ACM Symposium on Cloud Computing

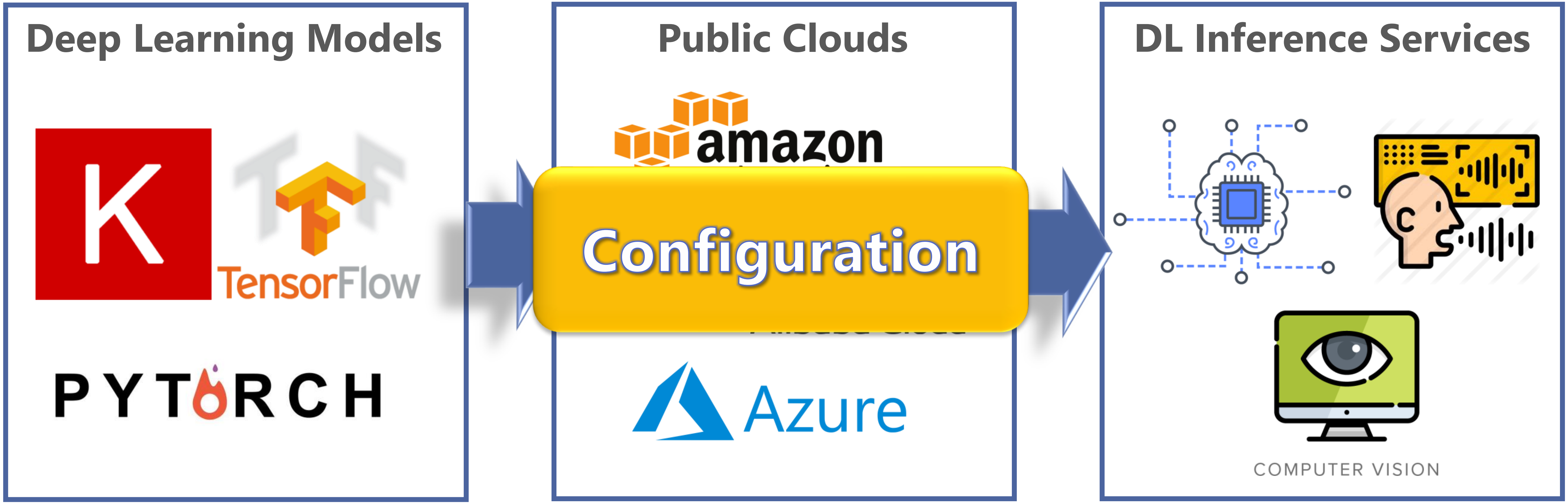
Serving Unseen Deep Learning Models with Near-Optimal Configurations: a Fast Adaptive Search Approach

Yuewen Wu *, Heng Wu *, Diaohan Luo + , Yuanjia Xu +, Yi Hu +, Wenbo Zhang *, Hua Zhong *

∗ *Institute of Software, Chinese Academy of Sciences*

+ *University of Chinese Academy of Sciences*

Serving deep learning models on public clouds becomes popular



It is essential to recommend near-optimal configurations for DL inference services, because...

Configuration includes: **runtime** & **resource** configurations.
Public clouds (e.g., Amazon EC2) provide **more than 1,000** configuration candidates.

Runtime configurations

batch size

… …

Resource configurations

GPU type, CPU number, GPU memory, … …

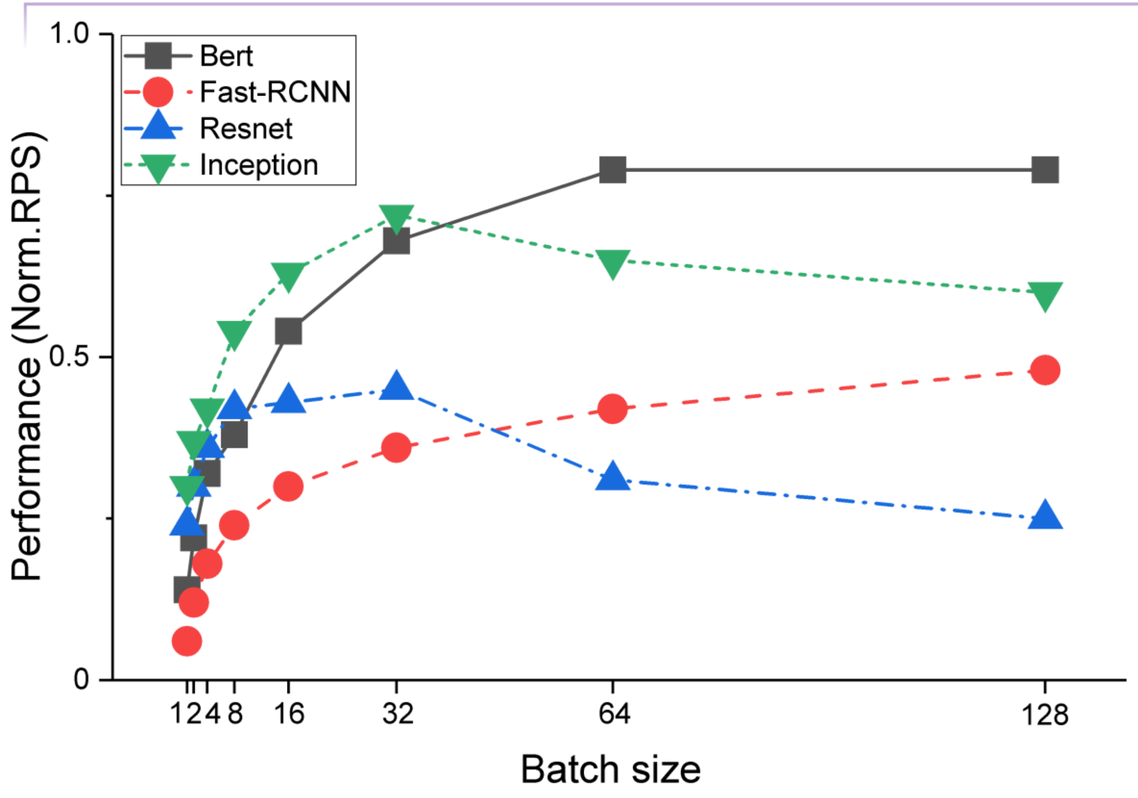
There are so many candidates that make it difficult for users to properly configure their DL models.

Choosing a near-optimal configuration has many benefits, including...

A near-optimal configuration can improve up to 8x performance and reduce over 60% budget

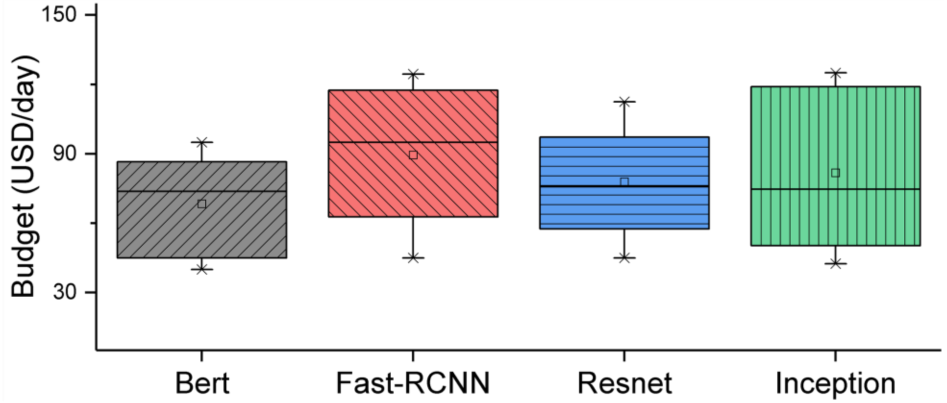
Improving performance

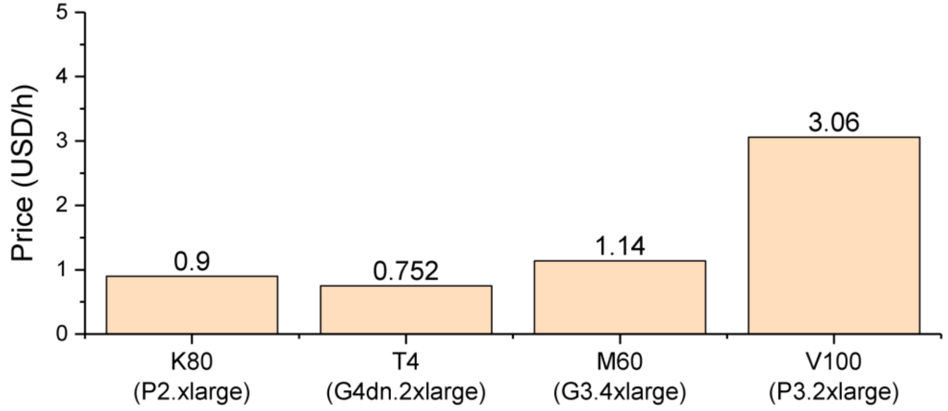
* The performance is evaluated by *request per second*, or RPS.



Reducing budget

* The budget is evaluated by *USD*, and it is calculated by *instance price* × *running time*.

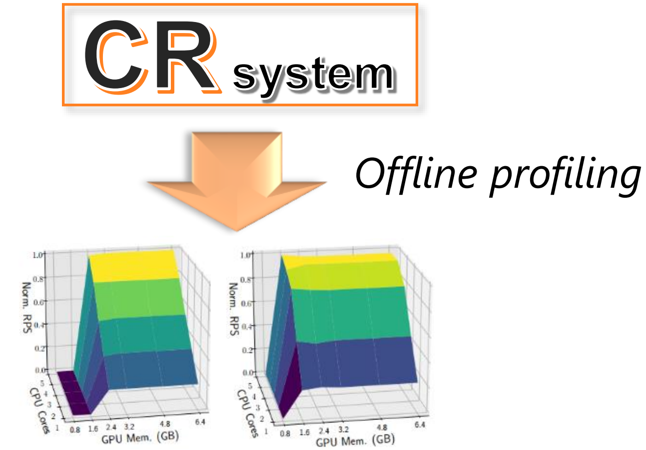




Existing configuration recommender (CR) systems suffer from a severe cold start problem, especially for unseen DL models

Existing CR systems, such as Morphling [1], reusing historical data from previous DL models to improve the configuration search of "seen" models. They work as follows:

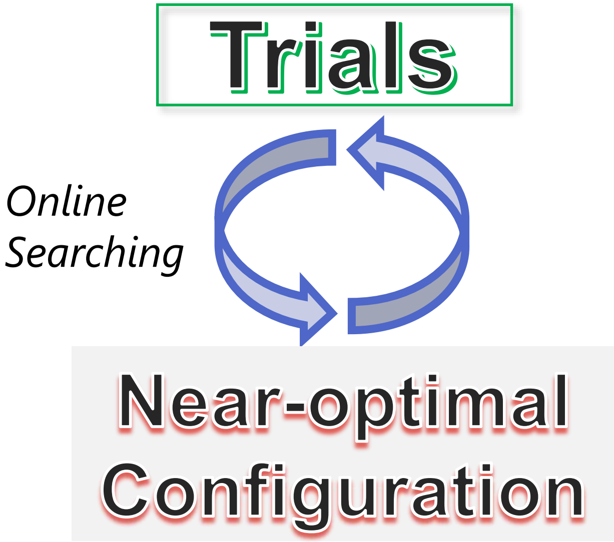
(a) Learning **model-level similarity** (resource sensitivity curves) by offline profiling.



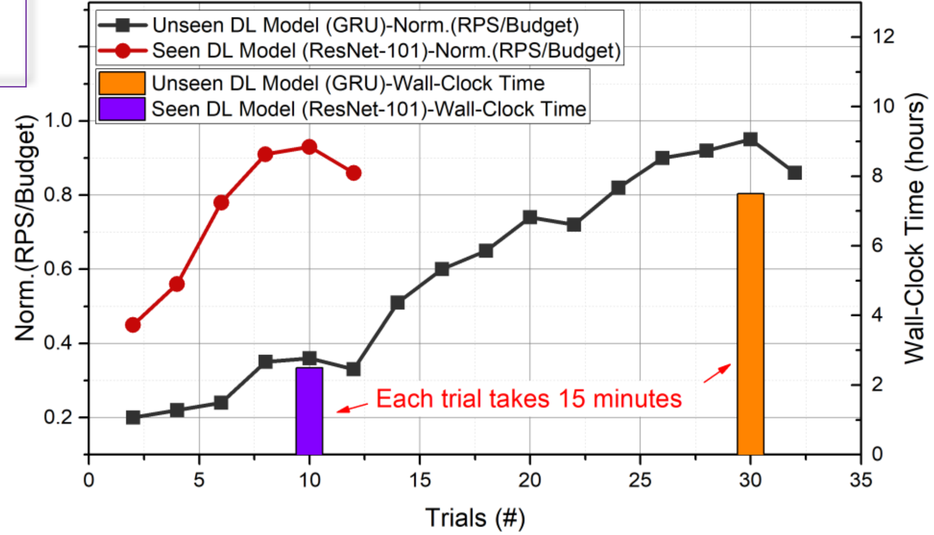
Model-level Similarity

(b) Running *trials*[1] for the target DL model to search online for a near-optimal configuration.

1. The *trial* is a stress test of the target DL model in a certain configuration to measure its performance.



(c) However, we find that **model-level similarity** suffers from severe **cold start problem**, especially for **unseen DL models**.



2.5 hours for **seen** DL models because of **high** model-level similarity

7.5 hours for **unseen** DL models because of **low** model-level similarity

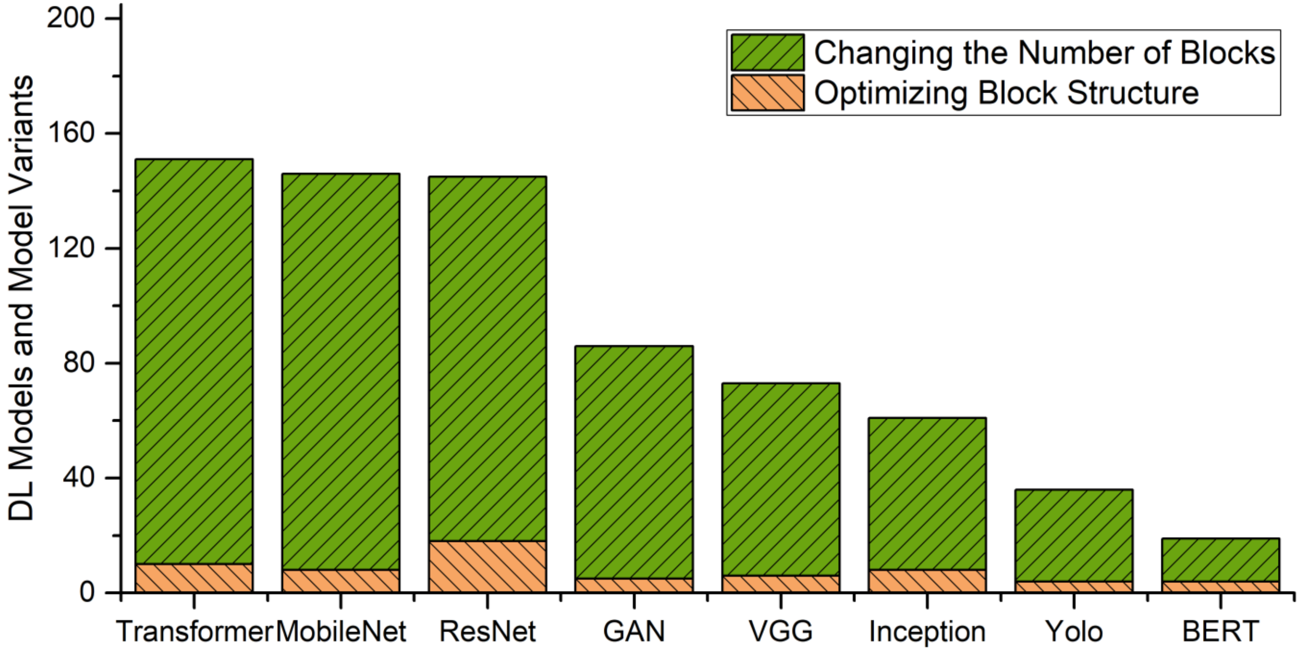
[1] Wang L, Yang L, Yu Y, et al. Morphling: Fast, Near-Optimal Auto-Configuration for Cloud-Native Model Serving[C]//Proceedings of the ACM Symposium on Cloud Computing. 2021: 639-653.

In fact, serving unseen DL models is a common requirement

Survey of over 1,200 DL models in *TensorFlow hub* shows that **newly developed DL models** and **model variants** are two main sources of unseen DL models.

For purposes such as improving model accuracy, improving performance, model variants include:

- Changing the number of blocks. For instance, *ResNet-152* has more blocks, deeper networks, and also requires more resources than other *ResNet models*.
- Optimizing block structure. For instance, *Inception V4* combines the residual network structure which was not present in previous versions of *Inception* (*v3/v2/v1*).



Partial statistics of the DL models and model variants.

Question: How to quickly adapt to unseen DL models?

Existing CR systems require dozens of *trials* to find near-optimal configurations for unseen DL models, mainly because:

- **Large search space:** over 1,000 configuration candidates.
- **Complex DL models and model variants**.
- **Poor model-level similarity**.

For a given unseen DL model,
how to find
a near-optimal configuration
over *a few trials*
to alleviate the cold start problem?

Key insight: Leveraging operator-level instead of model-level similarity

Although there are significant differences between DL models, **they all consist of a limited type of DL operators**[1].

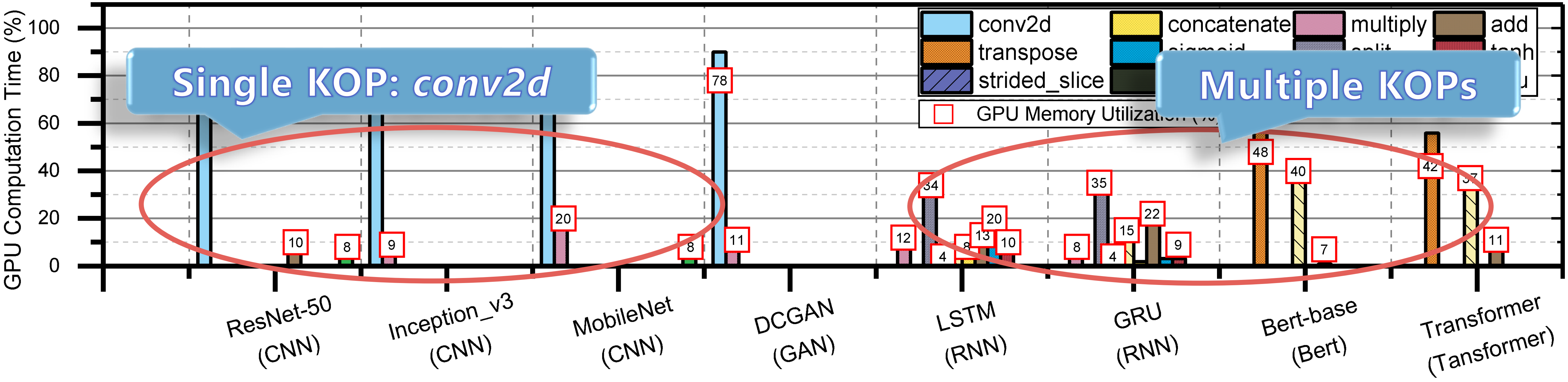
Operator-level similarity has the potential to quicky adapt to unseen DL models.

There are two important observations to support this insight...

1. The DL operator is the minimal execution unit of the DL model and it has independent resource requirements.

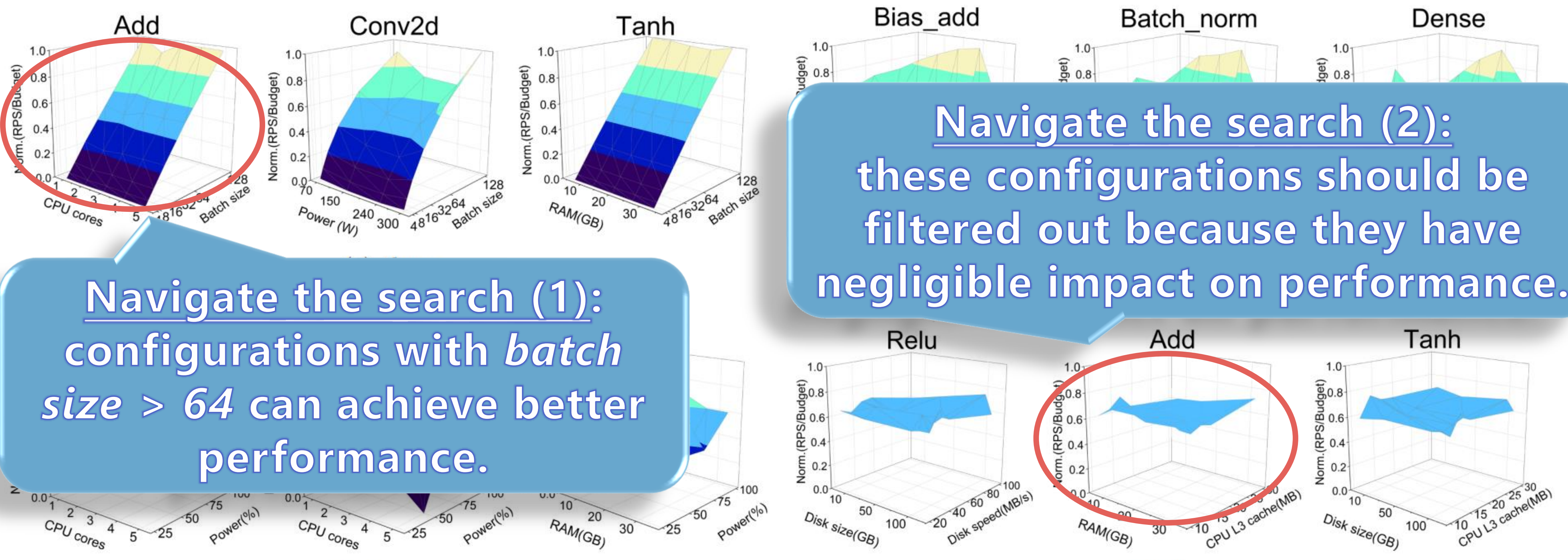
Observation 1: Key Operators (KOs) to depict DL model's performance

Through a large-scale evaluation on Amazon EC2 with 30 typical DL models, we find that DL operators are better suited to describe the performance of DL models.



For a given DL model, there are some Key Operators (KOPs) to depict its GPU computation time and GPU memory utilization

Observation 2: Key Operator Resource Curves (KOP-RCs) to navigate the search in a large search space



(c) Concave.

(d) Plane.

Navigate the search (1): configurations with *batch size > 64* can achieve better performance.

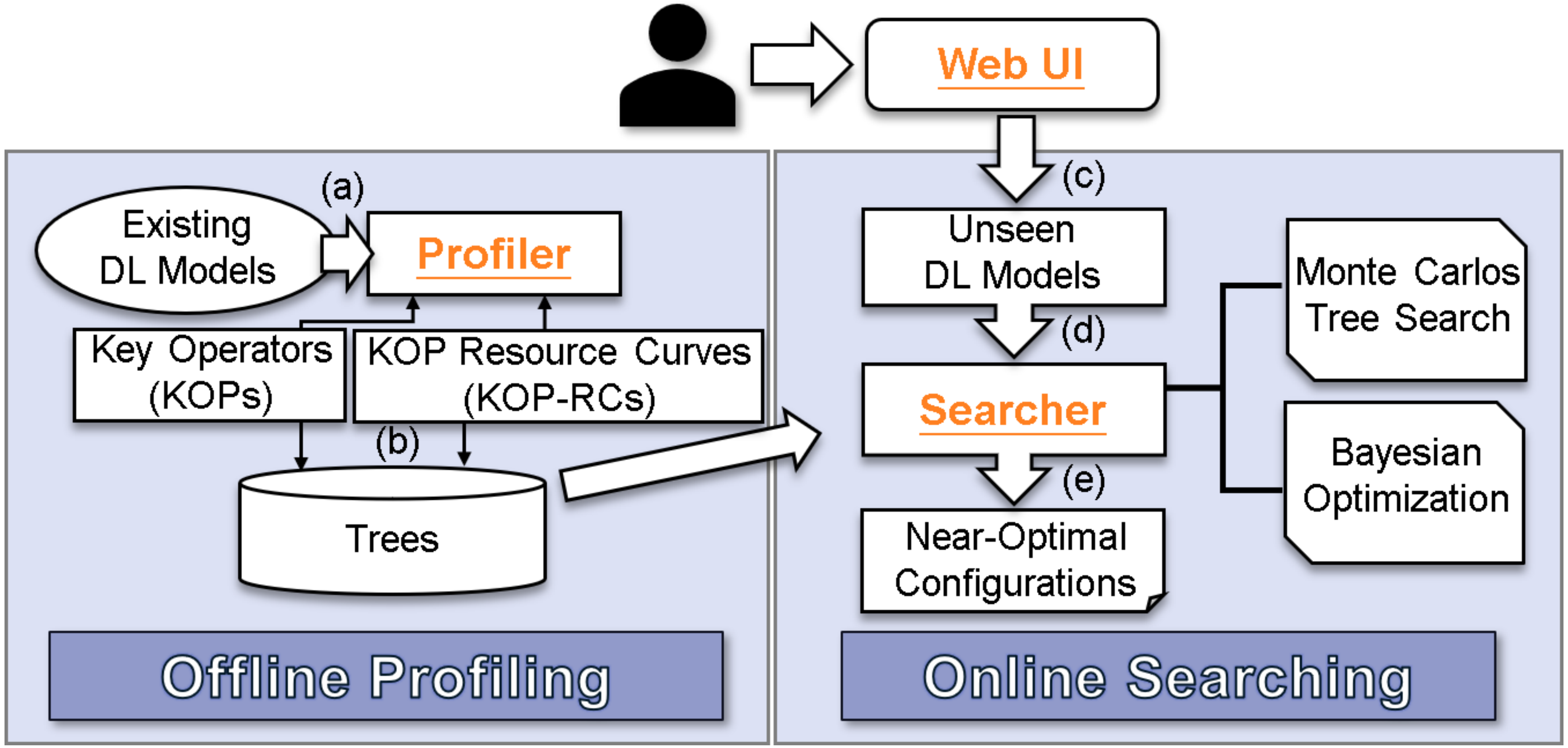
Navigate the search (2): these configurations should be filtered out because they have negligible impact on performance.

For each KOP, there are four typical Key Operator Resource Curves (KOP-RCs) to navigate the search of near-optimal configurations

Falcon: a Fast Adaptive Configuration Recommender System

Falcon works within a two-phase framework:

- **Offline Profiling:** learn KOPs and KOP-RCs.
- **Online Searching:** fast adaptive search by reusing KOPs and KOP-RCs.



1. Learning KOPs and KOP-RCs from a large-scale evaluation on Amazon EC2

DL models

- **Computer vision.** It includes VGG, ResNet, YOLO, DenseNet, etc.
- **Natural language process.** It includes LSTM, GRU, Bert, etc.
- **Generative adversarial network.** It includes DC-GAN, WG-AN, SGAN, etc.
- **Recommend system.** It includes NCF, DCN, DRN, etc.

Configuration knobs

- **Runtime configuration knobs.** We mainly consider the *batch size* as the runtime configuration knob, because it can profoundly impact the performance of DL models [28].
- **Resource configuration knobs.** These configuration knobs can be tuned when we deploy DL models on public clouds. They include GPU type, GPU memory, CPU cores, CPU L3 cache, RAM, GPU power [7], disk speed, disk size, network speed, etc.

We make the following efforts to better profile KOPs and KOP-RCs:

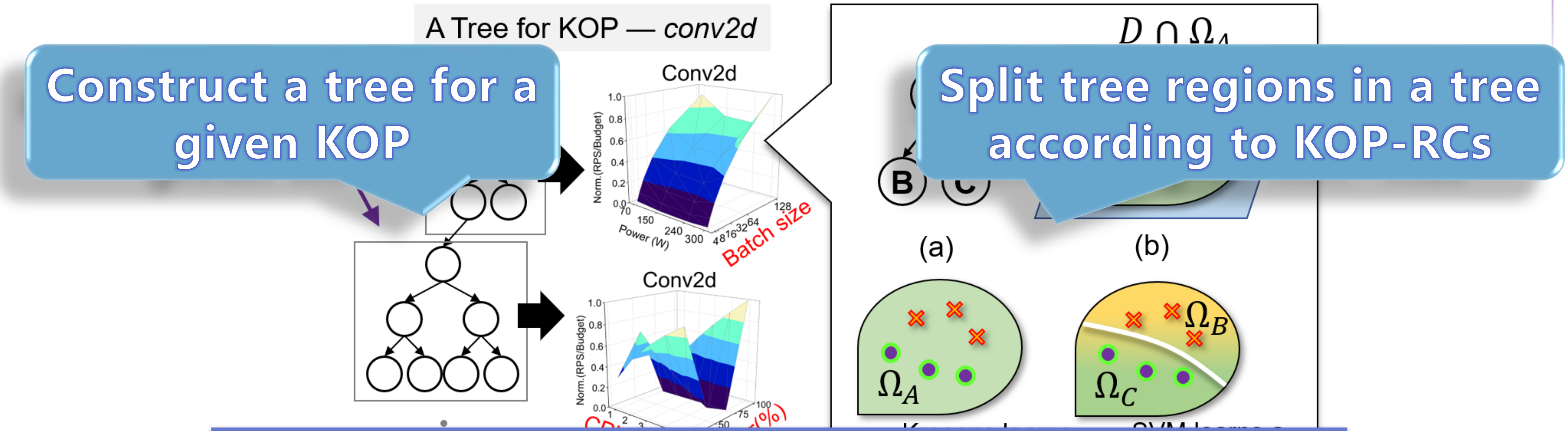
- **Setting thresholds for each model to better identify KOPs**.
- **Pruning redundant configurations in KOP-RCs:** PCA has been employed.

After the above steps, we learn an offline dataset *D* containing KOPs and KOP-RCs.

2. Constructing trees to represent KOPs and KOP-RCs

We choose the tree structure, because:

- **To better represent the complex relationship:** an unseen DL model has multiple KOPs, and a KOP has multiple KOP-RCs.
- **To partition a large search space into small search regions**.



As long as KOPs of an unseen model have been learned before, we are able to represent this model in trees

2. Constructing trees to represent KOPs and KOP-RCs

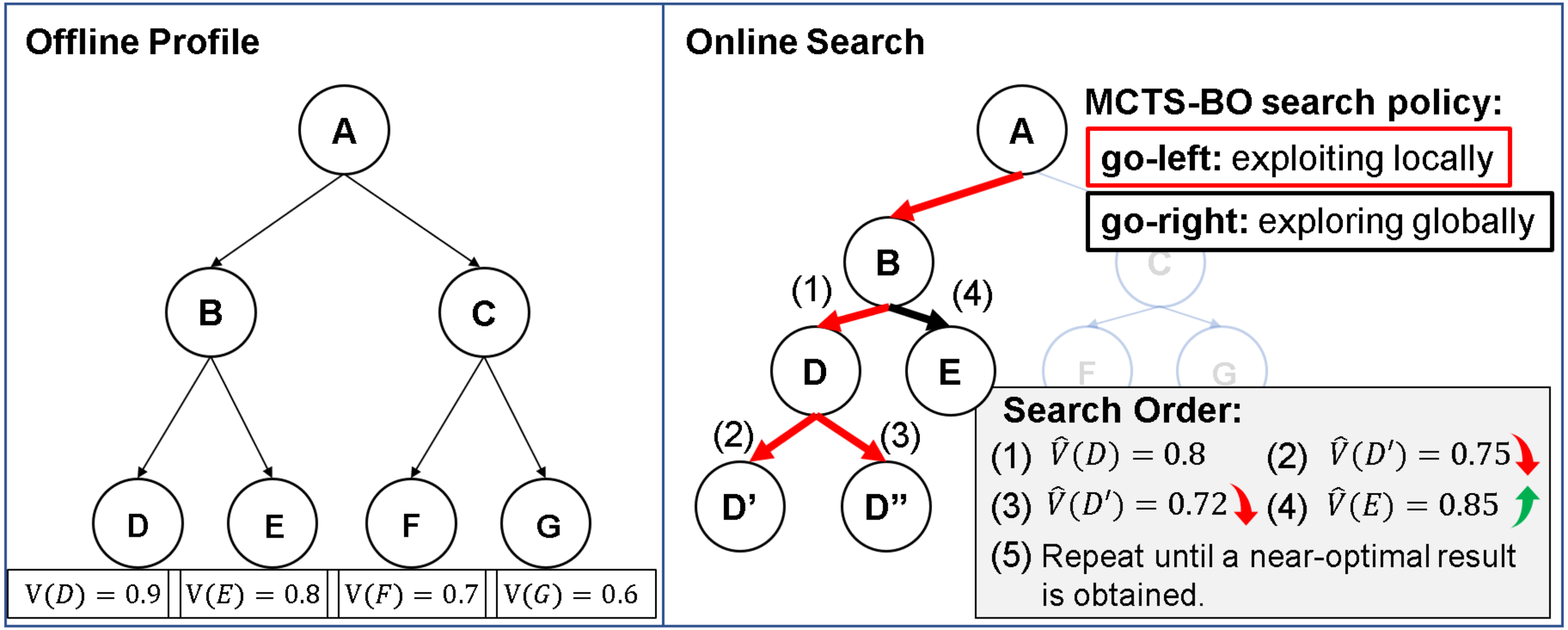
Since we have offline data, why do we still need online search when a target DL model arrives? There are two main reasons:

- **Complex DL model variants:** Their structure and parameters may be completely different, and their resource sensitivities may also different. Thus, reusing the best configuration from the offline dataset may lead to poor performance.
- **Conflicts in KOPs:** The resource sensitivity may also be obscured by conflicts in multiple KOPs. For instance, KOP *conv2d* requires a larger *batch size* of 128, while KOP *dense* achieves optimal norm.(RPS/Budget) when *batch size* is 64. Therefore, when different KOPs are in a same DL model, it is difficult to evaluate the impact of them via accurate estimation.

3. Fast adaptive search via Monte Carlo Tree Search and Bayesian Optimization (MCTS-BO)

We implement MCTS-BO because it can:

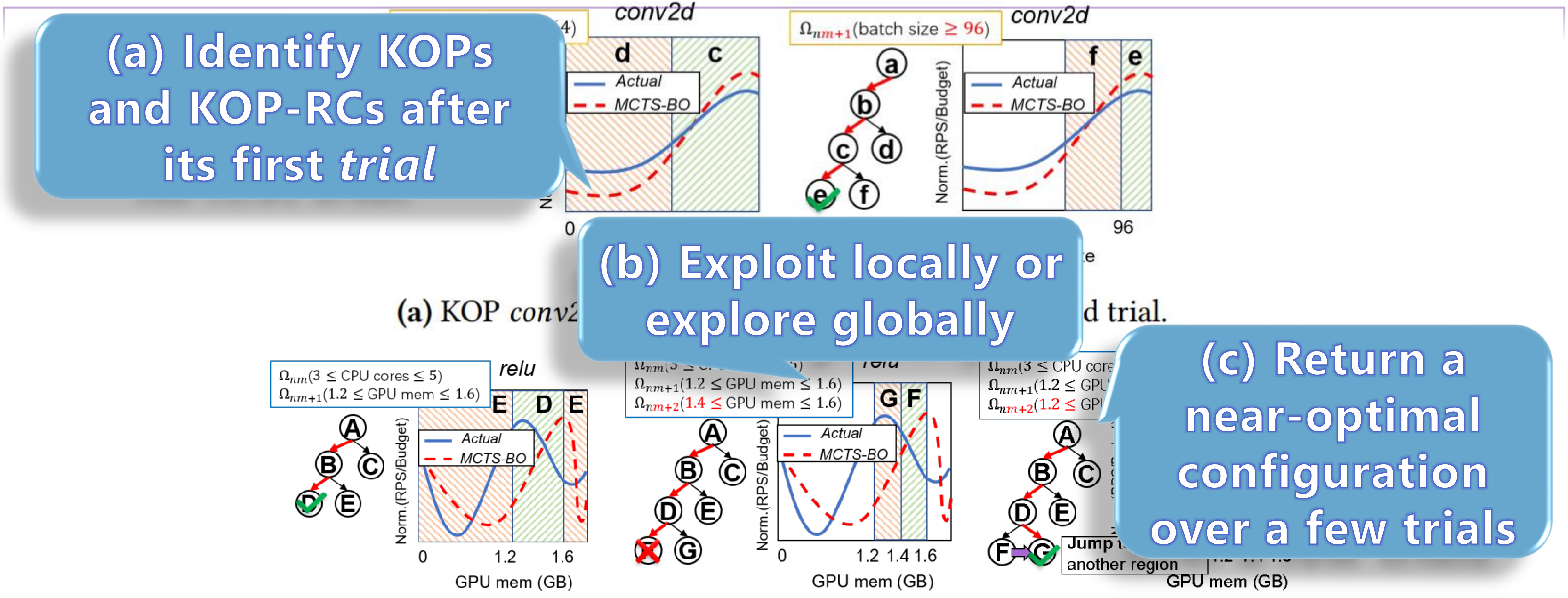
- **Reuse offline dataset:** it can reuse optimal configurations from offline dataset.
- **Balance exploitation and exploration:** it applies *go-left* strategy to exploit local tree regions, or applies *go-right* strategy to explore global tree regions.



MCTS-BO reuses offline datasets and searches them in different strategies

3. Fast adaptive search via Monte Carlo Tree Search and Bayesian Optimization (MCTS-BO)

Suppose an unseen DL model contains two KOPs *conv2d* and *relu*, MCTS-BO searches as follows:



(a) KOP conv2 … d trial.

(c) KOP *relu* the 1st trial. (d) KOP *relu* the 2nd trial. (e) KOP *relu* the 3rd trial.

Evaluation

DL models: 30 typical DL models of CNN, RNN, Bert, Transformer, and GAN.
Configuration search space: 1,440 configuration candidates.
Baselines: Morphling@SoCC'21, Vesta@ICPP'21, HeterBO@IPDPS'20, Ernest@NSDI'16.

DL models: the *source* set for offline profiling, and the *target* set for online searching.

	No.	Name		No.	Name
Source set	1	ResNet-152	Target set	19	MobileNet V2
	2	DenseNet-121		20	VGG 16
	3	WGAN		21	ResNet-101
	4	DCGAN		22	ResNet152 V2
	5	SGAN		23	Inception V2
	6	MobileNet V3		24	Inception V4
	7	Inception-ResNet V2		25	DenseNet-201
	8	Inception V3		26	Bert-large
	9	VGG19		27	GRU
	10	Fast-RCNN		28	RoBERTa
	11	Bert-base		29	Transformer
	12	NCF		30	Tacotron2
	13	DCN			
	14	DRN			
	15	NasNet-large			
	16	LSTM			
	17	EfficientNet-widese-b4			
	18	YOLO V5			

Configurations

- CPU cores: 1, 2, 3, 4, 5.
- GPU type: M60, T4, K80, V100.
- GPU memory (GB): 0.8, 1.2, 1.6, 2.4.
- Batch size: 4, 8, 16, 32, 64, 128.
- GPU power [11]: 50%, 75%, 100%.

Baselines

Morphling: it employs meta-learning and BO to reuse the model-level similarity.

Vesta: it leverages transfer learning to reuse historical data.

HeterBO: it provides heuristic rules to reuse prior features of other models.

Ernest: it abstracts and reuses patterns for different models.

Evaluation

Metrics: search accuracy, search overhead and practical benefits.

Search accuracy: the performance gap between the *recommended* and the *optimal* configurations

Search overhead: the number of *trials* & the wall-clock time of running these *trials*

Practical benefits: maximize normalized *request per budget*, or norm.(RPS/Budget)

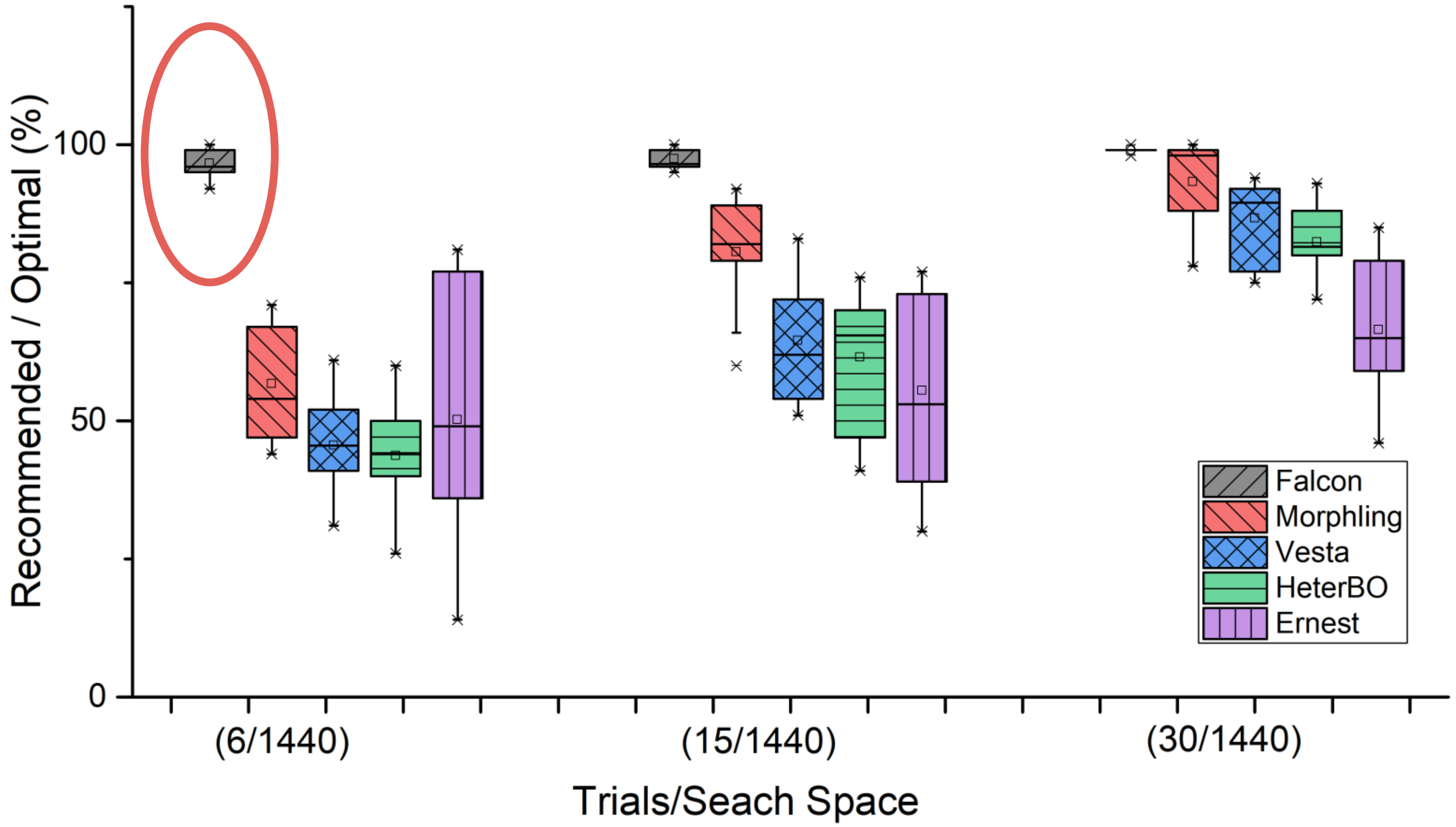
We evaluate Falcon by the following **experiment design**:

- **Effectiveness:** (1) alleviating the cold start problem, (2) apple-to-apple comparisons, (3) model-by-model comparisons, and (4) searching in good regions.
- **Robustness:** (5) and (6) applying different parameters to the methods used by Falcon.
- **Practical benefits for real-world applications:** (7) evaluating the practical benefit of recommending near-optimal configurations by using an enterprise-level DL benchmark [1].

[1] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, et al. 2020. Mlperf inference benchmark. In 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA). IEEE, 446–459.

Evaluation 1: Alleviating the cold start problem

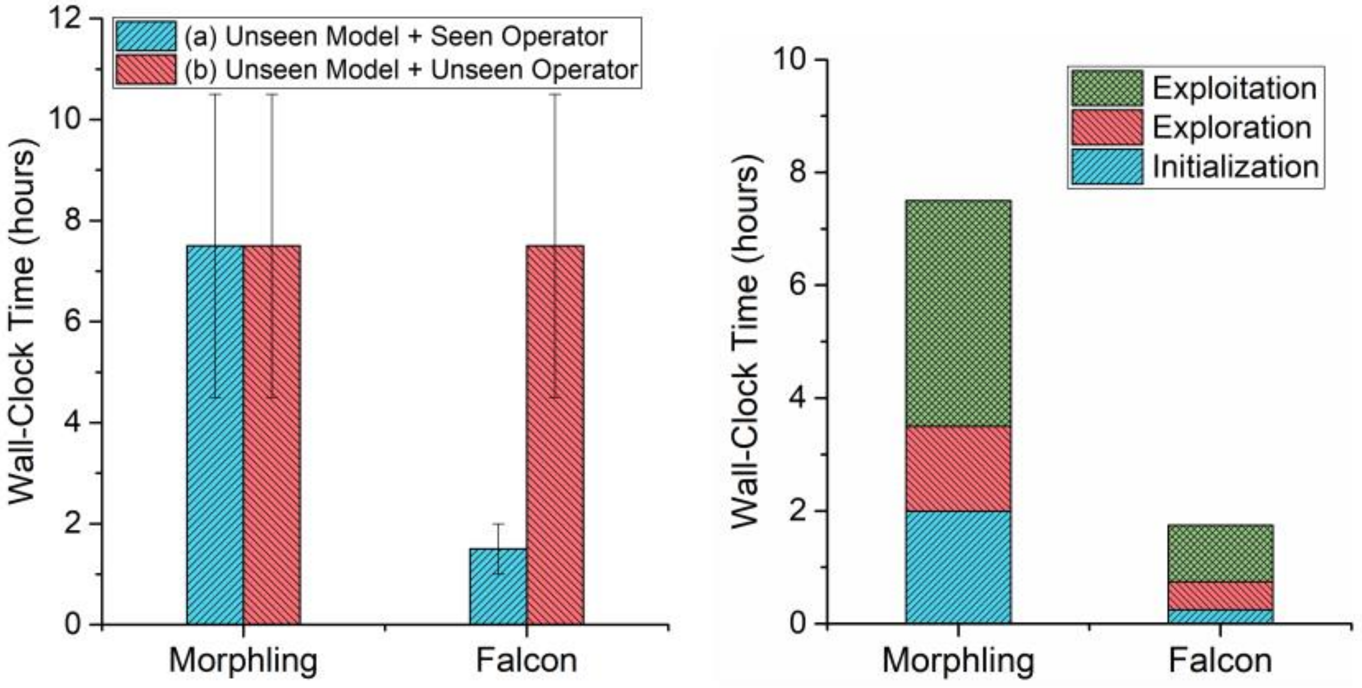
Comparison of different number of *trials*. The X-axis shows 6, 15, 30 *trials*, respectively. The Y-axis shows the search accuracy of the DL models in the *target* set.



For unseen DL models, only Falcon can find near-optimal configurations after 6 *trials*

Evaluation 2: Apple-to-apple comparison with Morphling

(a) Comparing the wall-clock time of the search in two cases.
(b) Comparing end-to-end time cost of the online search phase.

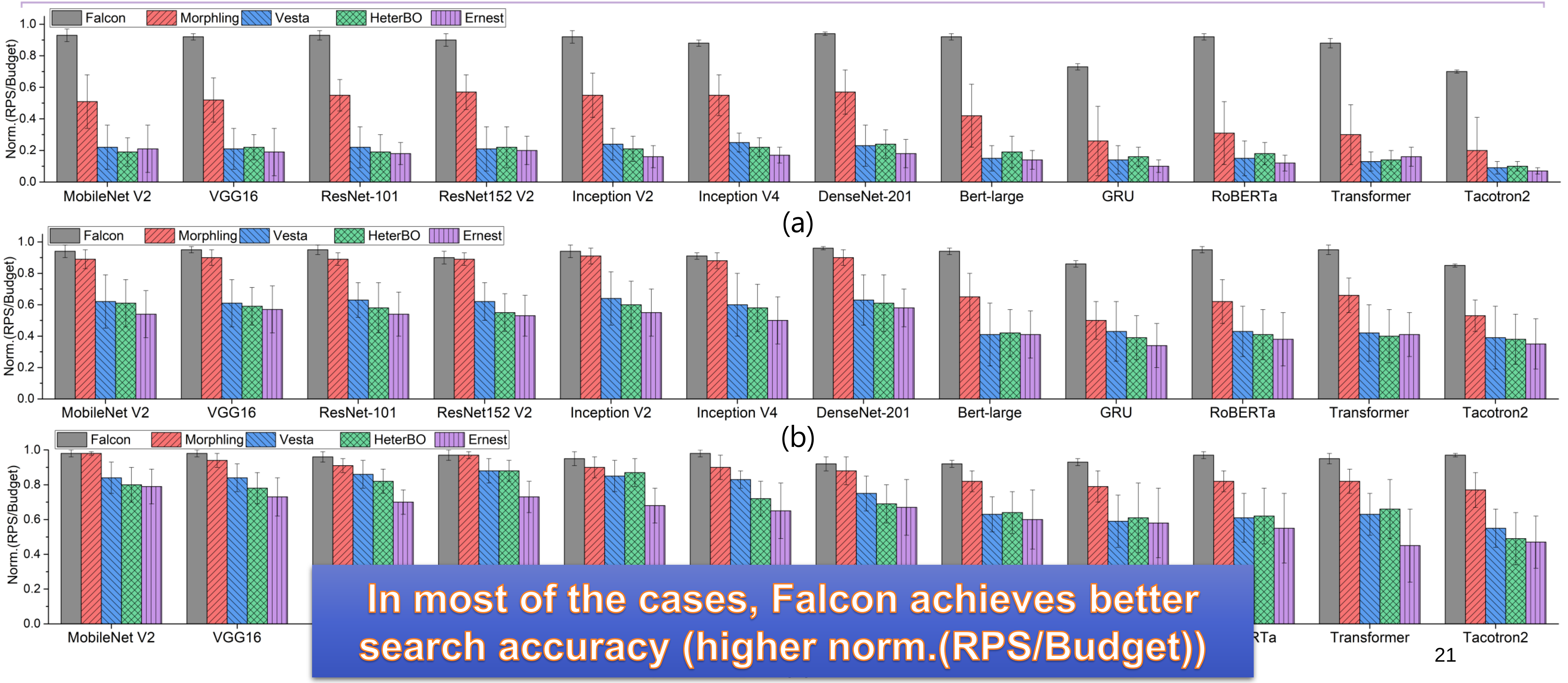


(a) Wall-clock time in two cases. **(b)** Wall-clock time of the online search phase.

Falcon can reduce up to 80% of search overhead by taking full advantage of KOPs and KOP-RCs

Evaluation 3: Model-by-model configuration optimization

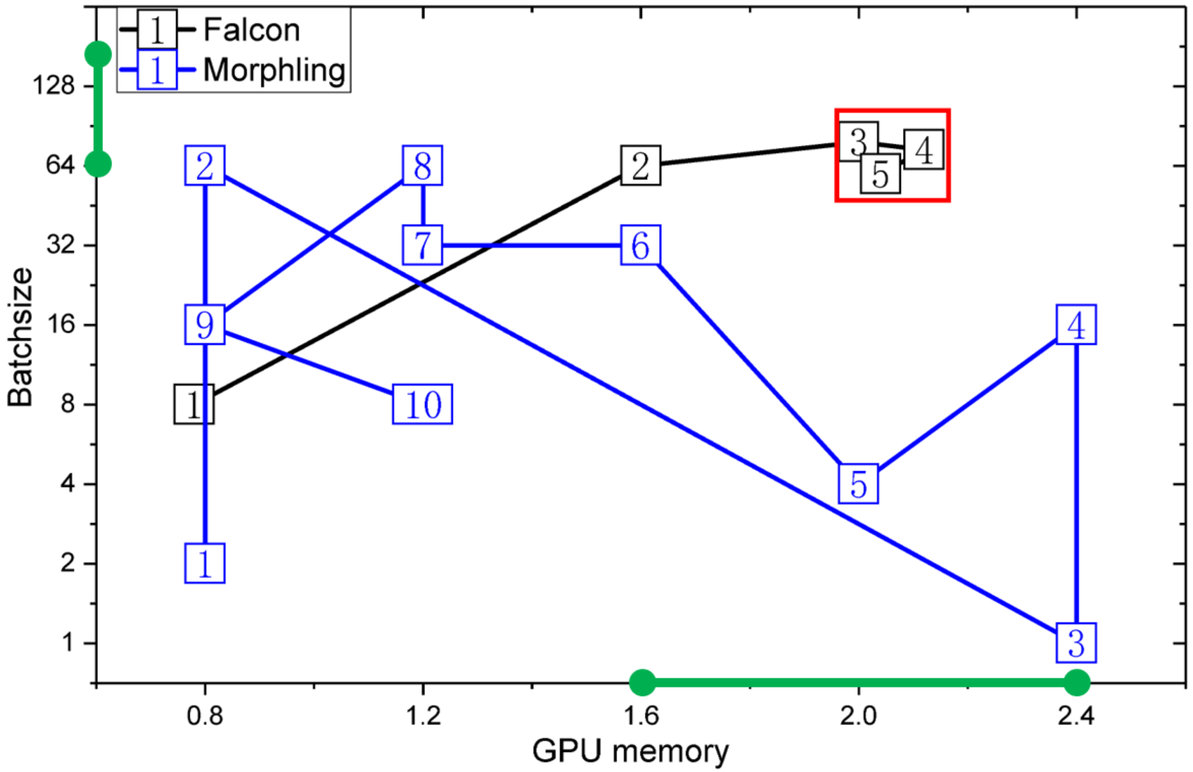
(a) Evaluating norm.(RPS/Budget) after 1.5 hours (six trials). (b) Evaluating norm.(RPS/Budget) after 3 hours and 45 minutes (15 trials). (c) Evaluating norm.(RPS/Budget) after 7.5 hours (30 trials).



In most of the cases, Falcon achieves better search accuracy (higher norm.(RPS/Budget))

Evaluation 4: Searching in good regions

Comparison of the search path for an unseen DL model. The number in the plot shows the xth trial. The green solid lines highlight good search regions. The red box highlights near-optimal configurations.

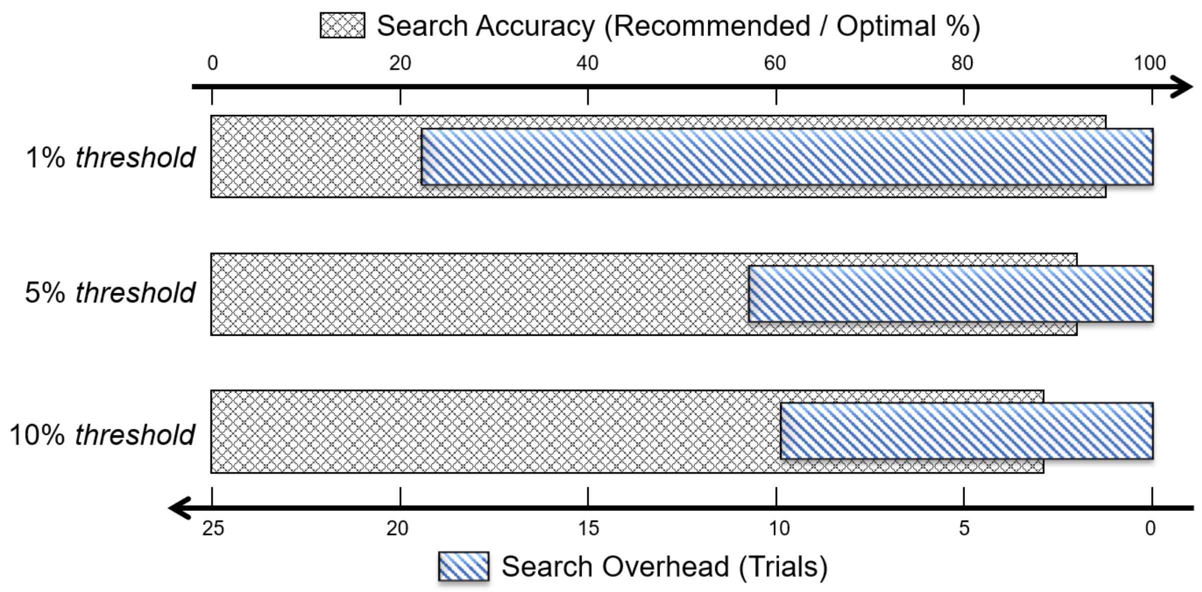


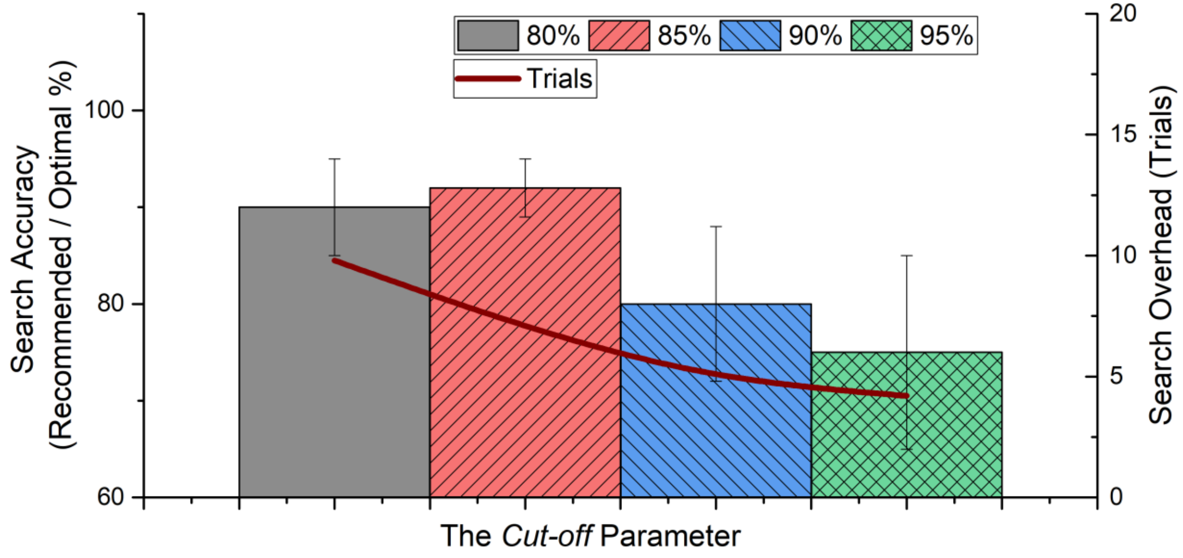
Falcon can quickly locate near-optimal search regions via MCTS-BO

Evaluation 5&6: Robustness

Evaluation 5: Tuning the *threshold* parameter to balance search accuracy and search overhead.

Evaluation 6: Tuning the *cut-off* parameter in PCA for pruning redundant configurations.

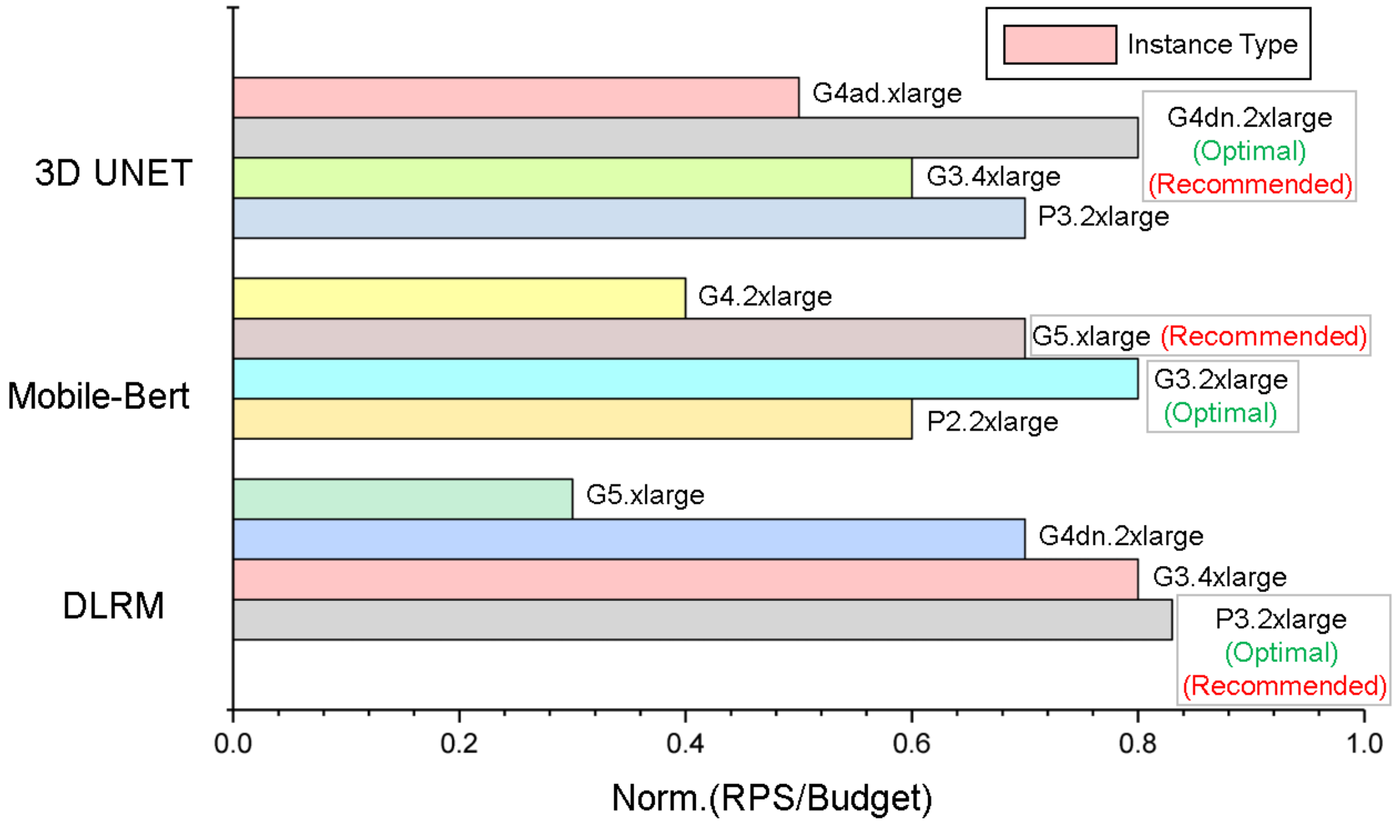




Falcon can strike a balance between search accuracy and search overhead

Evaluation 7: Practical benefits

Practical benefits of applying recommended configurations for three benchmark applications. The optimal configurations were found by exhaustive search.



Falcon can recommend an optimal (or a near-optimal) instance type to achieve high norm.(RPS/Budget)

Conclusion

Falcon is a novel CR system that can quickly adapt to unseen DL models. The main insight is that Falcon presents a new perspective to alleviate the cold start problem by leveraging Key Operators (KOPs) and Key Operator Resource Curves (KOP-RCs).

- **Learning KOPs and KOP-RCs:** Falcon launches a large-scale evaluation to cover typical DL models, 1,000+ configuration candidates and all types of DL operators.
- **Representing KOPs and KOP-RCs:** Falcon handles the complex relationship between DL models, KOPs and KOP-RCs in trees, and distinguishes good and bad search regions.
- **Fast adaptive searching via MCTS-BO:** Falcon makes a balance between exploitation and exploration. As a result, it can search more quickly and accurately than other CR systems.

Limitation: The overhead will increase when there are unseen operators.

Falcon is now available at https://github.com/dos-lab/Falcon.

Thank you!
Any questions?

E-mail：wuyuewen@otcaix.iscas.ac.cn