

- Consider this (ideal) case: build an index to store and query over a table of *n* rows with continuous integer keys, *i.e.*, Keys = [11, 12, 13, 14, 15, ...] and Pos = [0, 1, 2, 3, 4, …]
  - B-Tree: seeking Pos in time O(log *n*)
  - a learned function Pos = M(Key) = Key + *offset* : O(1)

- Main motivation: the **hidden yet useful distribution information** about the data to be indexed has not been fully explored and utilized in the classic index techniques
  - learned index: an automatic way to explore and utilize such information

Recursive-Model Index (RMI)



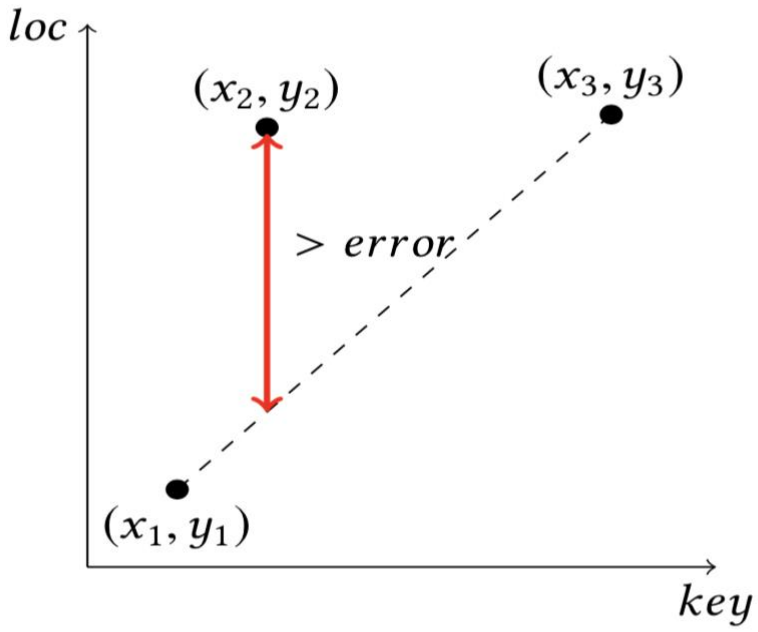
Root model

Sub-models

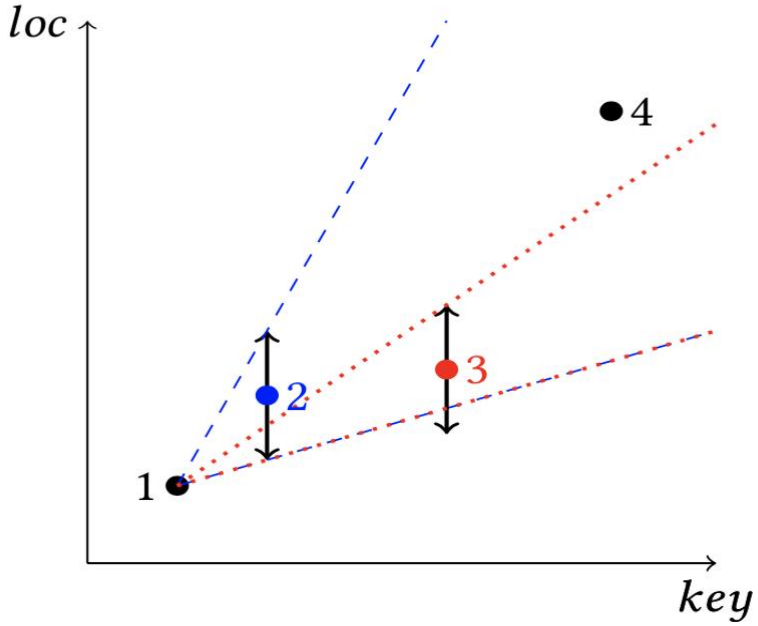
Data to be indexed

[Image source] Kraska et al., The case for learned index structures. *SIGMOD*, 2018

FITing-Tree



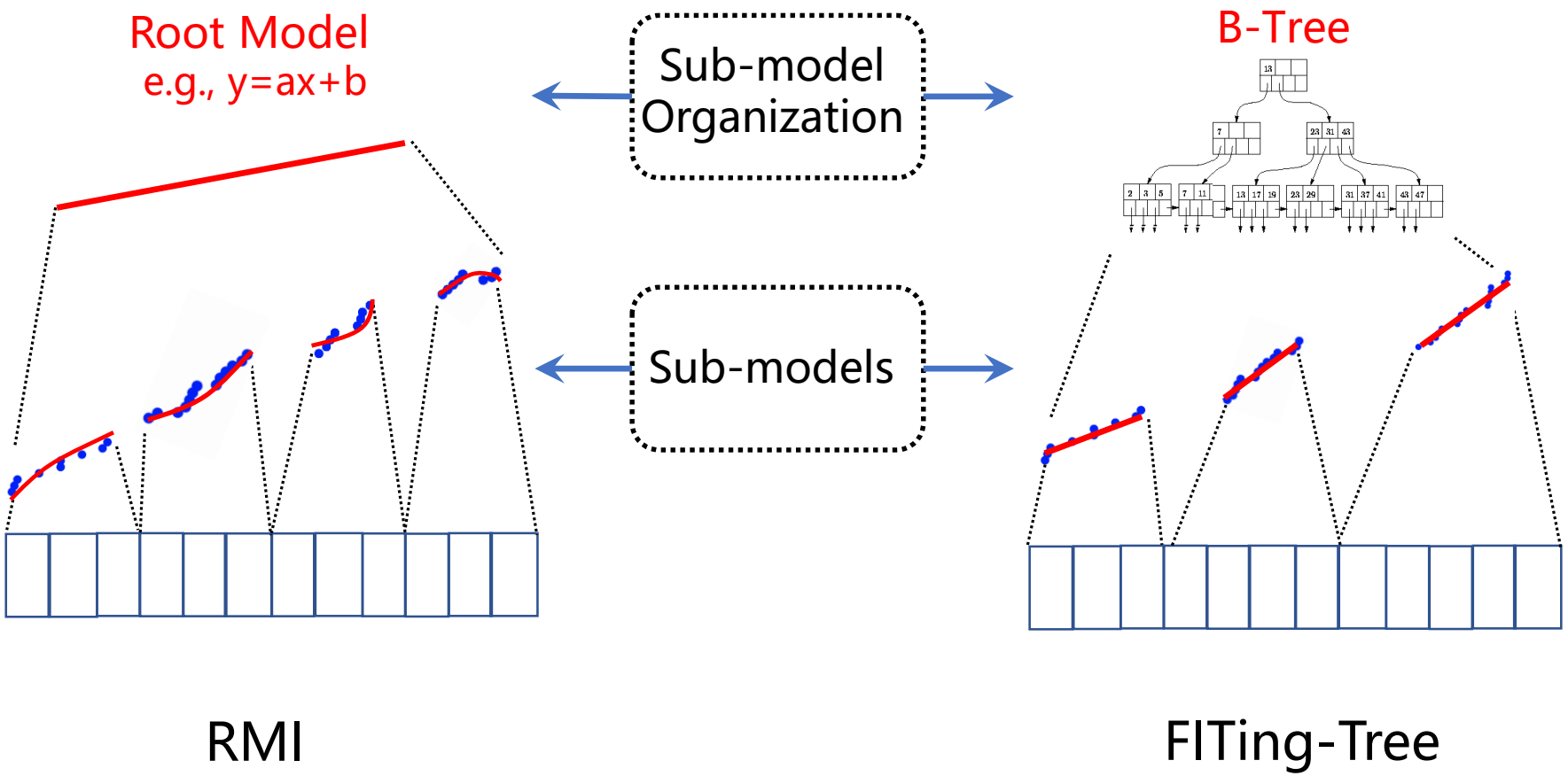
Error-Bounded Linear Segment: Given threshold $error$, a segment from (x_1, y_1) to (x_3, y_3) is *not valid* if (x_2, y_2) is further than $error$ from the interpolated line.



ShrinkingCone (building a segment): Point 1 is the origin of the cone. Point 2 is then added, resulting in the dashed cone. Point 3 is added next, yielding in the dotted cone. Point 4 is outside the dotted cone and therefore starts a new segment.

[Image source] Galakatos et al., FITing-Tree: A Data-aware Index Structure. SIGMOD, 2019

RMI v.s. FITing-Tree



More Learned Index Methods

- PGM [1] improves FITing-Tree by finding the optimal number of learned segments given an error bound.
- ALEX [2] proposes an adaptive RMI with workload-specific optimization, achieving high performance on dynamic workloads.
- RadixSpline [3] gains competitive performance with a radix structure while using a single-pass training.
- Multi-dimensional indexes: NEIST [4], Flood [5], Tsunami [6] and LISA [7].

More Learned Index Methods

- [1] The PGM-Index: A Fully-Dynamic Compressed Learned Index with Provable Worst-Case Bounds. *PVLDB*, 2020.
- [2] ALEX: An Updatable Adaptive Learned Index. *SIGMOD*, 2020.
- [3] RadixSpline: A Single-Pass Learned Index. *In aiDM Workshop on SIGMOD*, 2020.
- [4] NEIST: a Neural-Enhanced Index for Spatio-Temporal Queries. *TKDE*, 2019.
- [5] Learning Multi-dimensional Indexes. *SIGMOD*, 2020.
- [6] Tsunami: A Learned Multi-dimensional Index for Correlated Data and Skewed Workloads. *PVLDB*, 2020.
- [7] LISA: A Learned Index Structure for Spatial Data. *SIGMOD*, 2020.

Questions about Learned Indexes

How to systematically analyze and design machine learning based indexing methods?

More scalable index learning methods?

Which class of models suffice?

Task Definition

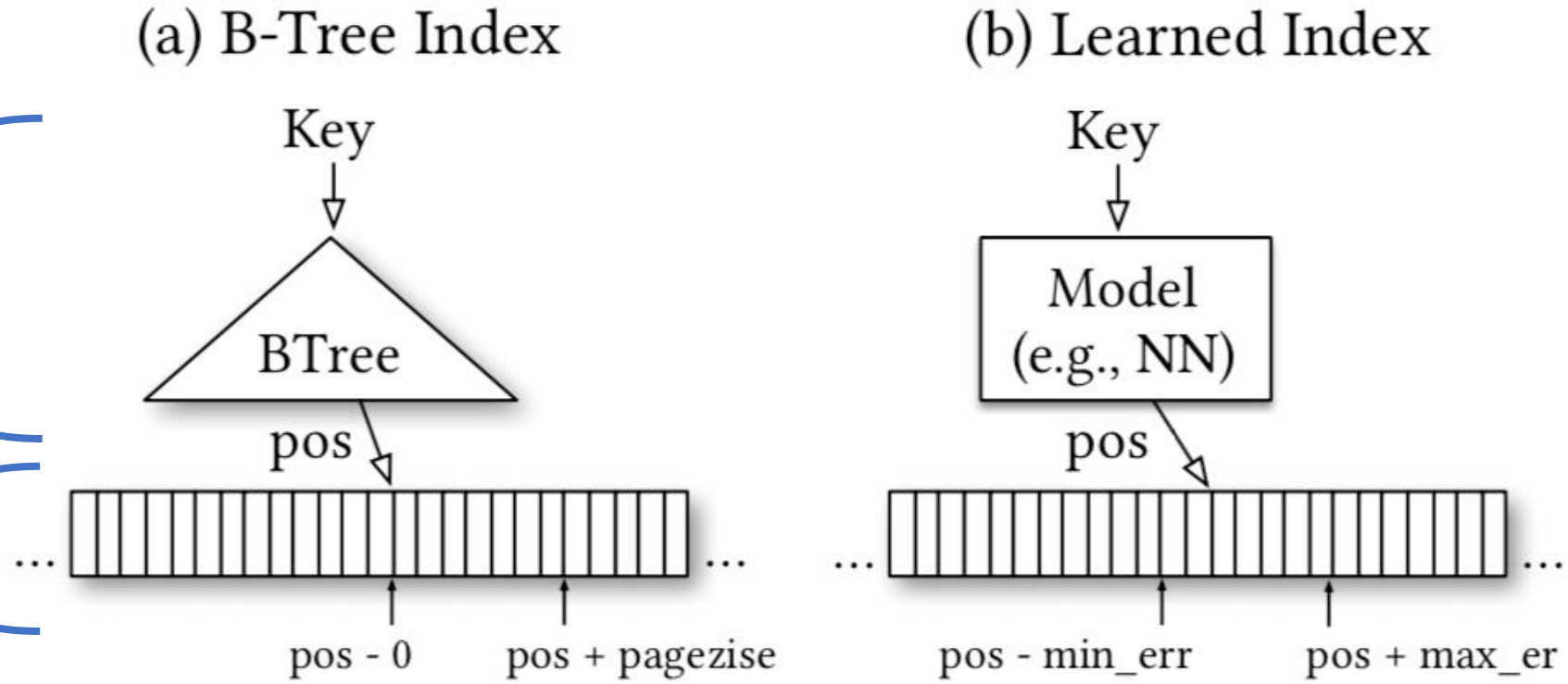
- Given a database $\mathcal{D}$ with n records (rows), let's assume that a *range index* structure will be built on a specific column x. For each record $i \in [n]$, the value of this column, x_i , is adopted as the *key*, and y_i is the *position* where the record is stored.

- We want to learn a *mechanism* M with the key x as input and outputs a *predicated position* $\hat{y} \leftarrow M(x)$ for accessing data.

Learning Index: A Machine Learning Task

$L(M)$
measures the cost of calculating $\hat{y} \leftarrow M(x)$

$L(\mathcal{D}|M)$
$= \mathbb{E}_{\{x,y\}\in\mathcal{D}} L(y, \hat{y})$



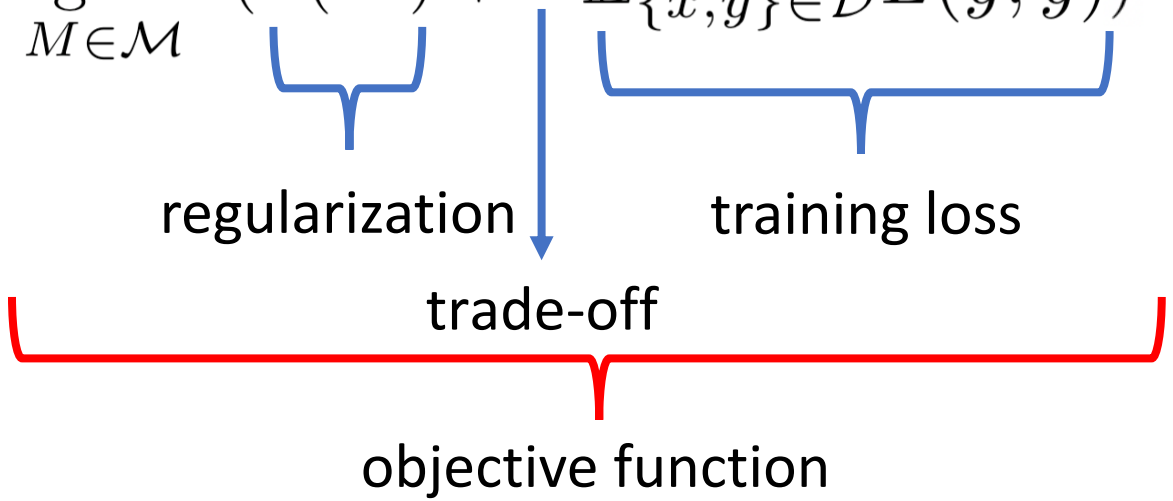
A Pluggable Learned Index Method via Sampling and Gap Insertion, https://arxiv.org/pdf/2101.00808.pdf

Learning Index: A Machine Learning Task

$$M^* = \arg\min_{M \in \mathcal{M}} \mathrm{MDL}(M, \mathcal{D})$$

$$= \arg\min_{M \in \mathcal{M}} \left(L(M) + \alpha L(\mathcal{D}|M)\right)$$

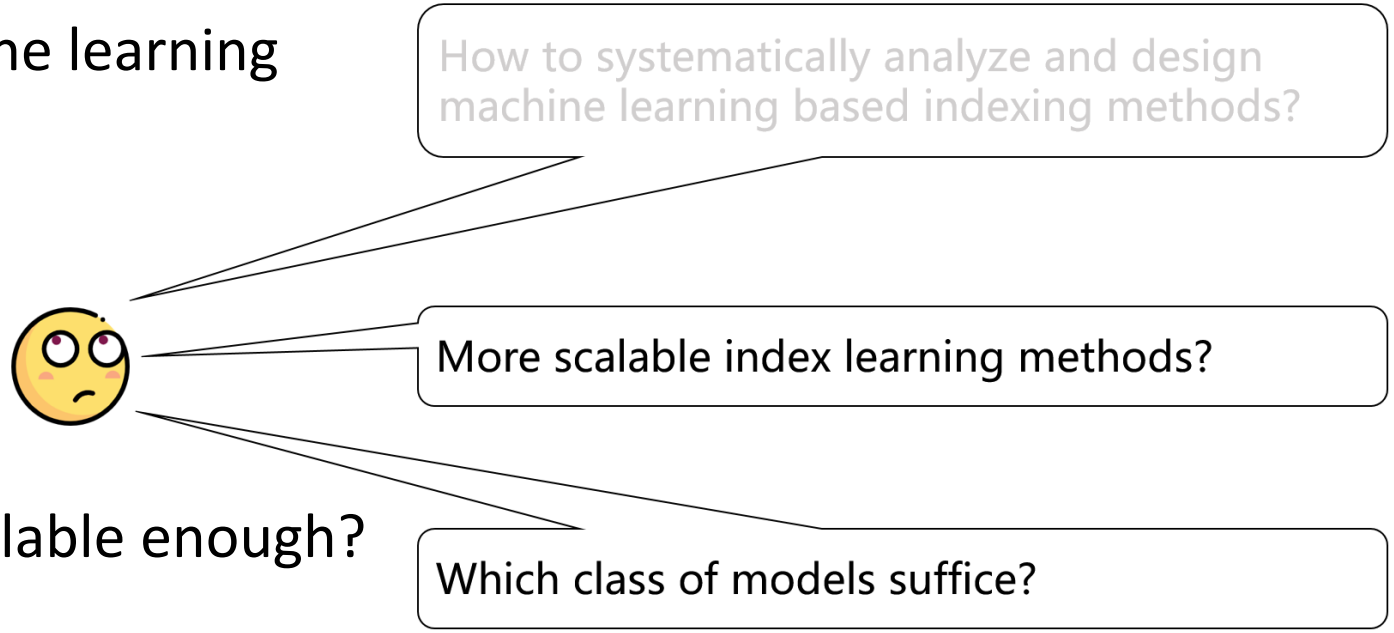
$$= \arg\min_{M \in \mathcal{M}} \left(L(M) + \alpha\, \mathbb{E}_{\{x,y\} \in \mathcal{D}} L(y, \hat{y})\right)$$



A Pluggable Learned Index Method via Sampling and Gap Insertion, https://arxiv.org/pdf/2101.00808.pdf

Benefits of Learned Index

- Smaller Size
- Faster Index Seek
- Better Handling Index Update
  - Generalization ability of machine learning
  - Incremental learning
- Question Mark
  - Is model training/inference scalable enough?



Learned Index with Sampling

$$
\begin{aligned}
M^* &= \arg\min_{M \in \mathcal{M}} \mathrm{MDL}(M, \mathcal{D}) \\
&= \arg\min_{M \in \mathcal{M}} \left(L(M) + \alpha L(\mathcal{D}|M)\right) \\
&= \arg\min_{M \in \mathcal{M}} \left(L(M) + \alpha\, \mathbb{E}_{\{x,y\} \in \mathcal{D}} L(y, \hat{y})\right)
\end{aligned}
$$

regularization | training loss | trade-off | objective function

- How large the sample needs to be?
  - n is the data size
  - M^* is fully optimized

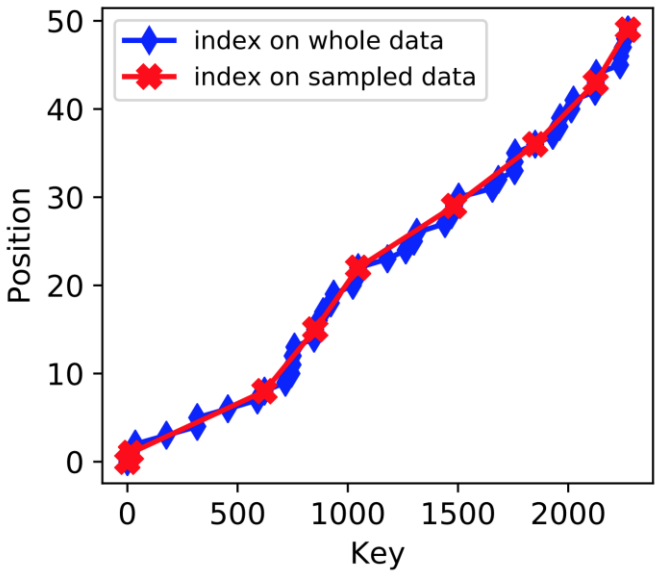


Fig: Illustration of sampling

THEOREM 1. *Consider the optimization problem:*

$$M^* = \arg\min_{M \in \mathcal{M}} \mathrm{MDL}(M, \mathcal{D}) = \arg\min_{M \in \mathcal{M}} \left(L(M) + \alpha L(\mathcal{D}|M)\right).$$

We can solve it on a random sample D_s *with size* $s = \mathrm{O}(\alpha^2 \log^2 n)$ *as*

$$\hat{M}^* = \arg\min_{M \in \mathcal{M}} \mathrm{MDL}(M, \mathcal{D}_s)$$

s.t., $\mathrm{MDL}(\hat{M}^*, \mathcal{D}) \leq \mathrm{MDL}(M^*, \mathcal{D}) + \mathrm{O}(1)$ *with high probability.*

A Pluggable Learned Index Method via Sampling and Gap Insertion, https://arxiv.org/pdf/2101.00808.pdf

Learned Index with Sampling

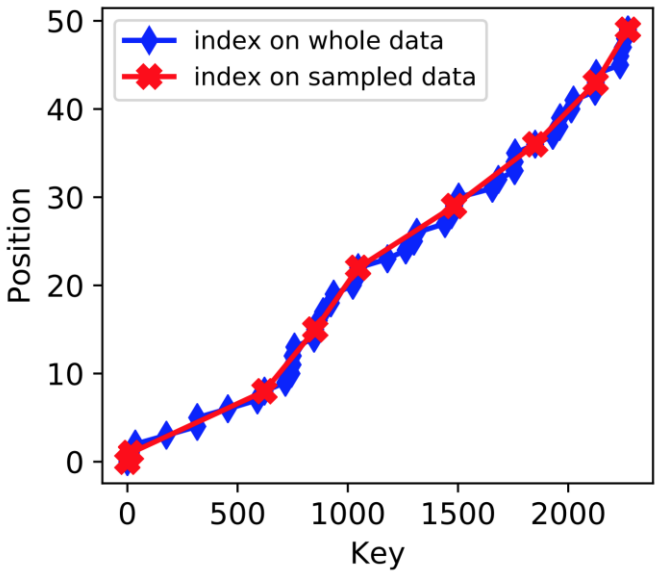
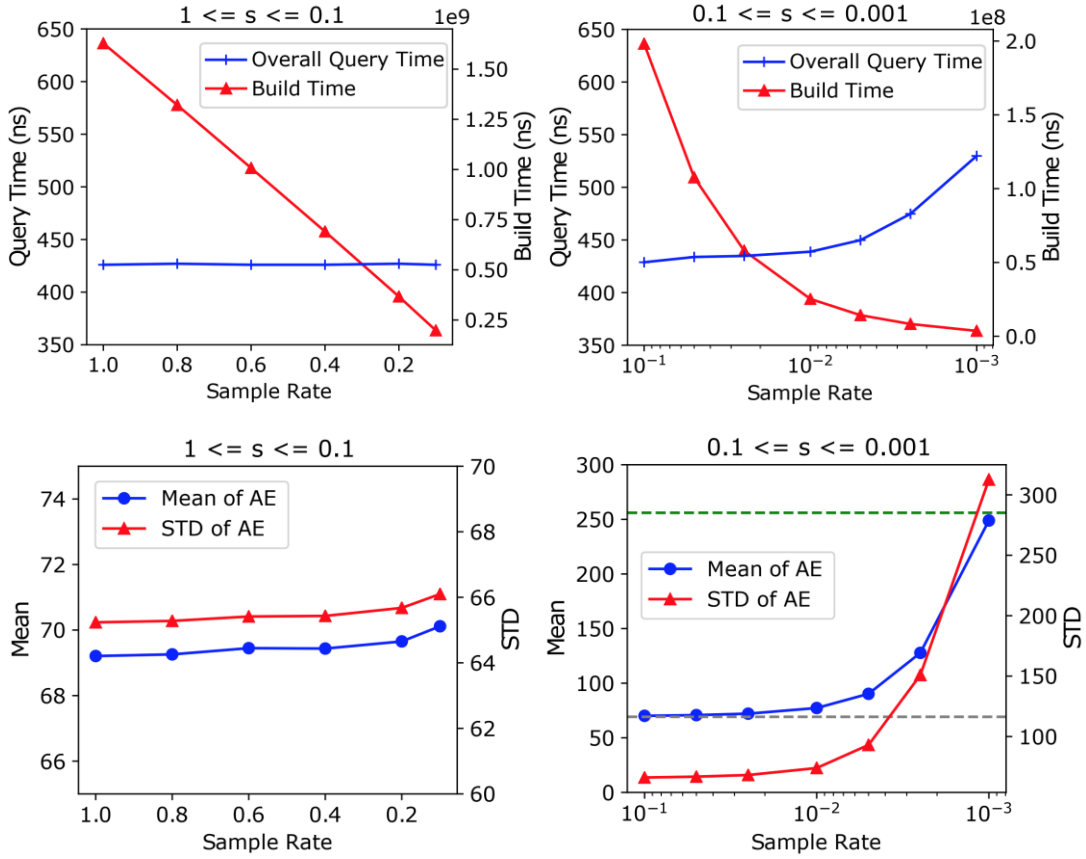


Fig: Illustration of sampling

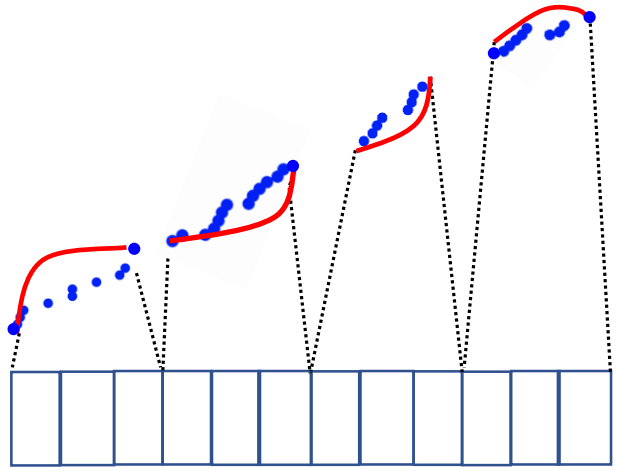
- Up to **78x** building speedup
- **Non-degraded performance** in terms of query time and prediction error)



A Pluggable Learned Index Method via Sampling and Gap Insertion, https://arxiv.org/pdf/2101.00808.pdf

Is Linear Model Sufficient?

- Linearization of a learned model

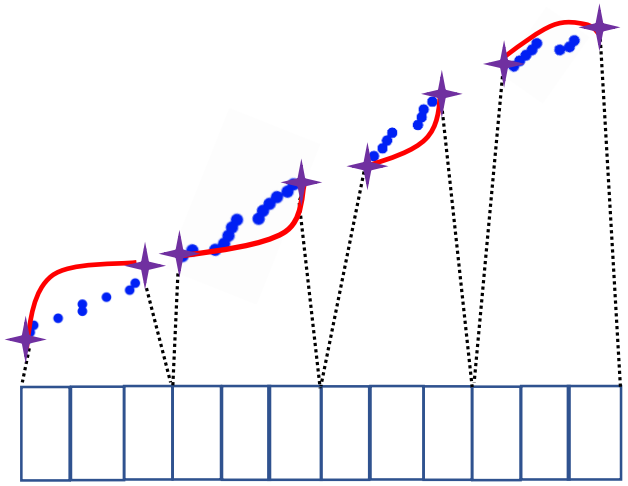


A learned model

$\bar{y} = M(x)$

Is Linear Model Sufficient?

- Linearization of a learned model

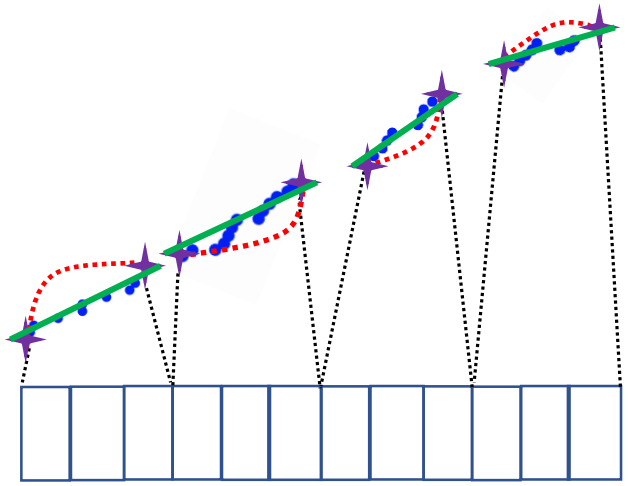


A learned model $\bar{y} = M(x)$

Landmark points $\ldots, (x_l, y_l), (x_r, y_r), \ldots$

Is Linear Model Sufficient?

- Linearization of a learned model



A learned model $\bar{y} = M(x)$

Landmark points $\ldots, (x_l, y_l), (x_r, y_r), \ldots$

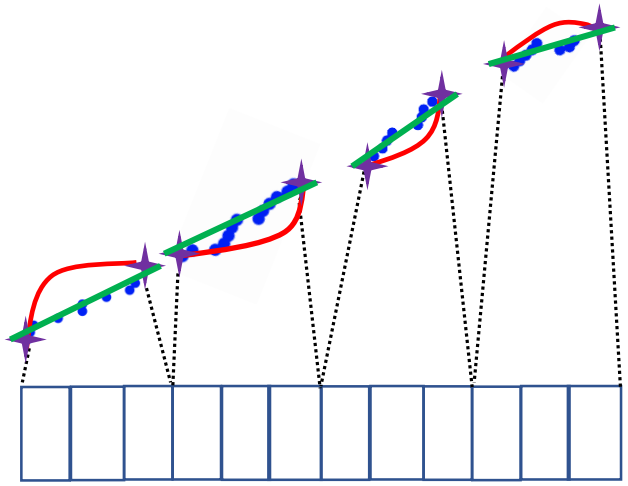
Linearized model $\hat{y} = M_L(x)$

connecting $(x_l, \bar{y}_l = M(x_l))$ **to** $(x_r, \bar{y}_r = M(x_r))$

Is Linear Model Sufficient?

Yes! As long as landmark points are dense enough

- Linearization of a learned model



A learned model $\overline{y} = M(x)$

Landmark points $\ldots, (x_l, y_l), (x_r, y_r), \ldots$

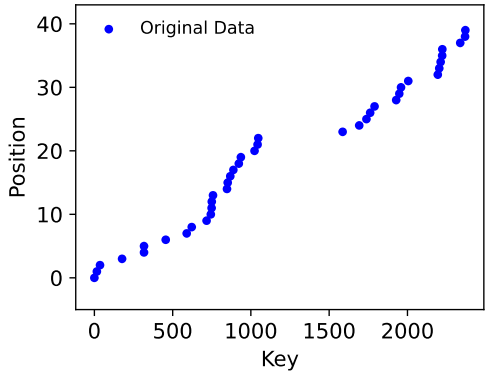
Linearized model $\hat{y} = M_L(x)$

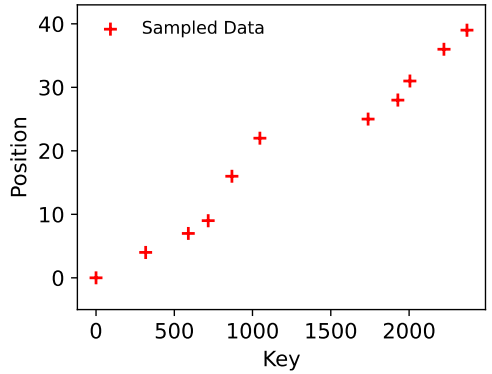
connecting $(x_l, \overline{y}_l = M(x_l))$ **to** $(x_r, \overline{y}_r = M(x_r))$

Theorem 2. Suppose $\forall x, |\overline{y} - y| \leq \epsilon$, after linearization, we have $\forall x, |\hat{y} - y| \leq 3\epsilon + 2(y_r - y_l)$.

A Pluggable Learned Index Method via Sampling and Gap Insertion, https://arxiv.org/pdf/2101.00808.pdf

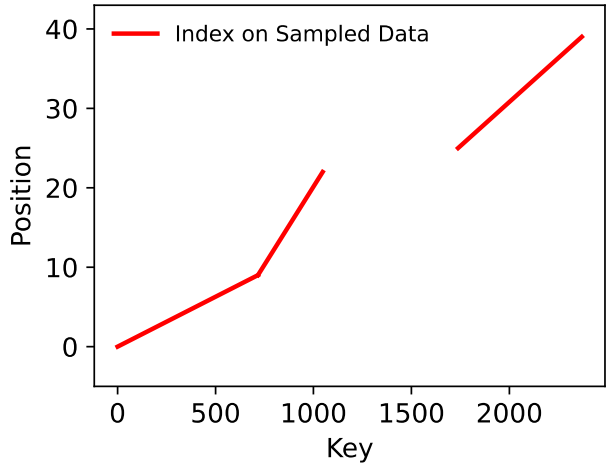
Sampling-Restriction-Linearization

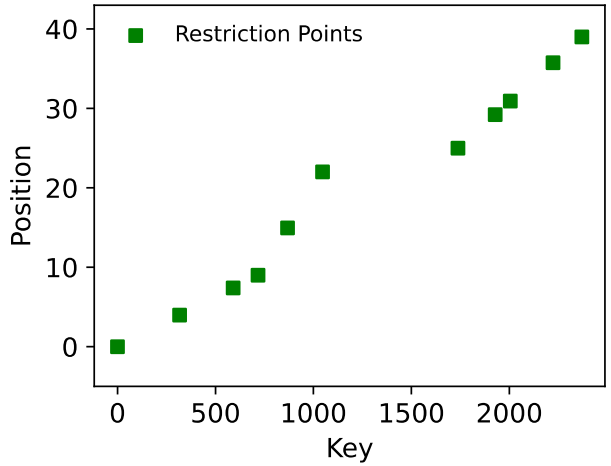


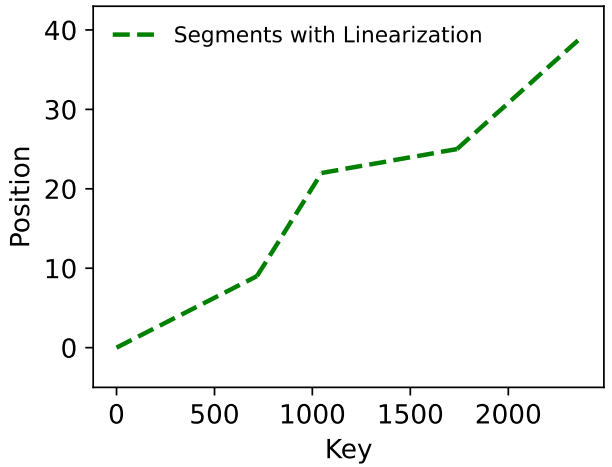


Sampled data points as landmark points:

$\ldots, (x_l, y_l), (x_r, y_r), \ldots$







Learned Index on Sampled Data → **Restriction** → **Linearization**

A Pluggable Learned Index Method via Sampling and Gap Insertion, https://arxiv.org/pdf/2101.00808.pdf

Open Questions

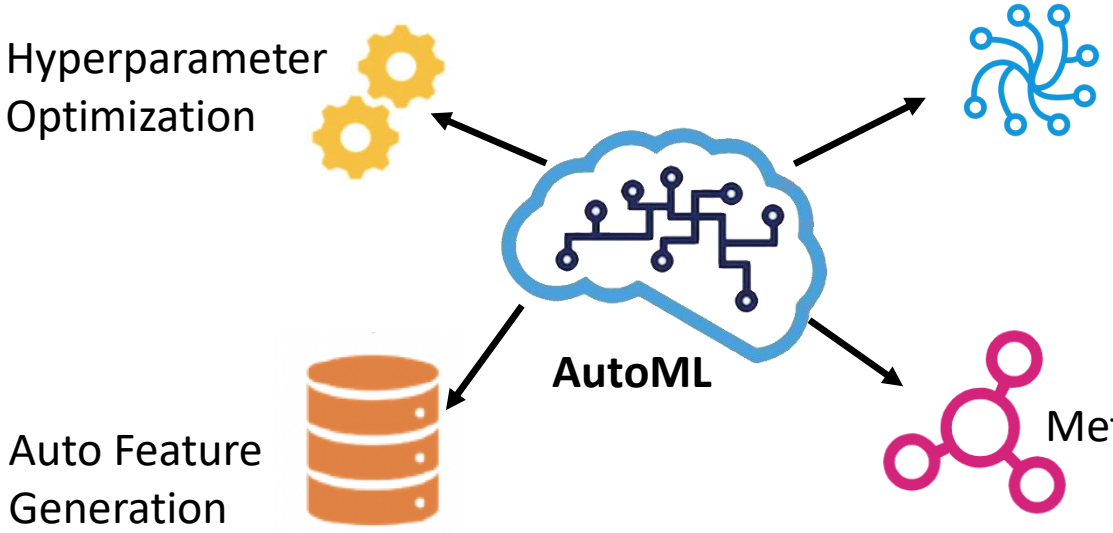
- How to handle extremely outlier keys?
- How to maintain index on updating data? [2]
- How to handle multi-dim data? [5, 6, 7]
- How to build it into real DB systems?
  - without too much modification to the current system

AutoML Tools

Availability

Learning to Mutate with Hypergradient Guided Population, NeurIPS 2020.

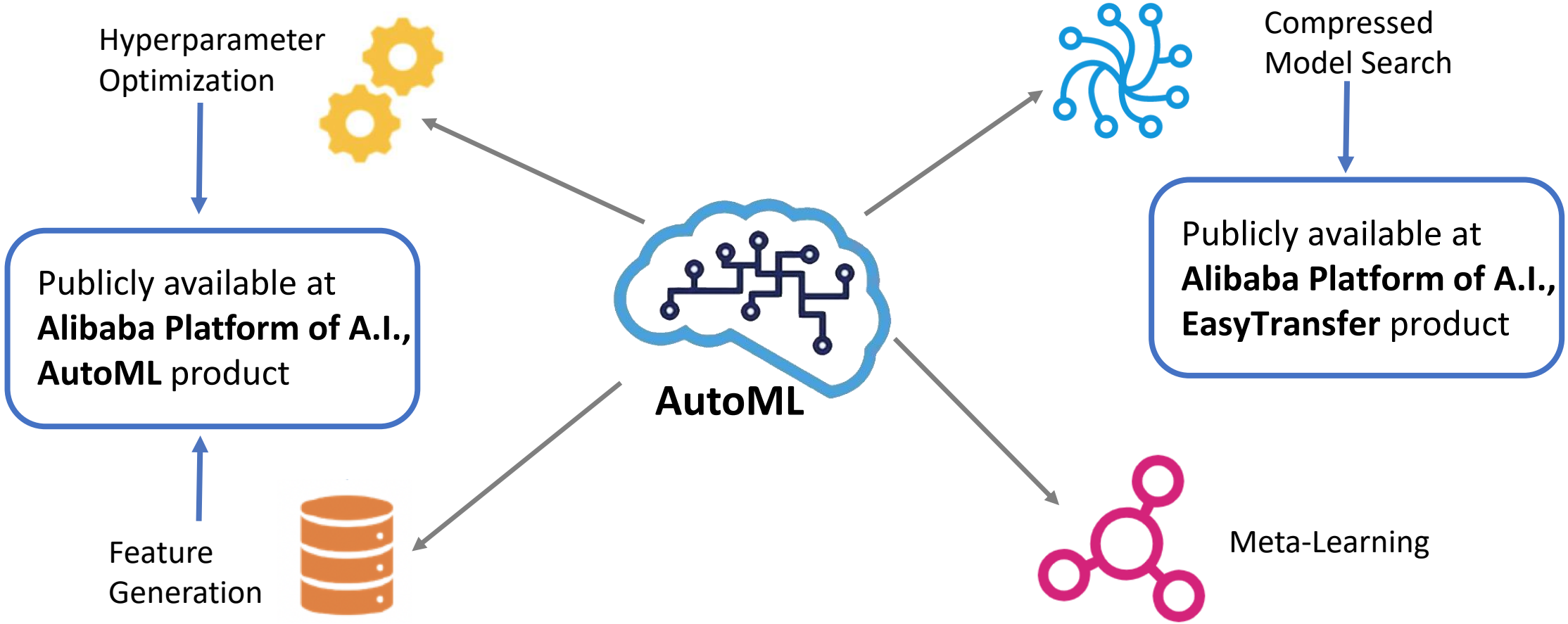
AdaBERT: Task-Adaptive BERT Compression with D-NAS, IJCAI 2020 https://arxiv.org/abs/2001.04246



FIVES: Feature Interaction Via Edge Search for Large-Scale Tabular Data, KDD 2021. https://arxiv.org/abs/2007.14573

Automated Relational Meta-learning, ICLR 2020. https://arxiv.org/abs/2001.00745

Availability



A Summary of AutoML Tools

Name	Authors	Functionalities	Algorithms	Language
Auto Tune Models (ATM)	MIT	AutoFeature, Model Selection, HPO	BO and Bandit	Python
AutoKeras	Texas A&M University	NAS	BO	Python
NNI	Microsoft	AutoFeature, HPO, NAS, Model Selection	Comprehensive	Python
emukit	Amazon	HPO	Meta-surrogate model	Python
Ray Tune	Berkeley	HPO	Comprehensive	Python
TPOT	University of Pennsylvania	AutoFeature, Model Selection, HPO	Genetic programming	Python
More AutoML packages include AutoFolio, Auto-sklearn, Auto-PyTorch, Auto-WEKA, etc.				

Tutorial Schedule

Yaliang Li, Background and Overview of AutoML
Hyperparameter Optimization

Zhen Wang, Neural Architecture Search
Meta-Learning

Yuexiang Xie, Automatic Feature Generation

Ce Zhang, VolcanoML: End-to-End AutoML via
Scalable Search Space Decomposition

Bolin Ding, Machine Learning Guided Database

Thank you!

Yaliang Li, Zhen Wang, Yuexiang Xie, Bolin Ding, and Ce Zhang

Email: yaliang.li@alibaba-inc.com

Please feel free to contact us if you have any questions,
or you are interested in **full-time** or **research intern** positions.