14th USENIX Symposium on Operating Systems Design and Implementation (**OSDI '20**)

Retiarii: *A Deep Learning Exploratory-Training Framework*

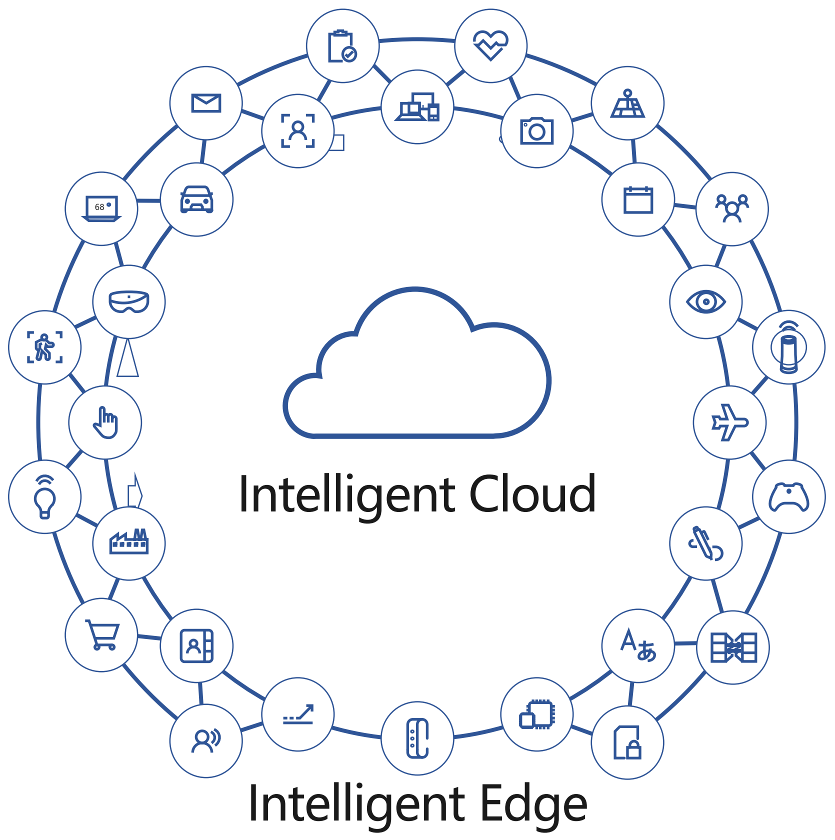
Quanlu Zhang, ***Zhenhua Han***, Fan Yang, Yuge Zhang, Zhe Liu, Mao Yang, Lidong Zhou

Microsoft Research

Microsoft

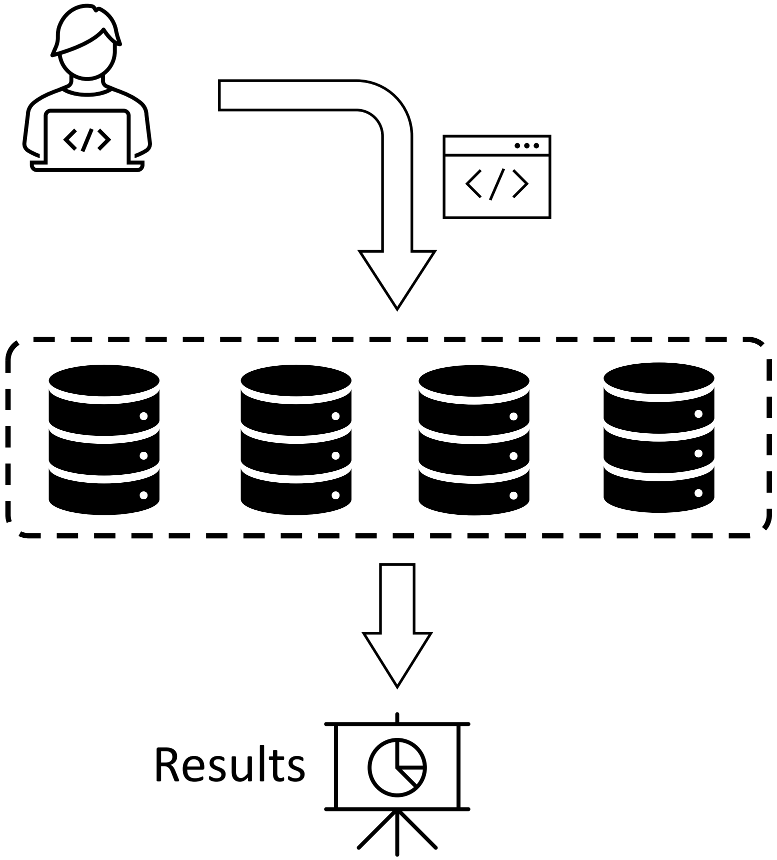
Deep Neural Network Becomes Prevalent

- DNN models are being adopted in Cloud and Edge
- More and more cloud/edge applications are powered by DNN techniques
- Important to design a good DNN model

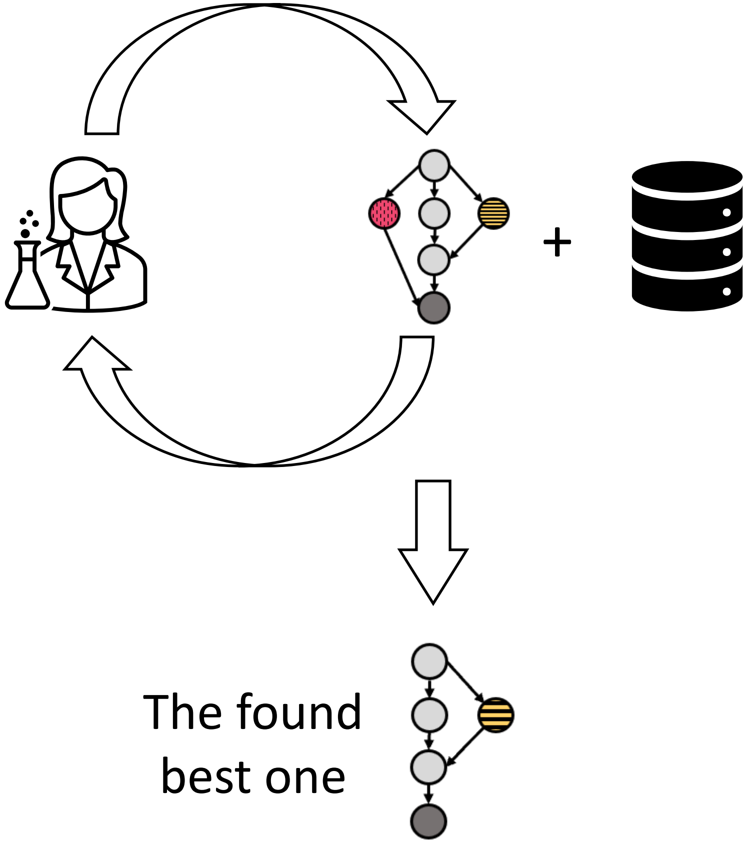


DNN Model Design: An Exploration Process

Big Data Tasks



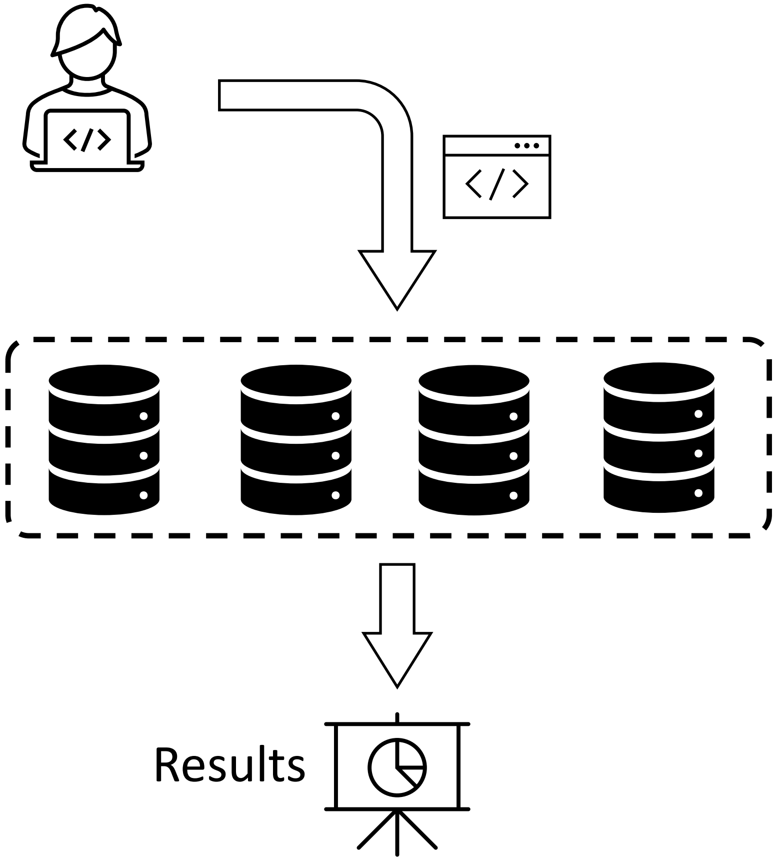
Deep learning tasks



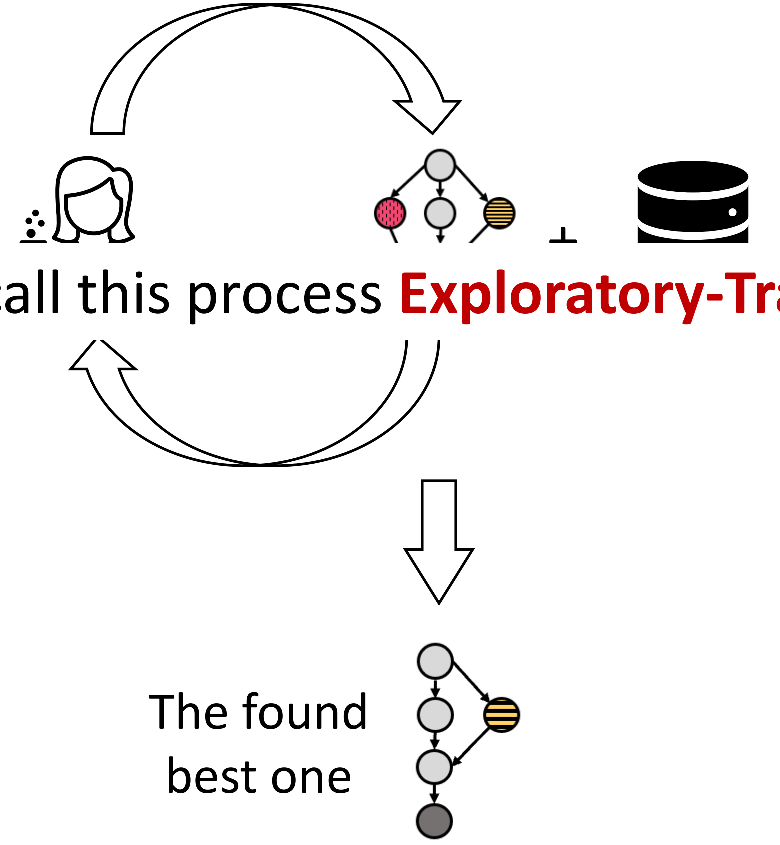
One-shot v.s. **Exploratory**

DNN Model Design: An Exploration Process

Big Data Tasks



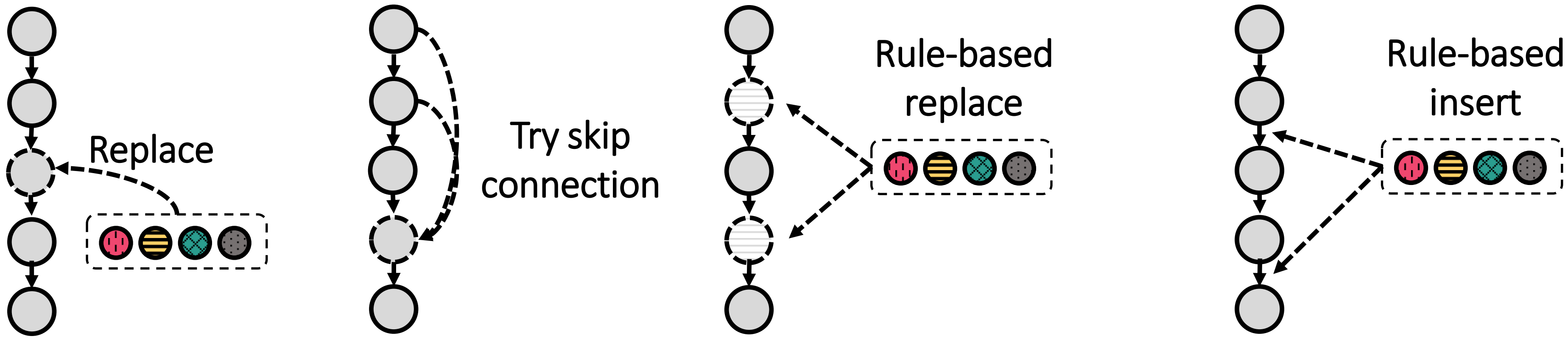
Deep learning tasks



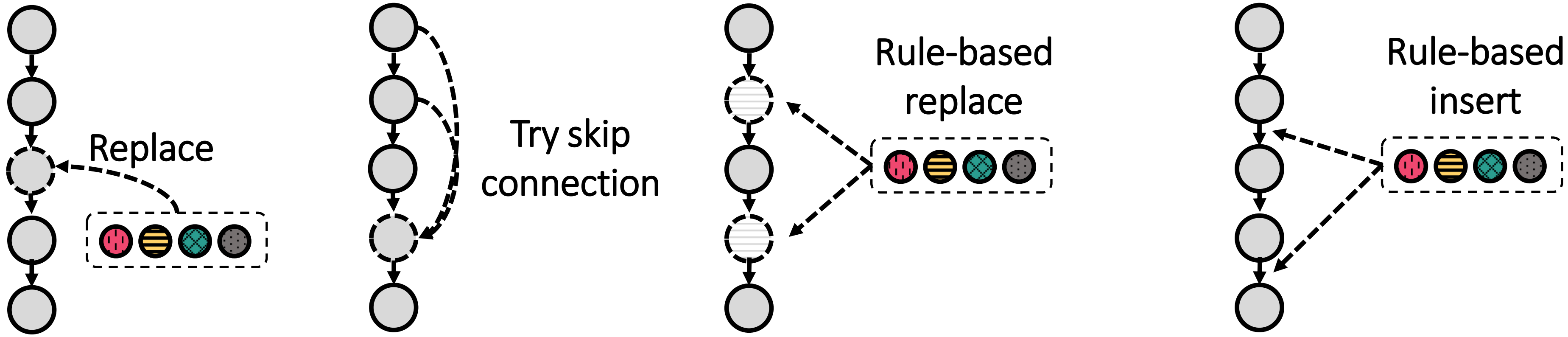
We call this process **Exploratory-Training**

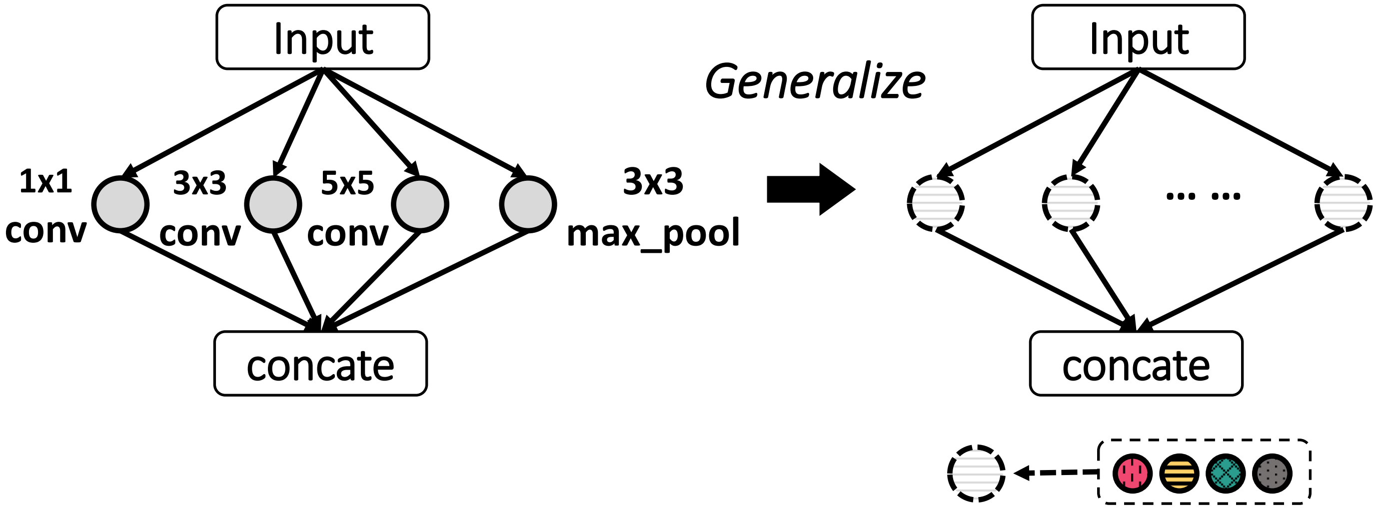
One-shot v.s. **Exploratory**

Examples of Exploratory-Training

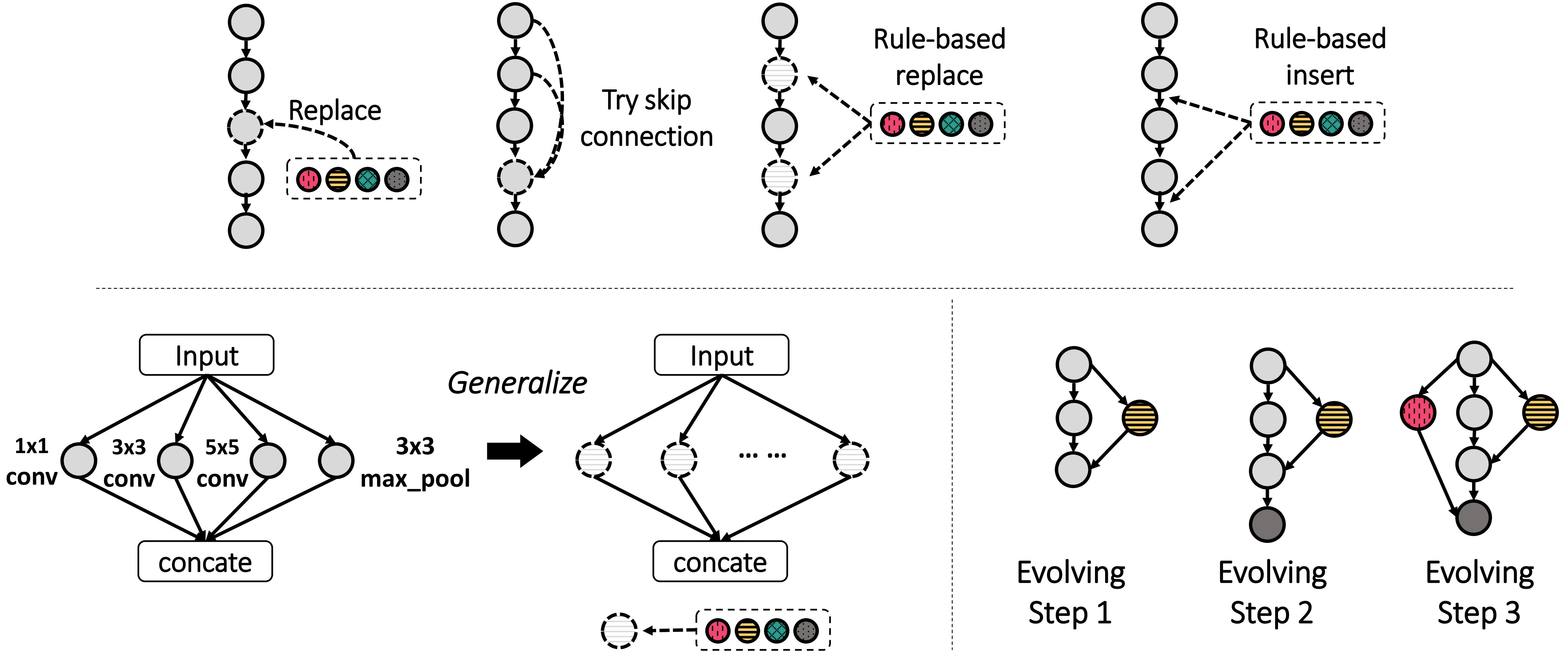


Examples of Exploratory-Training





Examples of Exploratory-Training



Weak Support to Exploratory-Training

- Existing deep learning frameworks focus on one single DNN model
  - Just one step of the entire exploratory-training process
- Tools for model exploration lack of modularity and programmability
  - Neural architecture search (NAS) or hyperparameter optimization (HPO)
  - One NAS/HPO solution only applicable to one kind of neural architectures
- Missed opportunities to speed up the model exploration process
  - Exploiting model similarities during the exploratory-training

Rethinking DNN Framework

No framework

Deep learning framework

Exploratory-Training framework

Programming with libraries

Making programming a DNN model easier and faster

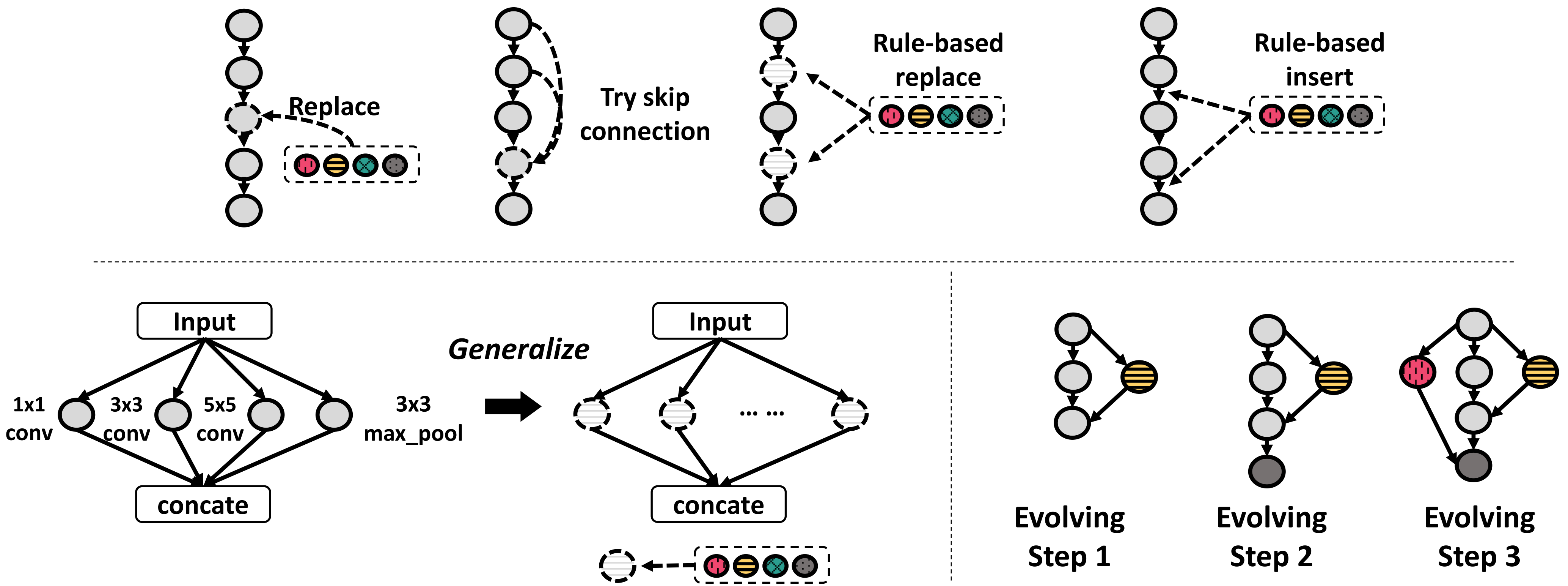
Making DNN model exploring easier and faster

The Goal of Retiarii

- A deep learning framework for exploratory-training, instead of the development of a single DNN model
- Making model exploration more systematic and programmable
- The go-to DNN framework when one designs a new DNN model

The Key Insight

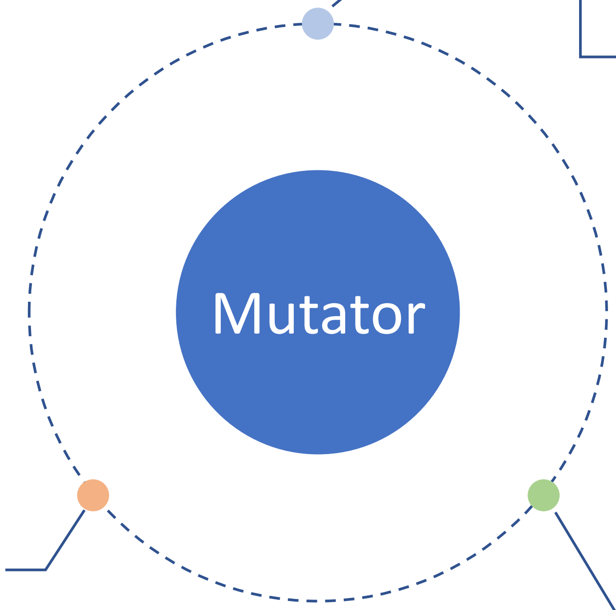
- Exploratory-training can be treated as a series of model **mutation** in a neural **model space**



Mutator as the Core Abstraction

Decoupling **model space** from model **exploration strategy**, while enabling both well-known and new **cross-model optimizations**

Defining arbitrary **model space** with mutators



Exposing the correlations between models for **cross-model optimizations**

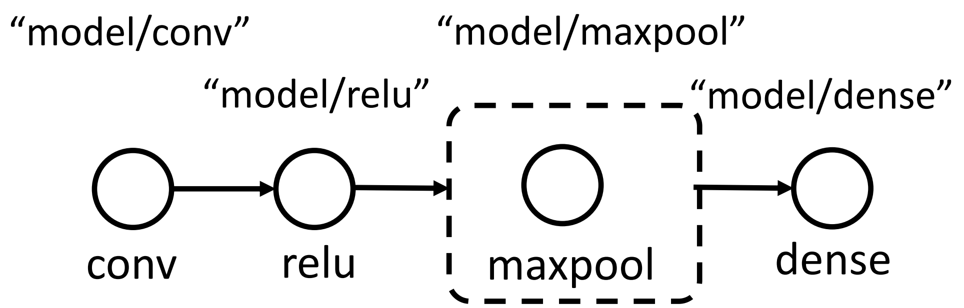
The model space can be understood by **exploration strategy**

The Highlights of Retiarii

- Mutator-based programming paradigm
  - Programming a model space, instead of programming a single model
- Highly composable between model space and exploration strategy
  - The decision of each mutation action in a model space during the exploratory-training is given to an exploration strategy (AutoML) or human (manual exploration)
  - Different exploration strategy can interact with different model space
- Exploiting rich optimizations exposed by model mutation
  - Speed up the exploration process by leveraging the similarity of explored models

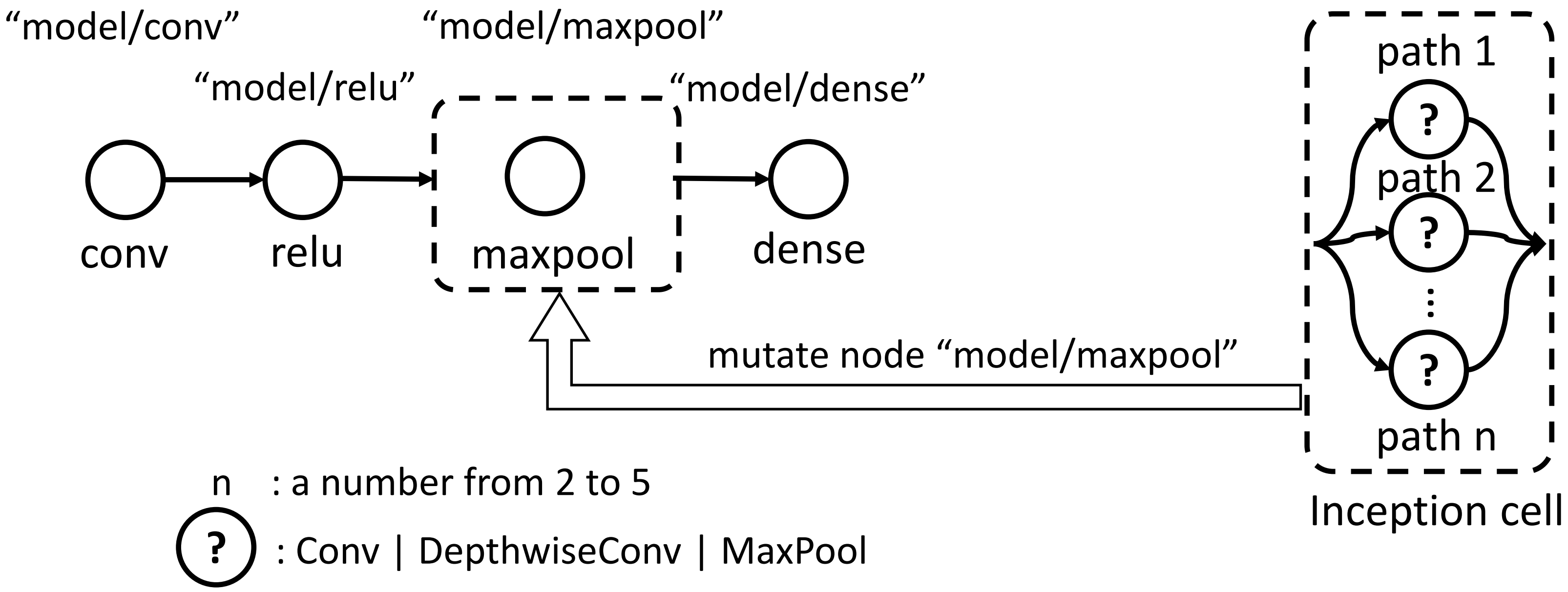
Mutator-Based Programming Paradigm

- Model Space = Base Model + Mutators



Mutator-Based Programming Paradigm

- Model Space = Base Model + Mutators

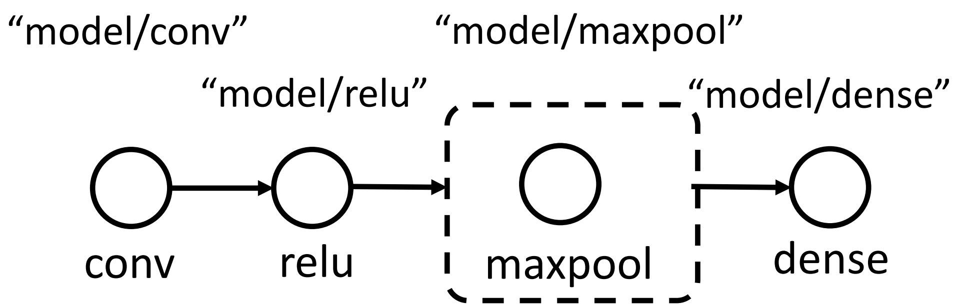


An example model space: the third node in a four-node base model is replaced with an inception cell

Mutator-Based Programming Paradigm

- Define and Apply 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):
```

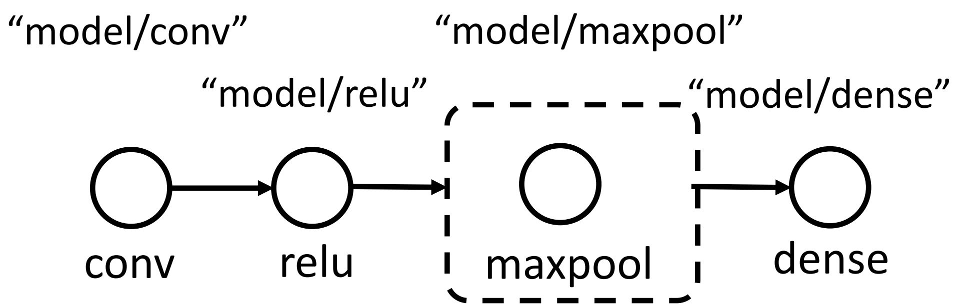


An example model space: the third node in a four-node base model is replaced with an inception cell

Mutator-Based Programming Paradigm

- Define and Apply 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):
    if not three_node_chain(targets):
      return err
```

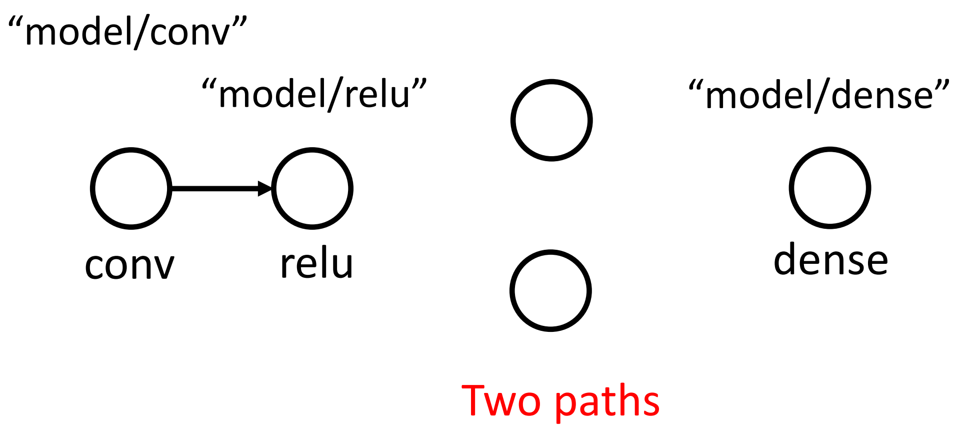


An example model space: the third node in a four-node base model is replaced with an inception cell

Mutator-Based Programming Paradigm

- Define and Apply Mutator

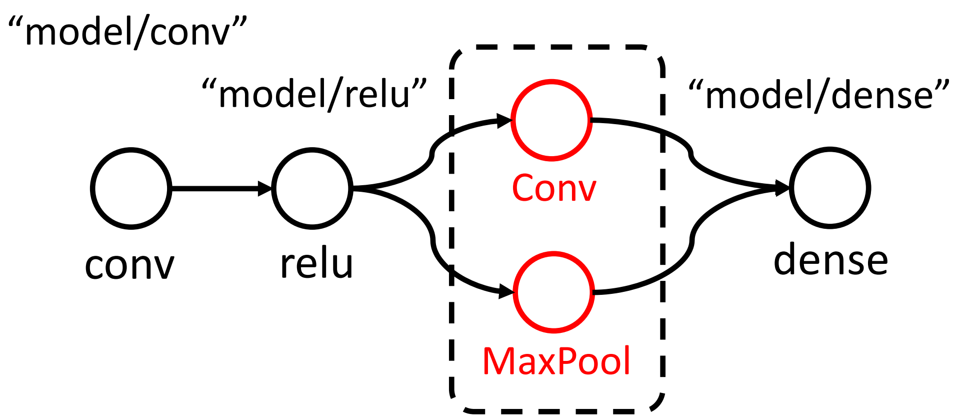
```
# define the graph mutation behavior
class InceptionMutator(BaseMutator):
  def __init__(self, paths_range, candidate_ops):
    self.paths_range = paths_range # [2, 3, 4, 5]
    self.ops = candidate_ops # {conv, dconv, ...}
  def mutate(self, targets):
    if not three_node_chain(targets):
     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



An example model space: the third node in a four-node base model is replaced with an inception cell

Mutator-Based Programming Paradigm

- Define and Apply Mutator

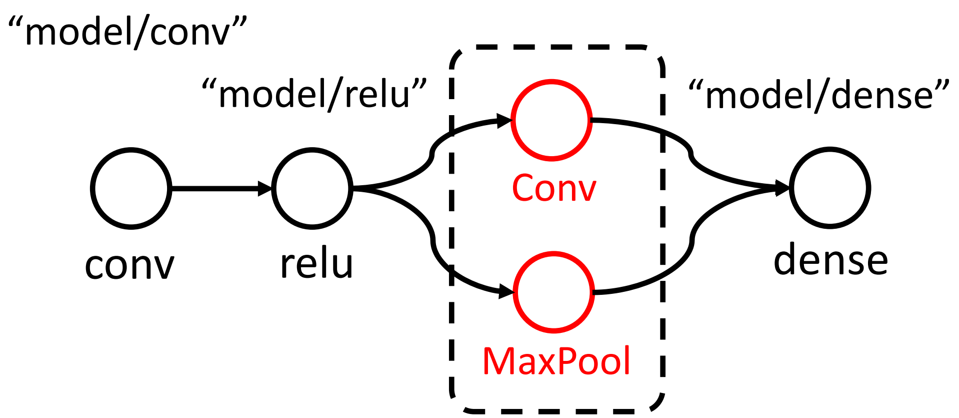



```
# define the graph mutation behavior
class InceptionMutator(BaseMutator):
  def __init__(self, paths_range, candidate_ops):
    self.paths_range = paths_range # [2, 3, 4, 5]
    self.ops = candidate_ops # {conv, dconv, ...}
  def mutate(self, targets):
    if not three_node_chain(targets):
     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

An example model space: the third node in a four-node base model is replaced with an inception cell

Mutator-Based Programming Paradigm

- Define and Apply Mutator

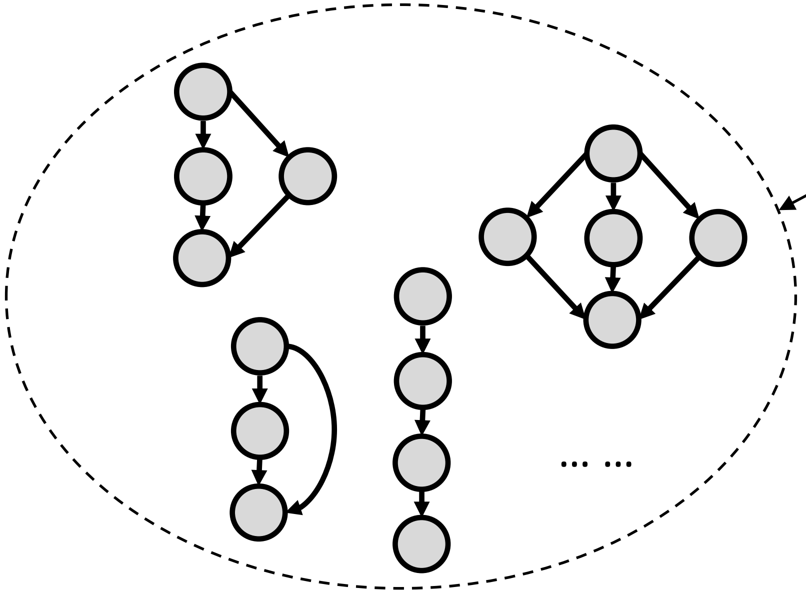


An example model space: the third node in a four-node base model is replaced with an inception cell

```python
# 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)

# mutation applied to the graph
apply_mutator(targets=["model/relu", "model/
    maxpool", "model/dense"],
    mutator=InceptionMutator(
    [2, 3, 4, 5], [conv, dconv, pool]))
```

Interaction between Model Space and Exploration Strategy



Exploratory-Training
of a model space

Exploration strategies

- Which models in the model space to try first?
- How long to train each model?
- Whether to share or inherit model weights?
- … …

Grid search

Evolution algorithm

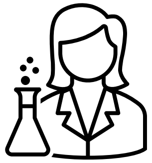
Reinforcement learning

Annealing algorithm

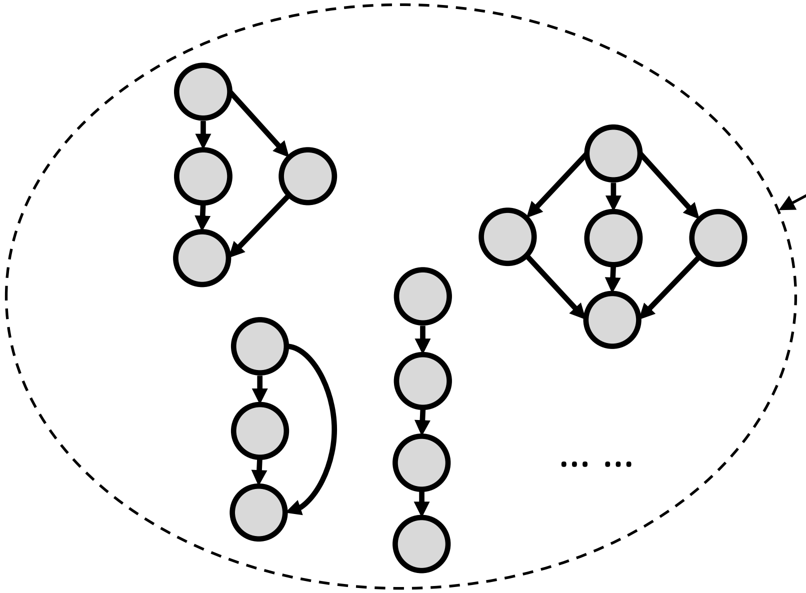
… …

Bayesian model

Gradient-based search



Interaction between Model Space and Exploration Strategy



Exploratory-Training of a model space

Exploration strategies are reusable

Exploration strategies

- Which models in the model space to try first?
- How long to train each model?
- Whether to share or inherit model weights?
- … …

Grid search

Evolution algorithm

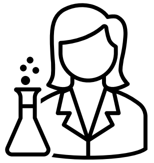
Reinforcement learning

Annealing algorithm

… …

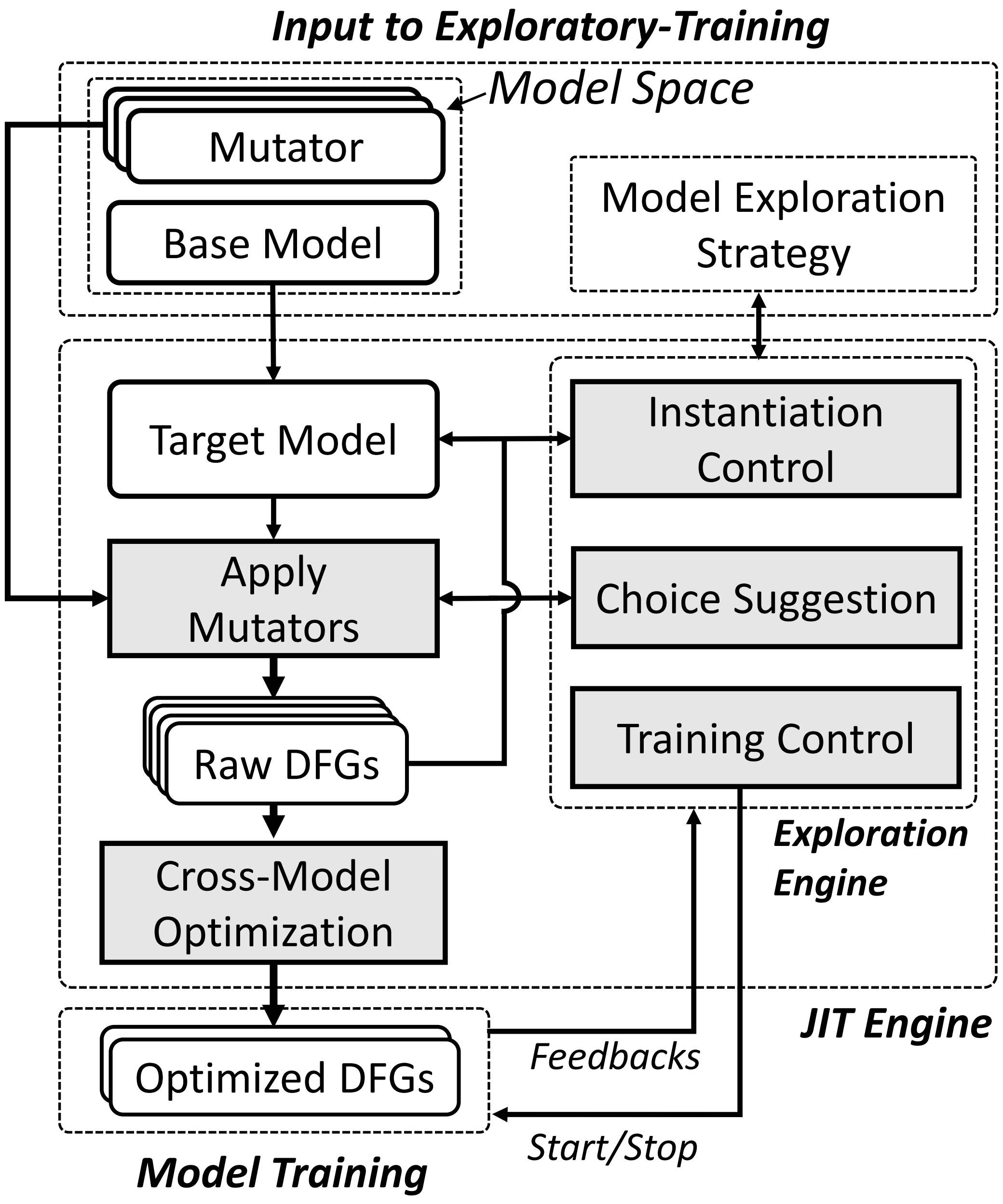
Bayesian model

Gradient-based search



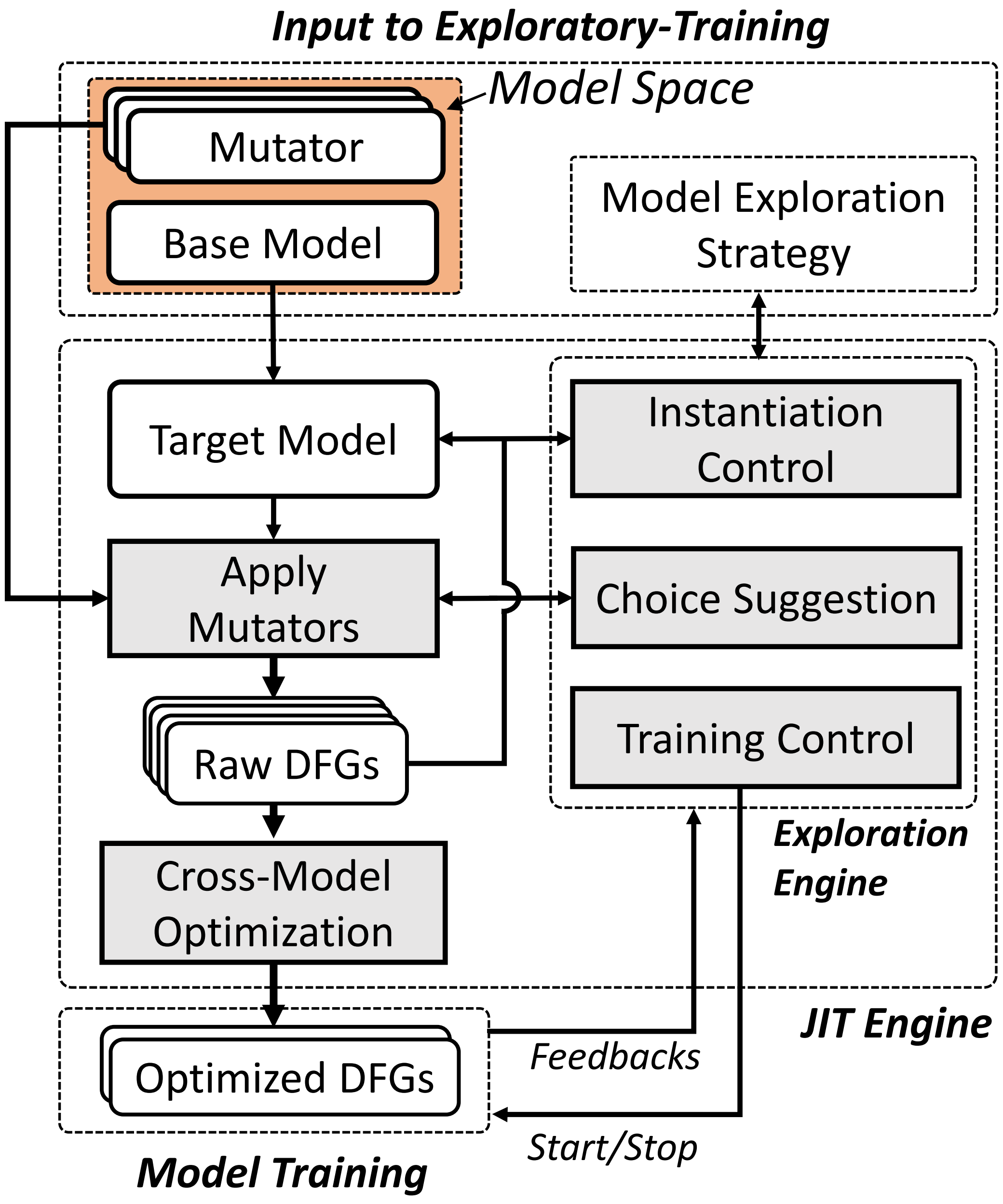
System Architecture

- Instantiate model following user specified model space
- Get suggestions from exploration strategy to instantiate models
- Optimize instantiated models to do model batching, merging and weight sharing
- Retrieve training feedbacks to feed in exploration strategy



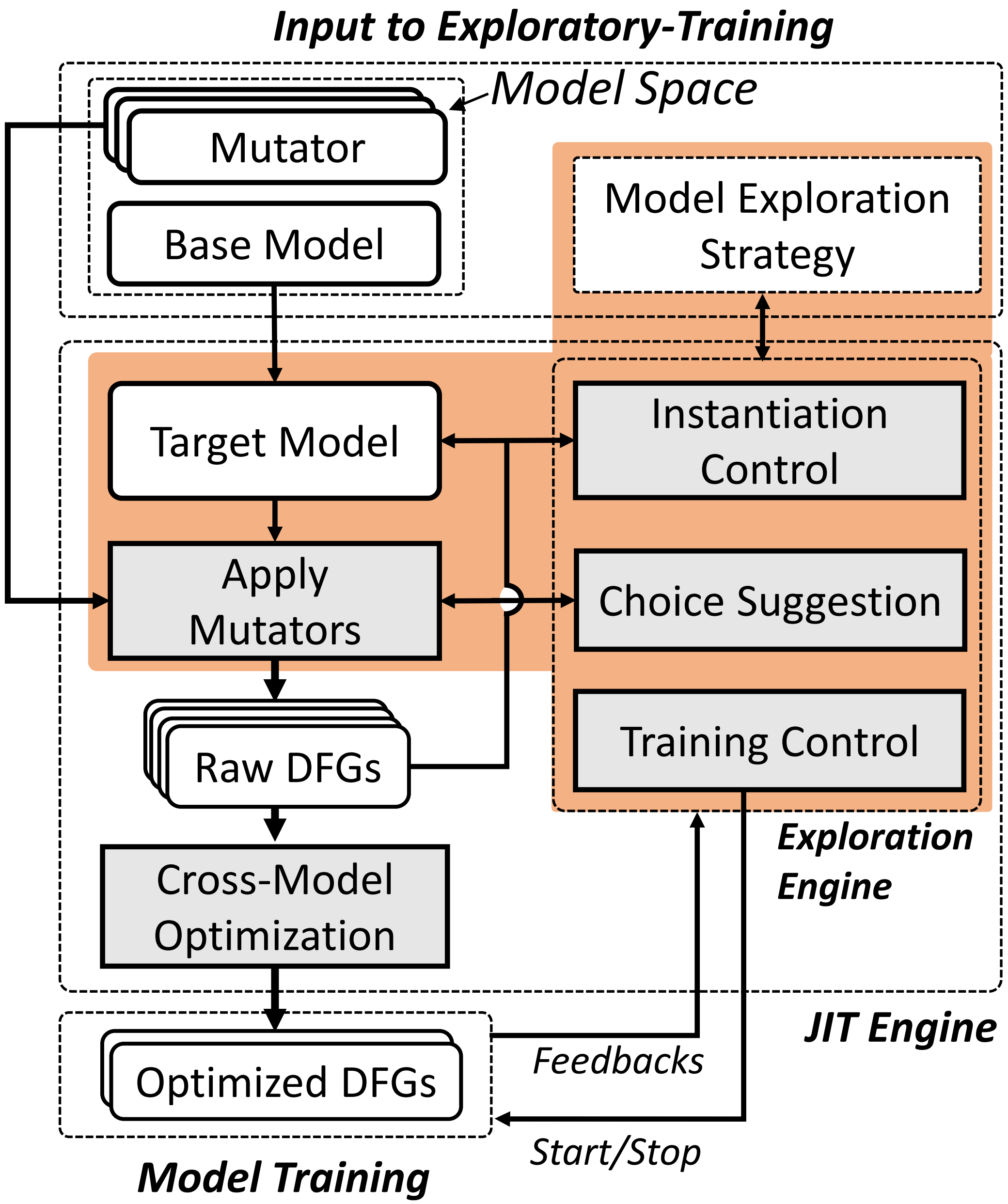
System Architecture

- Instantiate model following user specified model space
- Get suggestions from exploration strategy to instantiate models
- Optimize instantiated models to do model batching, merging and weight sharing
- Retrieve training feedbacks to feed in exploration strategy



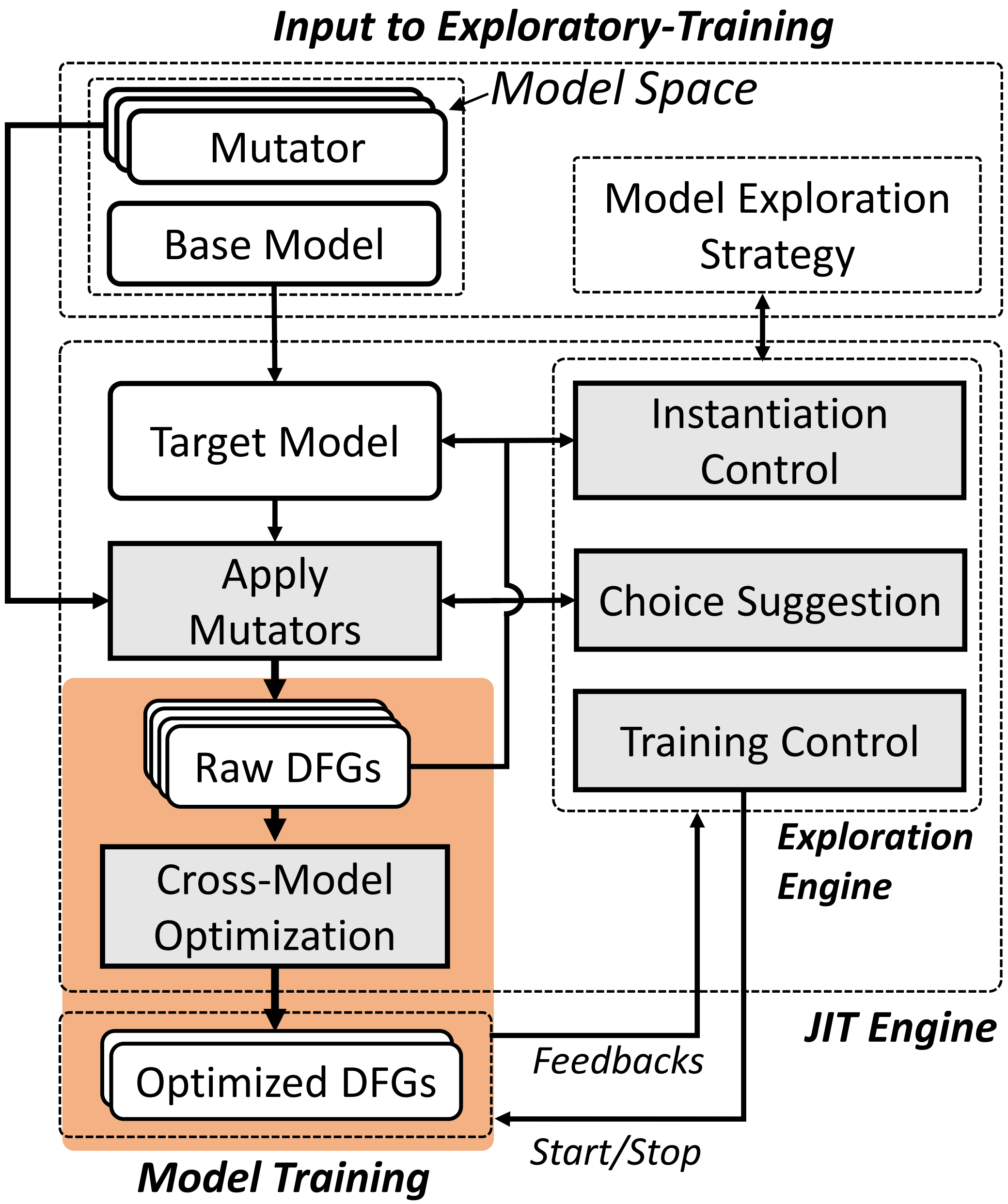
System Architecture

- Instantiate model following user specified model space
- Get suggestions from exploration strategy to instantiate models
- Optimize instantiated models to do model batching, merging and weight sharing
- Retrieve training feedbacks to feed in exploration strategy



System Architecture

- Instantiate model following user specified model space
- Get suggestions from exploration strategy to instantiate models
- Optimize instantiated models to do model batching, merging and weight sharing
- Retrieve training feedbacks to feed in exploration strategy



Expressiveness and Reusability

- The table shows 8 out of 27 NAS solutions currently supported by Retiarii

NAS Solution	Model Space	Exploration Strategy	Required Mutator Class			
			Input Mutator	*Operator Mutator*	*Inserting Mutator*	*Customized Mutator*
MnasNet [58]	MobileNetV2-based space	Reinforcement Learning		✓	✓	
NASNet [69]	NASNet cell	Reinforcement Learning	✓	✓		
ENAS-CNN [49]	NASNet cell variant	Reinforcement Learning	✓	✓		
AmoebaNet [50]	NASNet cell	Evolutionary	✓	✓		
Single-Path One Shot (SPOS) [26]	ShuffleNetV2-based space	Evolutionary		✓		
Weight Agnostic Networks [22]	Evolving space w/ adding/altering nodes adding connections	Evolutionary		✓		✓
Path-level NAS [12]	Evolving space w/ replication and split	Reinforcement Learning				✓
TextNAS [61]	TextNAS space	Reinforcement Learning	✓	✓		
...	...	...	...	...	...	...

Our open-sourced code: https://github.com/microsoft/nni/tree/retiarii_artifact

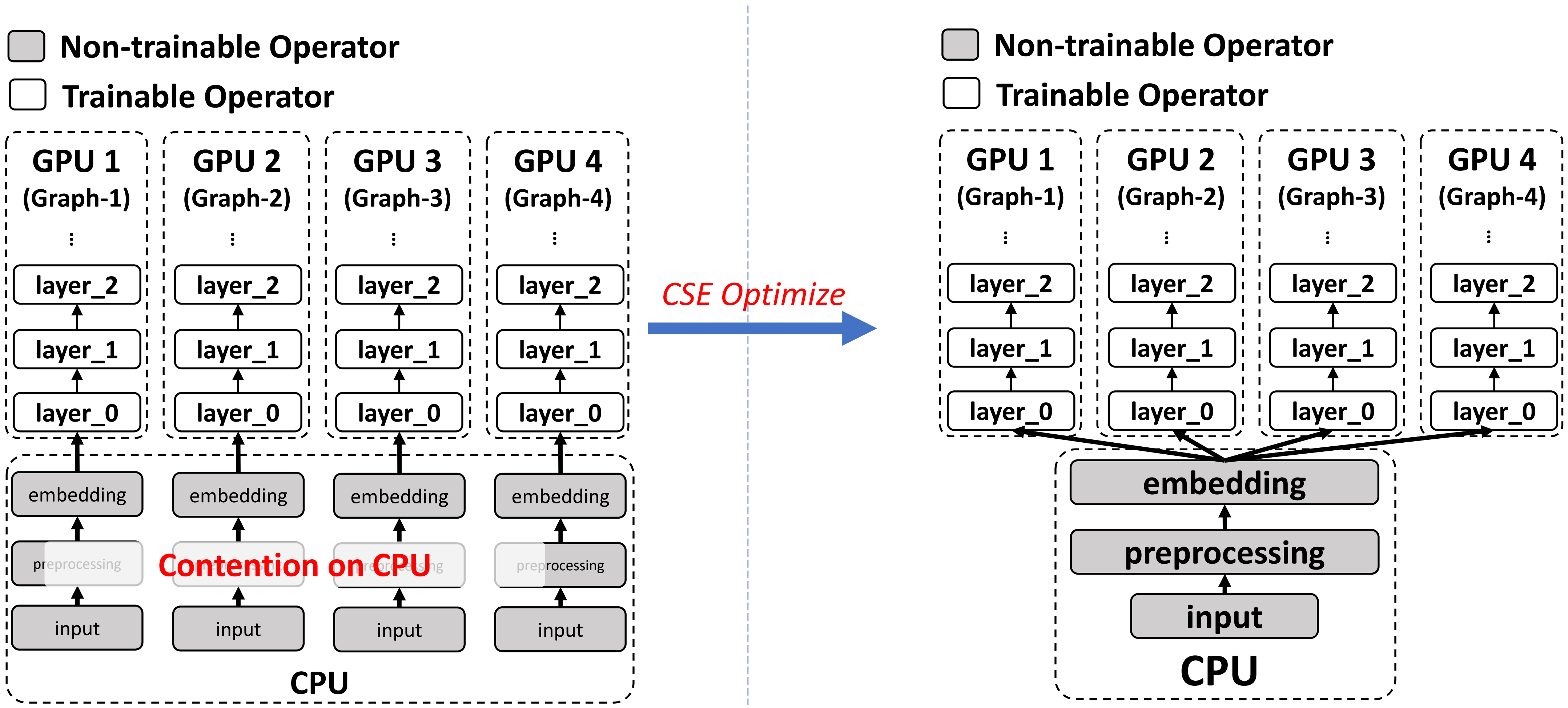
Exploiting Rich Optimizations

- There are plenty of optimization opportunities in Exploratory-Training
  - The same training data
  - The same data preprocessing
  - Similar neural architectures (e.g., common layers)
  - Weights shared among models

- Cross-model optimizations enabled with tracked correlations
  - Common sub-expression elimination (CSE)
  - Mixed parallelism for weight sharing
  - Operator batching

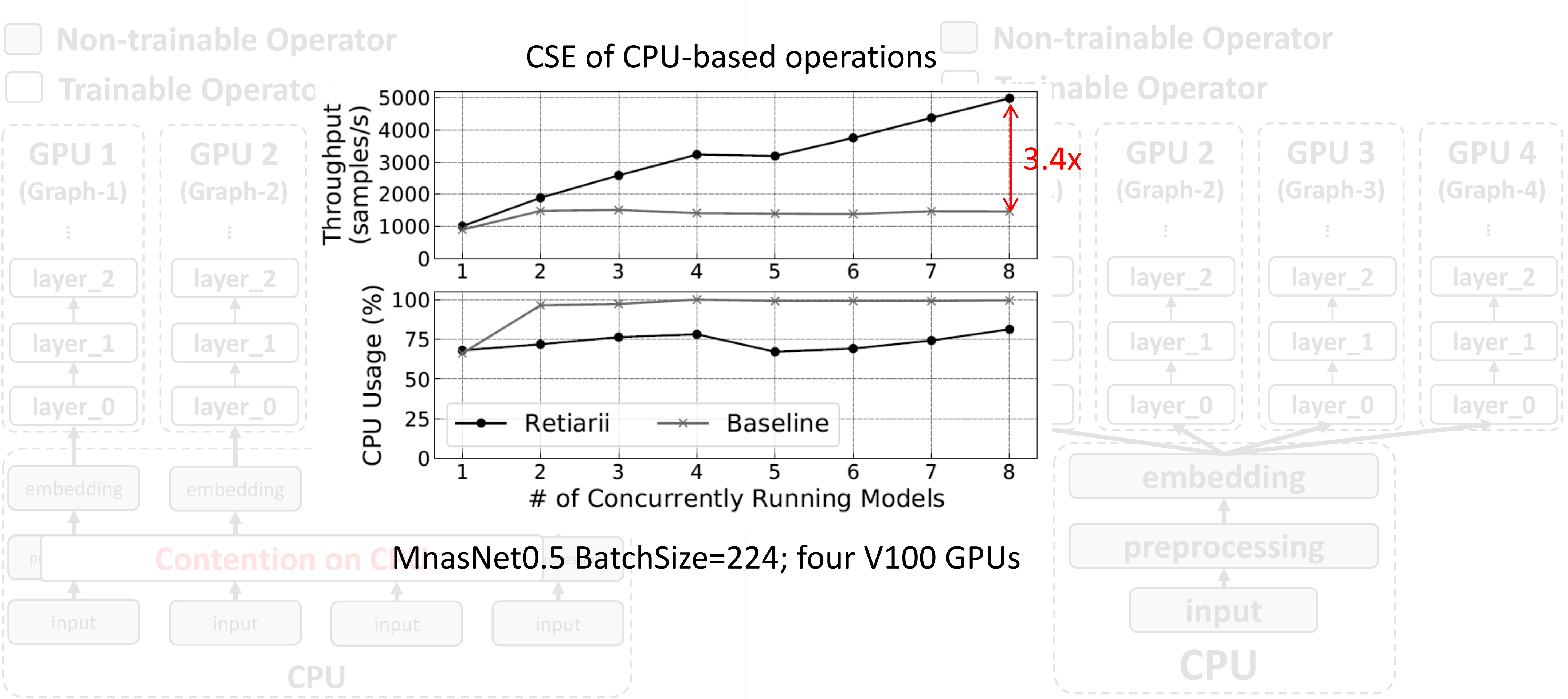
Common Sub-expression Elimination (CSE)

- De-duplicating CPU-based common prefix operations



Common Sub-expression Elimination (CSE)

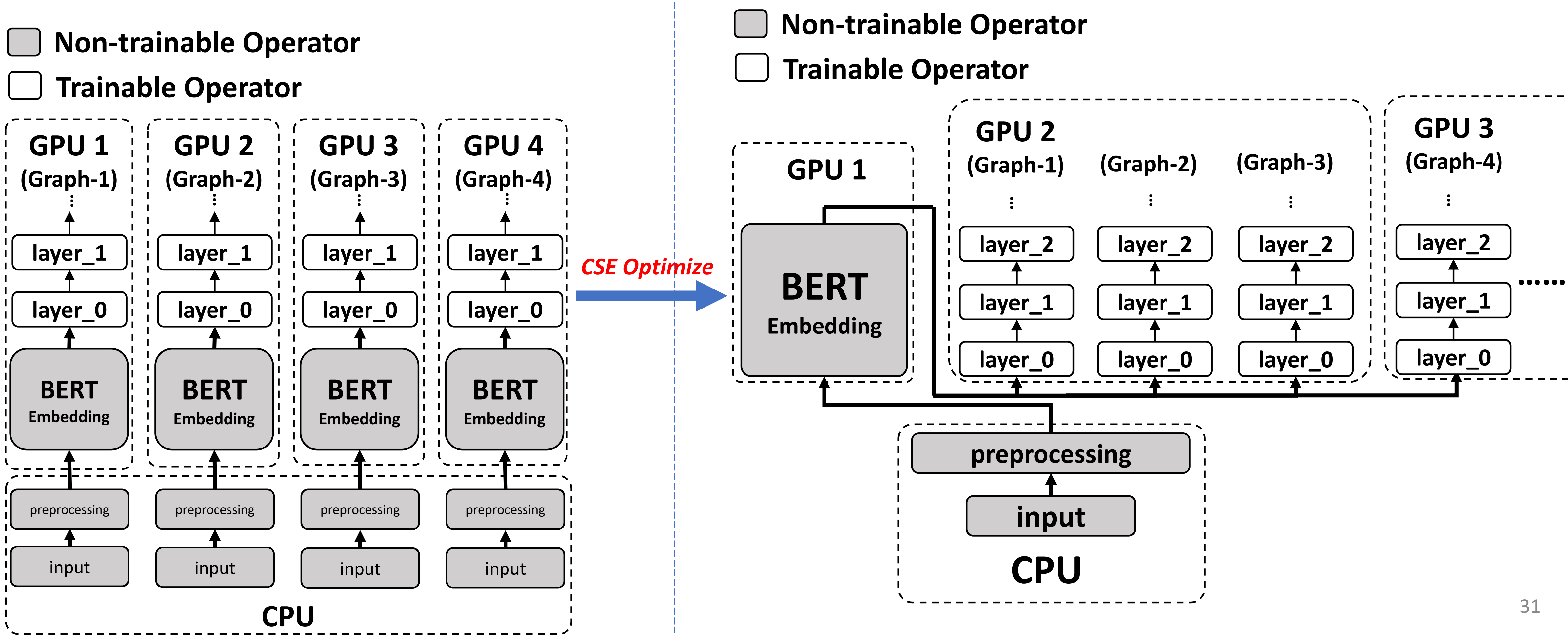
- De-duplicating CPU-based common prefix operations



MnasNet0.5 BatchSize=224; four V100 GPUs

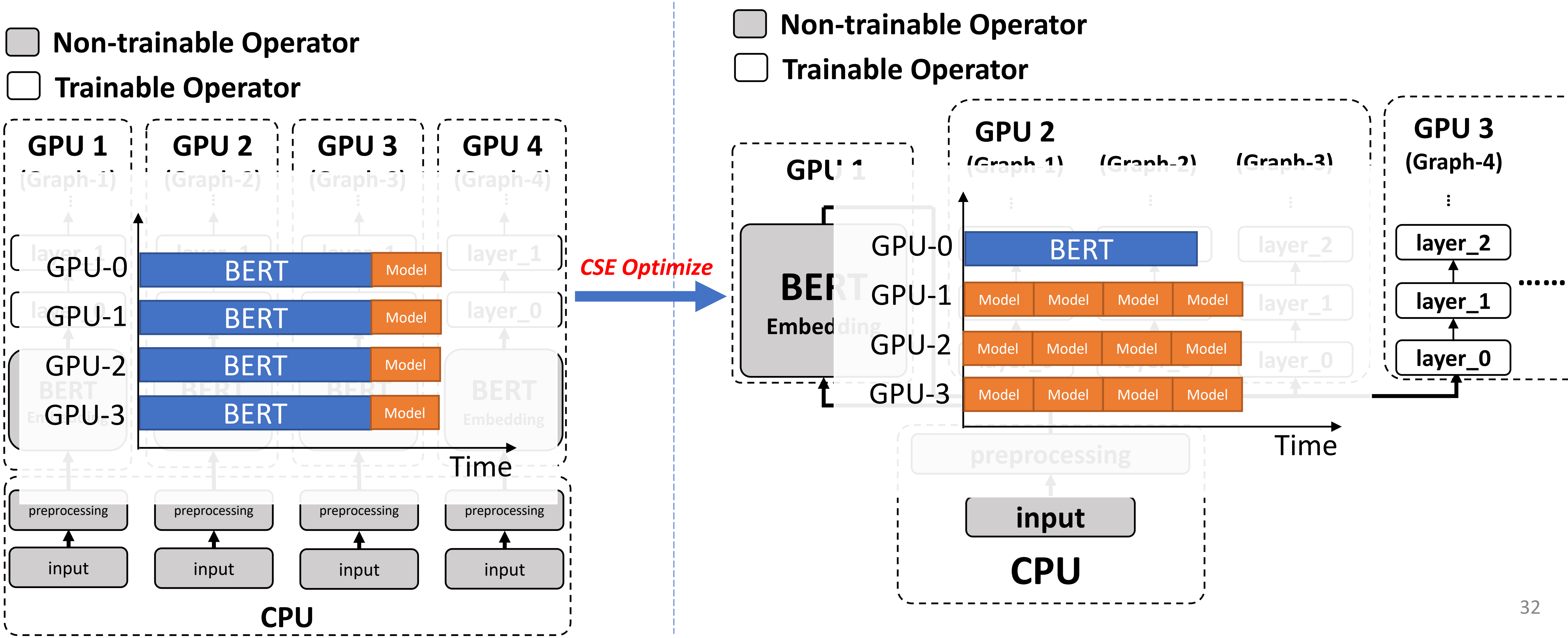
Common Sub-expression Elimination (CSE)

- CSE + Device Placement for GPU-based Embedding (e.g., BERT)



Common Sub-expression Elimination (CSE)

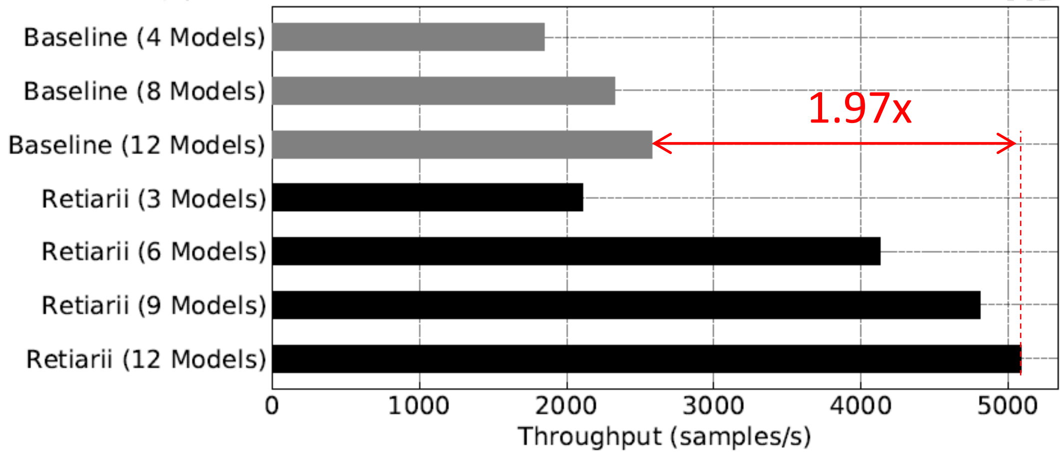
- CSE + Device Placement for GPU-based Embedding (e.g., BERT)



Common Sub-expression Elimination (CSE)

- CSE + Device Placement for GPU-based Embedding (e.g., BERT)

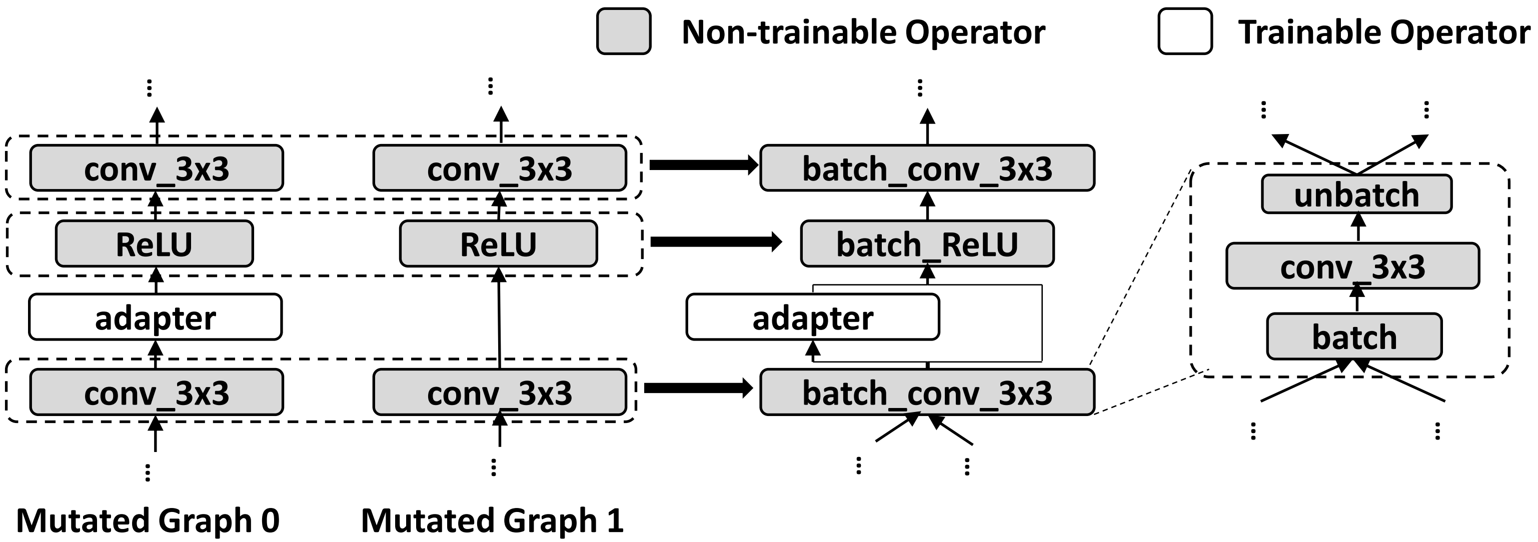
CSE of GPU-based BERT



TextNAS on SST dataset

Operator Batching

- De-duplicate common layers with different input



Please refer to our paper for details

Speeding up Neural Architecture Search (NAS)

- Three famous NAS solutions

NAS Solution	Search Space	Exploration Strategy
MnasNet [1]	Factorized Hierarchical Search Space	Reinforcement Learning
NASNet [2]	Normal Cell + Reduction Cell	Reinforcement Learning
AmoebaNet [3]	Normal Cell + Reduction Cell	Evolutionary Algorithm

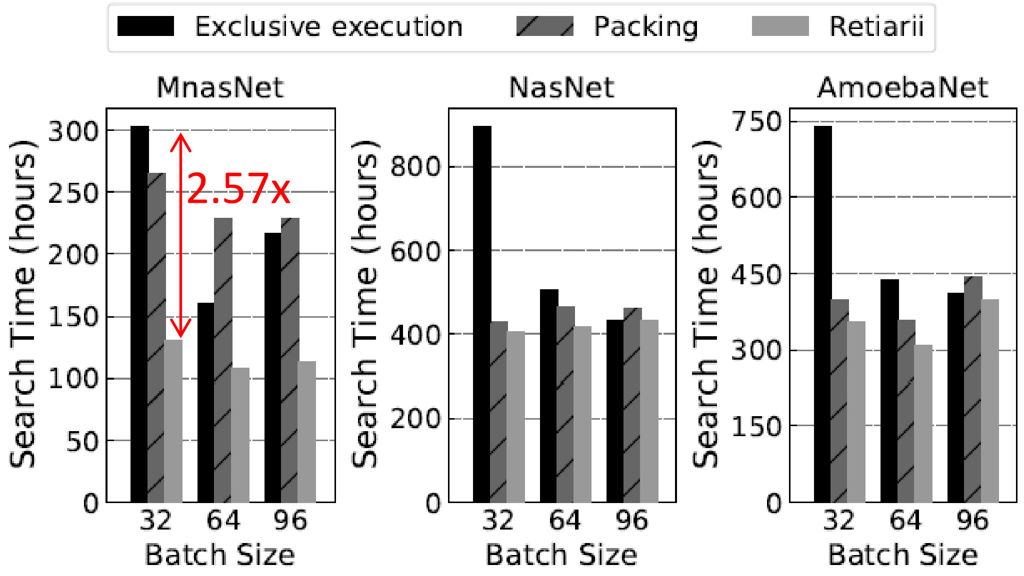
  - Time-consuming: they all need to explore over a large search space.
- Baselines
  - **Exclusive execution**: trains one model per GPU at a time
  - **Packing**: trains multiple models per GPU using NVIDIA CUDA MPS

Explore 1000 models on 4 V100 w/ 1 epoch training on ImageNet for each model

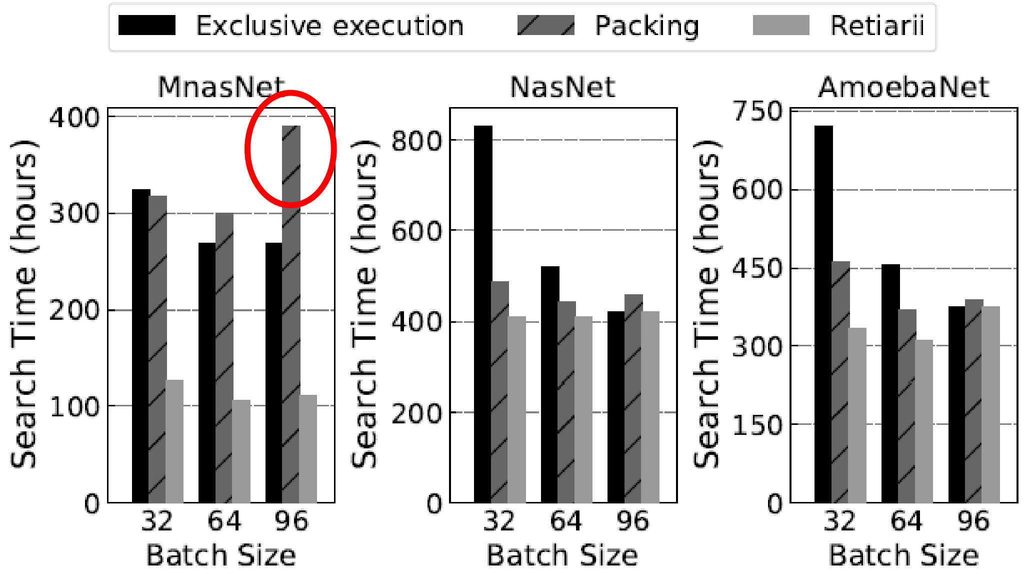
[1] Tan M, Chen B, Pang R, Vasudevan V, Sandler M, Howard A, Le QV. Mnasnet: Platform-aware neural architecture search for mobile. CVPR 2019
[2] Zoph B, Vasudevan V, Shlens J, Le QV. Learning transferable architectures for scalable image recognition. CVPR 2018
[3] Real E, Aggarwal A, Huang Y, Le QV. Regularized evolution for image classifier architecture search. AAAI 2019

End-to-End Experiment

Speeding up Neural Architecture Search (NAS)



(a) NVIDIA Data Loading Library (DALI)



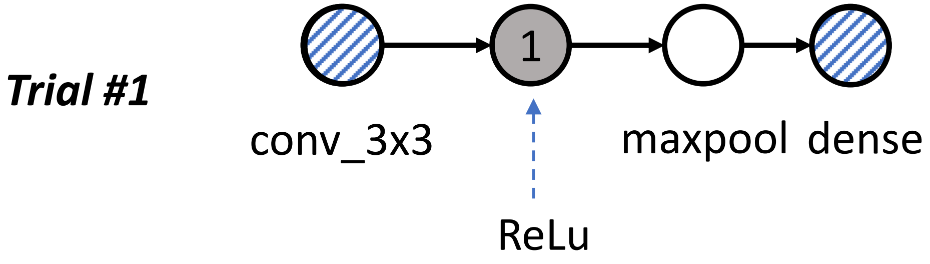
(b) PyTorch DataLoader

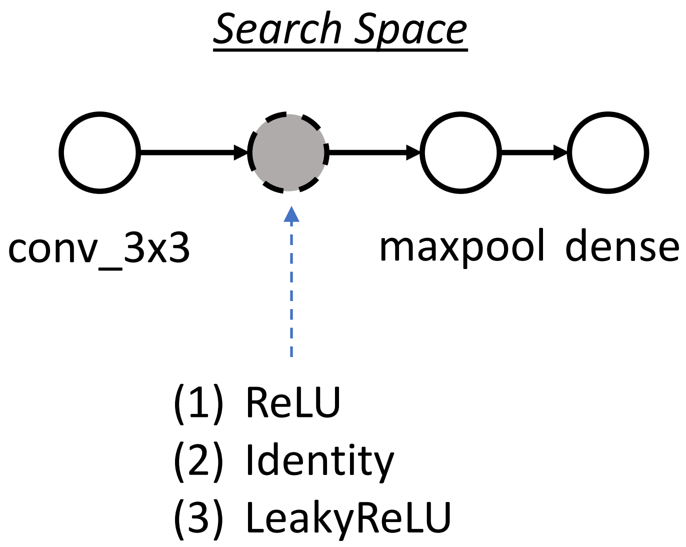
- Retiarii achieves up to 2.57 times speed up on three typical NAS solutions
  - Performance gain mainly from packing and CSE
  - Simultaneously run up to 22 of MnasNet models when Batch Size is 32

End-to-End Experiment

Speeding up Weight-Shared Training

- What is Weight Sharing?



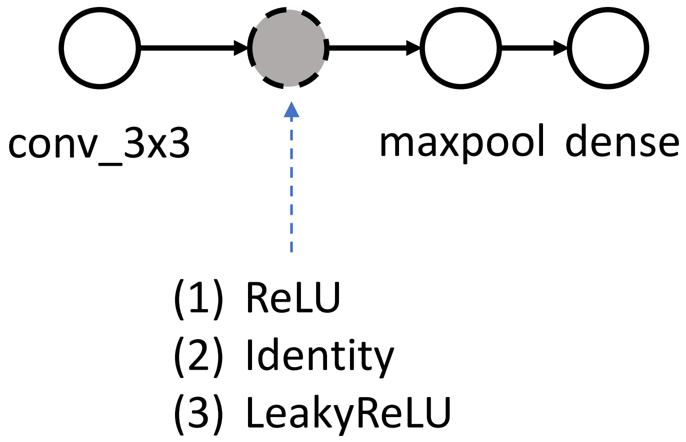


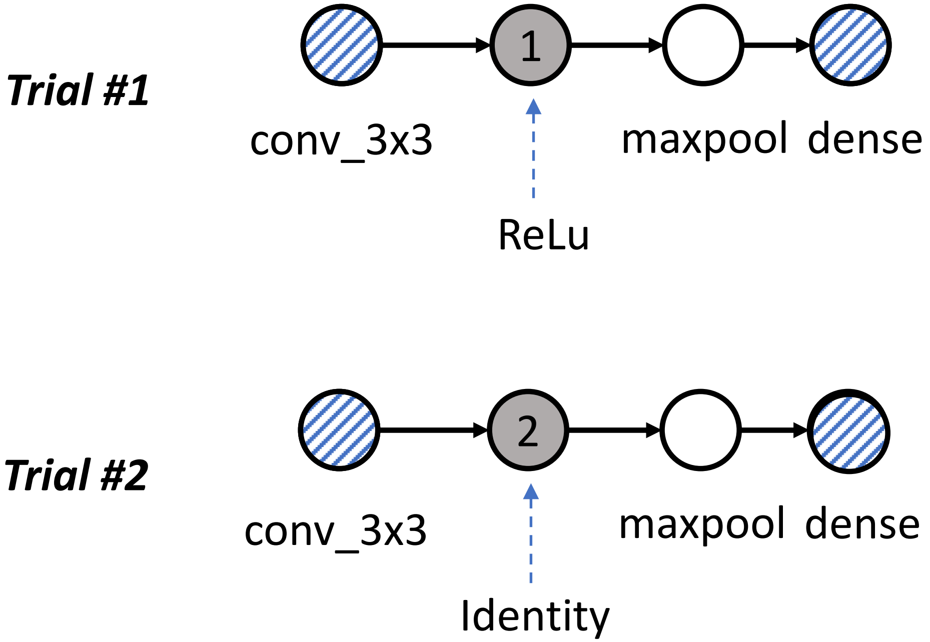
End-to-End Experiment

Speeding up Weight-Shared Training

- What is Weight Sharing?

Search Space

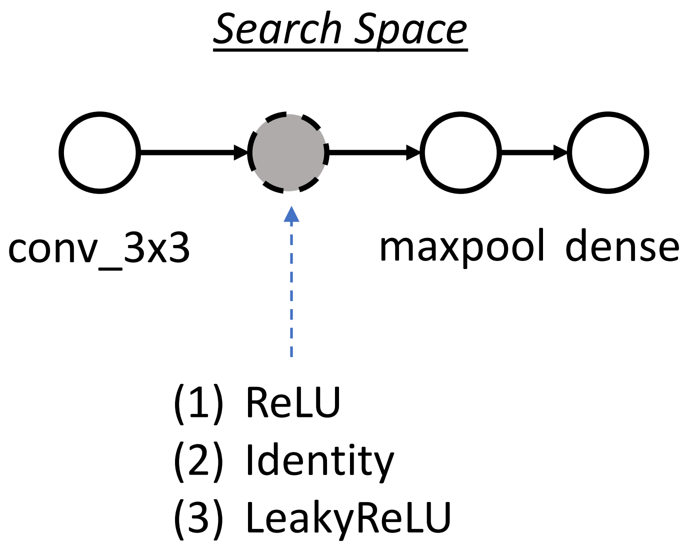


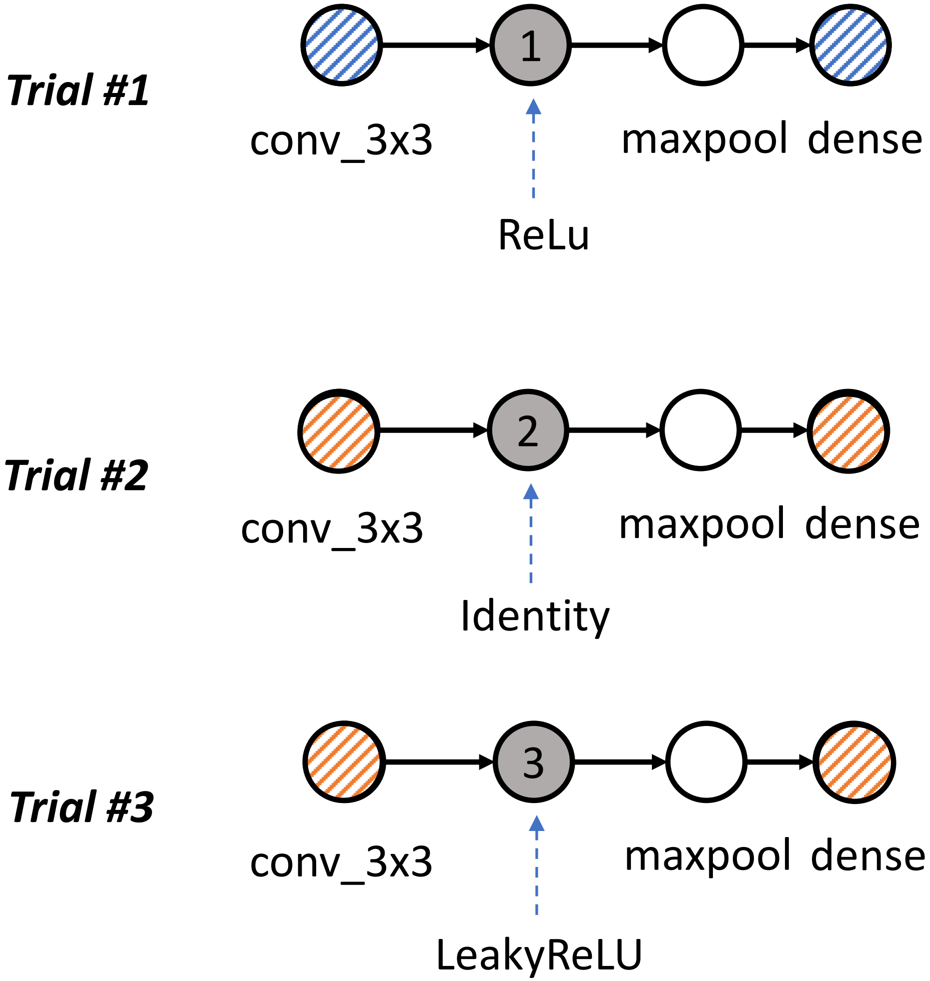


End-to-End Experiment

Speeding up Weight-Shared Training

- What is Weight Sharing?

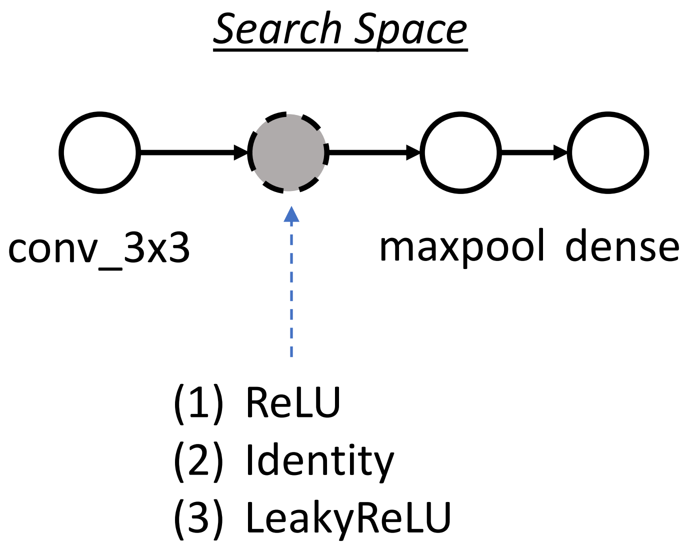


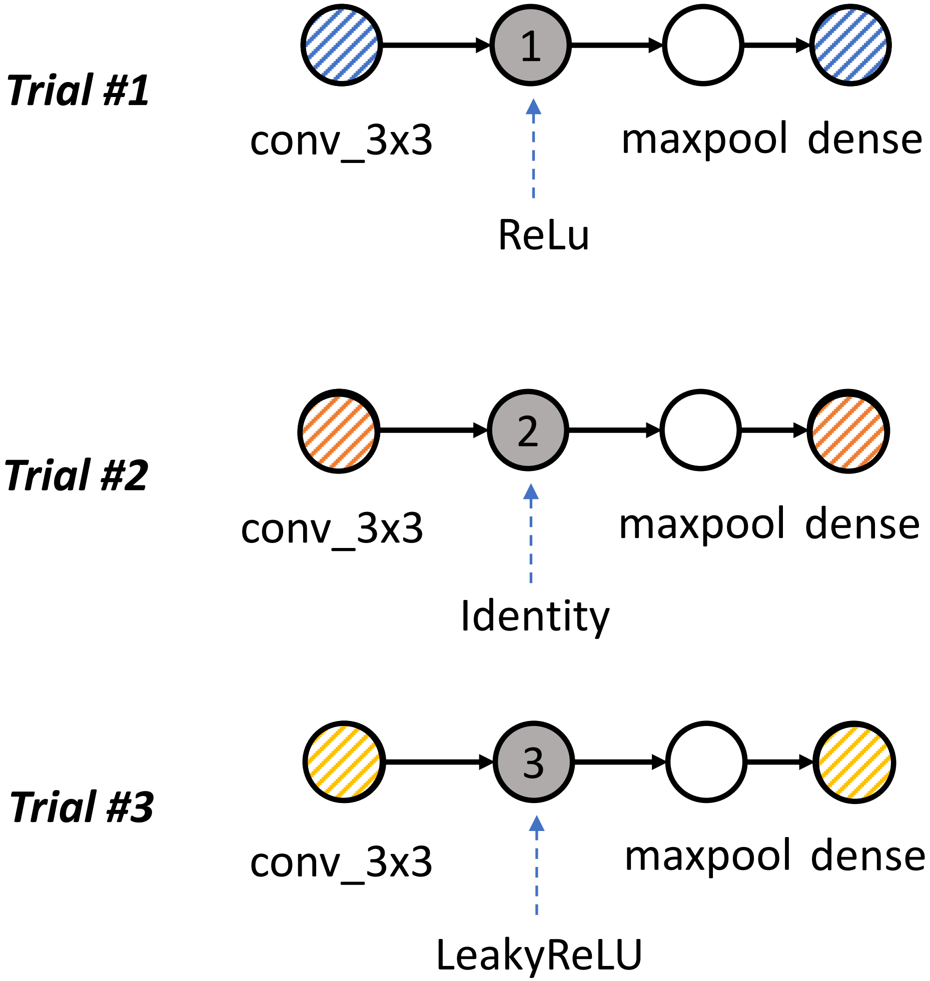


End-to-End Experiment

Speeding up Weight-Shared Training

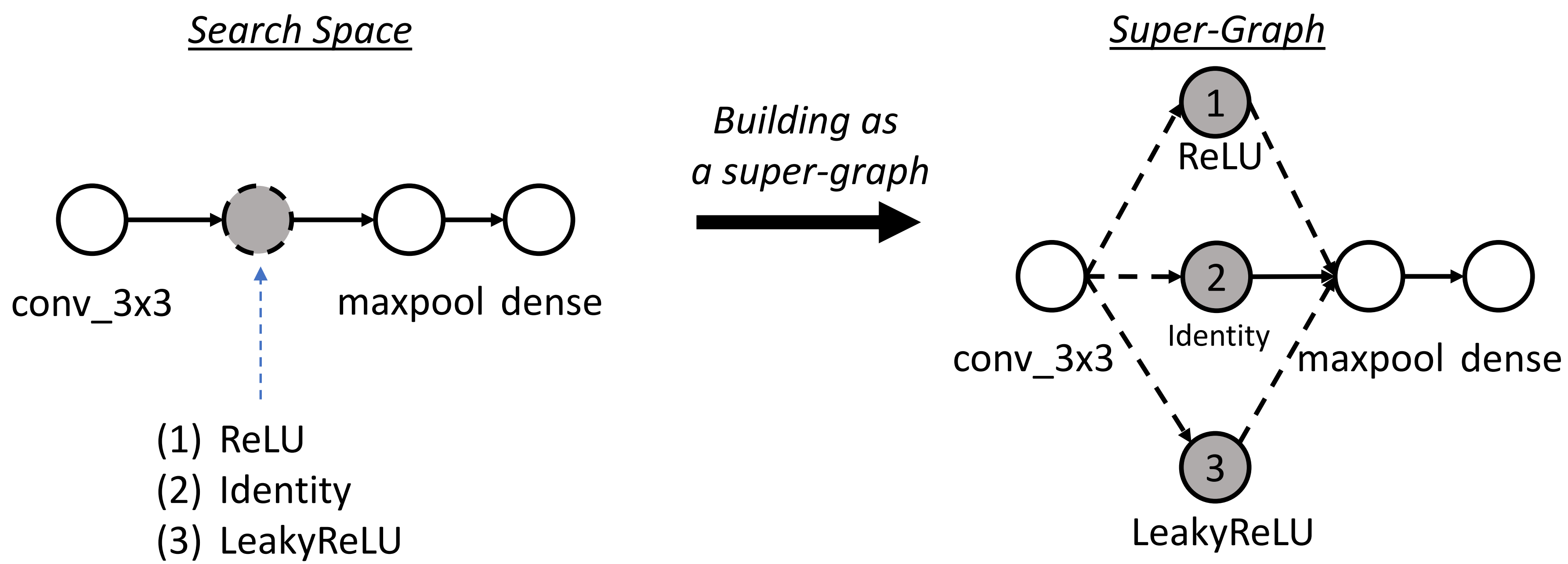
- What is Weight Sharing?





Speeding up Weight-Shared Training

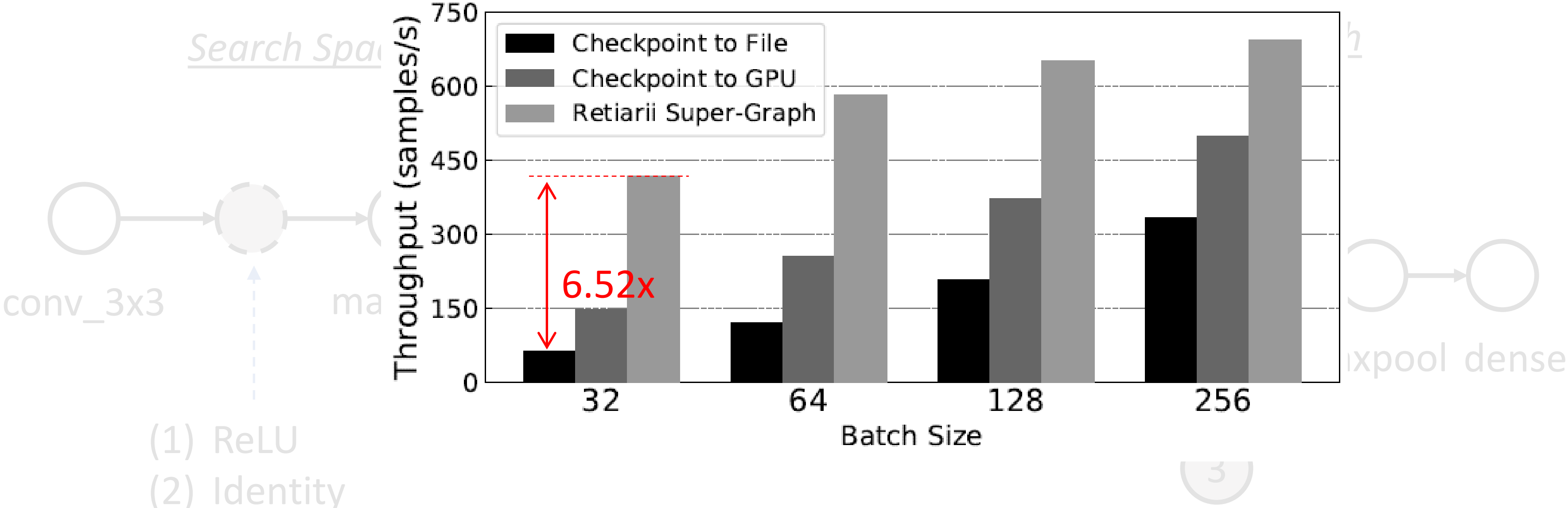
- Building a Super-Graph to encode the search space



End-to-End Experiment

Speeding up Weight-Shared Training

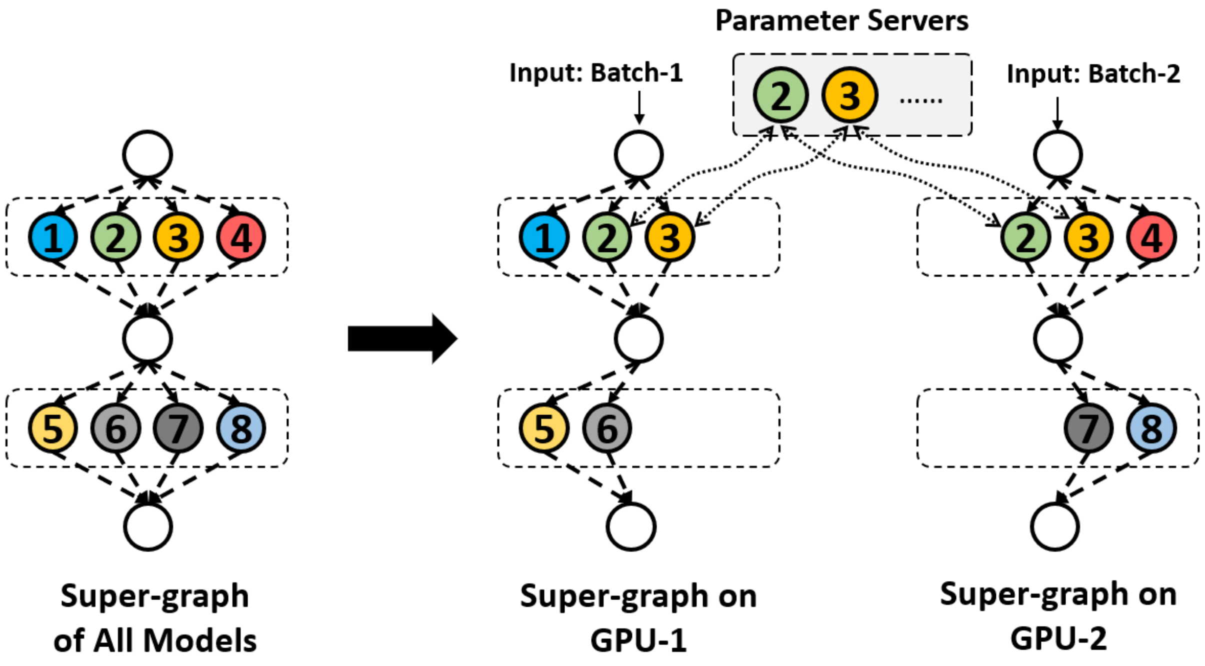
- Optimization of Super-Graph



Limited search space size!
Hard to scale to a large GPU cluster!

Speeding up Weight-Shared Training

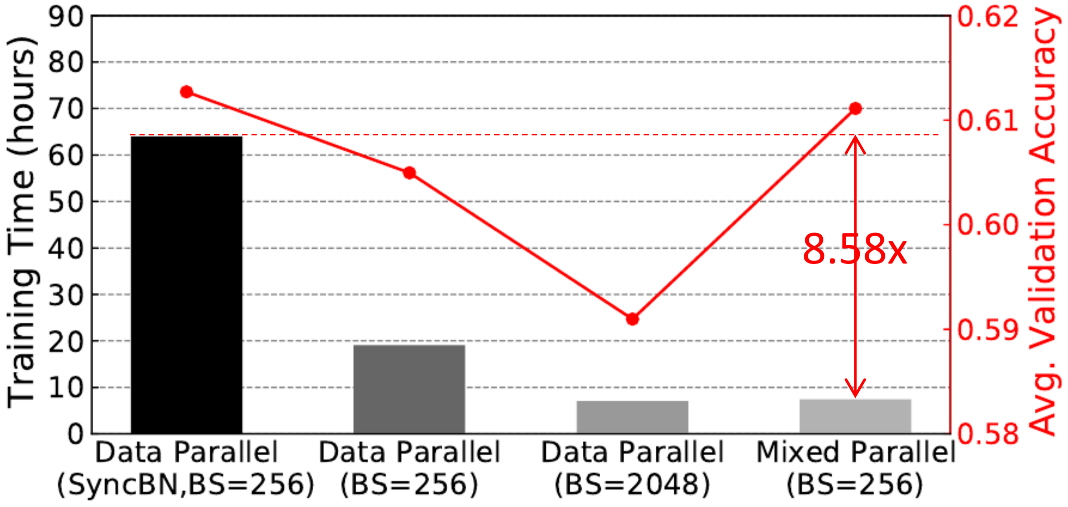
- Retiarii's Mixed Parallelism:
  - Model Parallelism: partitions the super-graph to multiple GPUs
  - Data Parallelism: feeds each partition with a different batch of data



End-to-End Experiment

Speeding up Weight-Shared Training

- Retiarii's mixed parallelism greatly reduces exploratory-training time (only 7.45 hours)
  - A famous weight-shared NAS: SPOS [1]
  - 8.58x speed-up over Data Parallel training w/ SyncBN on 8 V100 GPUs
  - Almost the same validation accuracy



[1] Guo Z, Zhang X, Mu H, Heng W, Liu Z, Wei Y, Sun J. Single path one-shot neural architecture search with uniform sampling. arXiv preprint arXiv:1904.00420. 2019 Mar 31.

Conclusion

- Retiarii is a new DNN framework designed for exploratory-training
- Retiarii provides new interfaces for DNN model developers to design & explore new models efficiently
- The simple but powerful Mutator abstraction
  - Expressiveness
  - Reusability of exploration strategies
  - Enabling cross-model optimization

https://github.com/microsoft/nni

Thanks! Q&A

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

https://github.com/microsoft/nni/tree/retiarii_artifact

No framework → Deep learning framework → Exploratory-Training framework

No framework	Deep learning framework	Exploratory-Training framework
	Caffe2, PyTorch, TensorFlow, mxnet, Chainer	Retiarii on NNI Neural Network Intelligence https://github.com/microsoft/nni
Programming with libraries	Making programming a DNN model easier and faster	Making DNN model exploring easier and faster