Retiarii: A Deep Learning Exploratory-Training Framework

Authors: Quanlu Zhang, Zhenhua Han, Fan Yang, Yuge Zhang, Zhe Liu, Mao Yang, Lidong Zhou

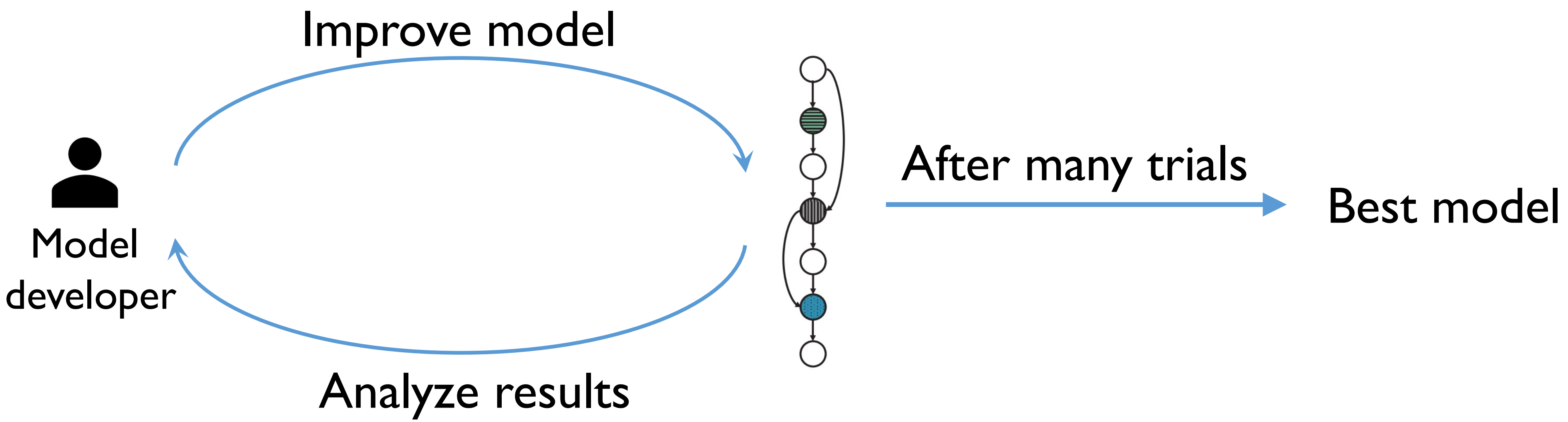
Members: Yibo Pi, Jiachen Liu and Qinye Li

Prevalence of Deep Neural Network (DNN)

- Success in perception-based tasks: vision and speech
- Widely used to empower many cloud/edge applications
- Important to design new DNN models

How DNNs Are Commonly Designed?

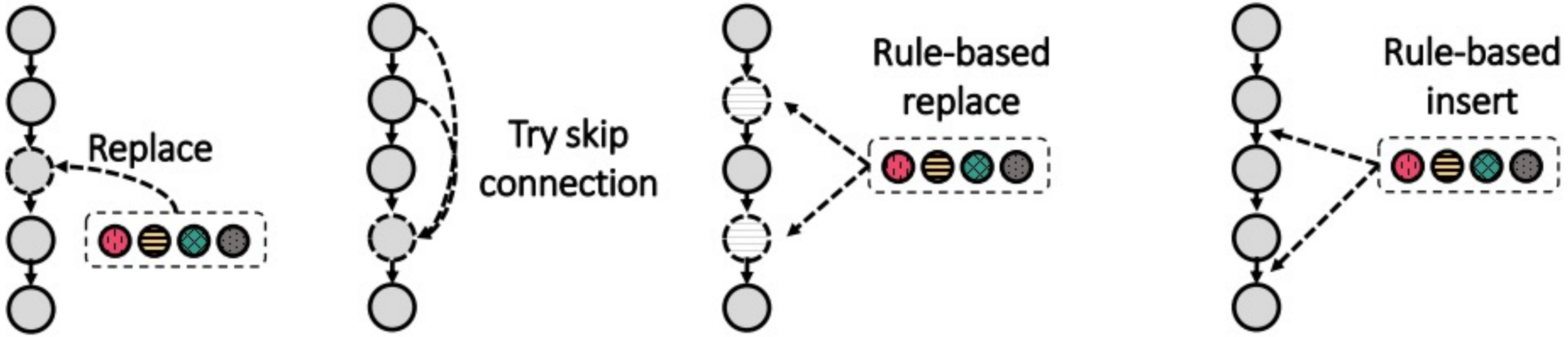
- Designing new DNNs is an exploratory process



This process is called exploratory-training.

Typical Types of Model Space Explorations

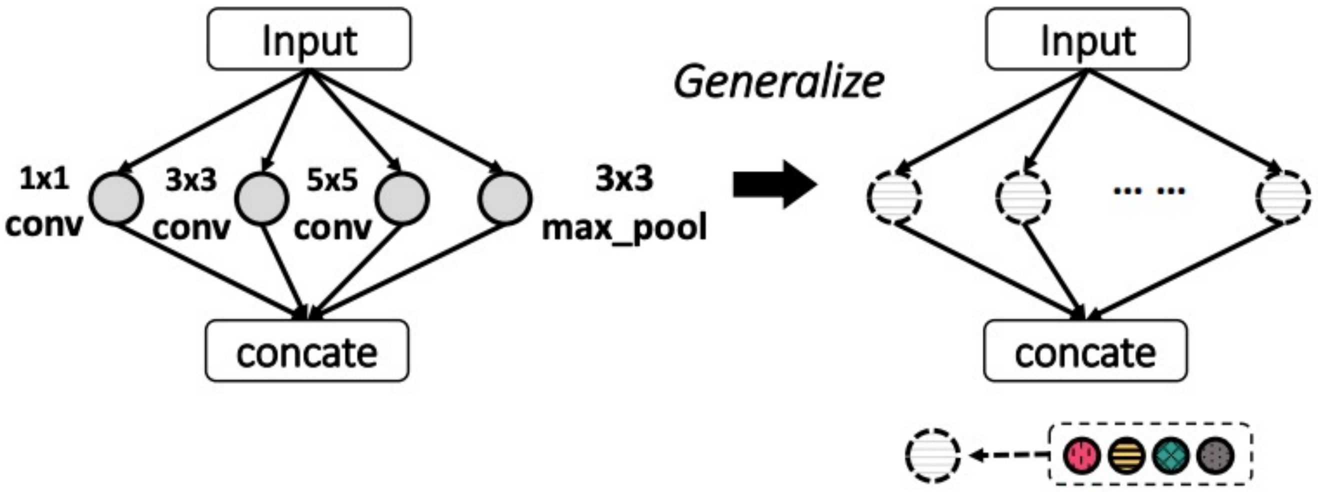
2. Adding a skip connection



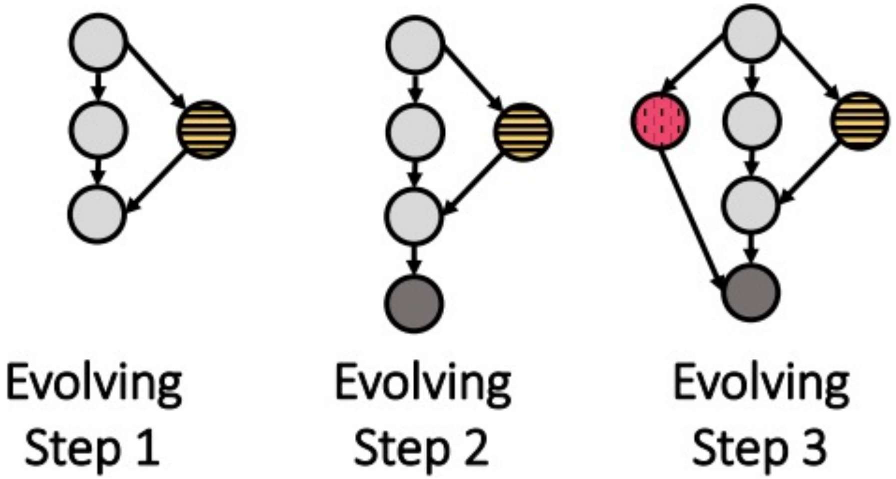
1. Replace an operator

3 & 4. Rule-based replacement and insertion, e.g., inserting *relu* between *conv* and *dense*

Typical Types of Model Space Explorations



5. Generalizing a cell structure to find a better one

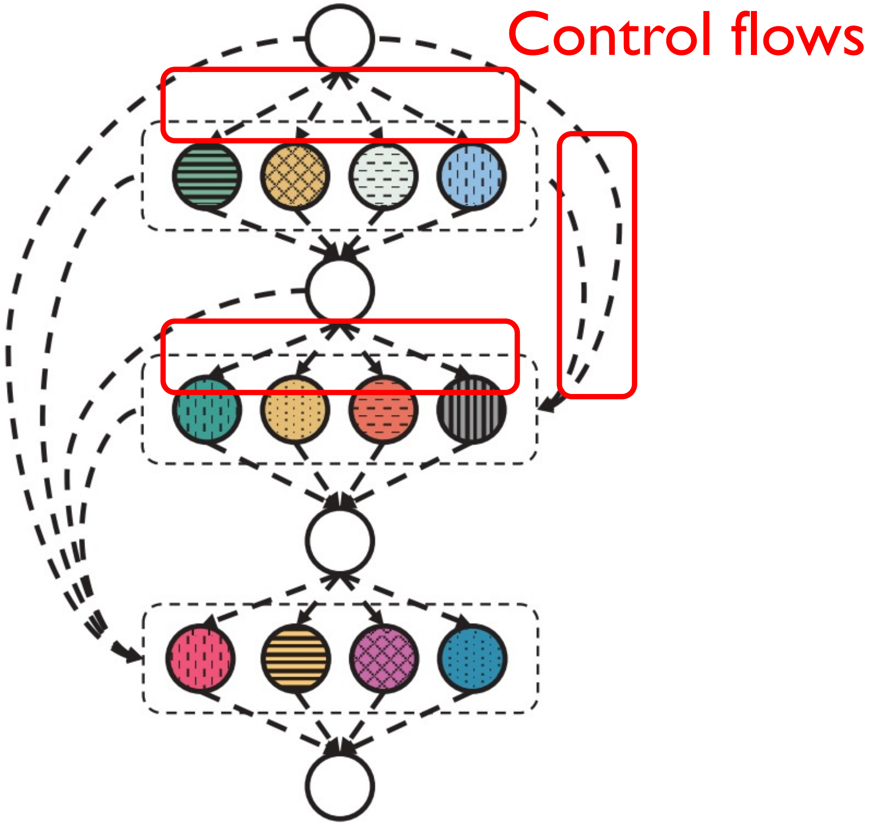


Evolving from a base model

Typical Implementation of Model Space Exploration

1. Specifying model space:
a jumbo model with many control flows

2. Exploring model space:
encoding an exploration strategy in the jumbo model



A coupling between model space and exploration strategy

Existing Ways to Achieve Exploratory-Training

- Manually try each new DNN model with deep learning frameworks, e.g., PyTorch

- Neural architecture search (NAS) algorithms or AutoML systems
  - **Lack of modularity**: coupling between model space, exploration strategy, and model training
  - **Lack of reusability**: model space and exploration strategy need to be parameterized

- Missed opportunities to exploit model similarities to speed up the exploration process

A Framework for Exploratory Training?

No framework → Deep learning framework → Exploratory-training framework

No framework

Program with libraries: C++, Matlab

Deep learning framework

Program DNN models easier and faster with Pytorch

Exploratory-training framework

Explore and choose the best model easier and faster

Retiarii: A Deep Learning Exploratory-Training Framework

- **Key idea**
  - Program the model space rather than single models
  - model space = base model + mutators (basic operations to modify models)

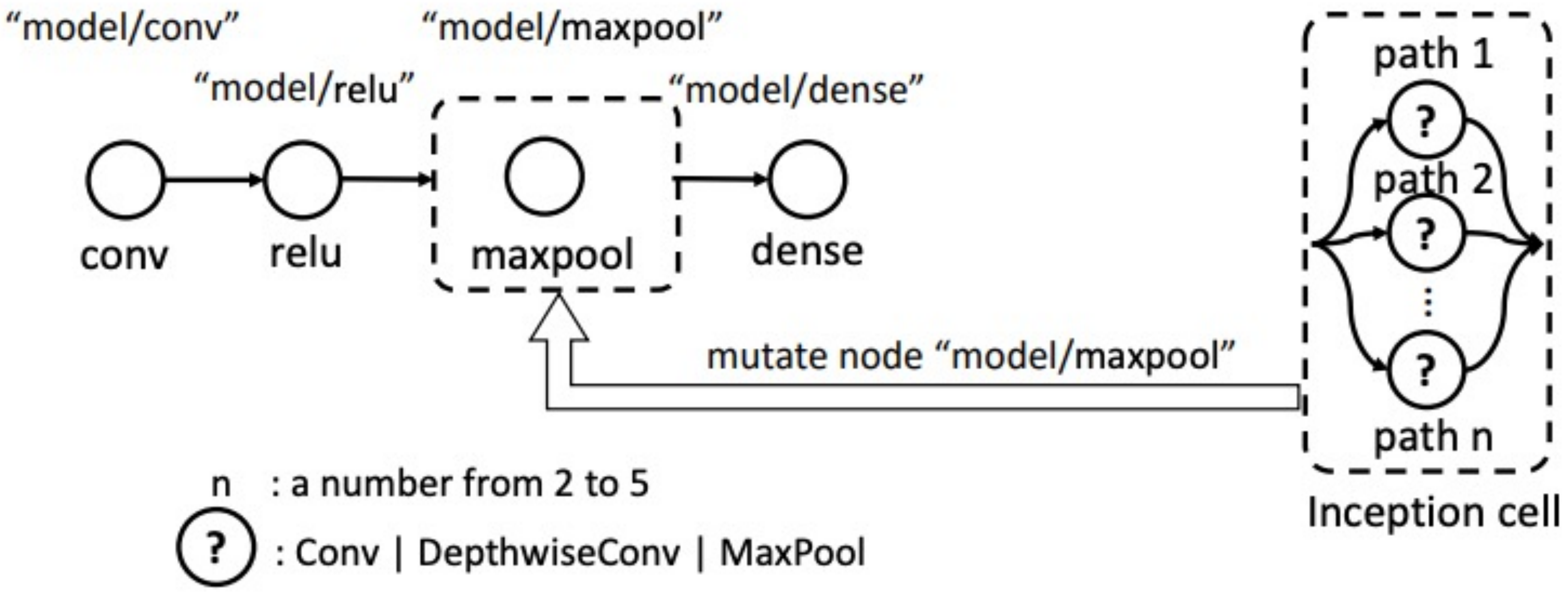
- **Mutator as the core abstraction**
  1. Defining arbitrary model space with mutators
  2. Decoupling model space from model exploration
     - Model exploration is a set of policies or rules
     - One model exploration strategy can be used for different model spaces
  3. Exposing correlations between models
     - Easy to analyze model similarities with mutators

Mutator-based Programming Paradigm

- Mutation primitives

```
create_node(name:str,op:Op,params:dict={})
delete_node(node:Node)
connect(src:NodeOutput,dst:NodeInput)
del_connect(src:NodeOutput,dst:NodeInput)
update_node(node:Node,op:Op=None,params:dict={},
            inputs:list=None)
choose(candidates:list,n_chosen:int=1,
       type:str="choice",ctx:dict=None)
```

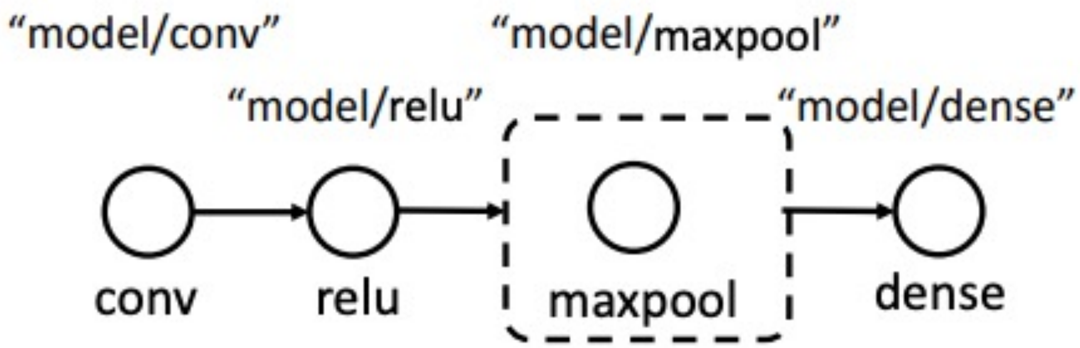
An Example of Mutating a Base Model



Replace the third node in this 4-node base model with an inception cell

An Example of Mutating a Base Model

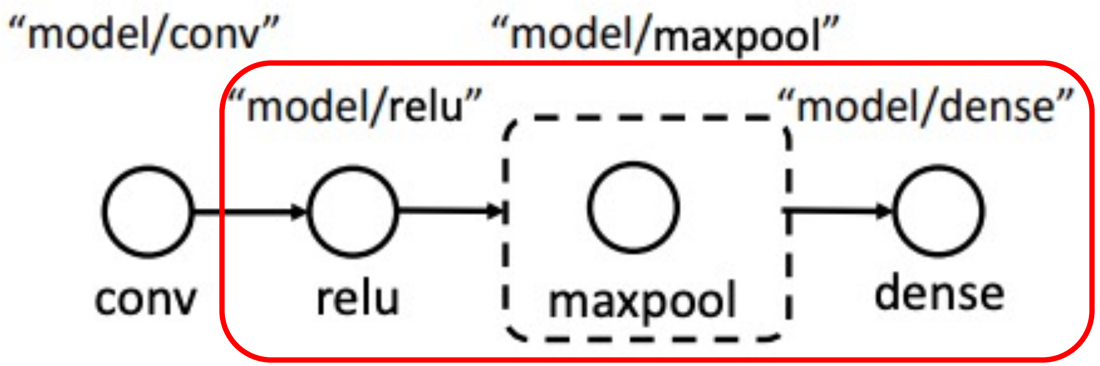
- Define the mutator




```
# define the graph mutation behavior
class InceptionMutator(BaseMutator):
  def __init__(self, paths_range, candidate_ops):
    self.paths_range = paths_range # [2, 3, 4, 5]
    self.ops = candidate_ops # {conv, dconv, ...}
  def mutate(self, targets):
```

An Example of Mutating a Base Model

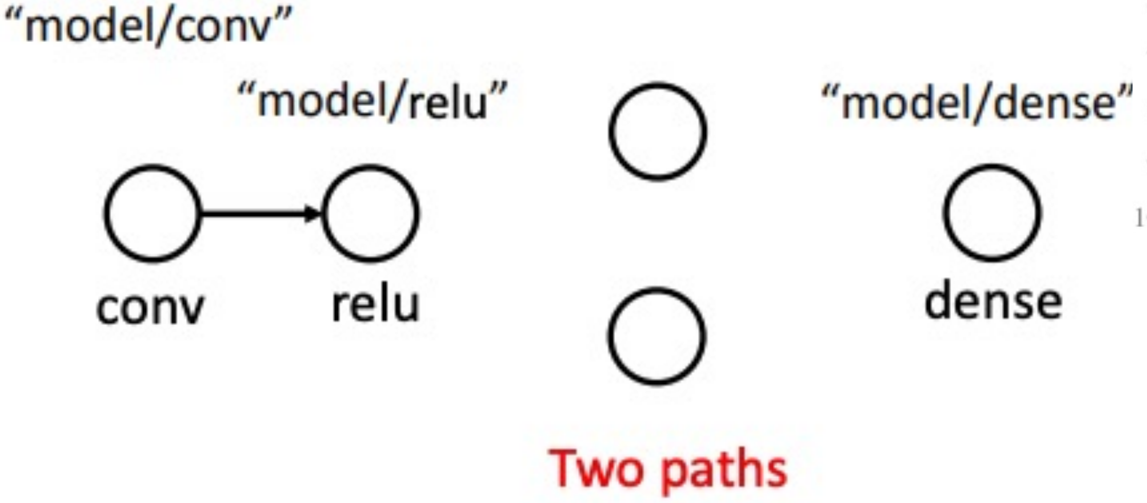
- Check the target node chain (relu, maxpool, dense)




```
# define the graph mutation behavior
class InceptionMutator(BaseMutator):
  def __init__(self, paths_range, candidate_ops):
    self.paths_range = paths_range # [2, 3, 4, 5]
    self.ops = candidate_ops # {conv, dconv, ...}
  def mutate(self, targets):
    if not three_node_chain(targets):
      return err
```

An Example of Mutating a Base Model

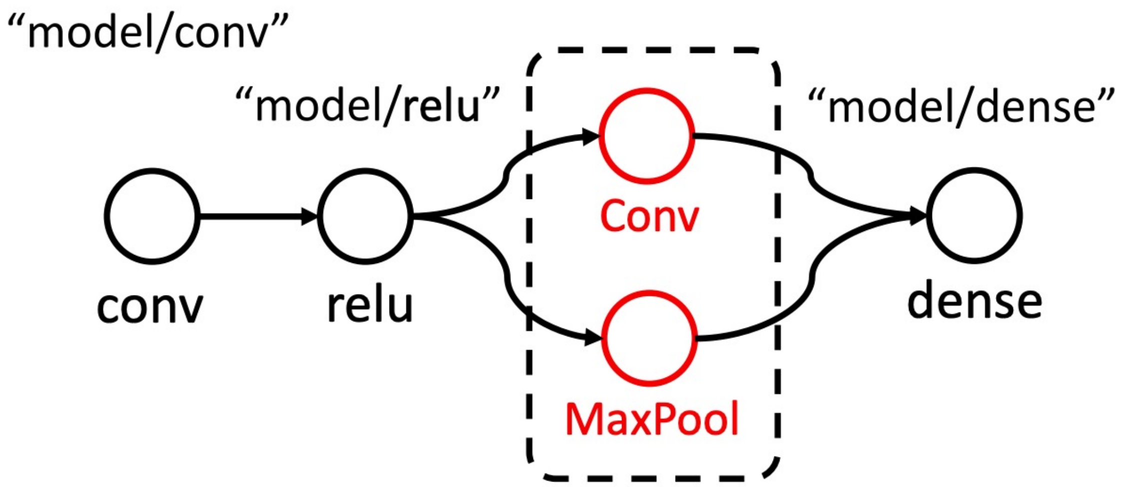
- Choose # of paths




```
# define the graph mutation behavior
class InceptionMutator(BaseMutator):
  def __init__(self, paths_range, candidate_ops):
    self.paths_range = paths_range # [2, 3, 4, 5]
    self.ops = candidate_ops # {conv, dconv, ...}
  def mutate(self, targets):
    if not three_node_chain(targets):
      return err
    n = choose(candidates=self.paths_range)
    delete_node(targets[1])
```

An Example of Mutating a Base Model

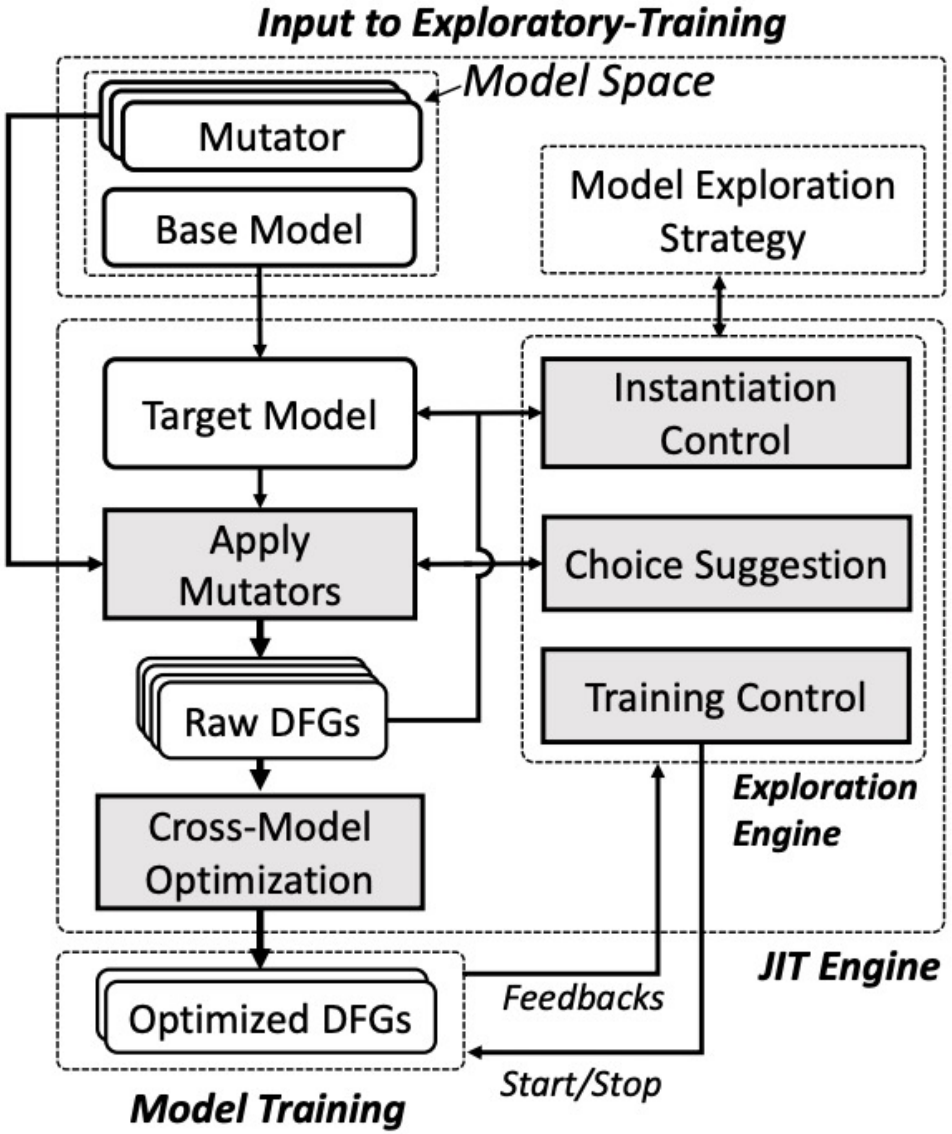
- Creating paths




```
# define the graph mutation behavior
class InceptionMutator(BaseMutator):
  def __init__(self, paths_range, candidate_ops):
    self.paths_range = paths_range # [2, 3, 4, 5]
    self.ops = candidate_ops # {conv, dconv, ...}
  def mutate(self, targets):
    if not three_node_chain(targets):
      return err
    n = choose(candidates=self.paths_range)
    delete_node(targets[1])
    for i in range(n): # create n paths
      op = choose(candidates=self.ops)
      nd = create_node(name='way_'+str(i), op=op)
      connect(src=targets[0].output, dst=nd.input)
      connect(src=nd.output, dst=targets[2].input)
```

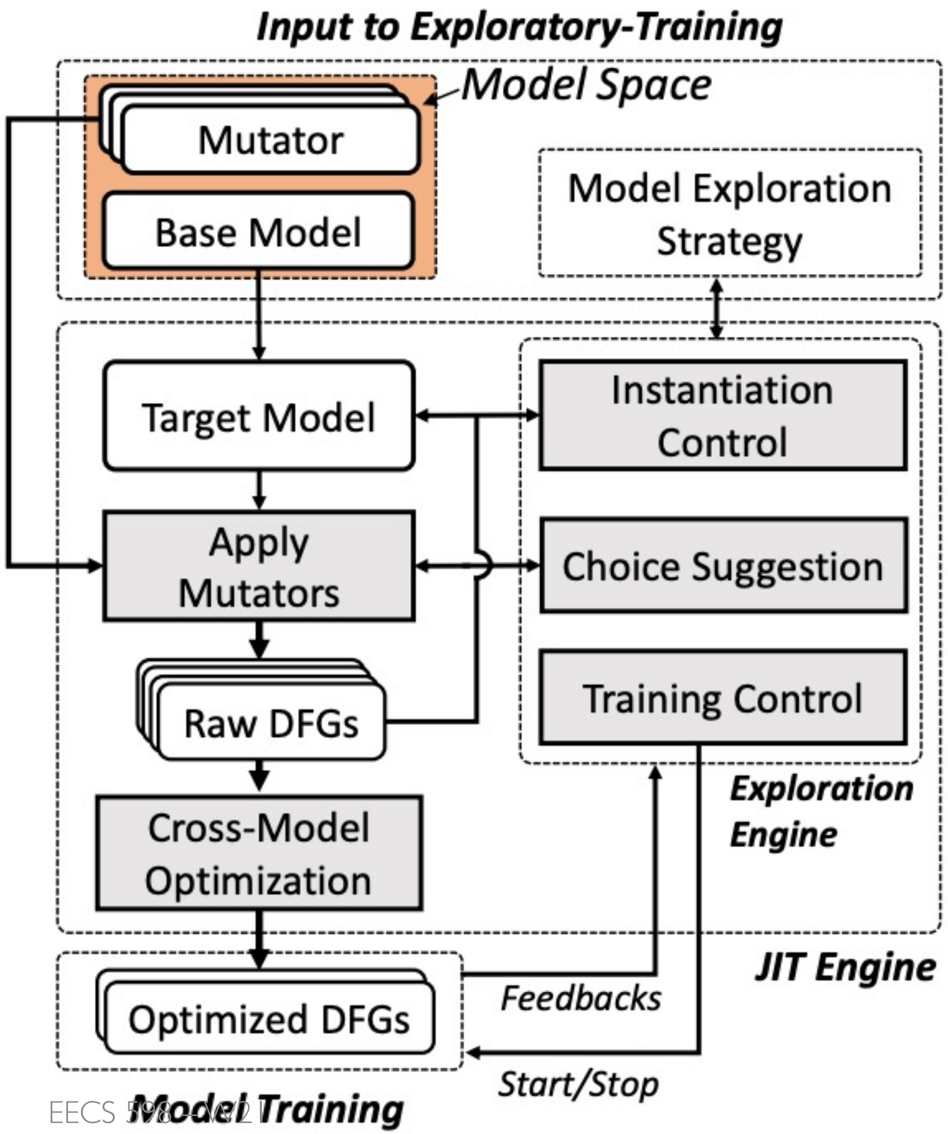
System Architecture

- Inputs
  - One or more base models
  - A set of mutators
  - A policy describing the exploration strategy



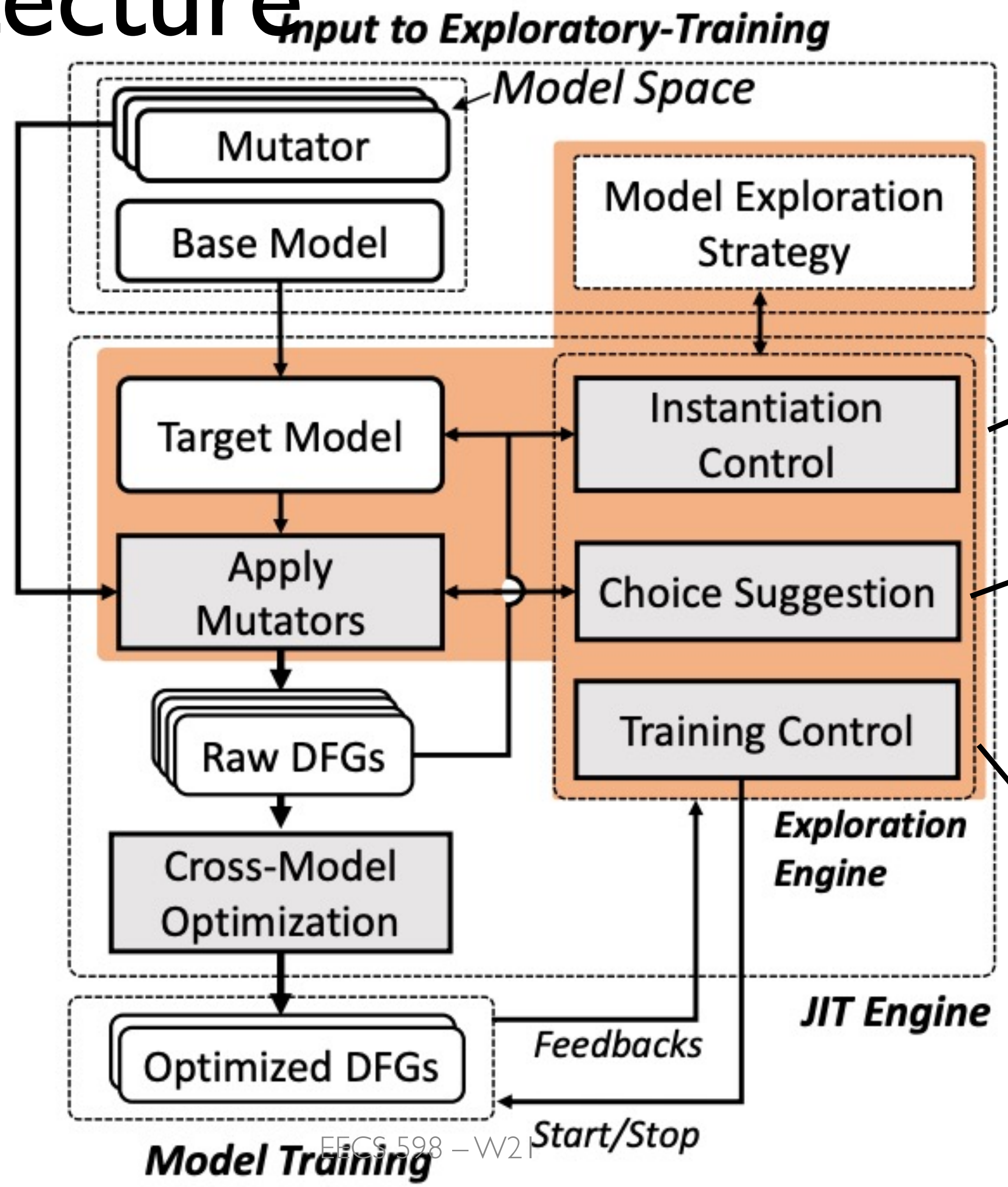
System Architecture

- Model space
  - Mutator
  - Base models



System Architecture

- Exploration engine
  - Instantiation control
  - Choice suggestion
  - Training control



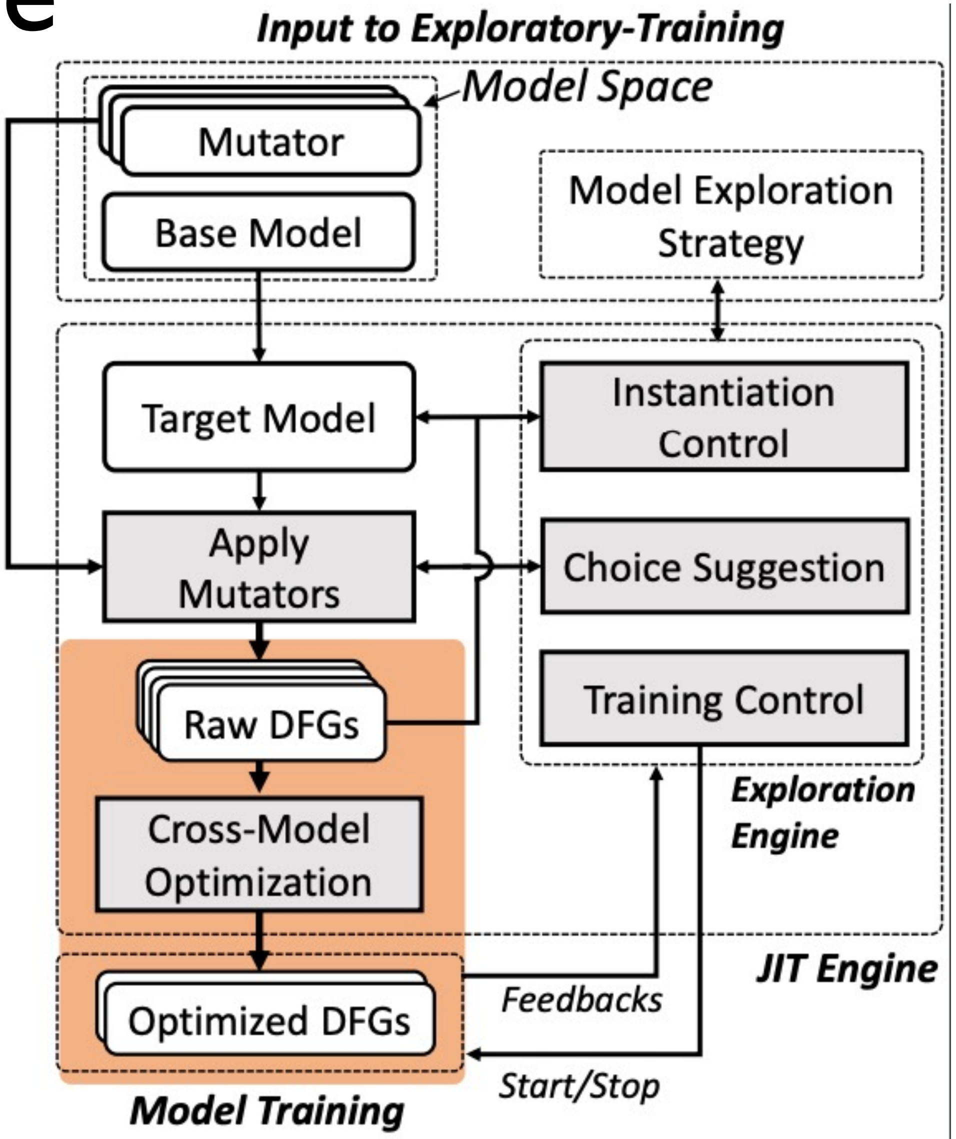
Which target model & mutators to use

Strategy for different combinations of mutators

Monitor training, collect feedbacks, and control priorities and resource allocation

System Architecture

- **Model training**
  - Optimized data-flow graphs



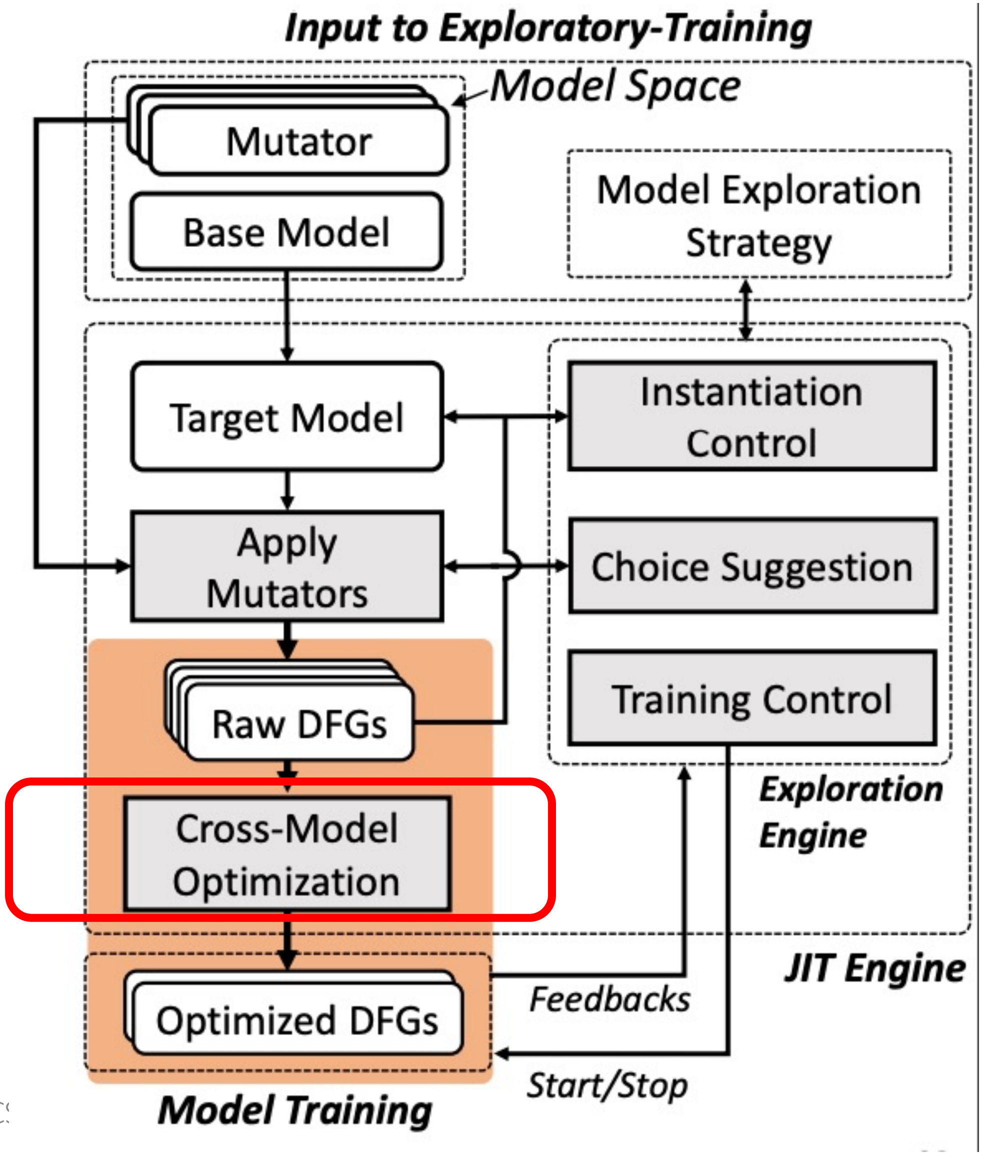
Expressiveness and Reusability

- Retiarii currently supports 27 neural architecture search (NAS) solutions

NAS Solution	Model Space	Exploration Strategy	Required Mutator Class			
			Input Mutator	*Operator Mutator*	*Inserting Mutator*	*Customized Mutator*
MnasNet [59]	MobileNetV2-based space	Reinforcement Learning		✓	✓	
NASNet [70]	NASNet cell	Reinforcement Learning	✓	✓		
ENAS-CNN [50]	NASNet cell variant	Reinforcement Learning	✓	✓		
AmoebaNet [51]	NASNet cell	Evolutionary	✓	✓		
Single-Path One Shot (SPOS) [27]	ShuffleNetV2-based space	Evolutionary		✓		
Weight Agnostic Networks [23]	Evolving space w/ adding/altering nodes adding connections	Evolutionary		✓		✓
Path-level NAS [13]	Evolving space w/ replication and split	Reinforcement Learning				✓
TextNAS [62]	TextNAS space	Reinforcement Learning	✓	✓		
...	...	...	...	...	...	...

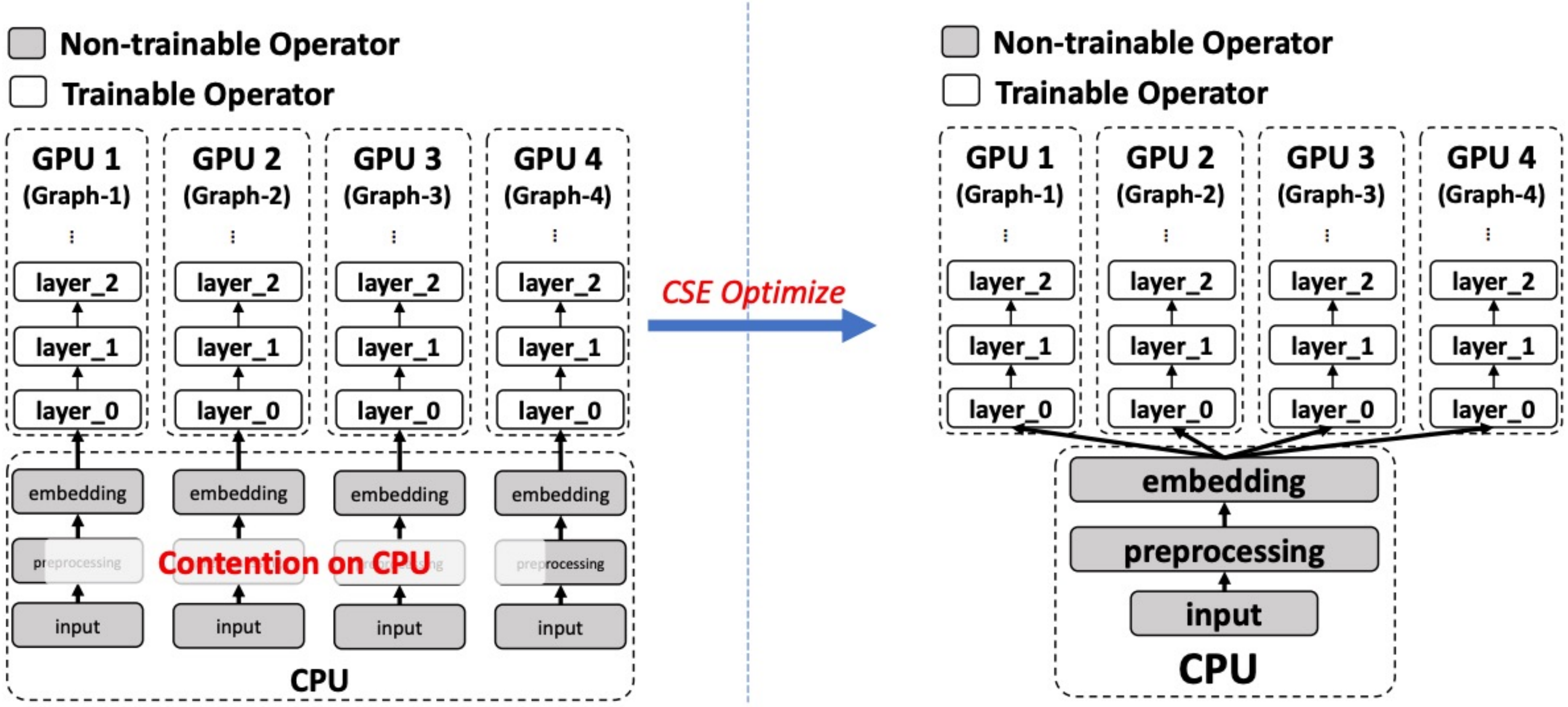
Cross-Model Optimization

- Optimization opportunities
  - Same training data
  - Same data preprocessing
  - Common layers
  - Weight sharing among models



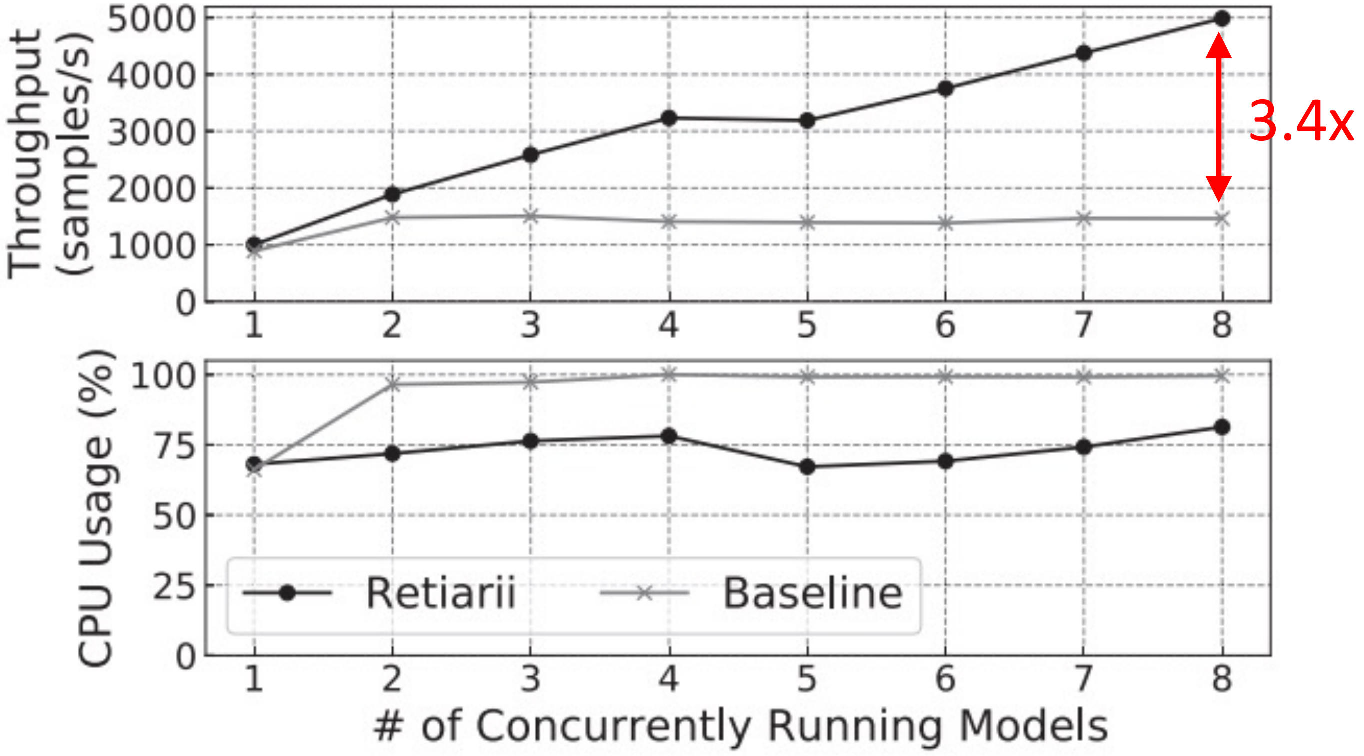
Common Sub-expression Elimination (CSE)

- De-duplicating CPU-based common prefix operations



Common Sub-expression Elimination (CSE)

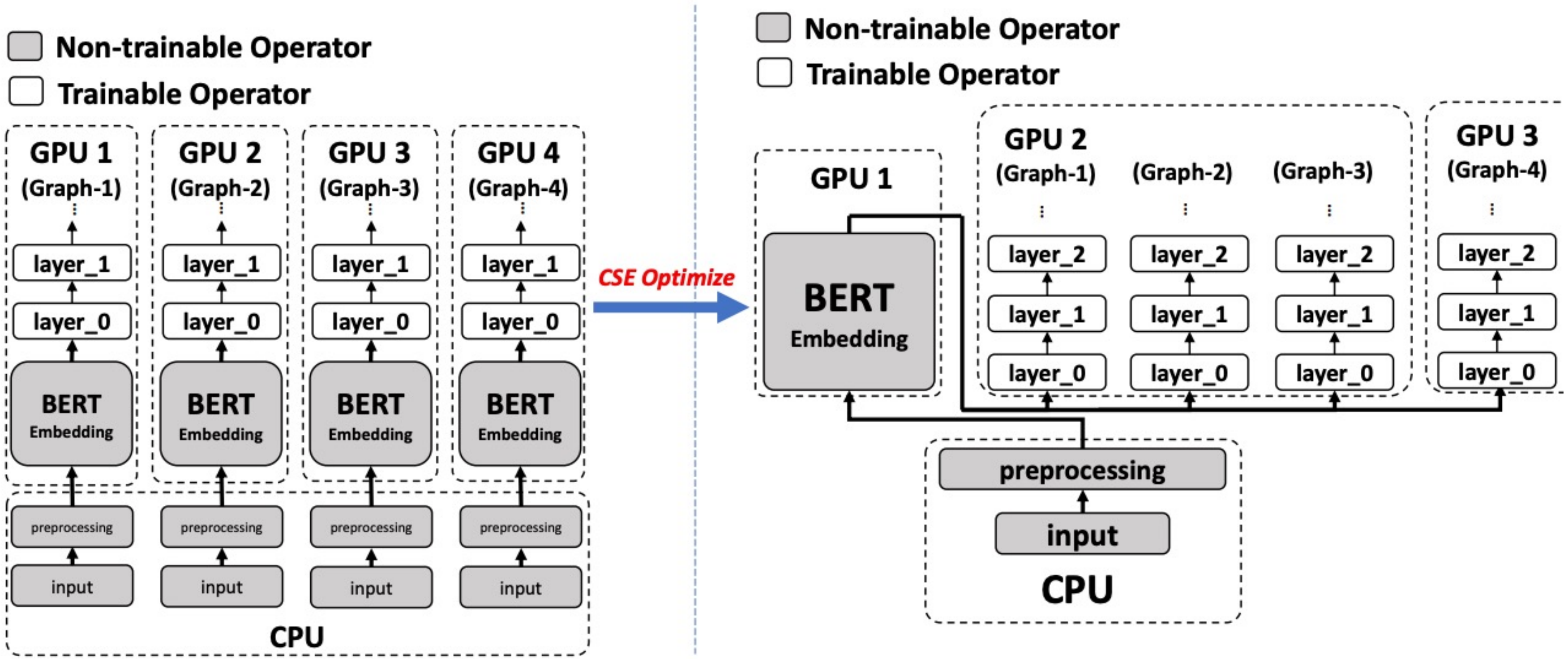
Experiment: training MnasNet0.5 on ImageNet with 4V100 GPUs and 20 CPU cores



CSE of CPU-based operation

Common Sub-expression Elimination (CSE)

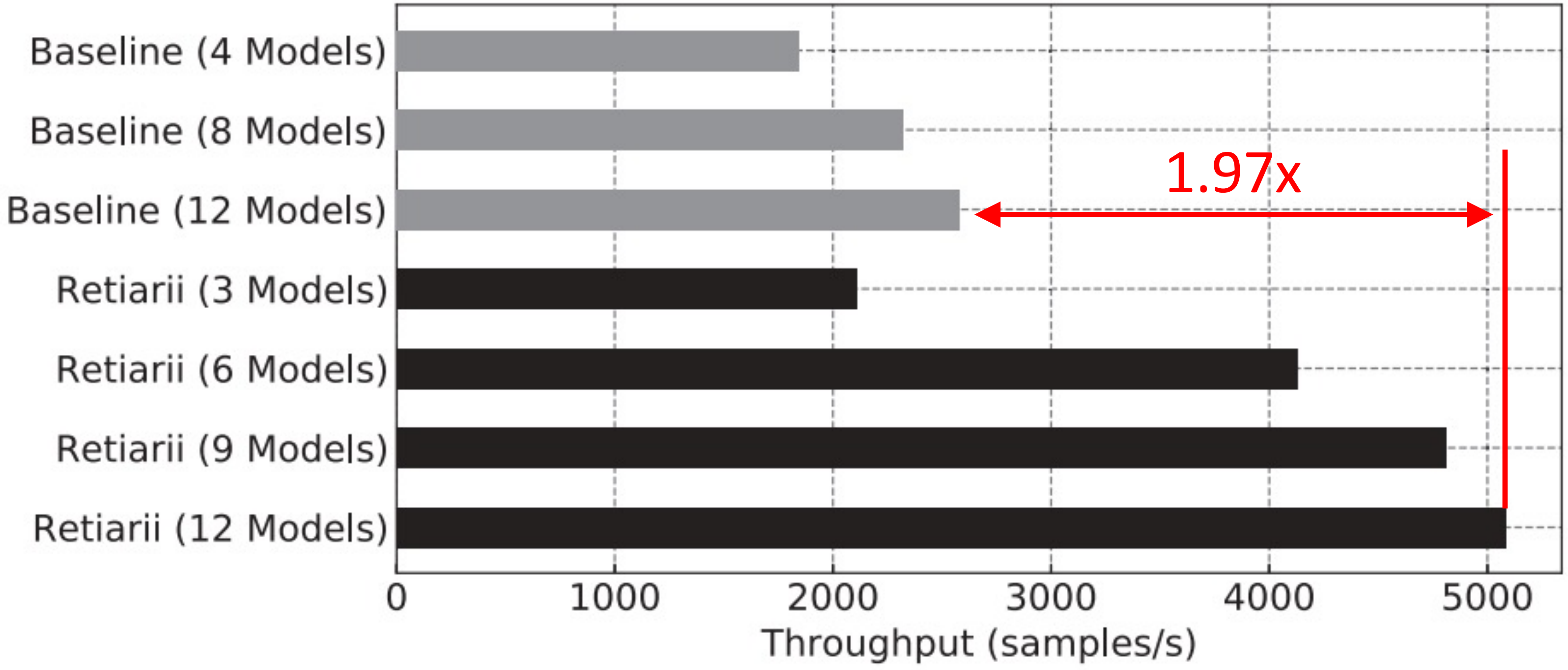
- CSE + device placement for GPU-based embedding



Dedicating one GPU for BERT-embedding improves pipeline and reduce memory consumption

Common Sub-expression Elimination (CSE)

Experiment: training TextNAS, one of the state-of-the art NLP models



Speeding up Neural Architecture Search (NAS)

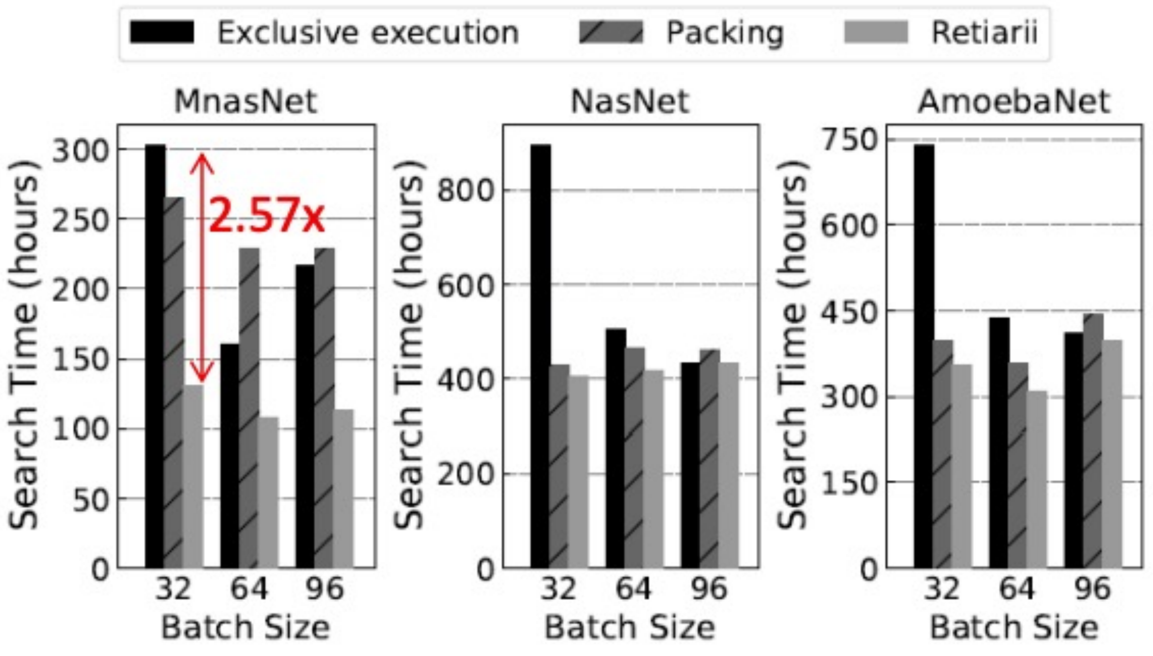
- Three popular NAS solutions

NAS Solution	Search Space	Exploration Strategy
MnasNet	Factorized Hierarchical Search Space	Reinforcement Learning
NASNet	Normal Cell + Reduction Cell	Reinforcement Learning
AmoebaNet	Normal Cell + Reduction Cell	Evolutionary Algorithm

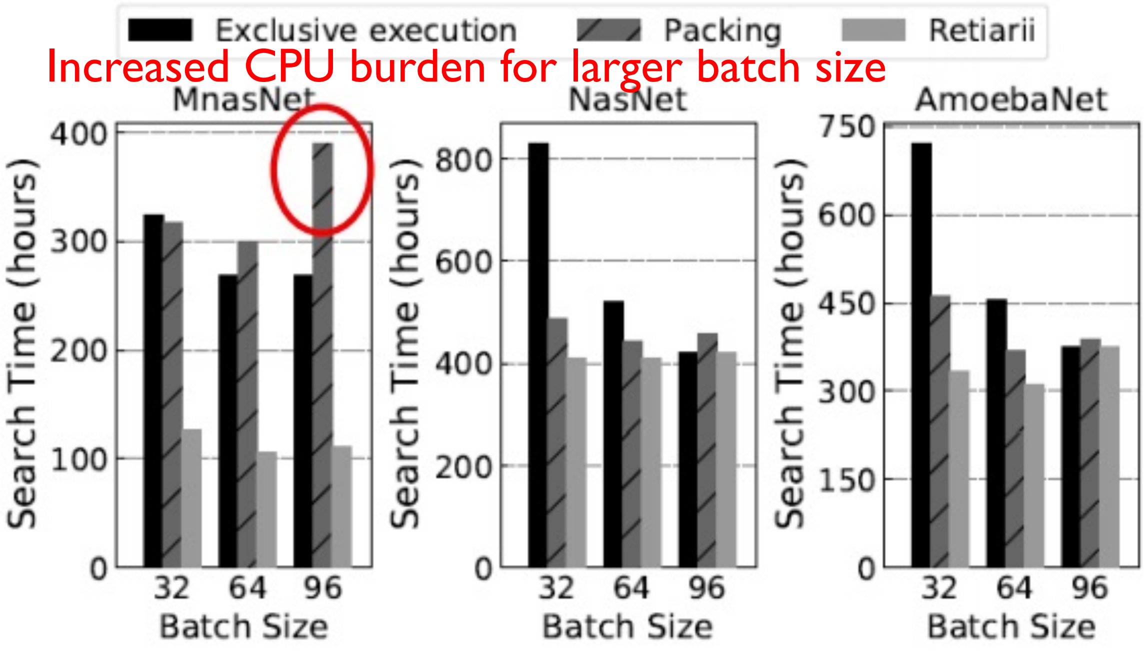
- Baselines
  - Exclusive execution: training one model per GPU at a time
  - Packing: training multiple models per GPU using NVIDIA CUDA MPS

Speeding up Neural Architecture Search (NAS)

- Running on 4 NVIDIA Tesla V100 GPUs
- Training 1000 models for 1 epoch on ImageNet



(a) NVIDIA Data Loading Library (DALI)



(b) PyTorch DataLoader

Conclusion

- Retiarii is a new DNN framework for exploratory training

- Retiarii provides new interfaces for model developers to explore new models efficiently

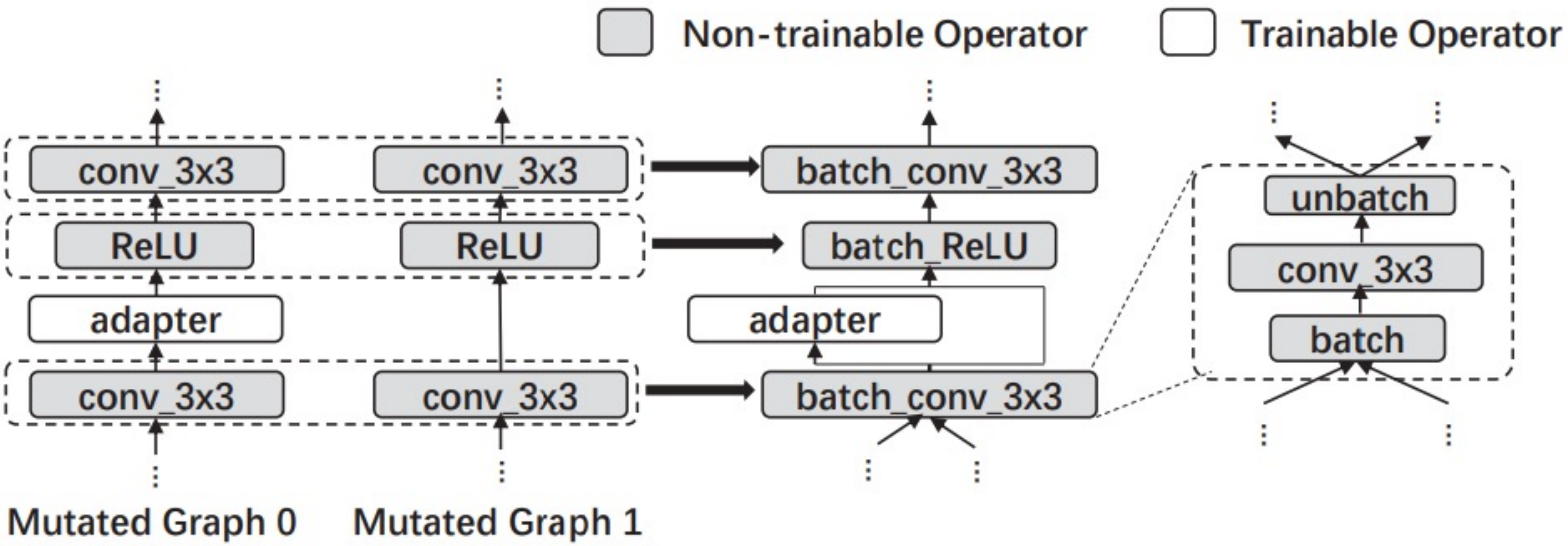
- Retiarii uses the Mutator abstraction to achieve
    - Strong expressiveness in model space
    - Reusability of exploration strategies
    - Cross-model optimization

Discussion

- **Limitations of Retiarii mentioned by the authors**
    1. Limited support to dynamic graphs
    2. Limited support to operator batching
    3. Possible shape mismatch between adjacent layers' input/output tensors

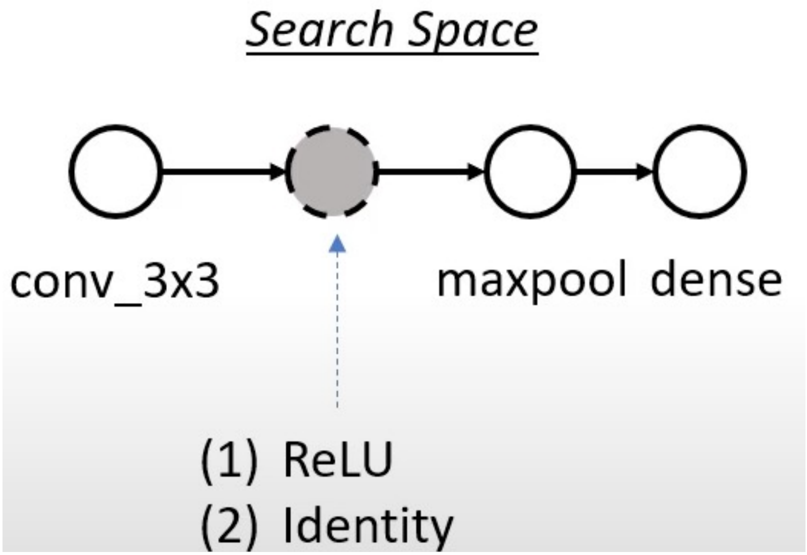
- **Other limitations:**
    1. A whitelist is used to identify operators requiring dedicated GPUs
    2. Retiarii greedily packs as many models as possible into one GPU
        - What if single models are too large to fit in to one GPU?
    3. What if there are significant mutations to the base model?
        - Is it easy for developers to manage these mutations?
        - How will cross-model optimization perform when models are very different?

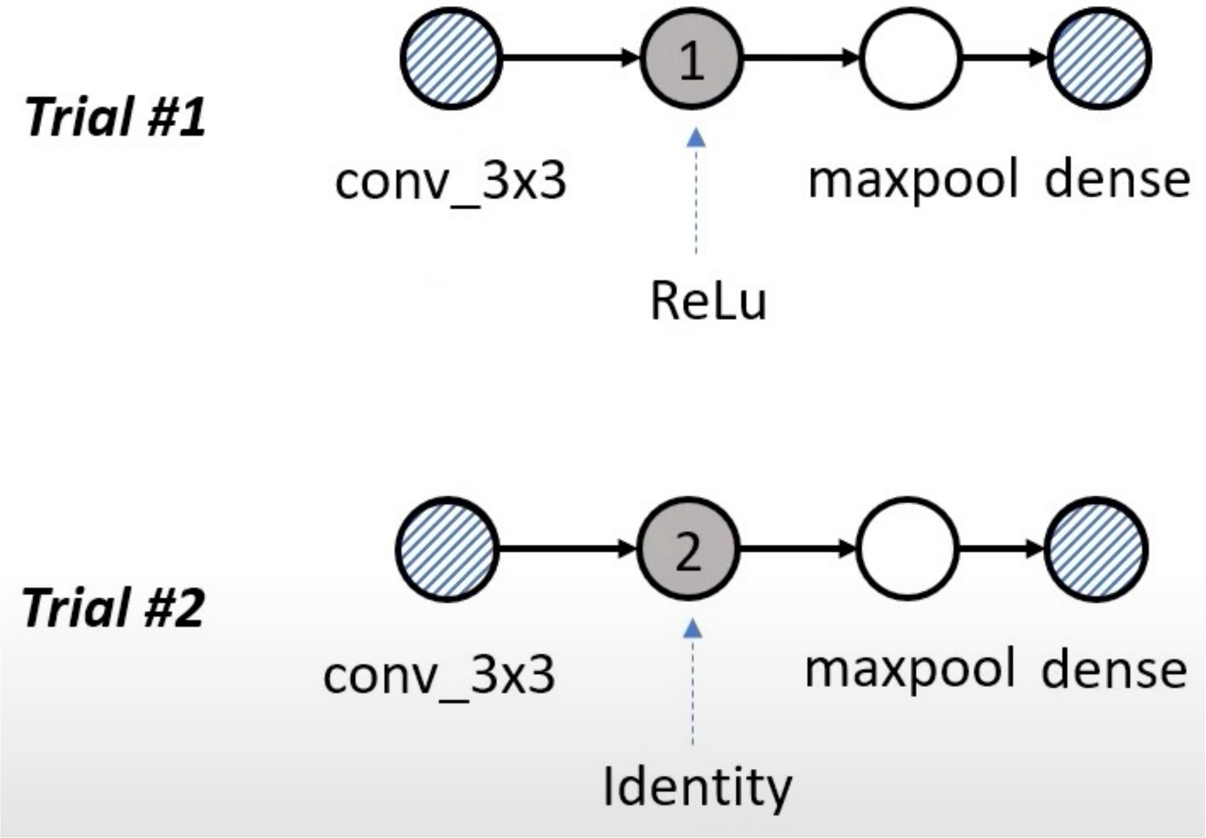
Operator Batching



Speed up Weight-Shared Training

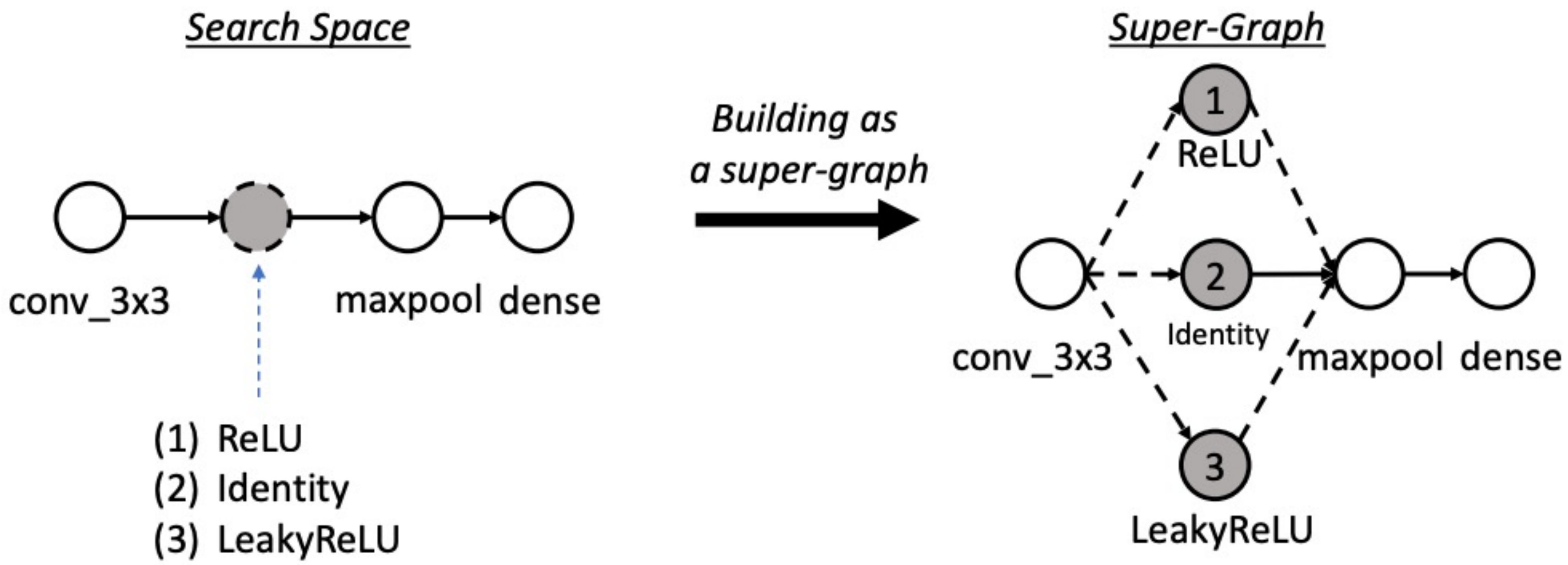
- What is weight sharing?





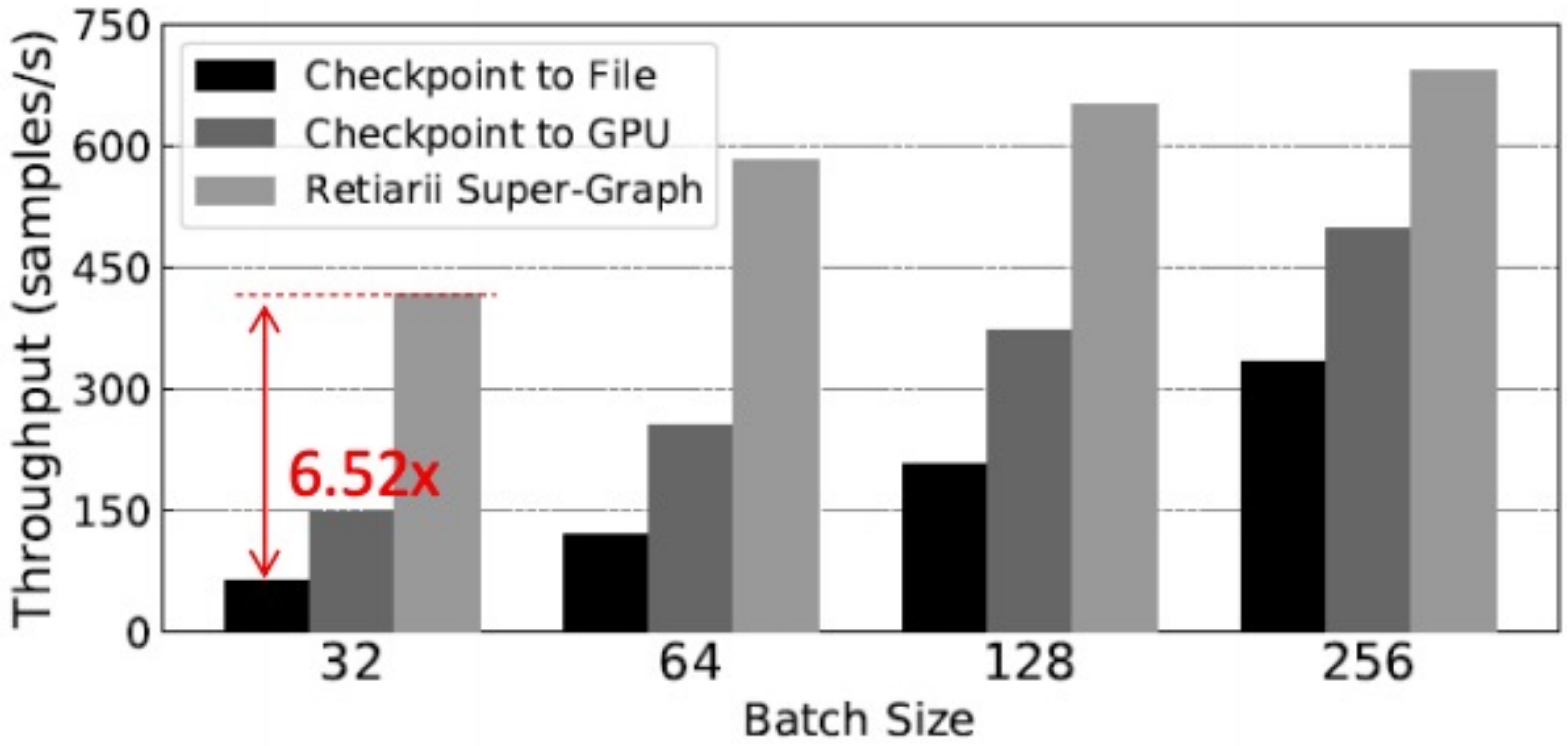
Speed up Weight-Shared Training

• Building a super-graph to encode the search space



Speed up Weight-Shared Training

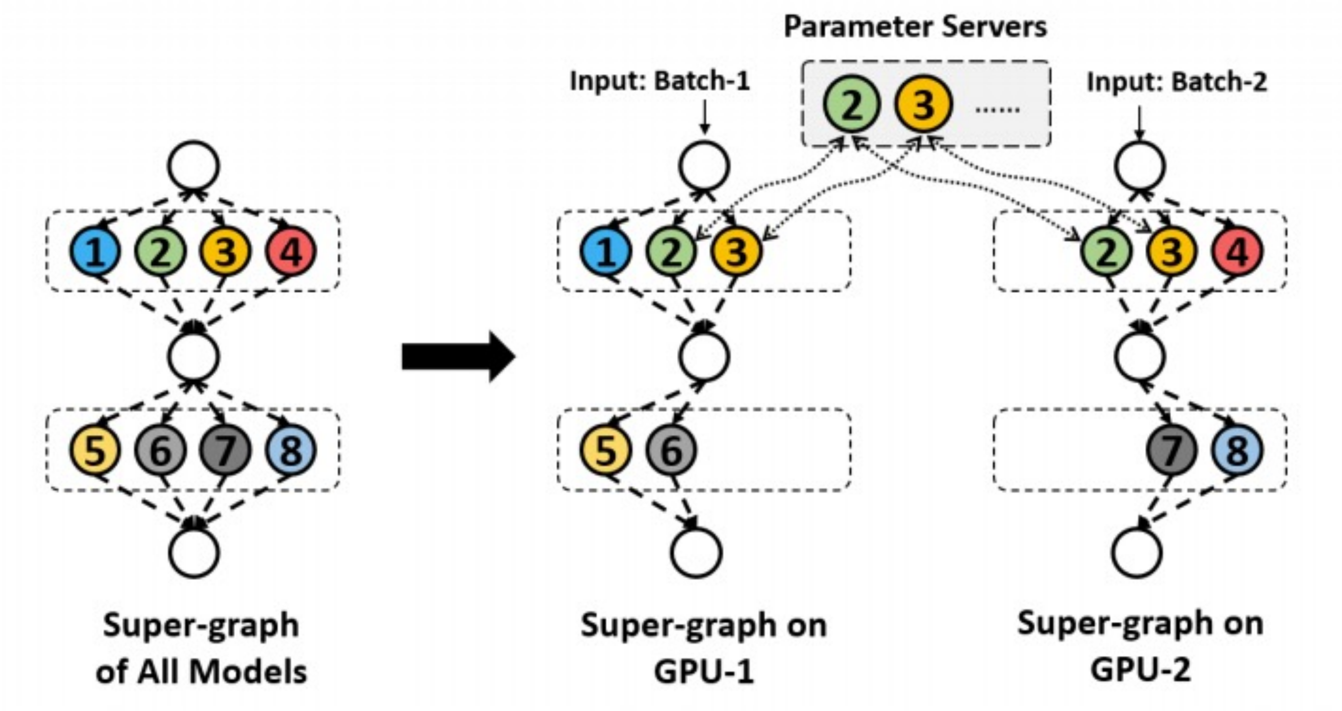
- Building a super-graph to encode the search space



Limited space size and hard to scale to a large GPU cluster

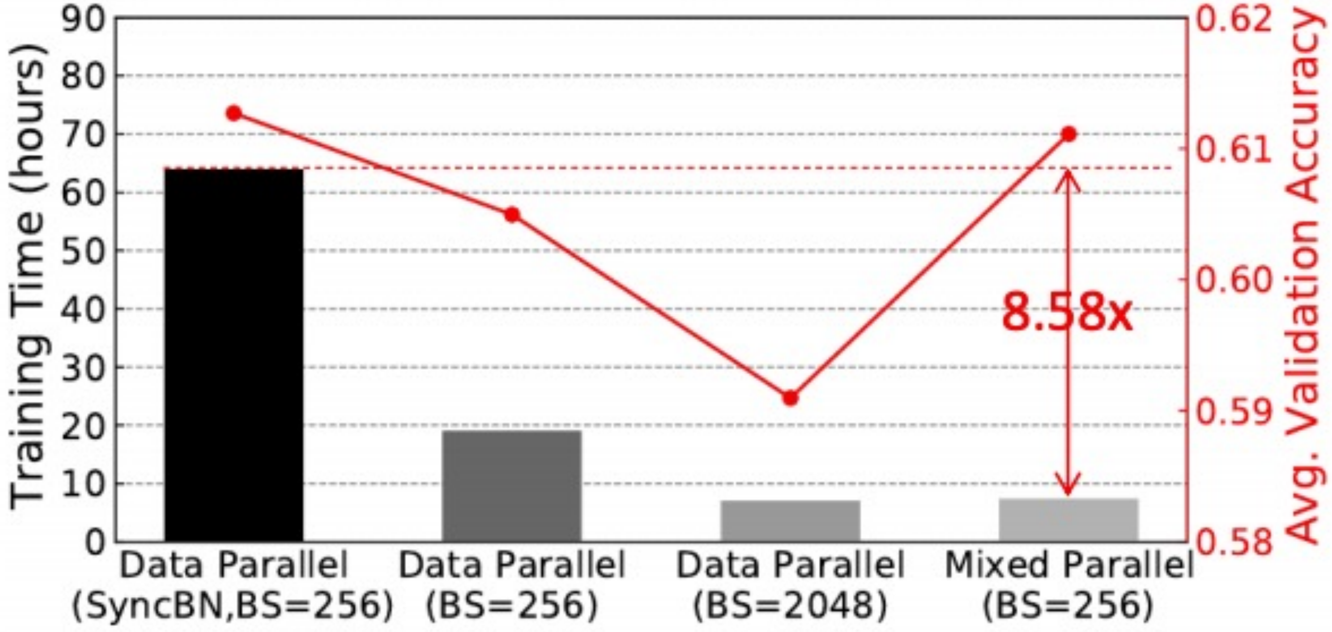
Speed up Weight-Shared Training

- **Mixed Parallelism for weight sharing**
  - Model parallelism partitions the super-graph to multiple GPUs
  - Data parallelism feeds each partition with a different batch of data



Speed up Weight-Shared Training

- Experiment with a popular weight-shared NAS, SPOS [*]



[*] Guo Z, et al. "Single path one-shot neural architecture search with uniform sampling". arXiv preprint. 2019 Mar 31.

Fluid: Resource-aware Hyperparameter Tuning Engine

Peifeng Yu[†], Jiachen Liu[†], Mosharaf Chowdhury

[†] Equal contribution

SymbioticLab

University of Michigan

Outline

1. Background and Motivation

2. Abstraction and Algorithms

3. Evaluation

Hyperparameter Tuning Today

- Hyperparameters
  - # of layers/# of neurons
  - Dropout rate
  - # of channels
  - Learning rate
  - Optimizer parameters
  - Etc.

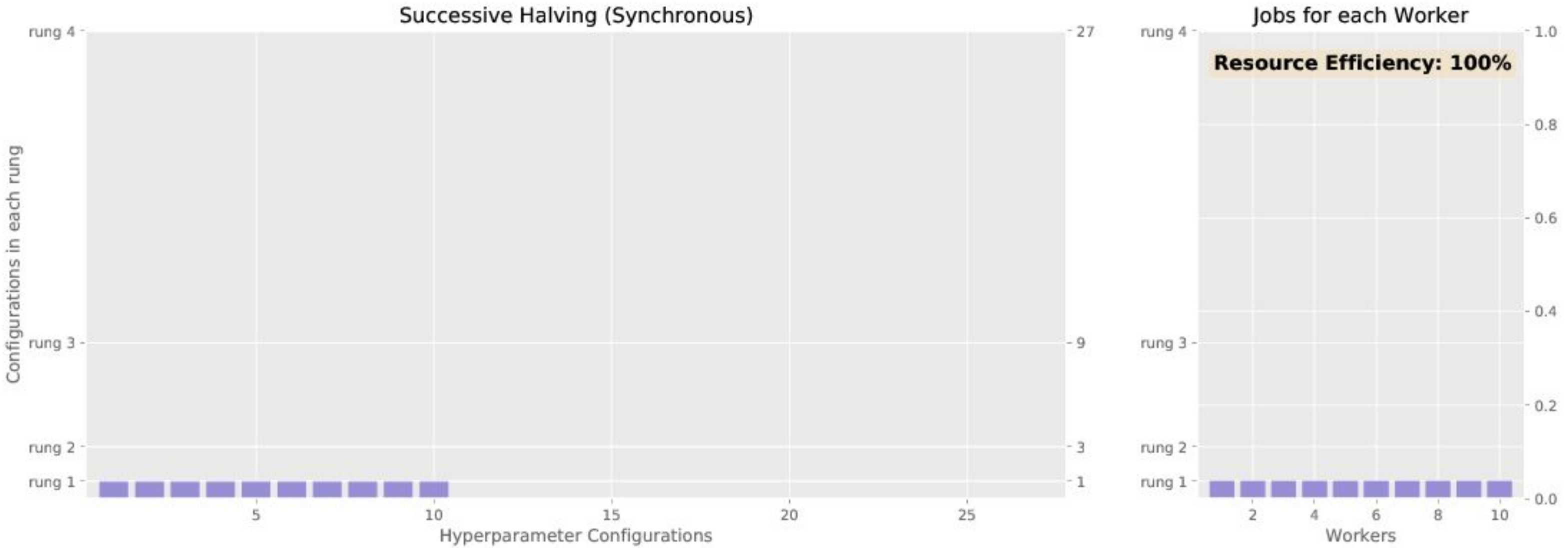
- Non-differentiable & high dimensional search space

Hyperparameter Tuning Today

- Evaluation of hyperparameters is time/resource consuming
  - train a model to know if it works

- Many algorithms & techniques
  - Random/Grid
  - Model-based config generation (BO, PTE ...)
  - Bandit-based / early-stopping (SHA, HB, BOHB...)
  - Many others (ASHA...)

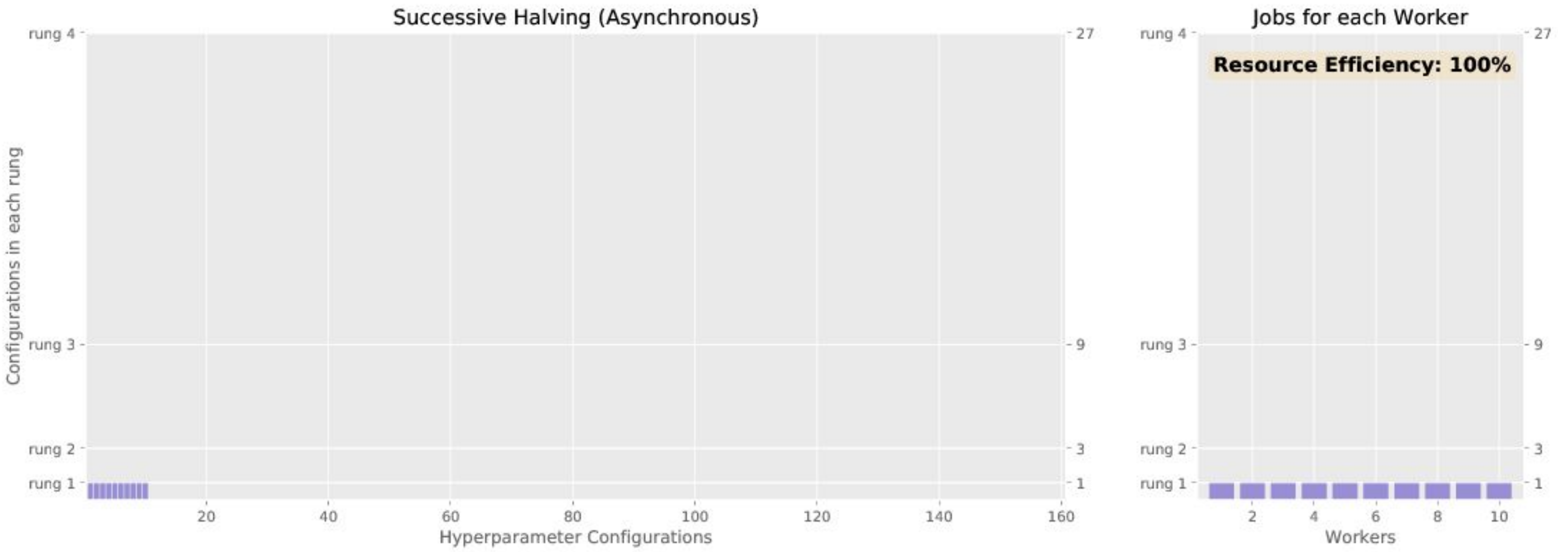
Example: SHA



1. Resource underutilized
2. Straggler problem

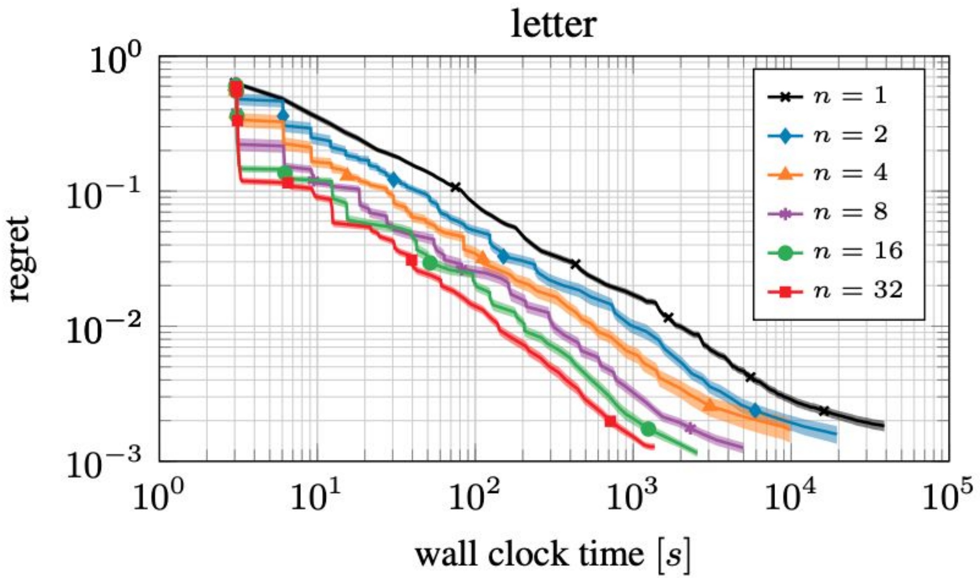
https://blog.ml.cmu.edu/2018/12/12/massively-parallel-hyperparameter-optimization/

Example: ASHA / BOHB



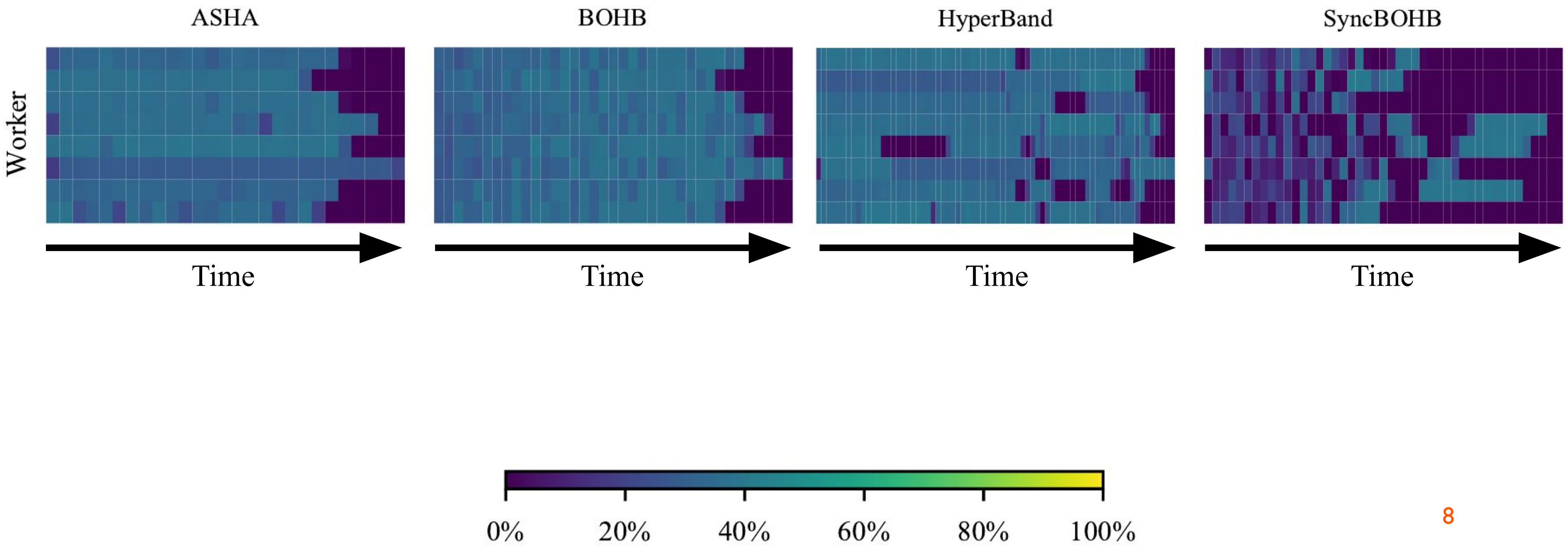
1. Maximize resource utilization by improving parallelism

Example: BOHB



1. Higher time to accuracy with the increase of #worker
2. Unknown resource to accuracy performance (goodput)
3. Simply improving parallelism may waste resources

Trials Execution
Resource utilization overtime



Resource Parallelism

Inter-GPU parallelism: Distributed training

- Fully utilize the idle resources

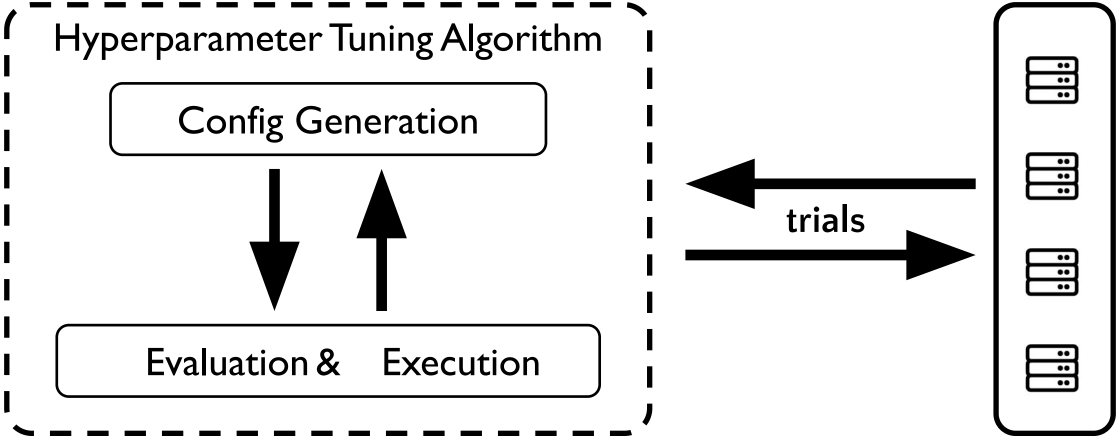
Intra-GPU parallelism: Nvidia MPS

- Execute more trials with under-utilized resources

Goal:

1. Improve resource utilization
2. Minimize the makespan

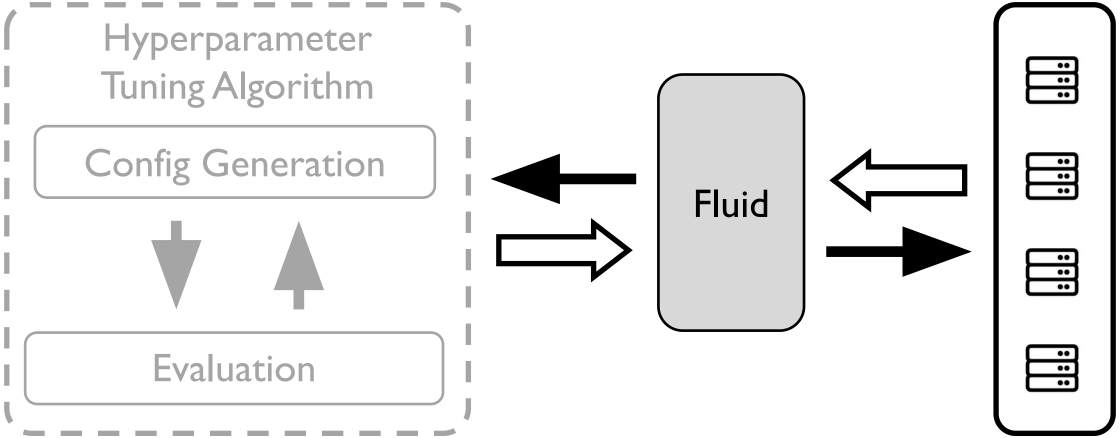
Hyperparameter Execution Engine: **Fluid**



- **Direct** interaction with the cluster to execute **trials**
- Trials gets executed in FIFO order

Hyperparameter Execution Engine: **Fluid**

Hyperparameter Execution Engine: **Fluid**



Challenge:

- Wide variety of tuning algorithms
  - Random/Iterative/Sequential

  ✔ **TrialGroup**

- Heterogeneity & dynamicity

✔ Integrated algorithm for leveraging multiple source of parallelism

Outline

1. Background and Motivation

2. Abstraction and Algorithms

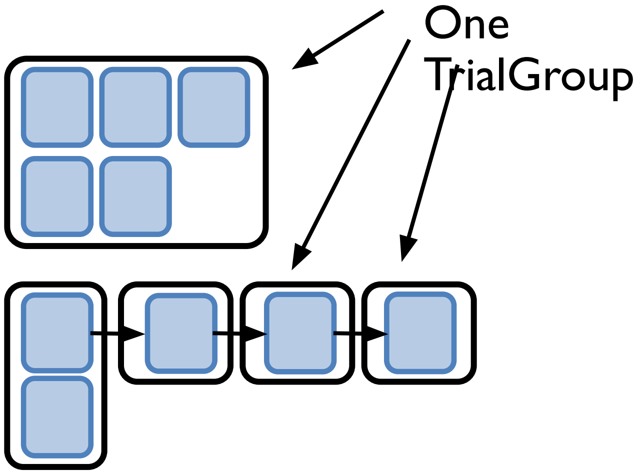
3. Evaluation

The Interface: **TrialGroup**

- Definition

  A group of training trials with a training budget associated to each trial.

- Example
  - Given 5 trials to evaluate:  x5
  - Grid/random search:
  - Sequential model-based algorithms:



The Interface: **TrialGroup**

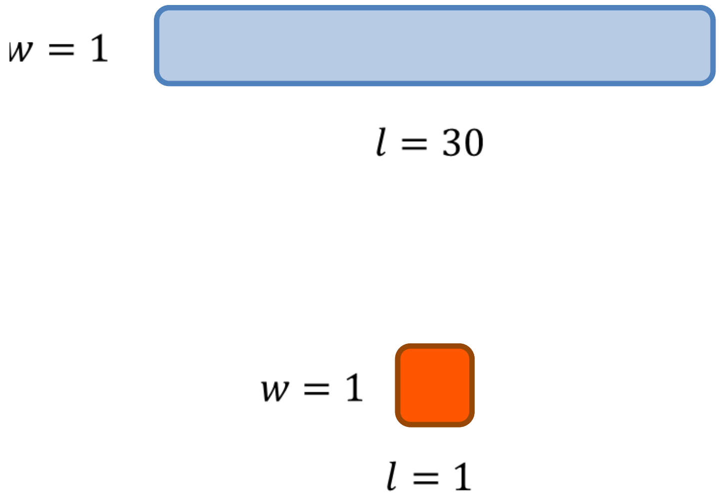
- Definition

  A group of training trials with a training budget associated to each trial.

- Generalization
  - All kinds of hyperparameter tuning algorithms could be expressed by **a sequence of TrialGroup** and executed by Fluid.

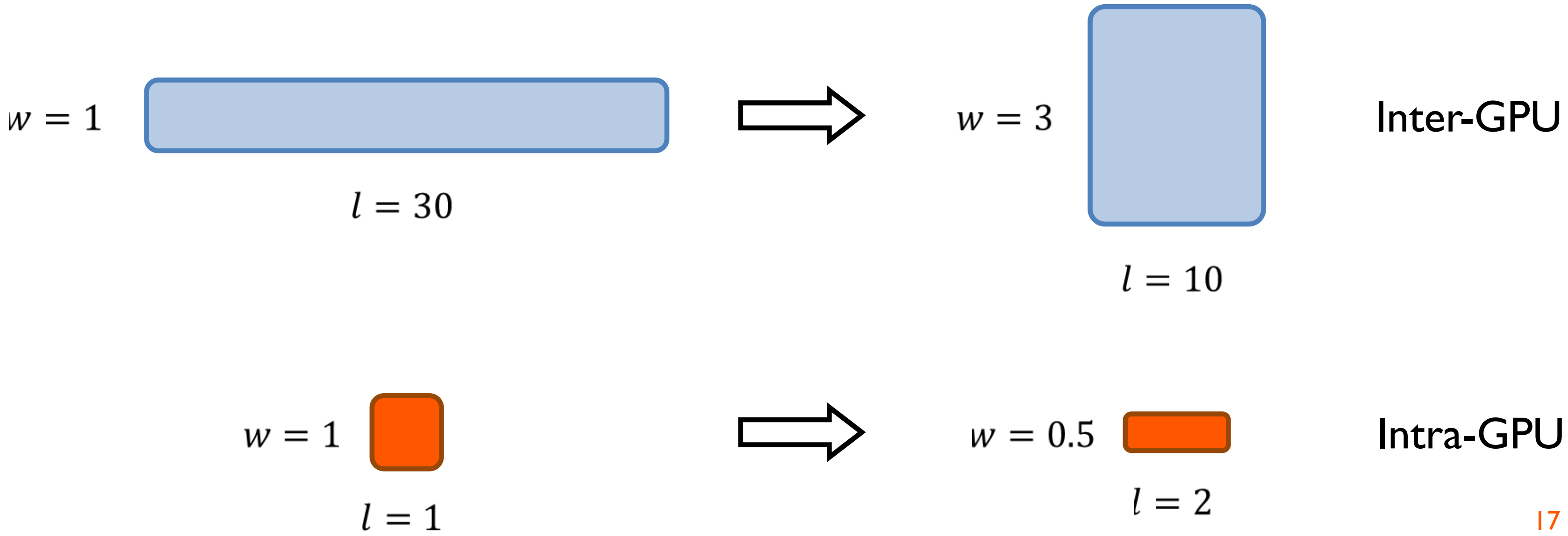
Problem Definition: Strip Packing

- Input: TrialGroup $A = \{a_1, a_2, \cdots, a_k\}$, resources $M = \{m_1, m_2, \cdots, m_n\}$
- Output: resource allocation $W = \{w_1, w_2, \cdots, w_n\}$
- Goal: minimize the length $\mathbf{L}$ of strips

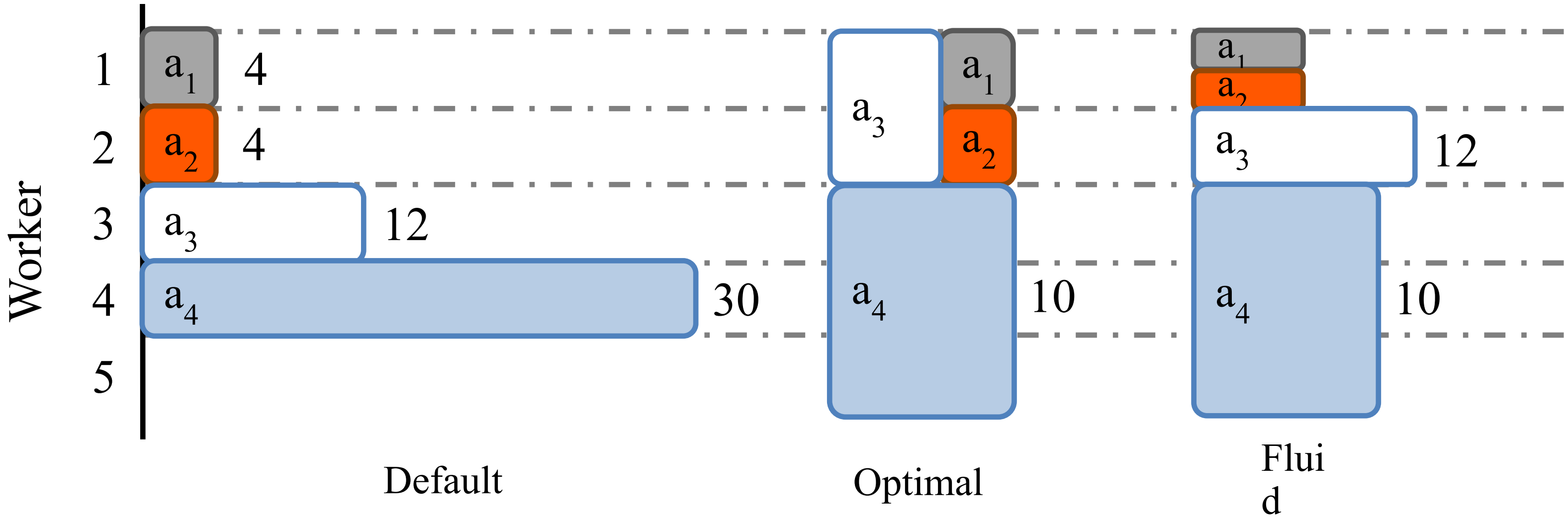


Problem Definition: Strip Packing

- Input: TrialGroup $A = \{a_1, a_2, \cdots, a_k\}$, resources $M = \{m_1, m_2, \cdots, m_n\}$
- Output: resource allocation $W = \{w_1, w_2, \cdots, w_n\}$
- Goal: minimize the length L of strips



Toy Example



Different solutions to execute 4 trials (1 TrialGroup) scheduled on 5 workers

Fully utilize the resources and mitigate the straggler

Algorithm: `StaticFluid`

$$w_i = \min(\max(\left\lfloor \frac{h_{i,1}}{\sum_j h_{j,1}} n \right\rfloor, \frac{1}{c}), d)$$

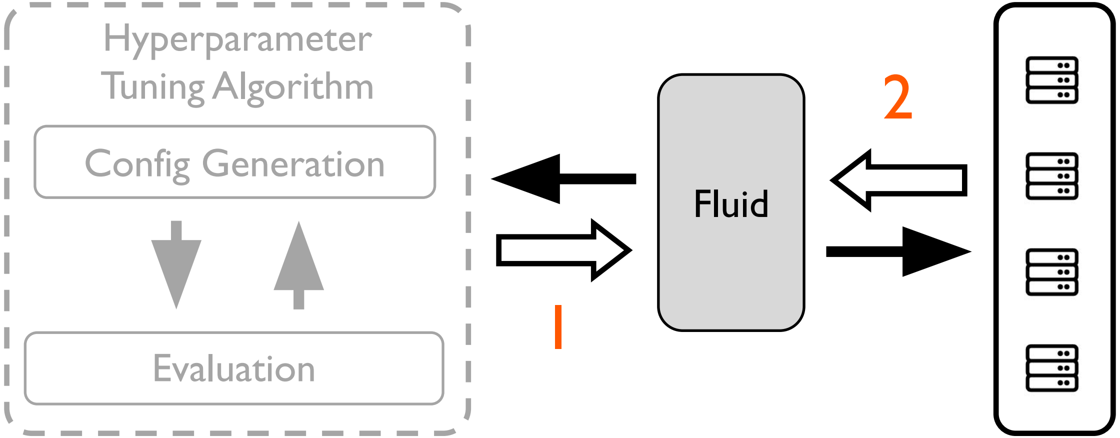
Intra-GPU overhead

Budget ratio

Inter-GPU overhead

- h: trial training budget
- n: available resources
- c: maximum intra-GPU parallelism (# of packing trials)
- d: maximum inter-GPU parallelism (# of distributed workers)

Algorithm: DynamicFluid



Fluid is <u>event-driven</u>:

1. Trials added / removed
2. Resource added / changed

Algorithm: DynamicFluid

$$w_i = \min(\max(\left\lfloor \frac{h_{i,1}}{\sum_j h_{j,1}} n \right\rfloor, \frac{1}{c}), d)$$

if $w'_i > w_i$ and $h_{i,w'_i} + \epsilon < h_{i,w_i}$
    Update a_i with w_i resources ▷ Scale up
else if $w'_i < w_i$ and $w'_i(h_{i,w'_i} + \epsilon) < w_i h_{i,w_i}$
    Update a_i with w_i resources ▷ Scale down

ϵ : scale up / down overhead

Outline

1. Background and Motivation

2. Abstraction and Algorithms

3. Evaluation

Evaluation Setup

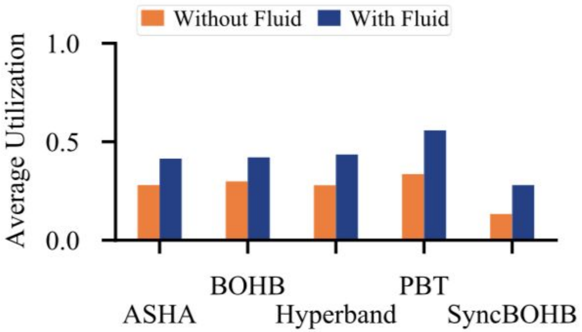
- Implementation: an alternative Ray[1] executor
- Workloads

Task	Base Model	# of Params.	Target
CIFAR-10	AlexNet	7	Acc. >= 90%
WLM	RNN	10	PPL <= 140
DCGAN	CNN	2	Inception >= 5.2

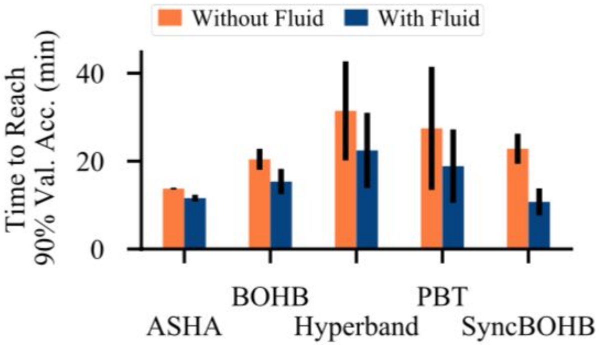
[1]:

Evaluation Results

- Average resource utilization: 10%-100% improvement
- Average job completion time: 10%-70% improvement



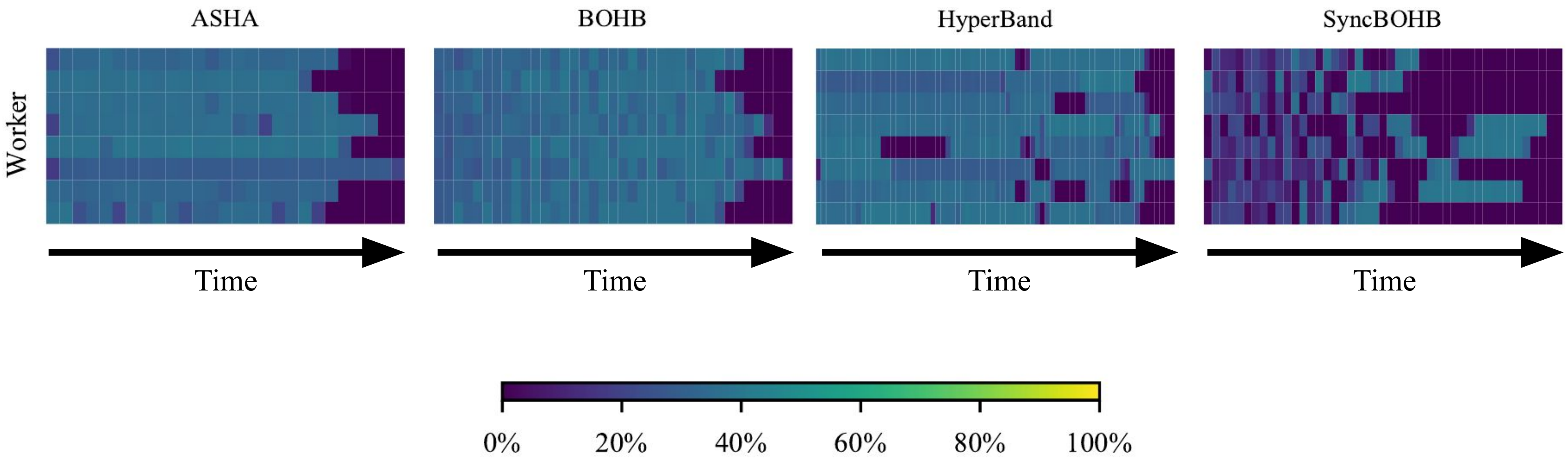
(a) CIFAR-10



(b) CIFAR-10

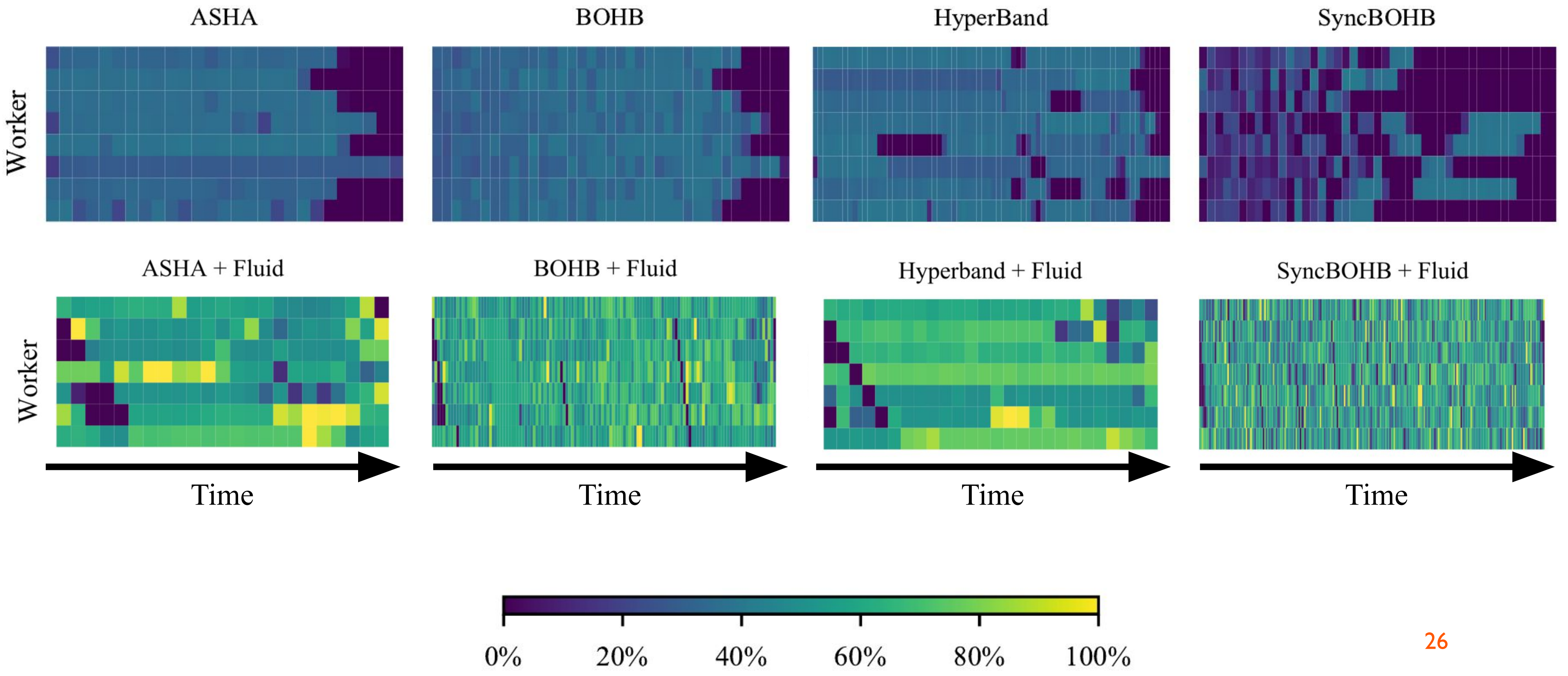
Evaluation Results: Visualization

Resource utilization over time



Evaluation Results: Visualization

Resource utilization over time



Conclusion

- Fluid
  - Hyperparameter tuning execution engine
  - Can be combined with most tuning algorithms
  - Improve utilization and end-to-end tuning time
- Open source
  - https://github.com/SymbioticLab/fluid
- Q&A

SymbioticLab

UNIVERSITY OF MICHIGAN