Microsoft

Neural Architecture Search : Part 2

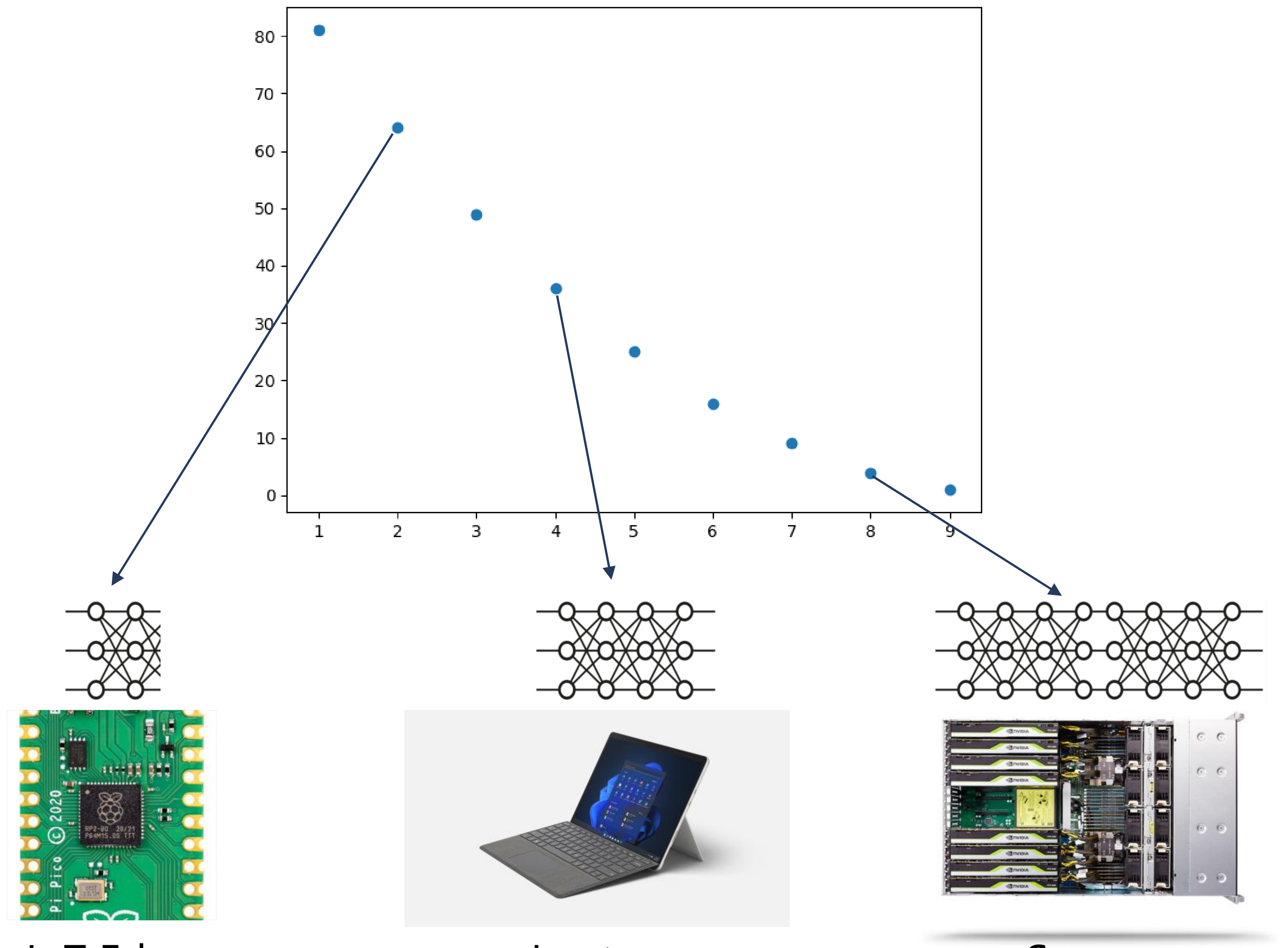
Debadeepta Dey, Microsoft Research (dedey@microsoft.com)
Colin White, Abacus.AI (colin@abacus.ai)

Roadmap

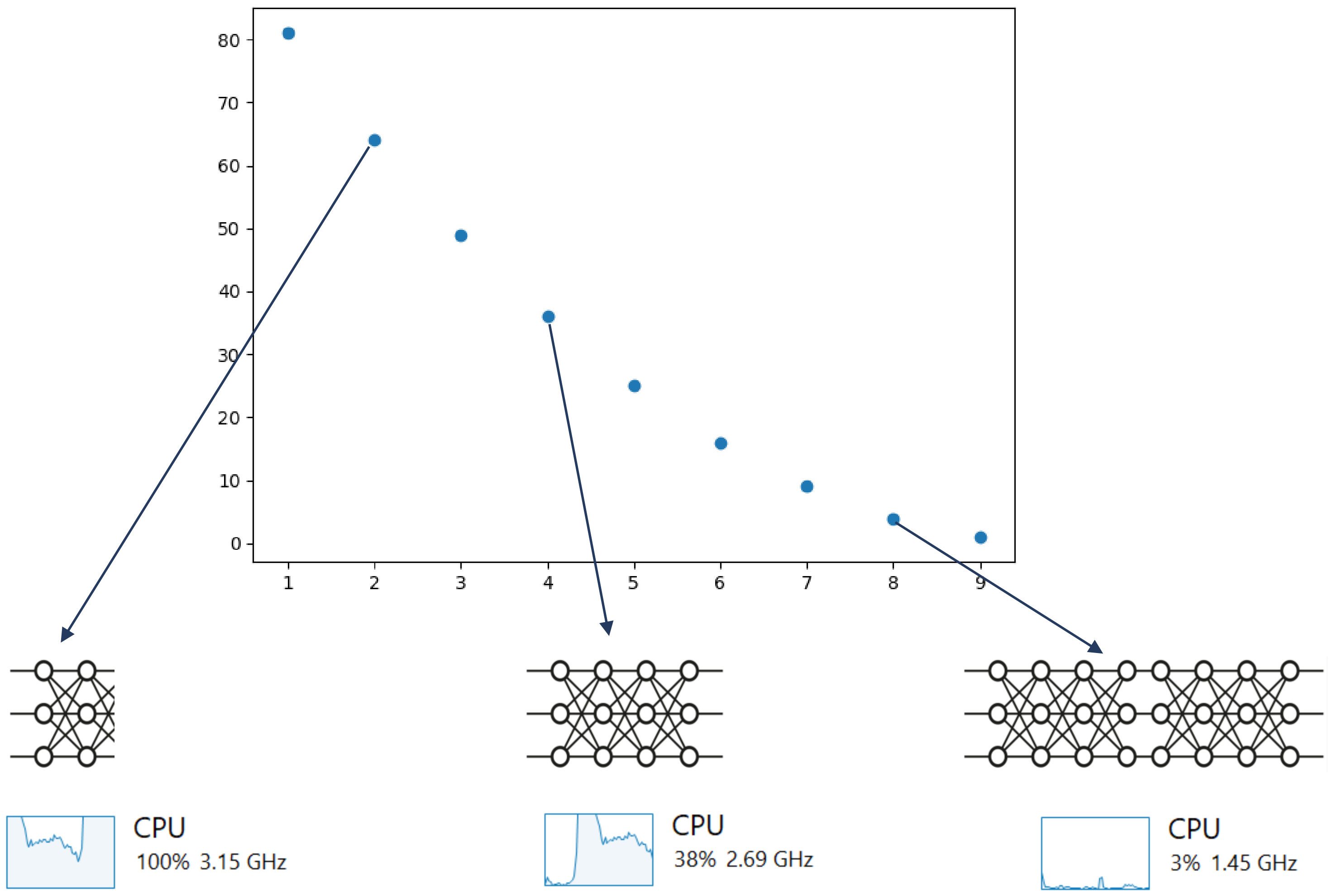
- The Power of Pareto-Frontiers
- Weight-sharing Methods
  - ENAS
  - OFA
- Recent Transformer-based Search Spaces
- Petridish
- Reproducibility, Fair Comparison, Best Practices
- Open Problems

The Power of Pareto-Frontiers

The Power of Pareto-Frontier: Varying compute ability

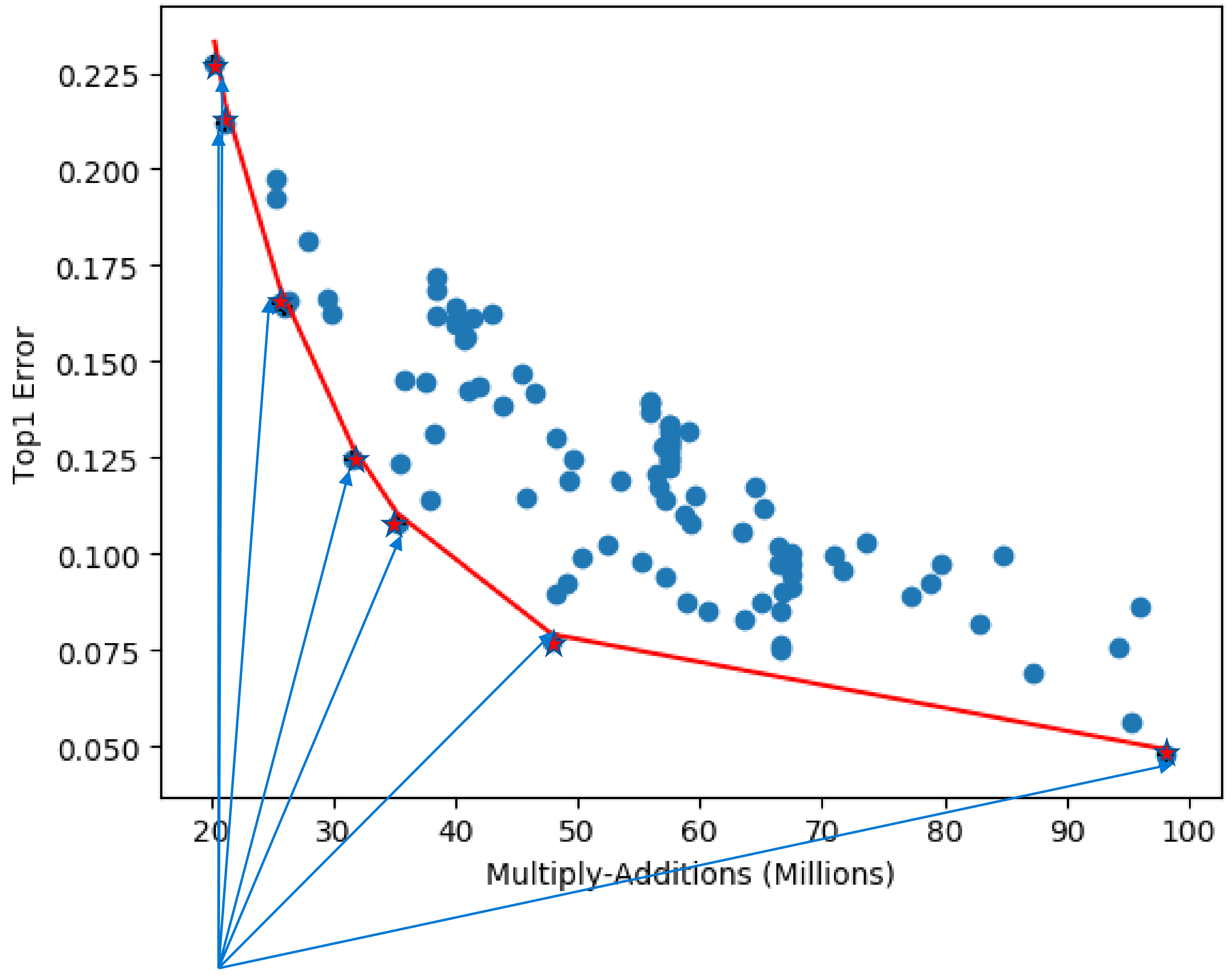


The Power of Pareto-Frontier: Dynamic device load



Pareto-frontiers are *generalization* of model compression!

Search Once, Deploy Everywhere!



Train models on the frontier!

Pareto-Frontier Search Methods

- The vast majority of methods in current NAS literature do *NOT* output pareto-frontiers!
- Combining multiple objectives via scalarization does *NOT* output pareto-frontiers!
- Can we leverage single-objective search methods and turn them into pareto-frontier output methods?

Bag of Baselines for Multi-objective Joint Neural Architecture Search and Hyperparameter Optimization

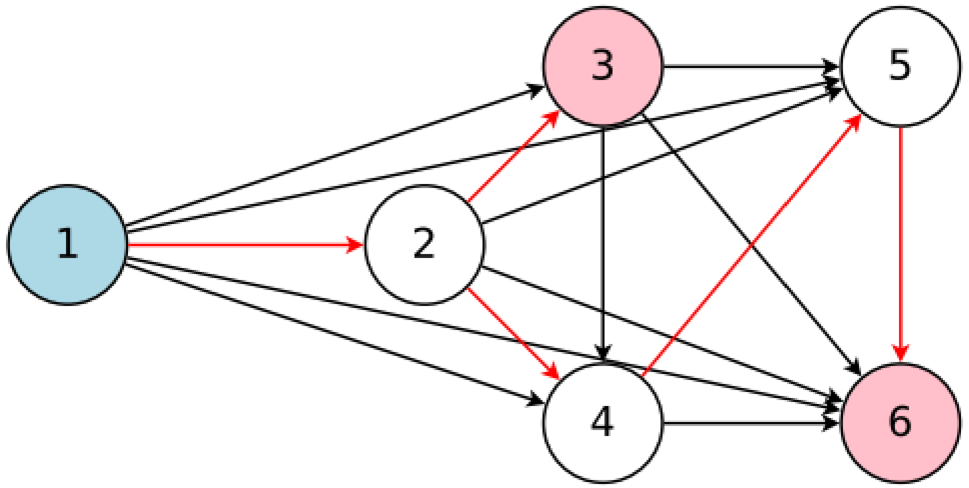
Guerrero-Viu et al., AutoML Workshop at ICML 2021

Weight-sharing Methods

Efficient Neural Architecture Search via Parameter Sharing
Pham et al, ICML 2018

ENAS

ENAS



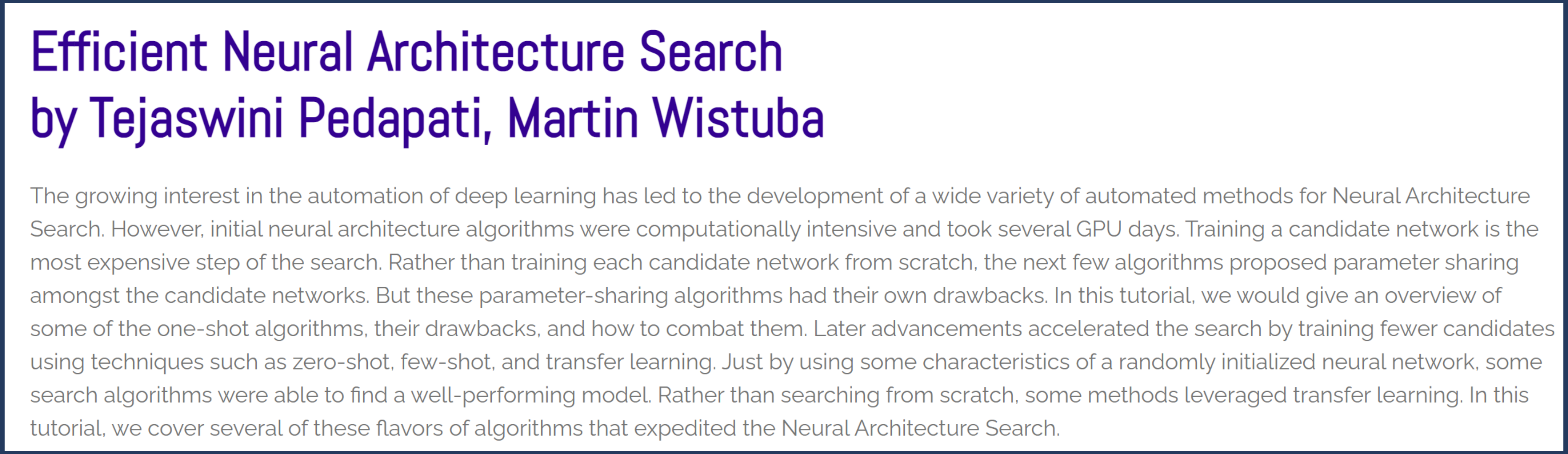
"The main contribution of this work is to improve the efficiency of NAS by forcing all child models to share weights to eschew training each child model from scratch to convergence."

- Uses a single Nvidia 1080Ti GPU!
  - Search < 16 hours!
  - Compared to NAS via RL, 1000x reduction in search time!

Diagram credit:
ENAS ICML 2018

Microsoft

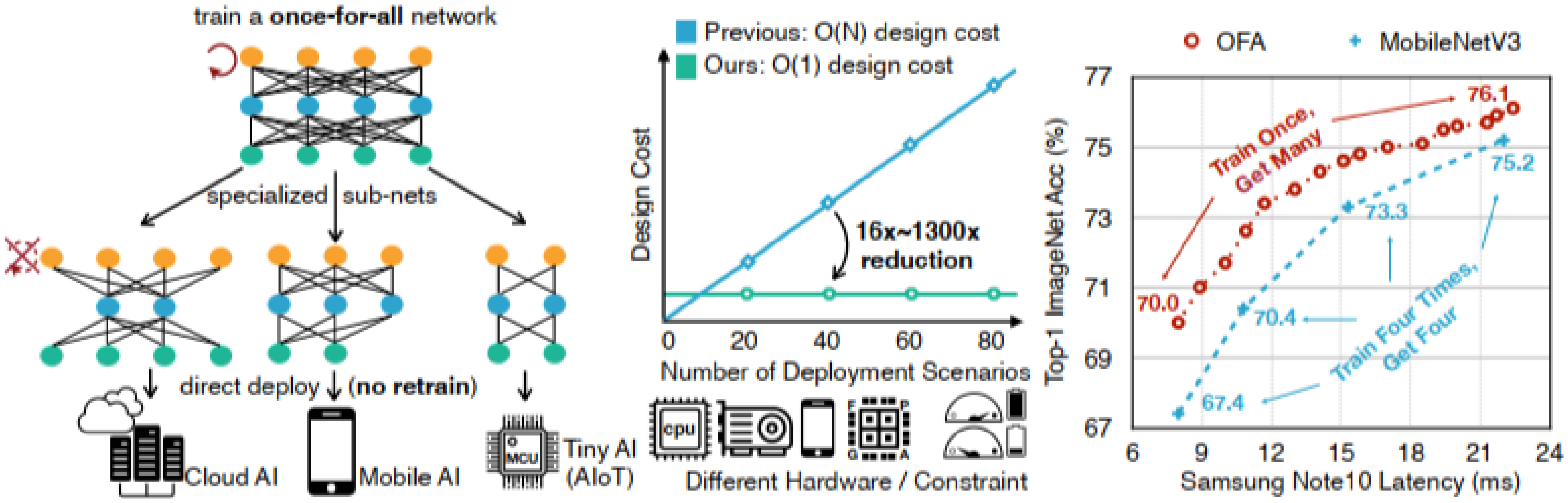
Please attend NAS 2 for weight-sharing in-depth!



Efficient Neural Architecture Search
by Tejaswini Pedapati, Martin Wistuba

The growing interest in the automation of deep learning has led to the development of a wide variety of automated methods for Neural Architecture Search. However, initial neural architecture algorithms were computationally intensive and took several GPU days. Training a candidate network is the most expensive step of the search. Rather than training each candidate network from scratch, the next few algorithms proposed parameter sharing amongst the candidate networks. But these parameter-sharing algorithms had their own drawbacks. In this tutorial, we would give an overview of some of the one-shot algorithms, their drawbacks, and how to combat them. Later advancements accelerated the search by training fewer candidates using techniques such as zero-shot, few-shot, and transfer learning. Just by using some characteristics of a randomly initialized neural network, some search algorithms were able to find a well-performing model. Rather than searching from scratch, some methods leveraged transfer learning. In this tutorial, we cover several of these flavors of algorithms that expedited the Neural Architecture Search.

Once-for-All: Train One Network and Specialize it for Efficient Deployment
Cai et al., ICLR 2020



train a once-for-all network
specialized sub-nets
direct deploy (no retrain)
Cloud AI
Mobile AI
Tiny AI (AIoT)
Previous: O(N) design cost
Ours: O(1) design cost
Design Cost
16x~1300x reduction
0
20
40
60
80
Number of Deployment Scenarios
Different Hardware / Constraint
OFA
MobileNetV3
Top-1 ImageNet Acc (%)
Train Once, Get Many
Train Four Times, Get Four
76.1
75.2
73.3
70.4
70.0
67.4
Samsung Note10 Latency (ms)

Diagram credit: OFA ICLR 2020

Phase 1: Train supergraph

$$\min_{W_o} \sum_{arch_i} \mathcal{L}_{val}\big(C(W_o, arch_i)\big)$$

- Want to find weights such that every subgraph is competitive wrt the subgraph being independently trained!
- Exponentially many subgraphs!
  - Infeasible to enumerate and train each separately. ☹
- Can randomly sample a few each step and update shared weights (remember ENAS!)
  - Updates interfere with each other leading to reduced performance ☹
- Solution: Train the biggest and progressively shrink down!

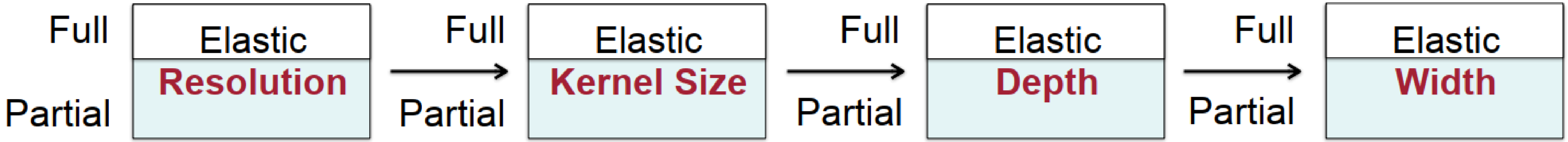
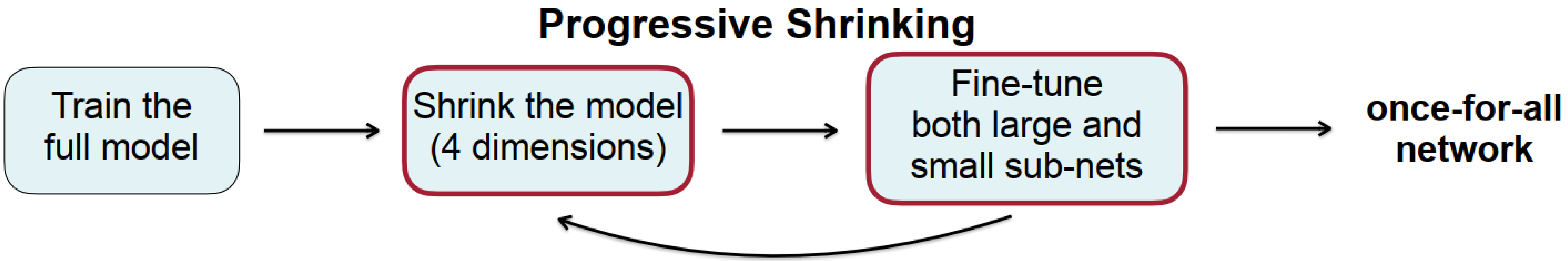
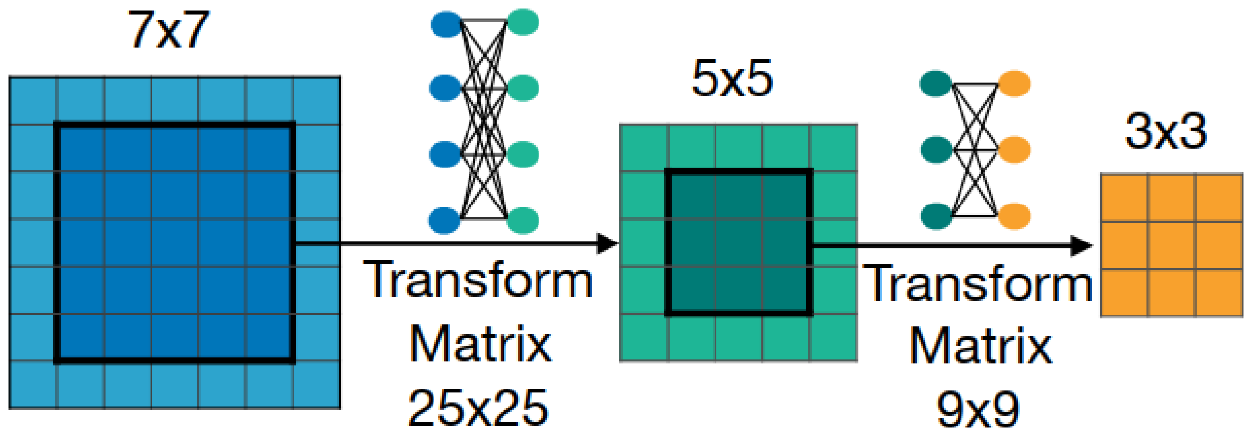


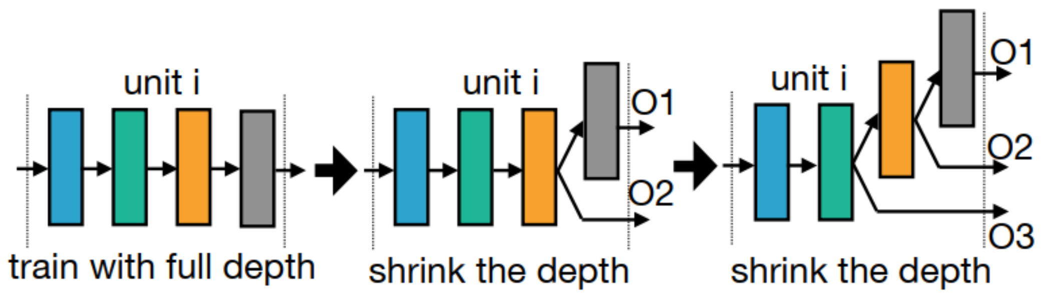
Diagram credit: OFA ICLR 2020

Phase 1: Train supergraph


Progressive Shrinking
Train the full model
Shrink the model (4 dimensions)
Fine-tune both large and small sub-nets
once-for-all network



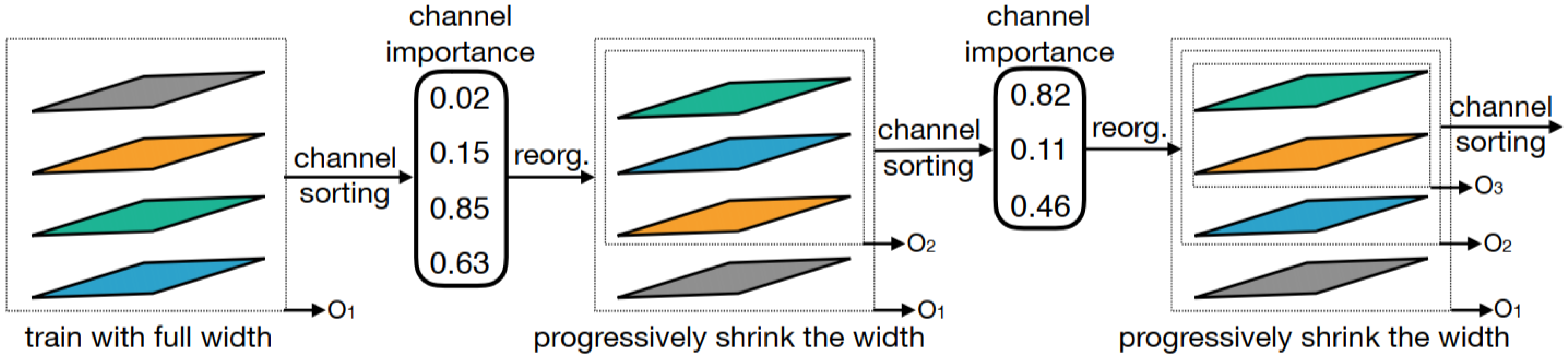
Kernel Shrinking



Depth Shrinking

Diagram credit: OFA ICLR 2020

Phase 1: Train supergraph



Width shrinking

- Throughout kernel, depth and width shrinking sample different input resolutions.
- Important detail: Use distillation to guide training of smaller architectures!
- Phase 1 cost: 1200 GPU hours (~3 days with 16 GPUs)

Diagram credit: OFA ICLR 2020

Phase 2: Train regressors

- Sample 16k different architectures – input image sizes and measure accuracy on validation set to generate (architecture, accuracy) tuples.
  - Train small NN to predict accuracy given architecture as input.
- Do same on each target platform to get (architecture, latency) tuples.
  - Train small NN to predict latency given architecture as input.

- Phase 2 cost: ~40 GPU hours

Search

- Simple! Use evolutionary search/RL/random search against the simulators (regressors from Phase 2)
- Search cost: a few minutes on a laptop!

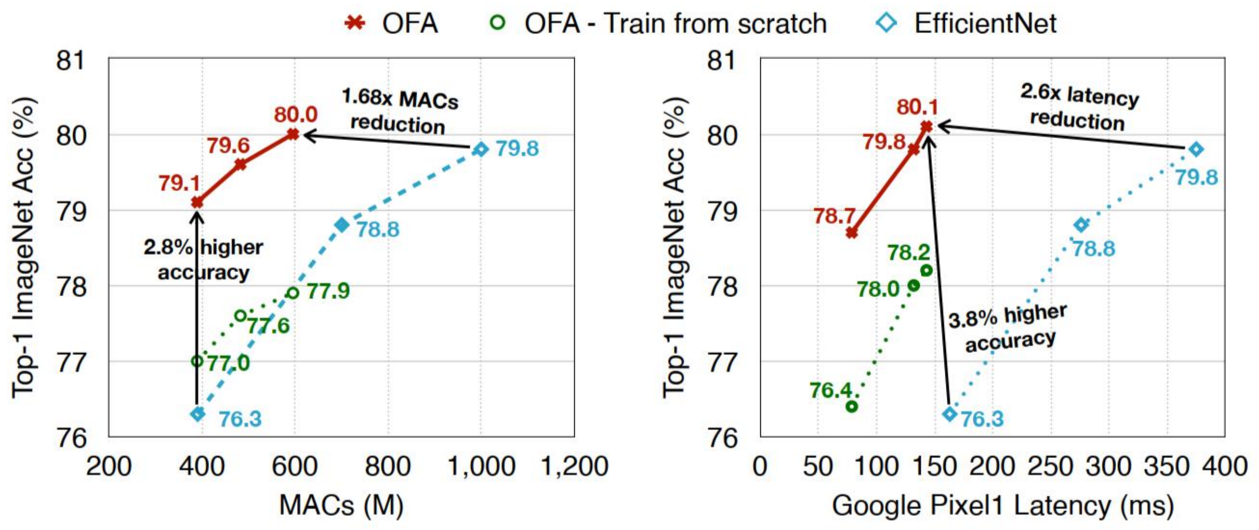
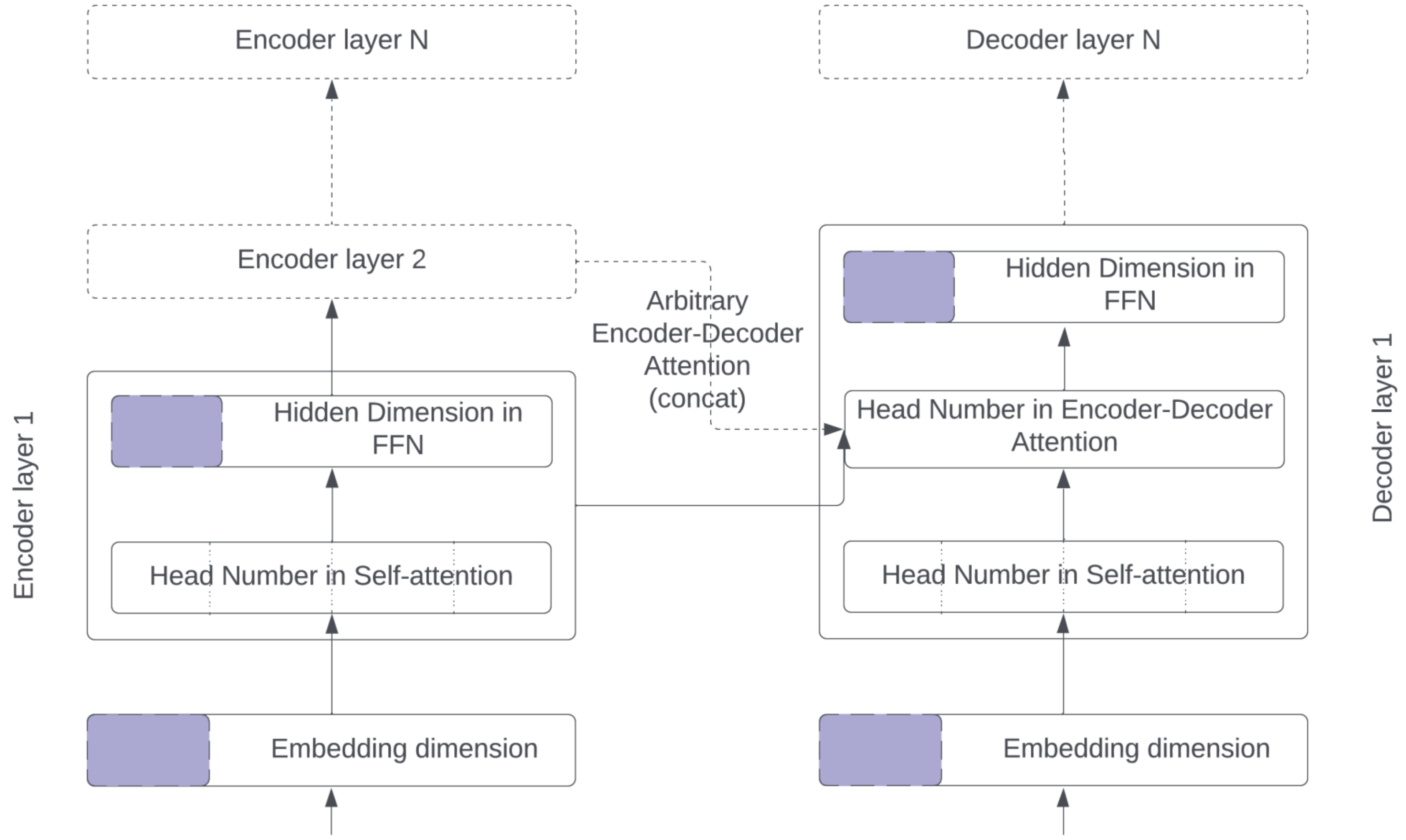


Figure 9: OFA achieves 80.0% top1 accuracy with 595M MACs and 80.1% top1 accuracy with 143ms Pixel1 latency, setting a new SOTA ImageNet top1 accuracy on the mobile setting.

Diagram credit: OFA ICLR 2020

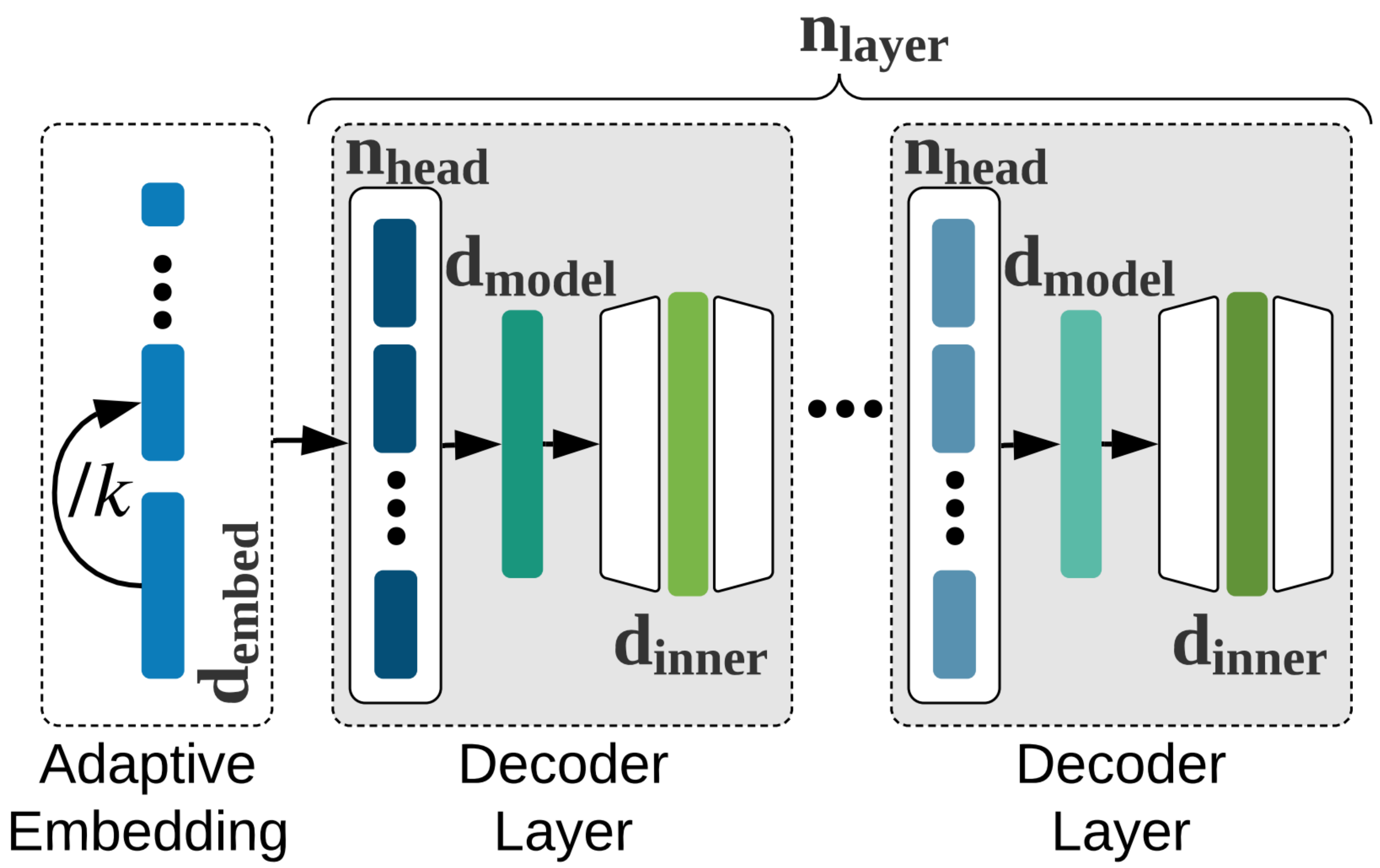
Recent Transformer-based Search Spaces

HAT: Hardware-Aware Transformers for Efficient Natural Language Processing



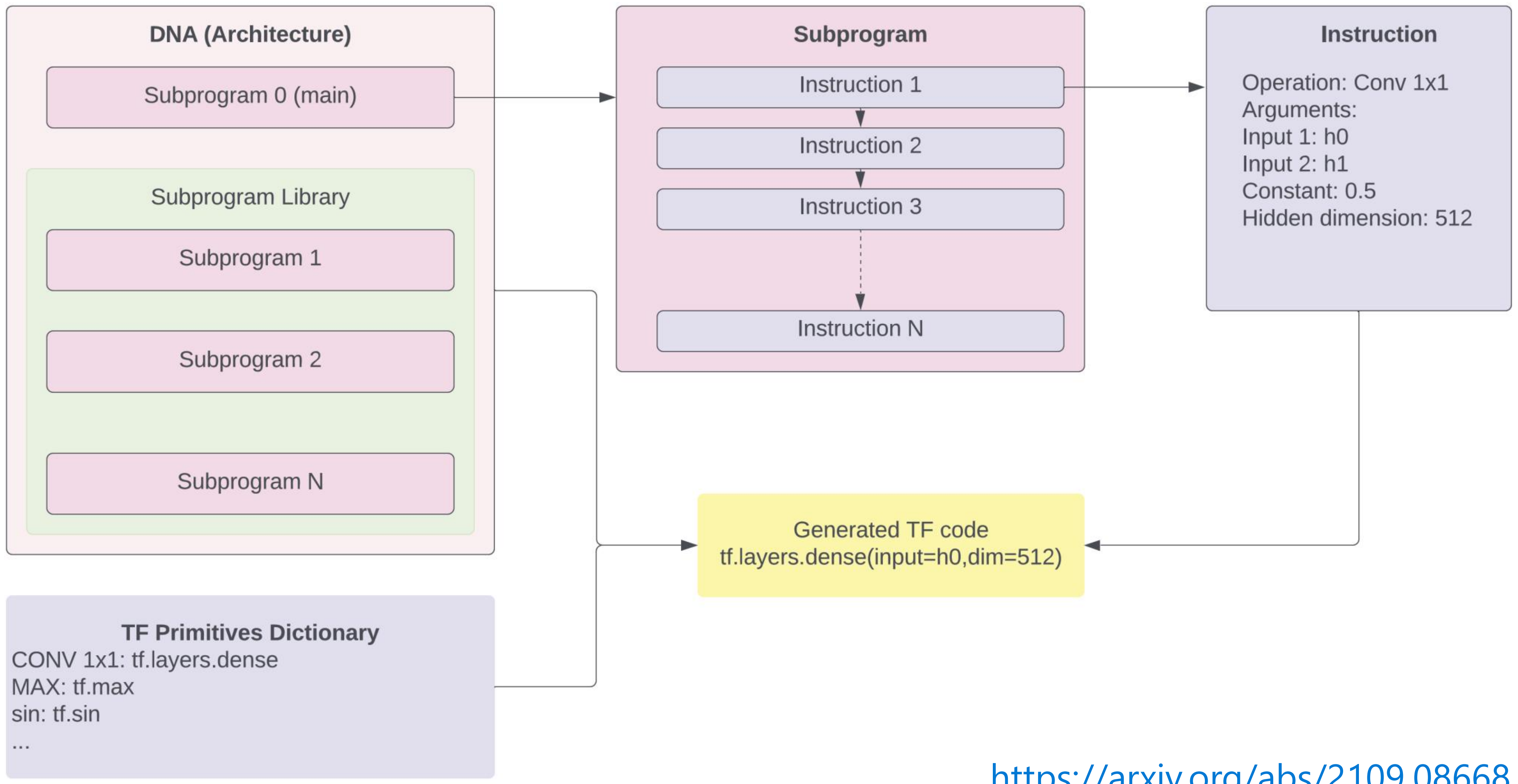
https://arxiv.org/abs/2005.14187

LiteTransformerSearch: Training-free On-device Search for Efficient Autoregressive Language Models



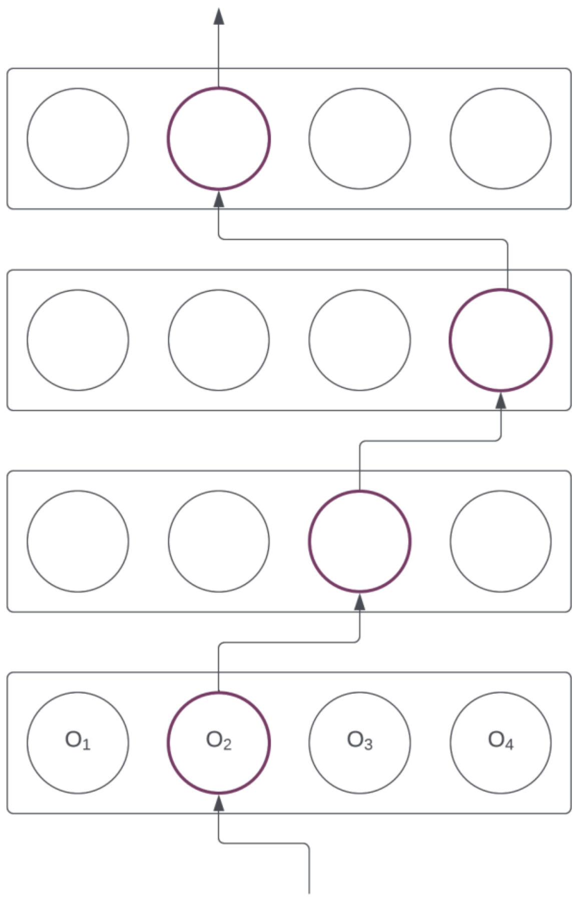
https://arxiv.org/pdf/2203.02094.pdf

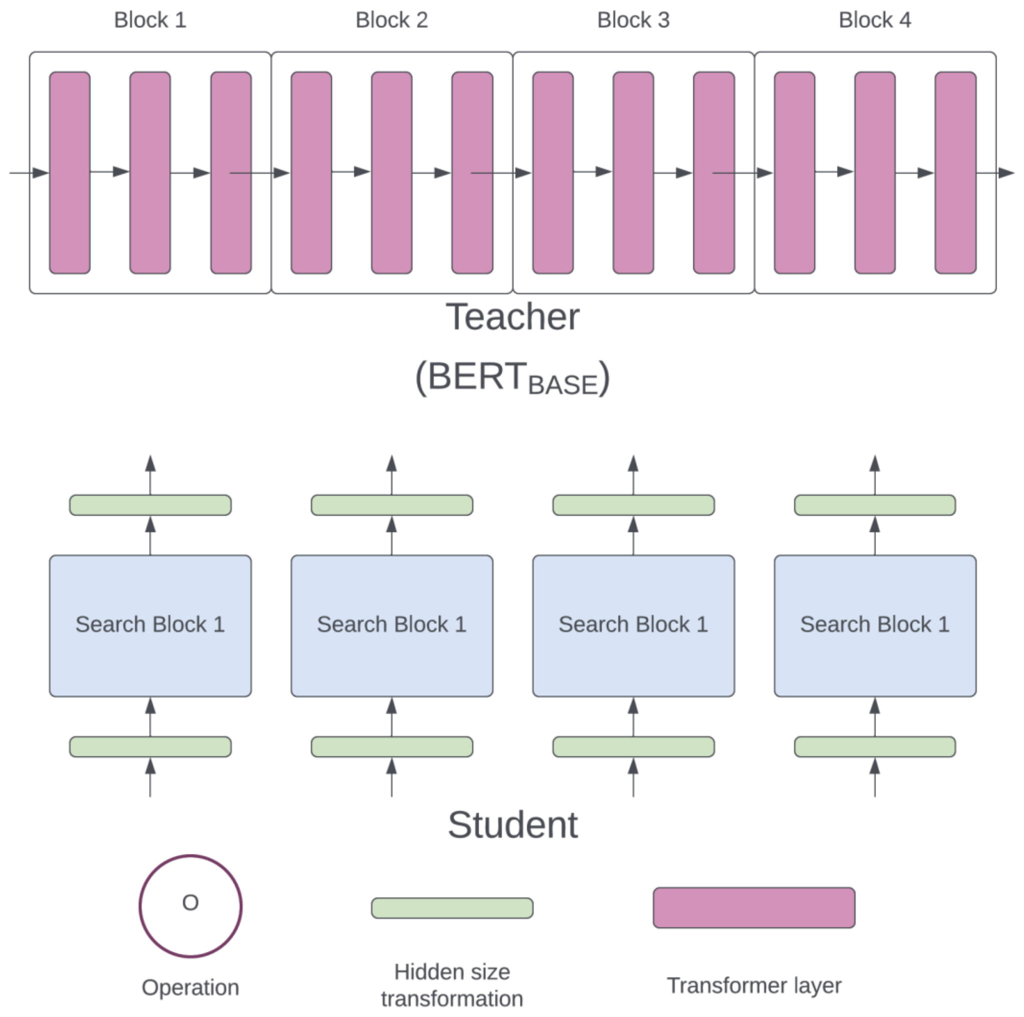
Primer: Searching for Efficient Transformers for Language Modeling



https://arxiv.org/abs/2109.08668

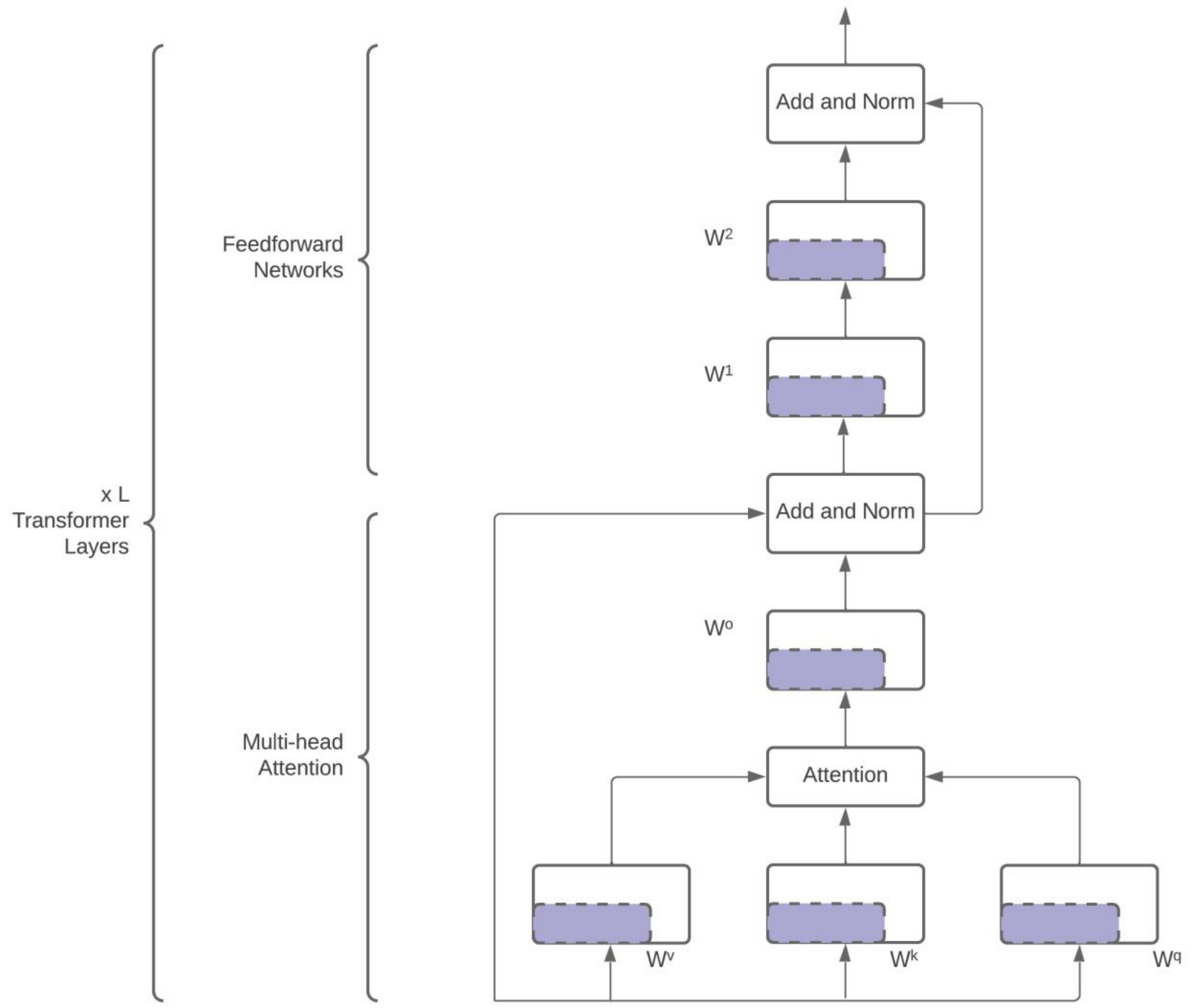
NAS-BERT: Task-Agnostic and Adaptive-Size BERT Compression with Neural Architecture Search





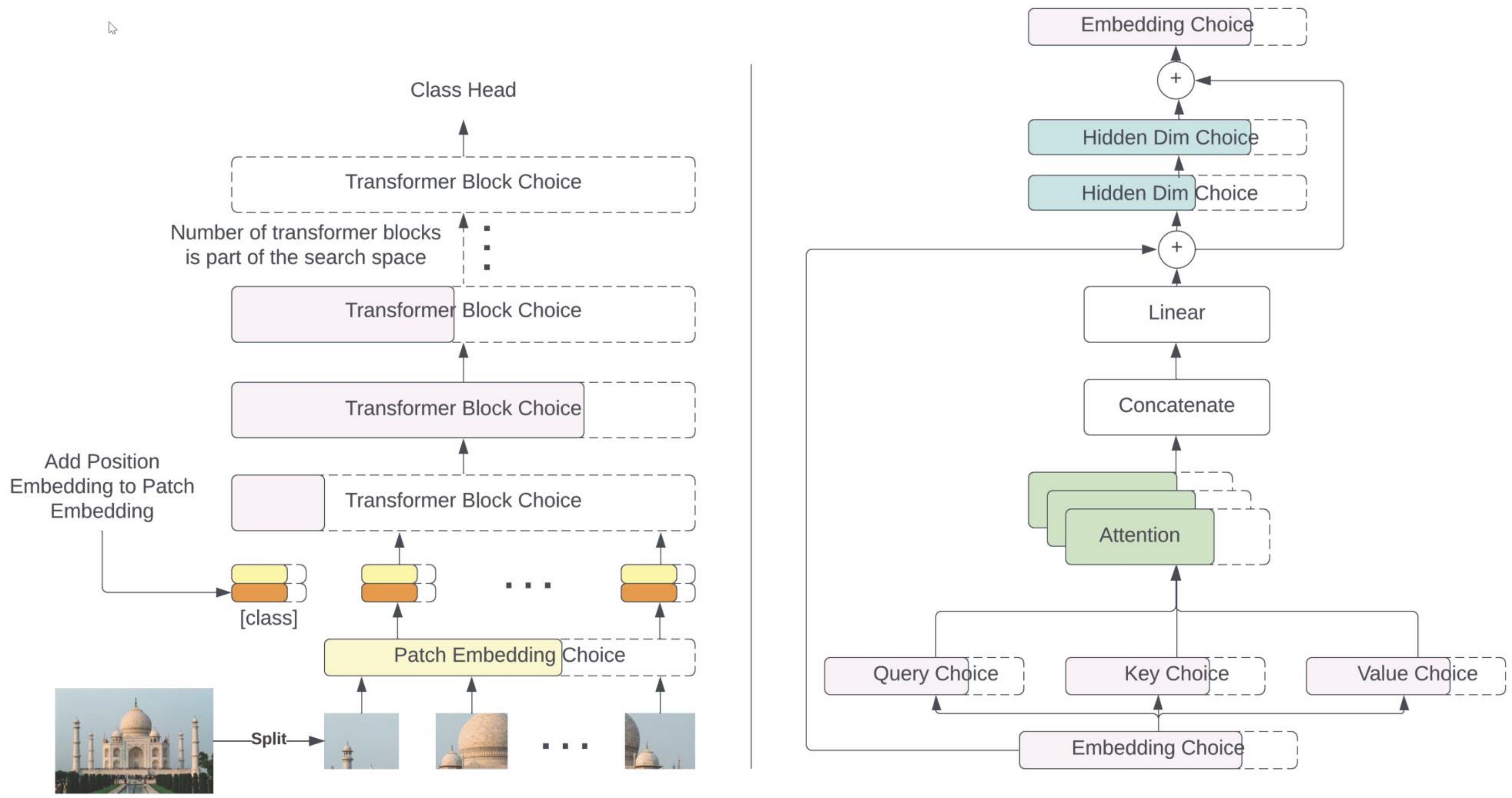
https://arxiv.org/abs/2105.14444

AutoTinyBERT: Automatic Hyper-parameter Optimization for Efficient Pre-trained Language Models



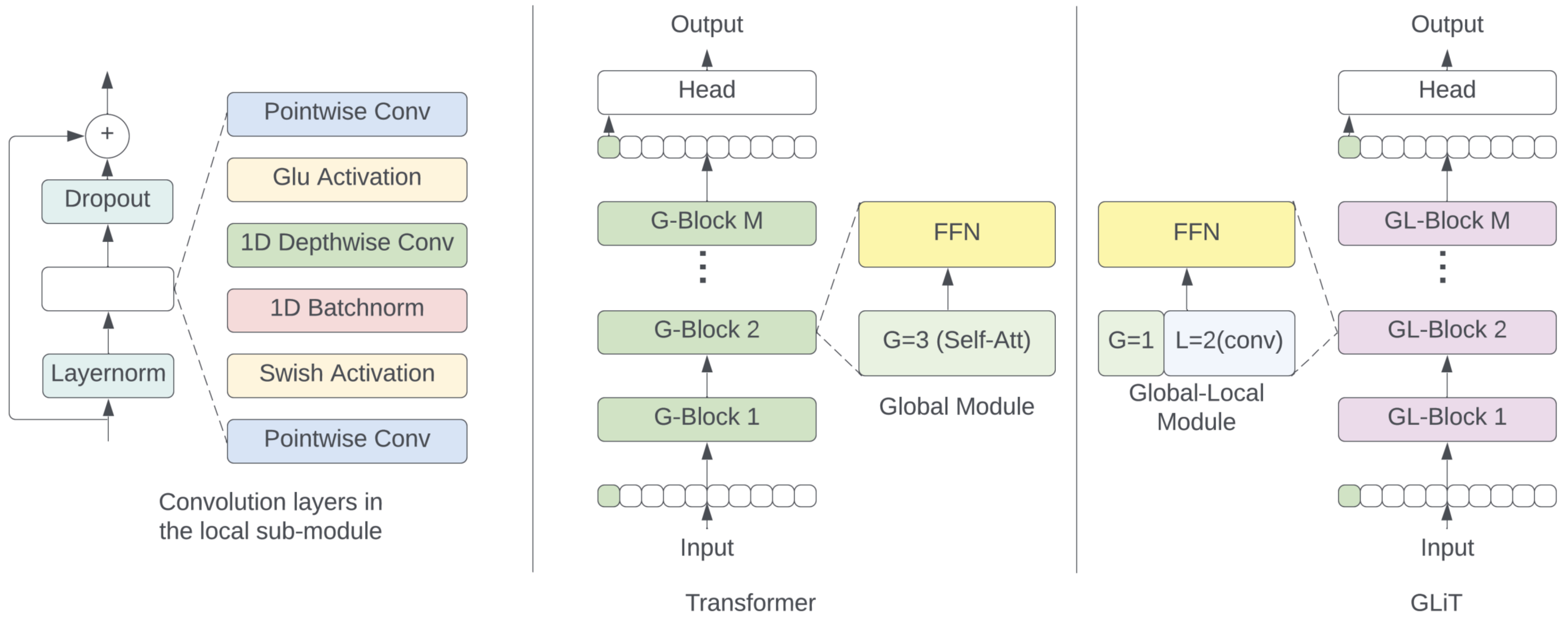
https://aclanthology.org/2021.acl-long.400.pdf

AutoFormer: Searching Transformers for Visual Recognition



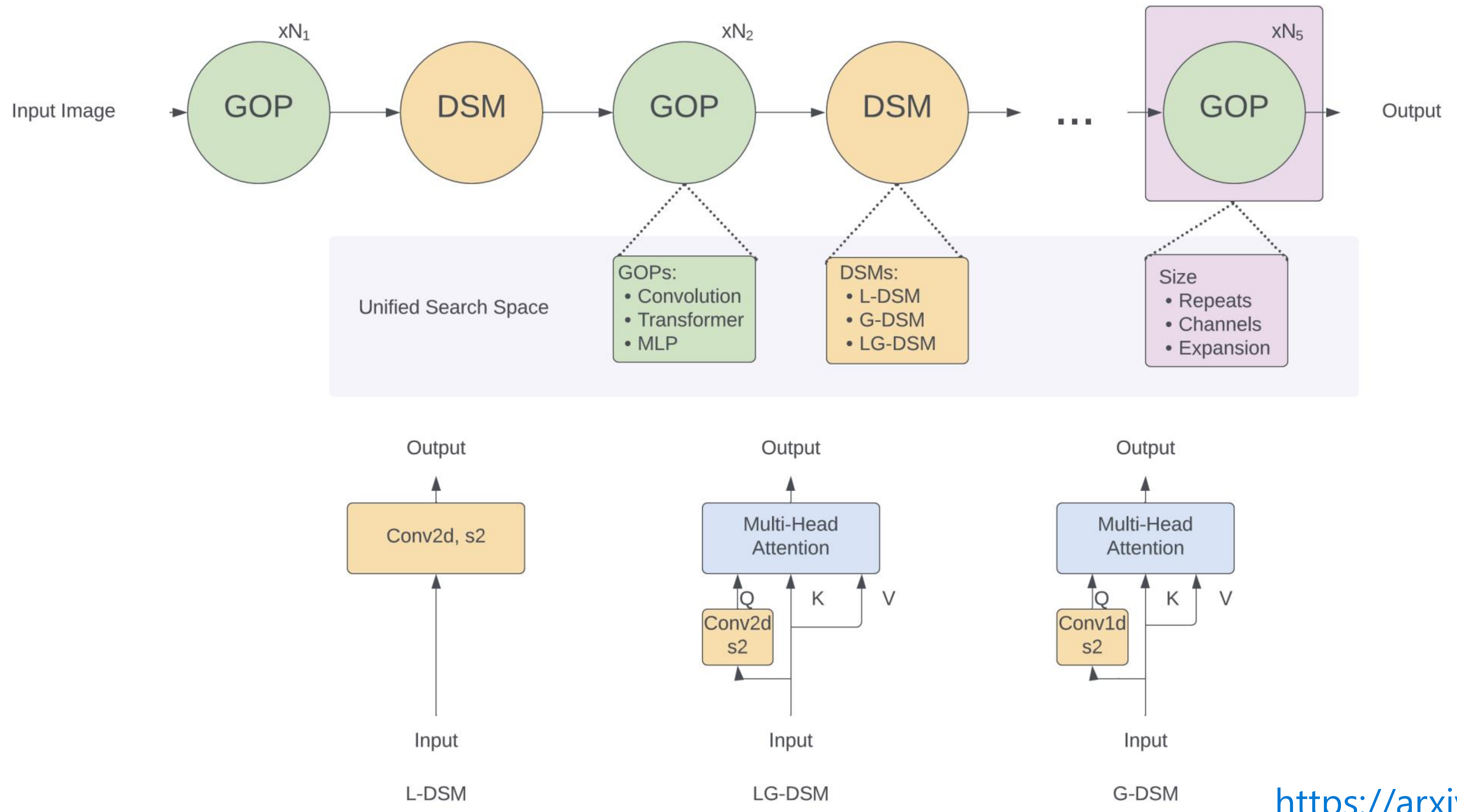
https://arxiv.org/abs/2107.00651

GLiT: Neural Architecture Search for Global and Local Image Transformer



https://arxiv.org/abs/2107.02960

UniNet: Unified Architecture Search with Convolution, Transformer, and MLP

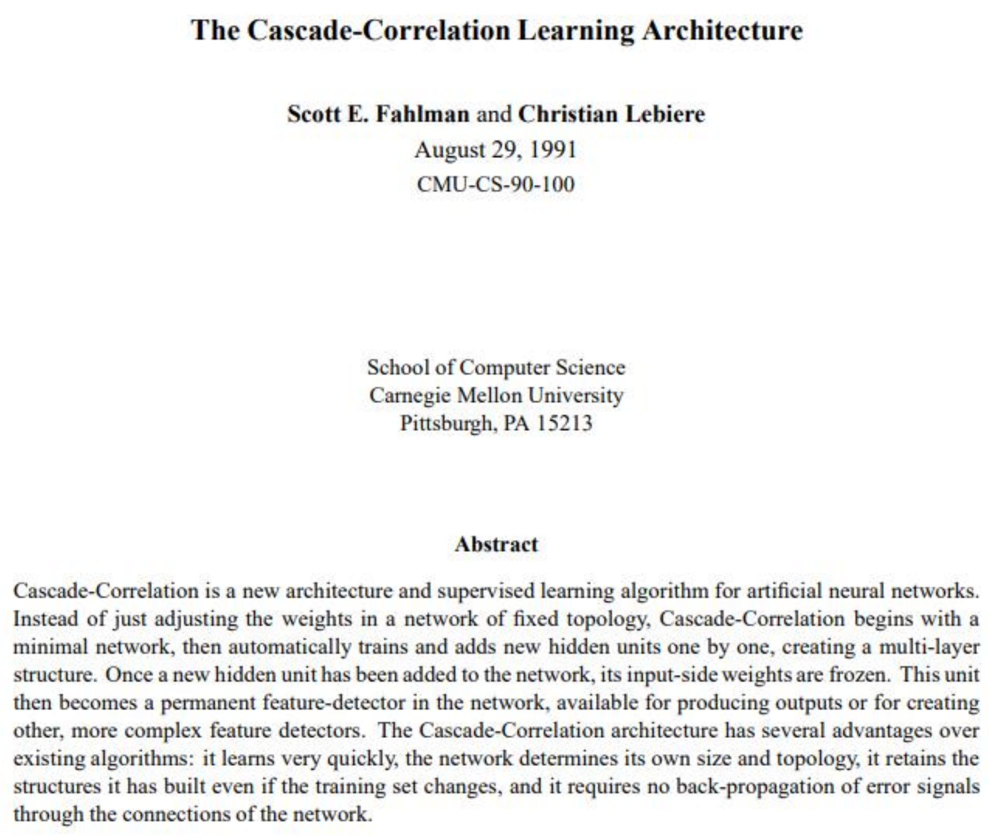


https://arxiv.org/abs/2110.04035

Petridish: Efficient Forward Architecture Search
Hu et al, NeuRIPS 2019

Petridish overview

- Warm start
  - Inspired by gradient boosting.
- Expand the search tree
  - Focus on the most cost-effective ones.
  - Directly search the pareto-frontier.
- Predict performance.
  - Utilizing model initialization to select children to train.



The Cascade-Correlation Learning Architecture

Scott E. Fahlman and **Christian Lebiere**

August 29, 1991

CMU-CS-90-100

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Cascade-Correlation is a new architecture and supervised learning algorithm for artificial neural networks. Instead of just adjusting the weights in a network of fixed topology, Cascade-Correlation begins with a minimal network, then automatically trains and adds new hidden units one by one, creating a multi-layer structure. Once a new hidden unit has been added to the network, its input-side weights are frozen. This unit then becomes a permanent feature-detector in the network, available for producing outputs or for creating other, more complex feature detectors. The Cascade-Correlation architecture has several advantages over existing algorithms: it learns very quickly, the network determines its own size and topology, it retains the structures it has built even if the training set changes, and it requires no back-propagation of error signals through the connections of the network.

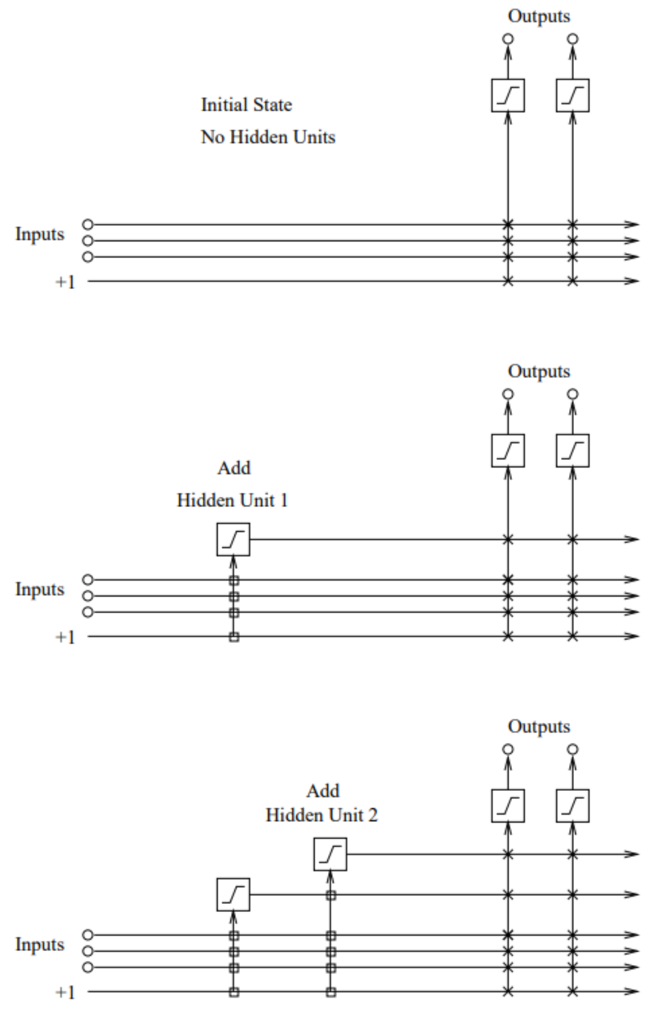
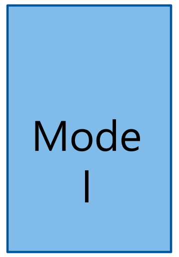
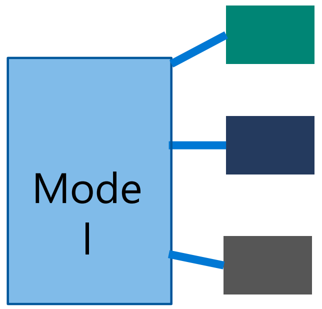


Figure 1: The Cascade architecture, initial state and after adding two hidden units. The vertical lines sum all incoming activation. Boxed connections are frozen, X connections are trained repeatedly.

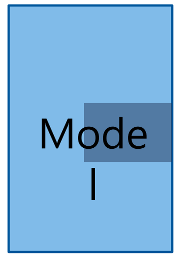
Incremental Training



Phase 0
Original model

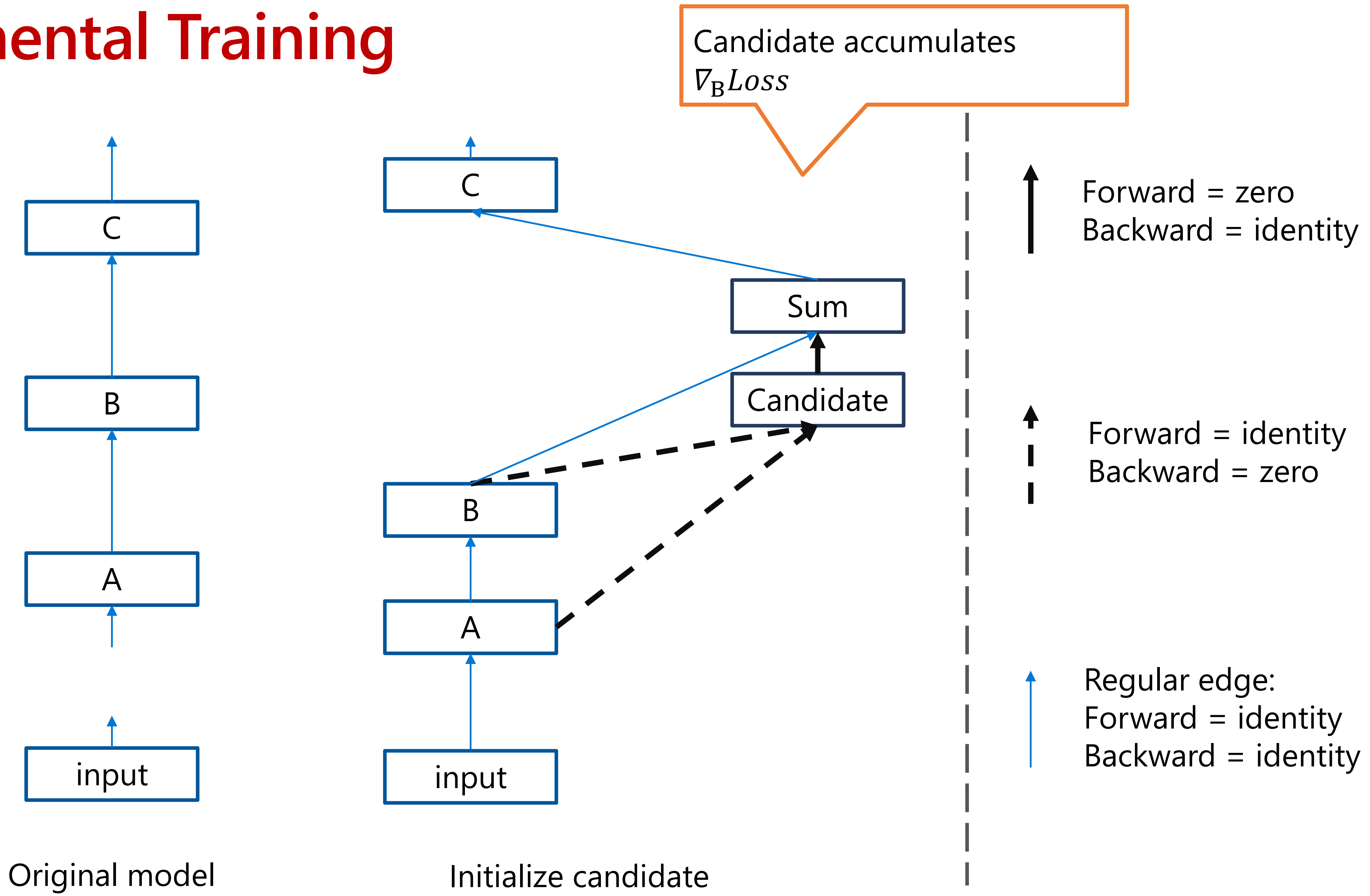


Phase 1
Initialize candidates, but do not allow candidates to affect the original model.

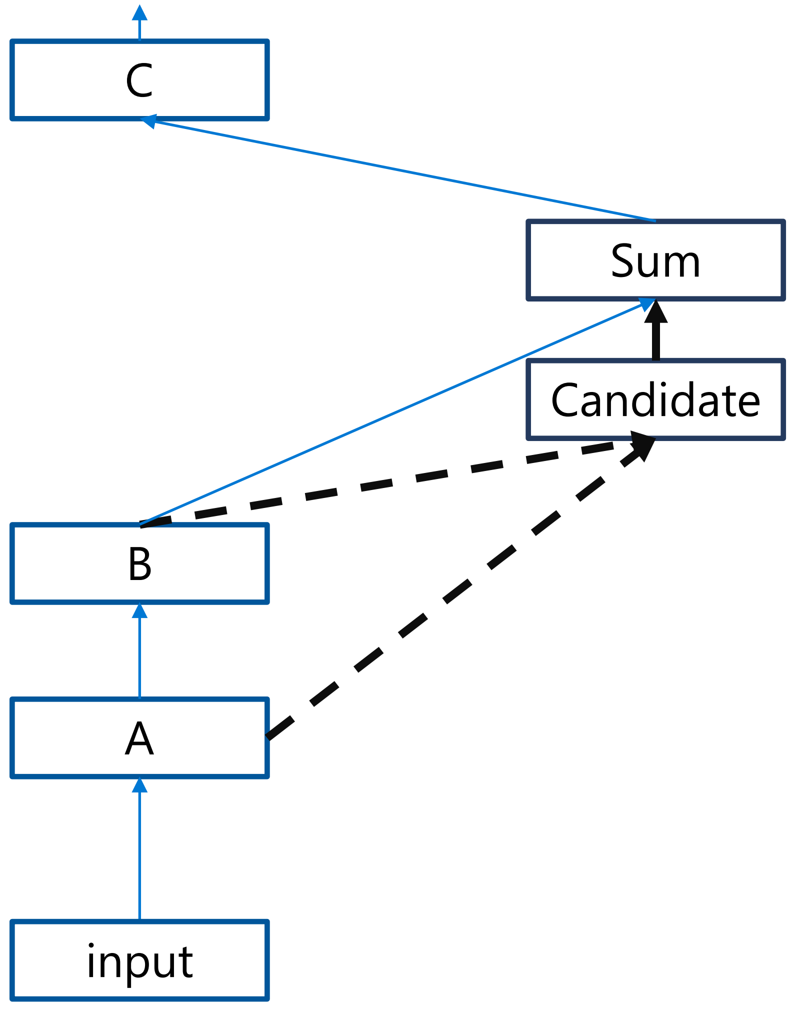


Phase 2
Officially add an candidate to model. Now the candidate can affect the original.

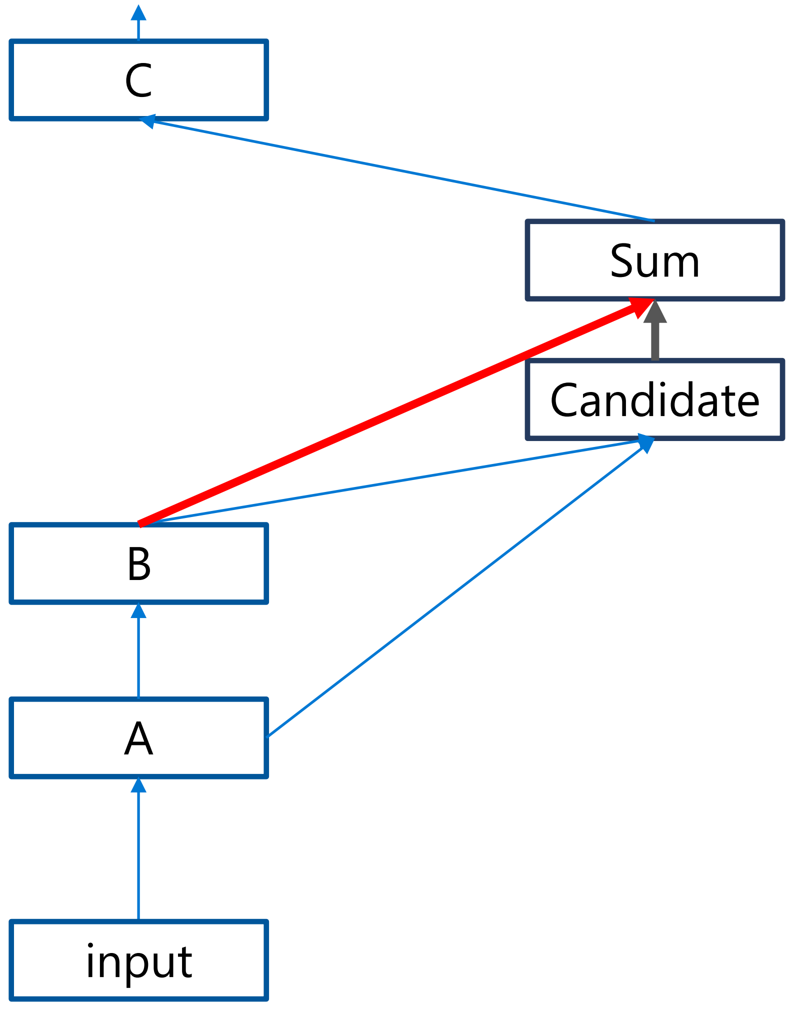
Incremental Training



Incremental Training



Initialize candidate

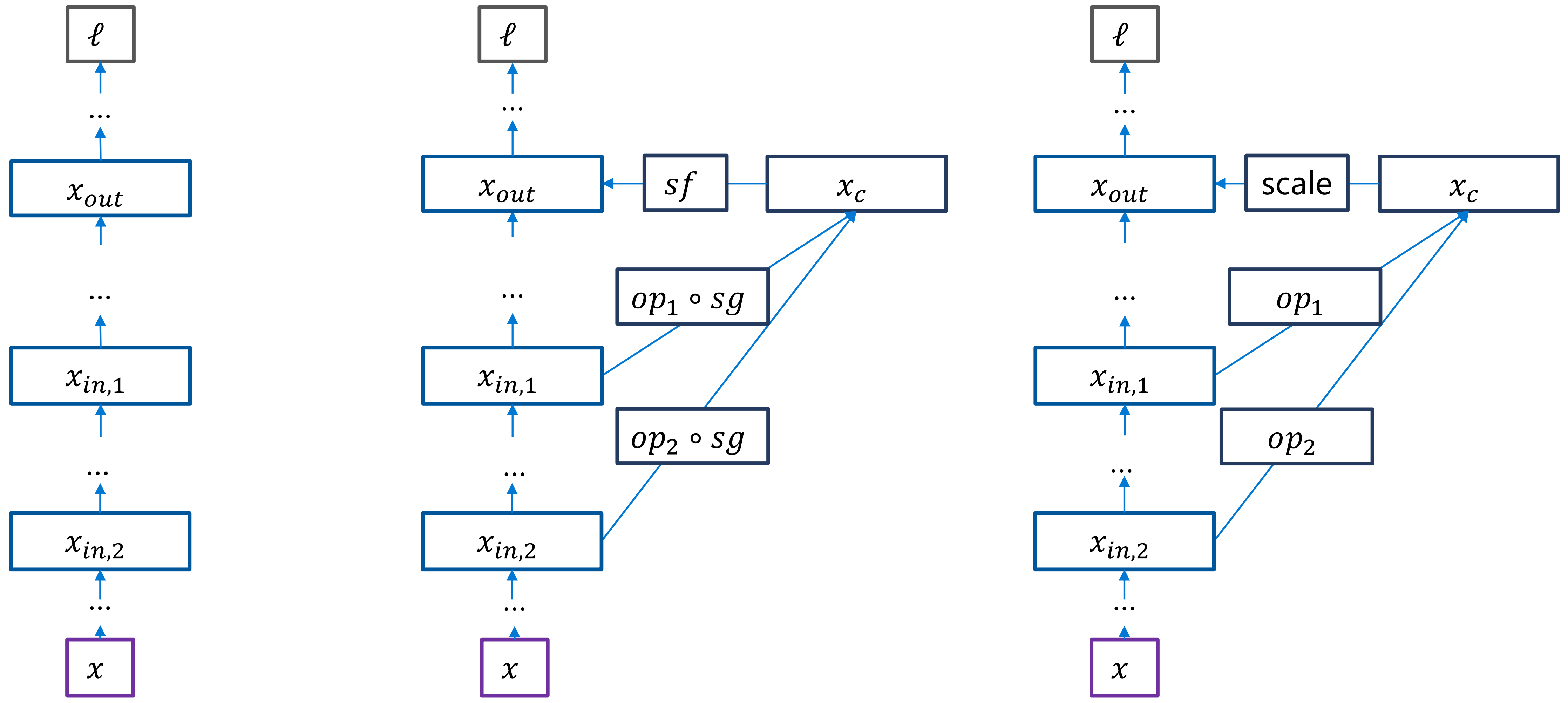


Officially add candidate to model

Scale the input.
Initial scale = 1

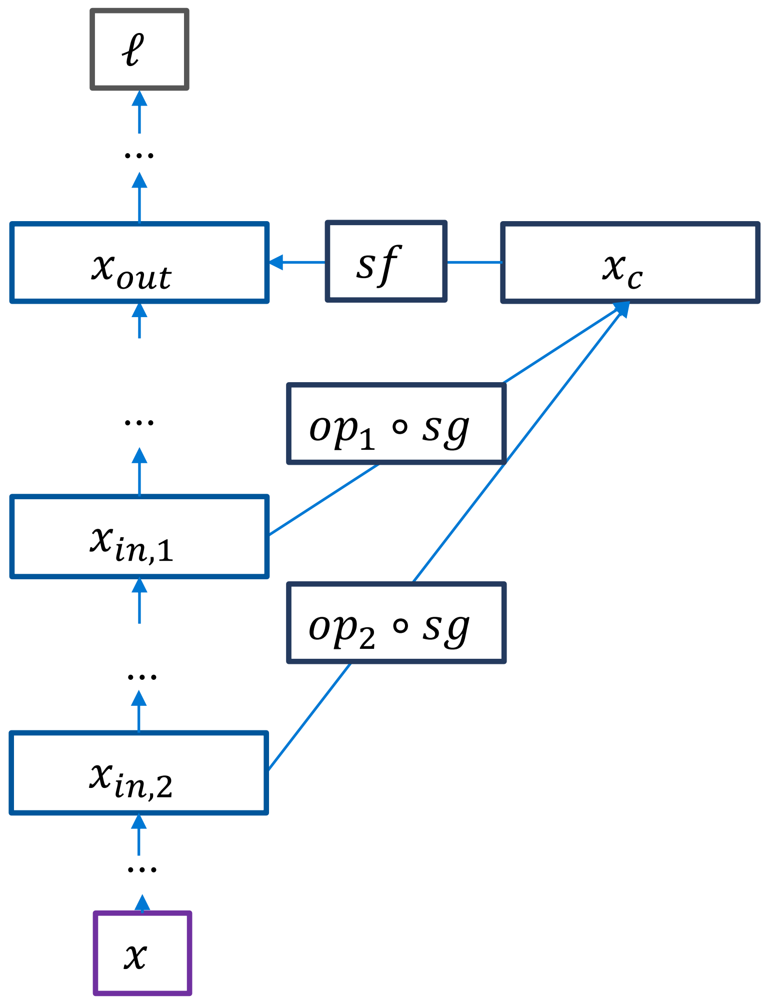
Scale the input.
Initial scale = 0

Incremental Training (Summary)

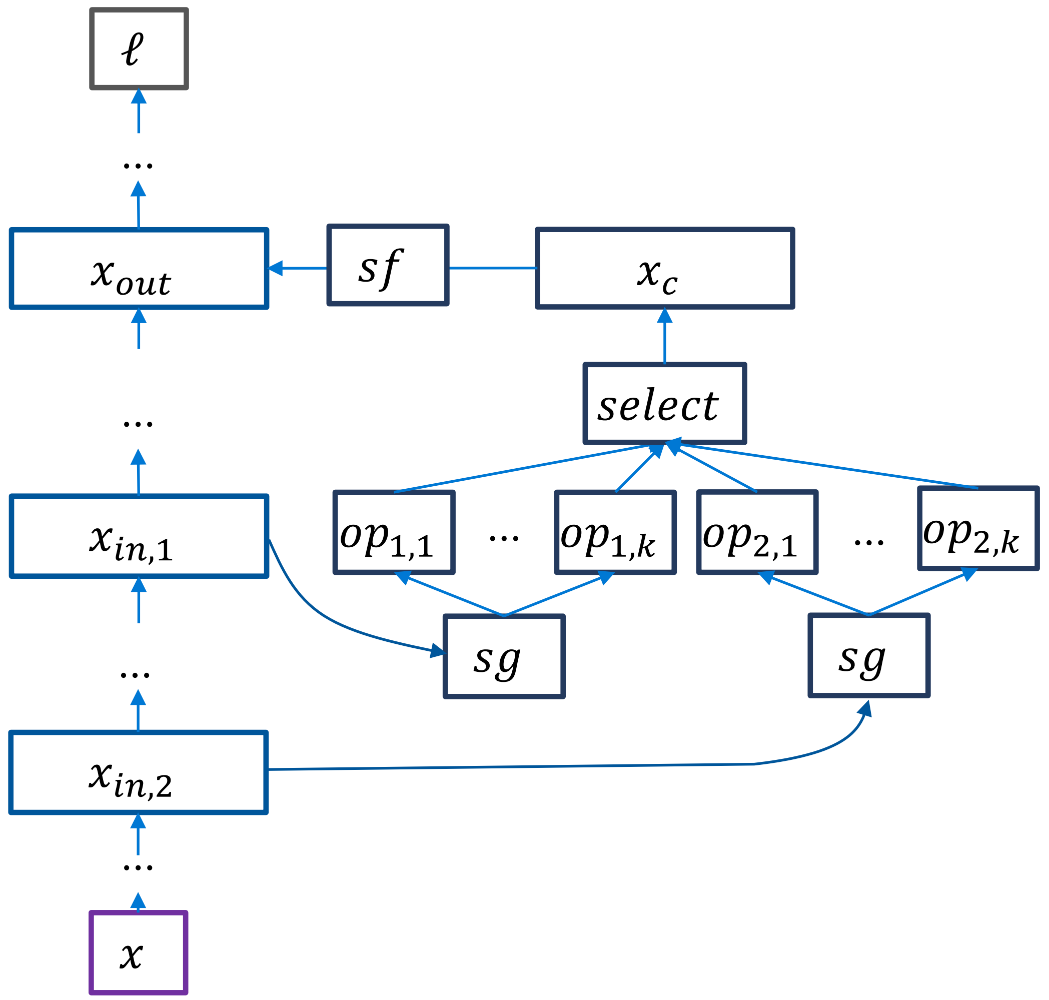


(a) (b) (c)

Incremental Training (Choice of Candidates)

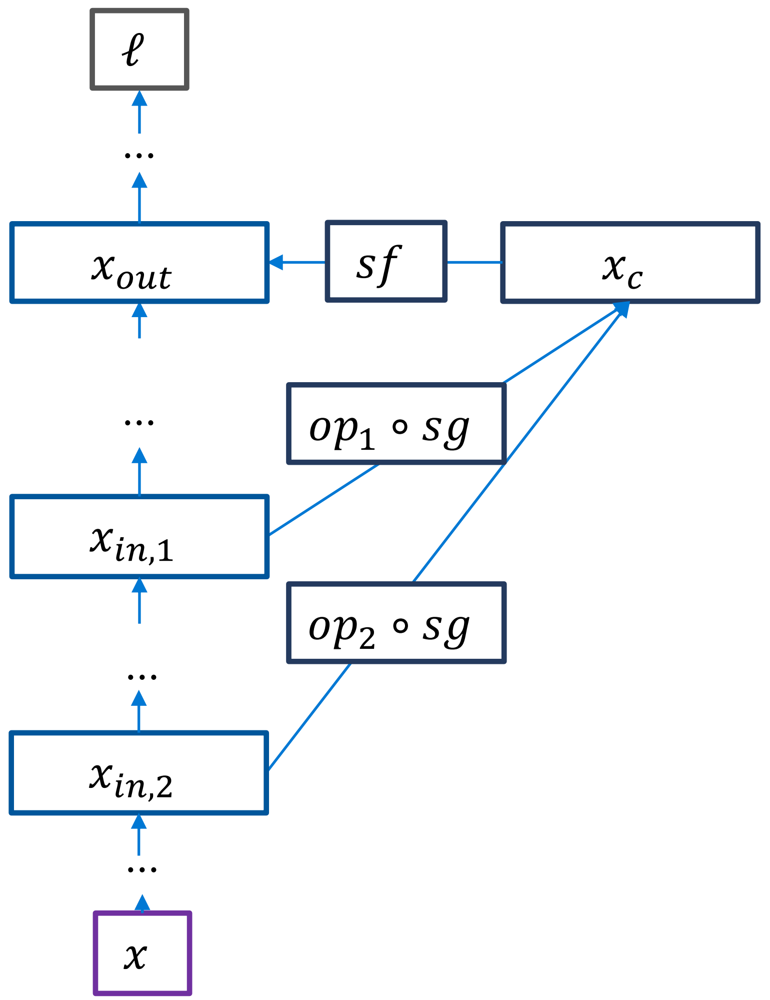


(b)

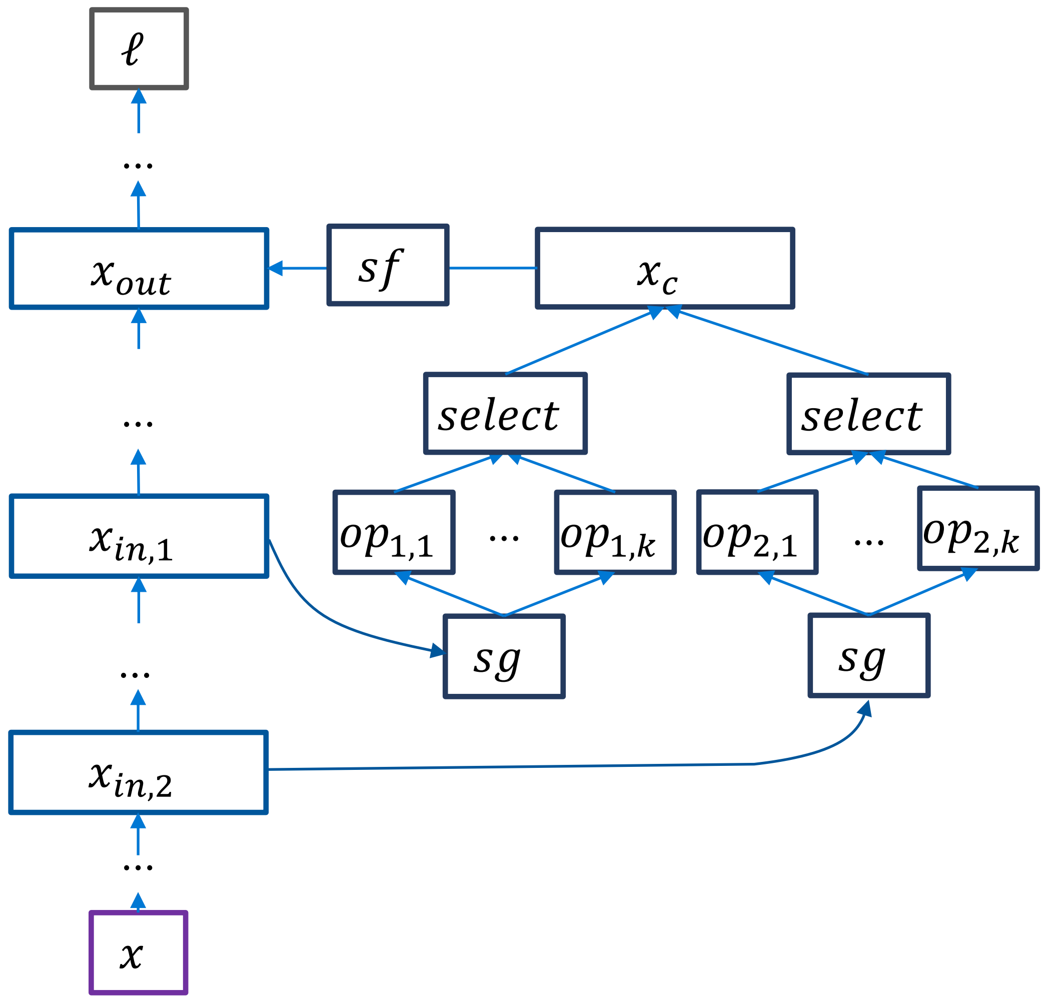


(b′)

Incremental Training (Choice of Candidates)



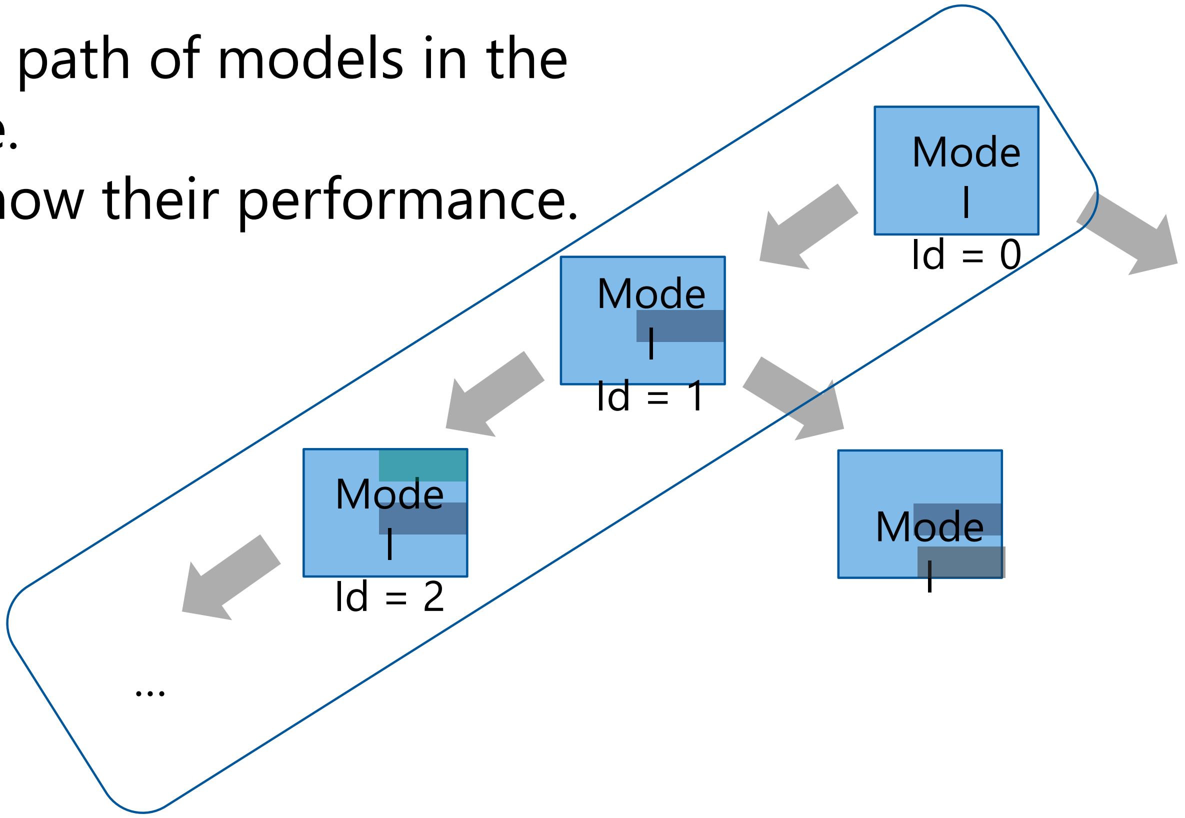
(b)



(b′)

Incremental training during search

Consider a path of models in the search tree.
Want to know their performance.



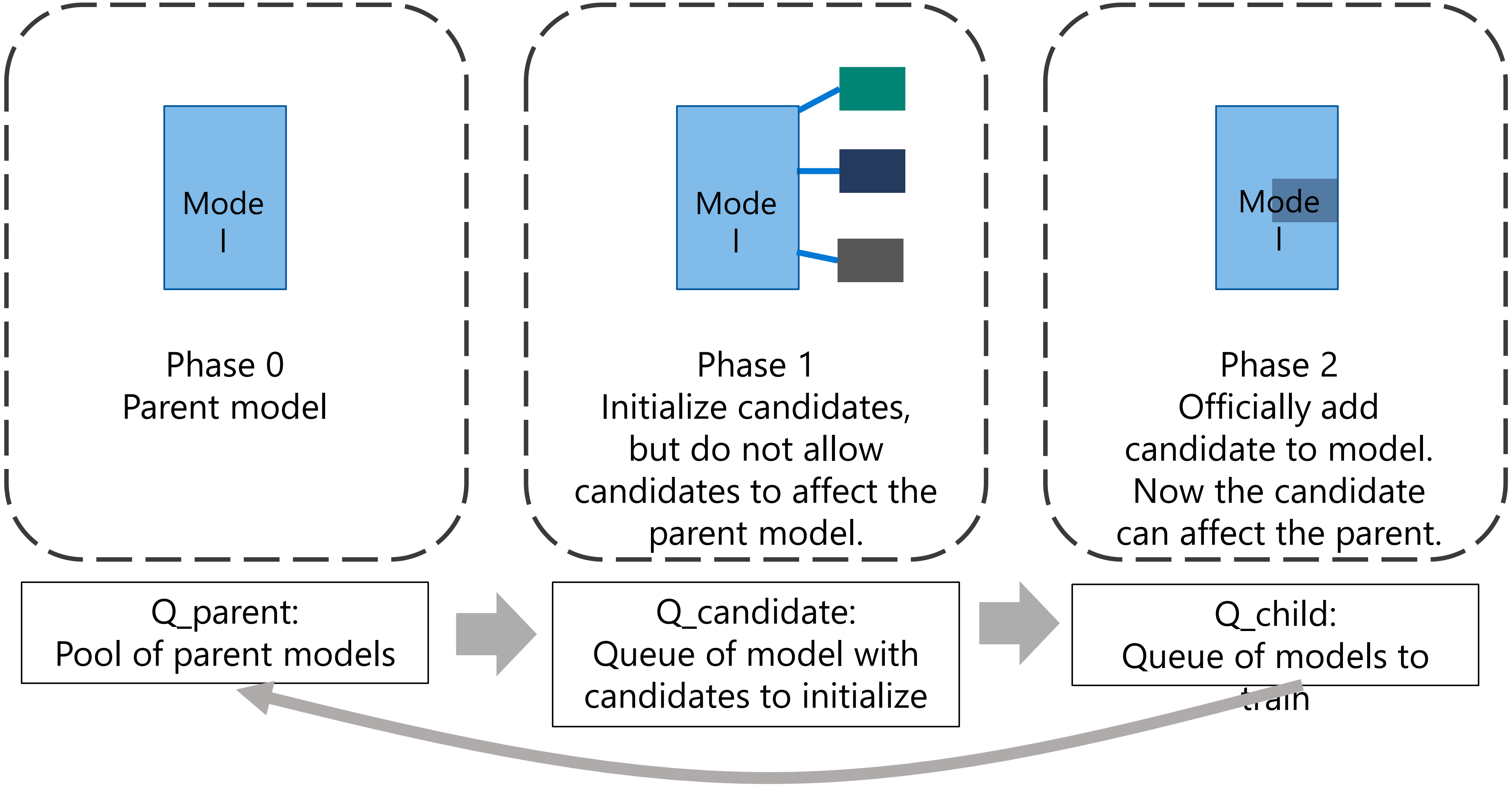
Option 1 (From-scratch) :

- Train models independently.
- 300 epochs per model

Option 2 (Incremental) :

- Start from parent; initialize children
- 40 epochs per model

Search on distributed systems

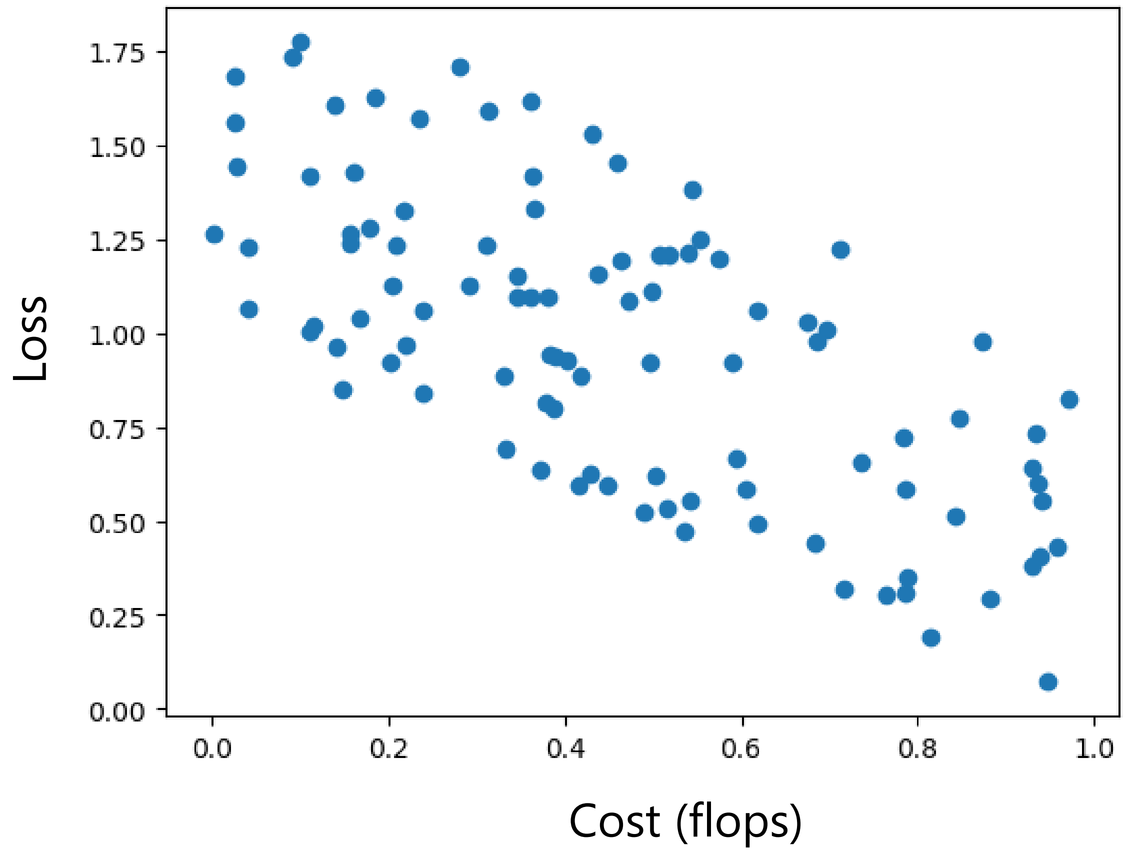

Mode
l
Phase 0
Parent model
Mode
l
Phase 1
Initialize candidates, but do not allow candidates to affect the parent model.
Mode
l
Phase 2
Officially add candidate to model. Now the candidate can affect the parent.
Q_parent:
Pool of parent models
Q_candidate:
Queue of model with candidates to initialize
Q_child:
Queue of models to train

Search on distributed systems

- Q_parent: explore-exploit a diverse set of good models to extend.
- Q_candidate: initialize promising candidates
- Q_children: train promising children

- How do we know a model is good?

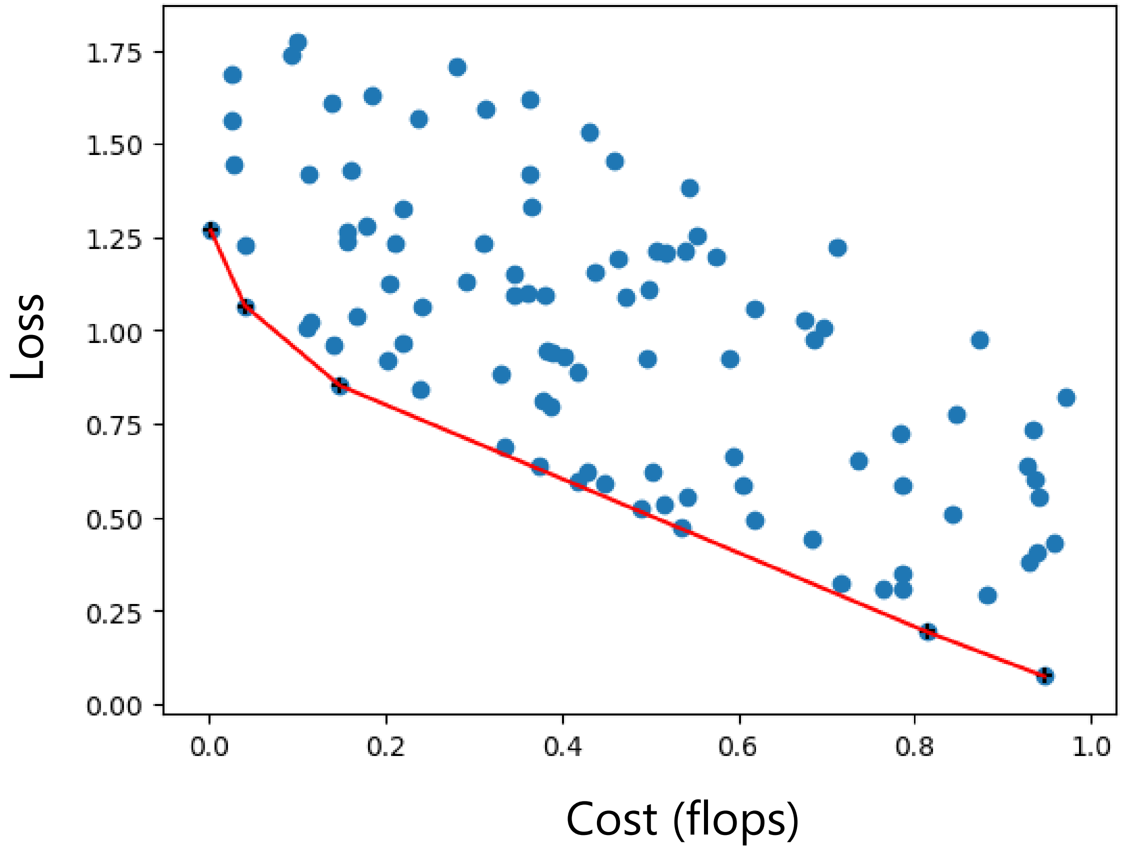
Expanding the Most Cost-efficient Models



This figure is for illustration only

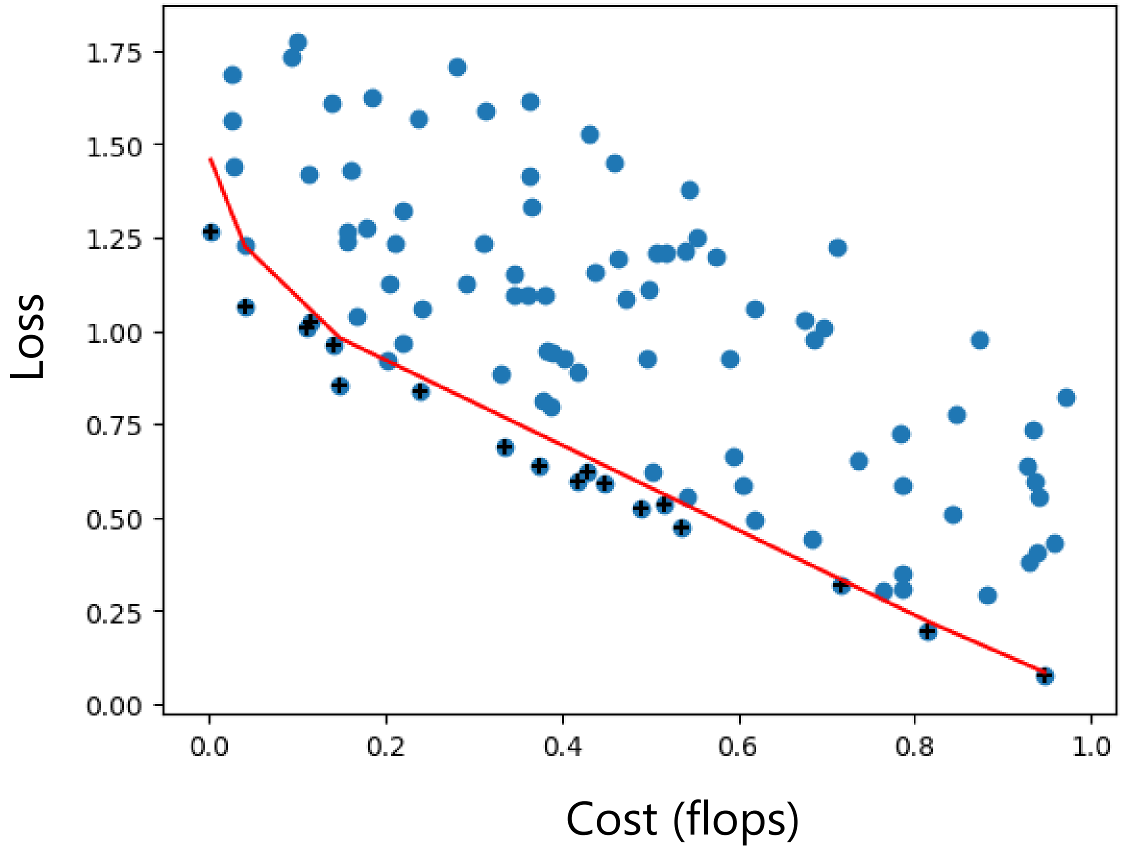
Expanding the Most Cost-efficient Models

- Convex hull



Expanding the Most Cost-efficient Models

- Epsilon- convex hull



Key advantage:

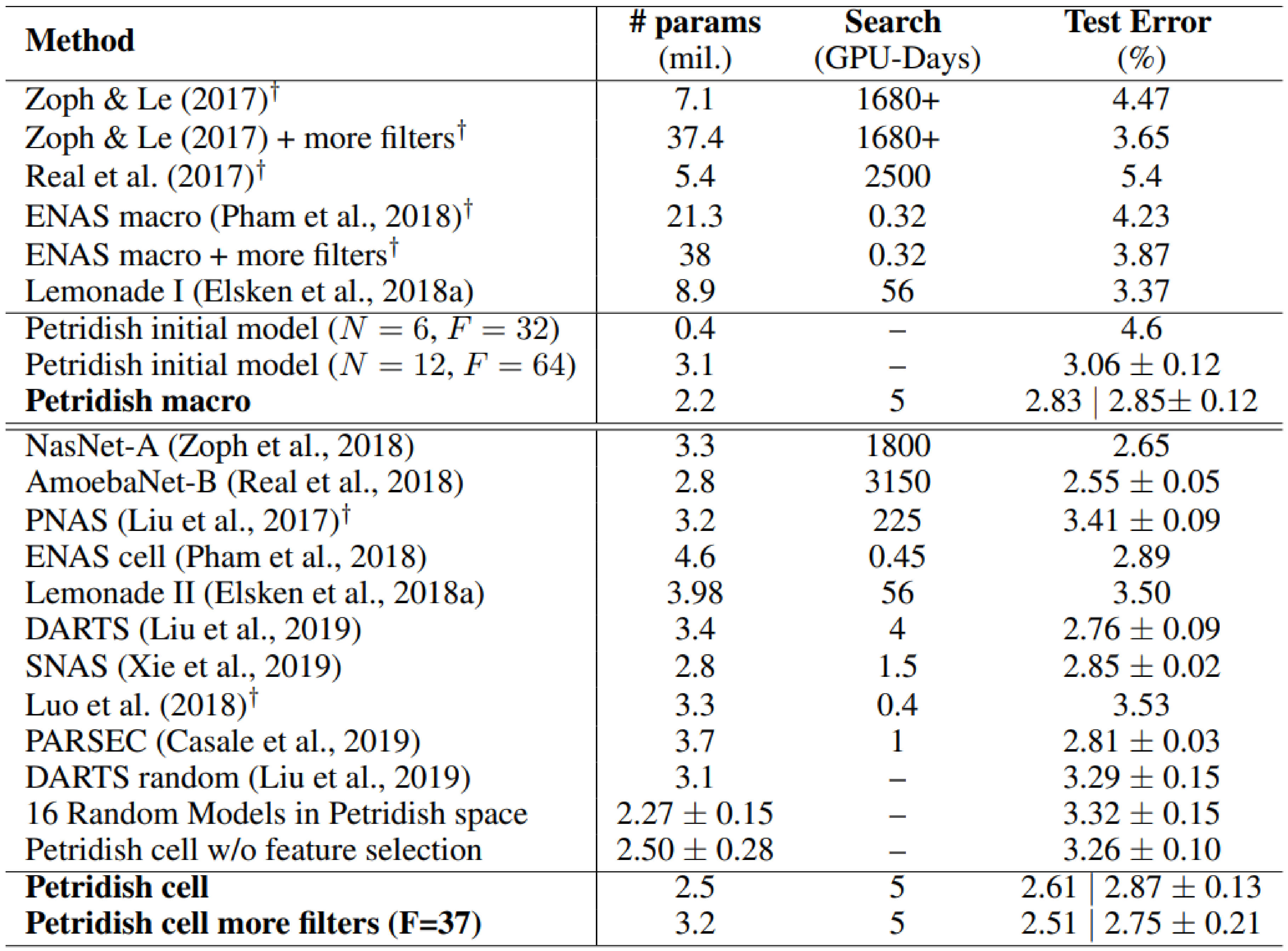
Method naturally produces a 'gallery' of models which are nearly-optimal for every serving time budget need.

This is critical to production serving needs.

Search on CIFAR10

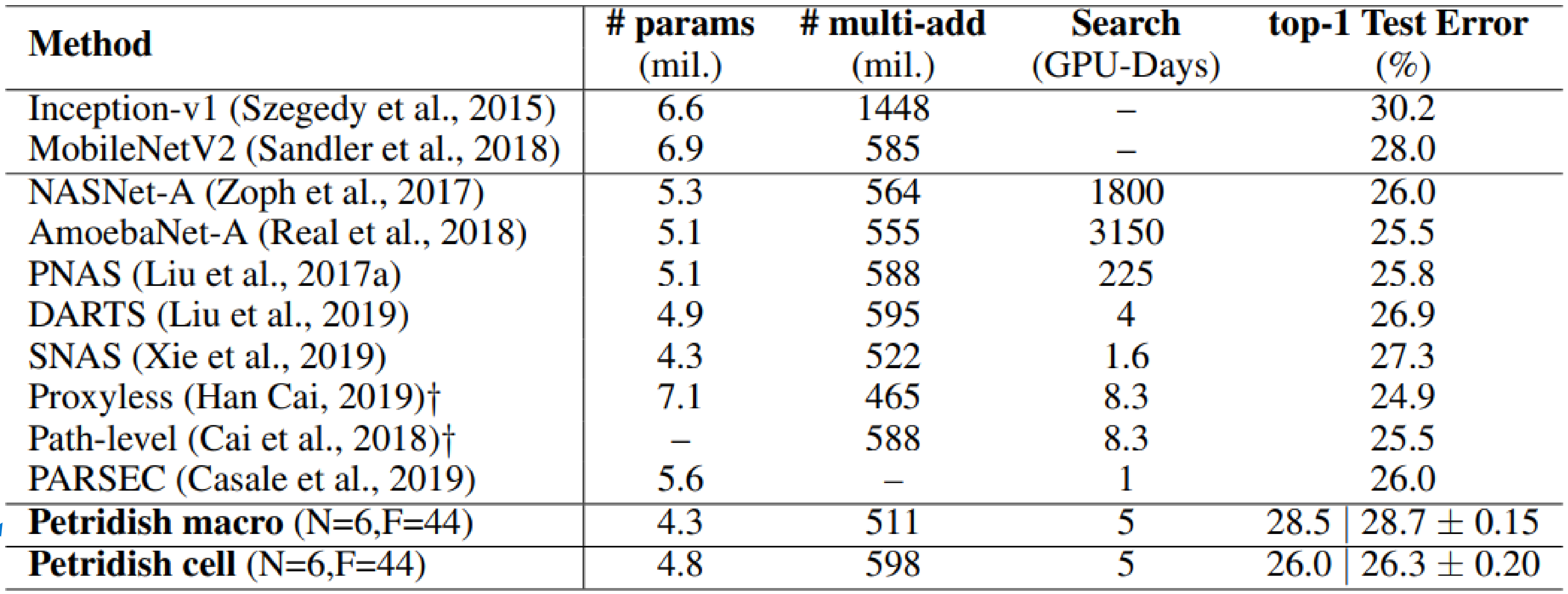
Petridish on macro search space

Petridish on cell search space



Method	# params (mil.)	Search (GPU-Days)	Test Error (%)
Zoph & Le (2017)†	7.1	1680+	4.47
Zoph & Le (2017) + more filters†	37.4	1680+	3.65
Real et al. (2017)†	5.4	2500	5.4
ENAS macro (Pham et al., 2018)†	21.3	0.32	4.23
ENAS macro + more filters†	38	0.32	3.87
Lemonade I (Elsken et al., 2018a)	8.9	56	3.37
Petridish initial model ($N = 6$, $F = 32$)	0.4	–	4.6
Petridish initial model ($N = 12$, $F = 64$)	3.1	–	3.06 ± 0.12
Petridish macro	2.2	5	2.83 | 2.85 ± 0.12
NasNet-A (Zoph et al., 2018)	3.3	1800	2.65
AmoebaNet-B (Real et al., 2018)	2.8	3150	2.55 ± 0.05
PNAS (Liu et al., 2017)†	3.2	225	3.41 ± 0.09
ENAS cell (Pham et al., 2018)	4.6	0.45	2.89
Lemonade II (Elsken et al., 2018a)	3.98	56	3.50
DARTS (Liu et al., 2019)	3.4	4	2.76 ± 0.09
SNAS (Xie et al., 2019)	2.8	1.5	2.85 ± 0.02
Luo et al. (2018)†	3.3	0.4	3.53
PARSEC (Casale et al., 2019)	3.7	1	2.81 ± 0.03
DARTS random (Liu et al., 2019)	3.1	–	3.29 ± 0.15
16 Random Models in Petridish space	2.27 ± 0.15	–	3.32 ± 0.15
Petridish cell w/o feature selection	2.50 ± 0.28	–	3.26 ± 0.10
Petridish cell	2.5	5	2.61 | 2.87 ± 0.13
Petridish cell more filters (F=37)	3.2	5	2.51 | 2.75 ± 0.21

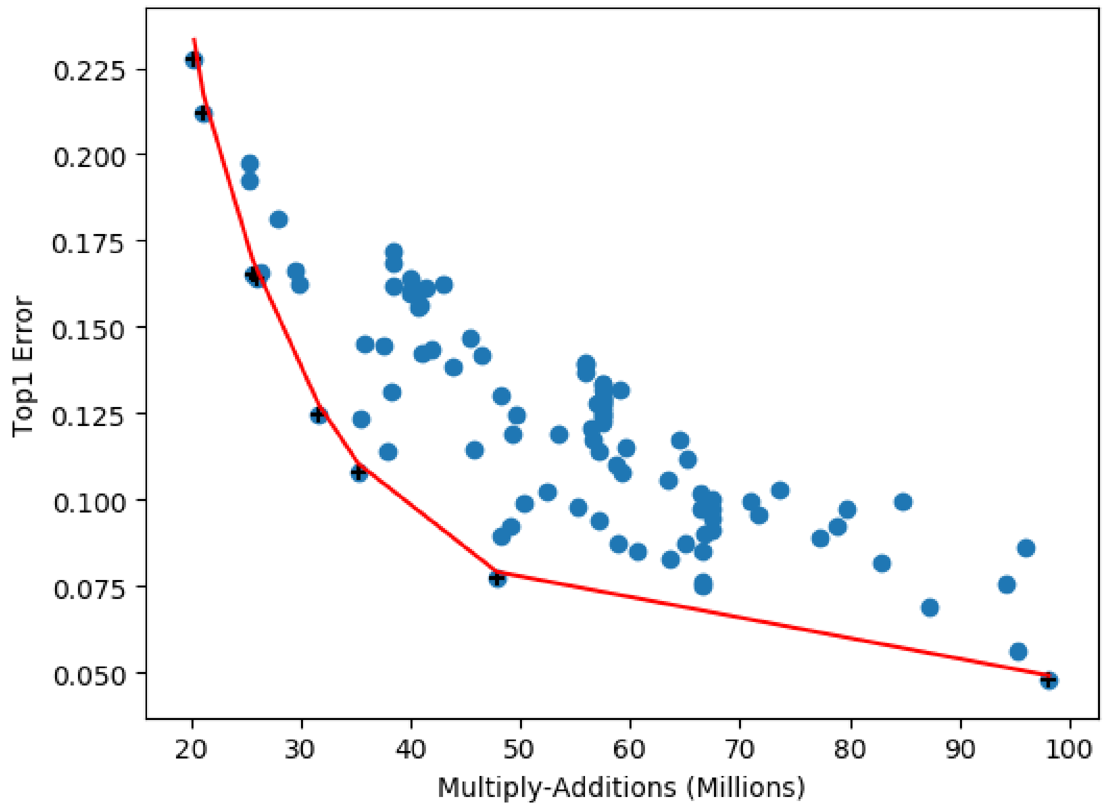
Transfer to ImageNet

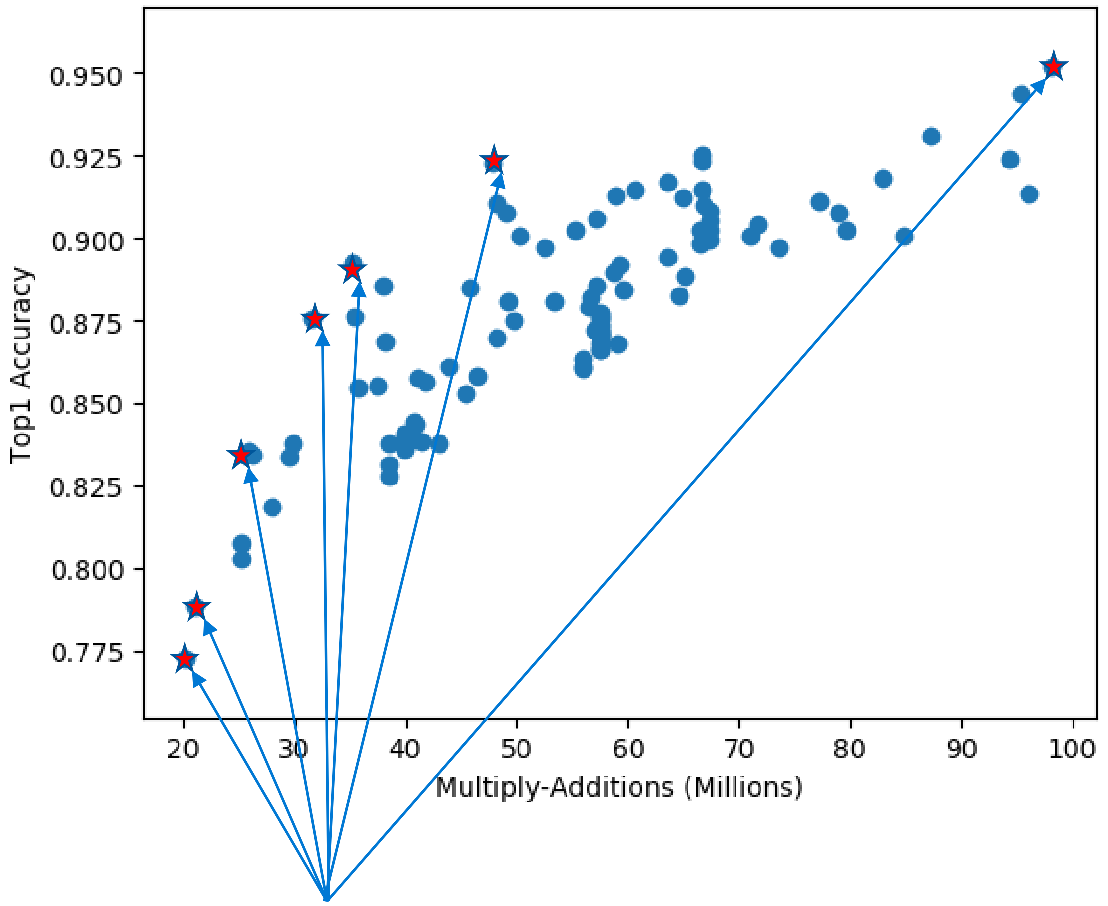


Method	**# params** (mil.)	**# multi-add** (mil.)	**Search** (GPU-Days)	**top-1 Test Error** (%)
Inception-v1 (Szegedy et al., 2015)	6.6	1448	–	30.2
MobileNetV2 (Sandler et al., 2018)	6.9	585	–	28.0
NASNet-A (Zoph et al., 2017)	5.3	564	1800	26.0
AmoebaNet-A (Real et al., 2018)	5.1	555	3150	25.5
PNAS (Liu et al., 2017a)	5.1	588	225	25.8
DARTS (Liu et al., 2019)	4.9	595	4	26.9
SNAS (Xie et al., 2019)	4.3	522	1.6	27.3
Proxyless (Han Cai, 2019)†	7.1	465	8.3	24.9
Path-level (Cai et al., 2018)†	–	588	8.3	25.5
PARSEC (Casale et al., 2019)	5.6	–	1	26.0
Petridish macro (N=6,F=44)	4.3	511	5	28.5 | 28.7 ± 0.15
Petridish cell (N=6,F=44)	4.8	598	5	26.0 | 26.3 ± 0.20

No domain-knowledge injection in architecture design at all!

Search Once, Deploy Everywhere!





Train models on the frontier with every orthogonal trick!
(data augmentation, distillation….)

Example search on
CIFAR10

Reproducibility, Fair Comparison and Best Practices!

Difficult to compare approaches!

- Search spaces and datasets.
- Training routine used.
  - Does it have all the tips and tricks?
- Hardware-Software used.
  - TPU vs. GPU vs. driver version vs. cuda version vs. framework.
- Stochasticity in training on GPUs.

NAS Evaluation is Frustratingly Hard, Yang et al., ICLR 2020
Random Search and Reproducibility for Neural Architecture Search, Li and Talwalker, UAI 2020

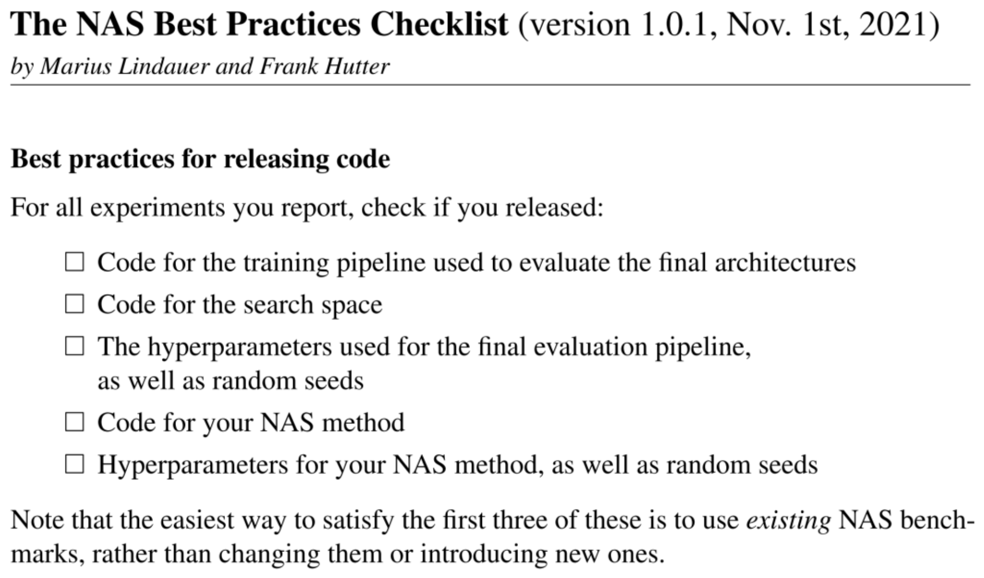
Benchmarks Help!

- NASBench-101
  - Cannot evaluate weight-sharing, DARTS-like search spaces.
- NASBench-201
  - Uses different search space than 101.
- NASBench-1Shot1
  - Leverages 101 to make it amenable for weight-sharing.
- NASBench-301
  - 60,000 models sampled from DARTS search space trained on CIFAR10 to train surrogate model.
- NASBench-NLP
  - 14k RNN architectures trained on Penn Tree Bank.
- NASBench-ASR
  - 8k architectures trained on TIMIT audio dataset for speech recognition.

Benchmarks Help!

- NAS-HPO-Bench-II
  - 4K cell-based CNNs with different learning rates, batch sizes.
- HW-NAS-Bench
  - Evaluate NAS-Bench-201 and FBNet search spaces on 6 devices (edge devices, FPGA, ASIC).
- On Network Design Spaces for Visual Recognition
  - Over 100k architectures evaluated on CIFAR-10 from different search spaces.
- NAS-Bench-Suite
  - Collection of NAS Benchmarks through unified interface.
- NAS-Bench-360
  - 10 diverse tasks which are not just traditional vision tasks.

Checklist *Before* Starting Project



The NAS Best Practices Checklist (version 1.0.1, Nov. 1st, 2021)
by Marius Lindauer and Frank Hutter

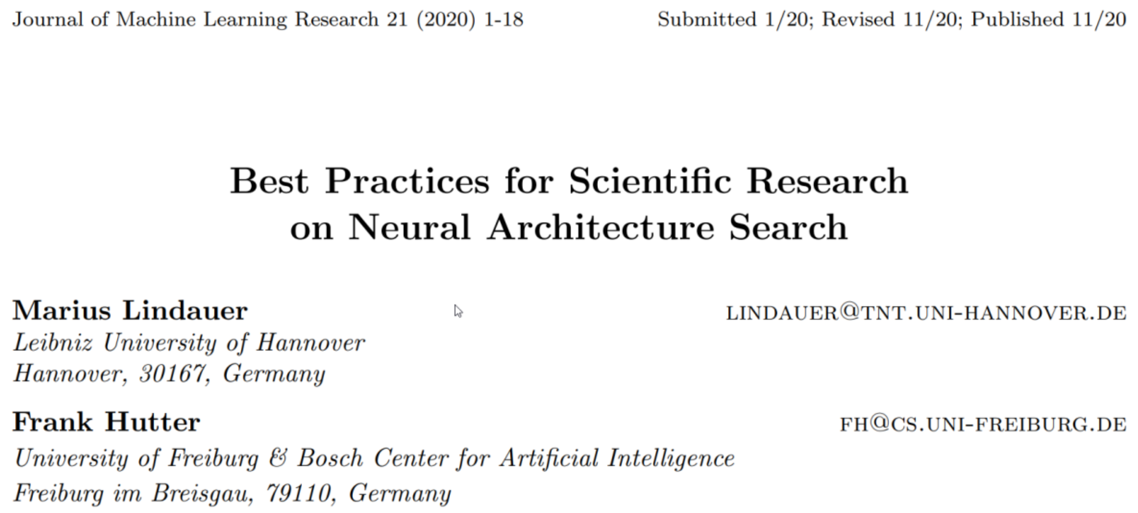
Best practices for releasing code

For all experiments you report, check if you released:

- ☐ Code for the training pipeline used to evaluate the final architectures
- ☐ Code for the search space
- ☐ The hyperparameters used for the final evaluation pipeline, as well as random seeds
- ☐ Code for your NAS method
- ☐ Hyperparameters for your NAS method, as well as random seeds

Note that the easiest way to satisfy the first three of these is to use *existing* NAS benchmarks, rather than changing them or introducing new ones.

https://www.automl.org/nas_checklist.pdf



Journal of Machine Learning Research 21 (2020) 1-18 Submitted 1/20; Revised 11/20; Published 11/20

Best Practices for Scientific Research on Neural Architecture Search

Marius Lindauer LINDAUER@TNT.UNI-HANNOVER.DE
Leibniz University of Hannover
Hannover, 30167, Germany

Frank Hutter FH@CS.UNI-FREIBURG.DE
University of Freiburg & Bosch Center for Artificial Intelligence
Freiburg im Breisgau, 79110, Germany

https://www.jmlr.org/papers/volume21/20-056/20-056.pdf

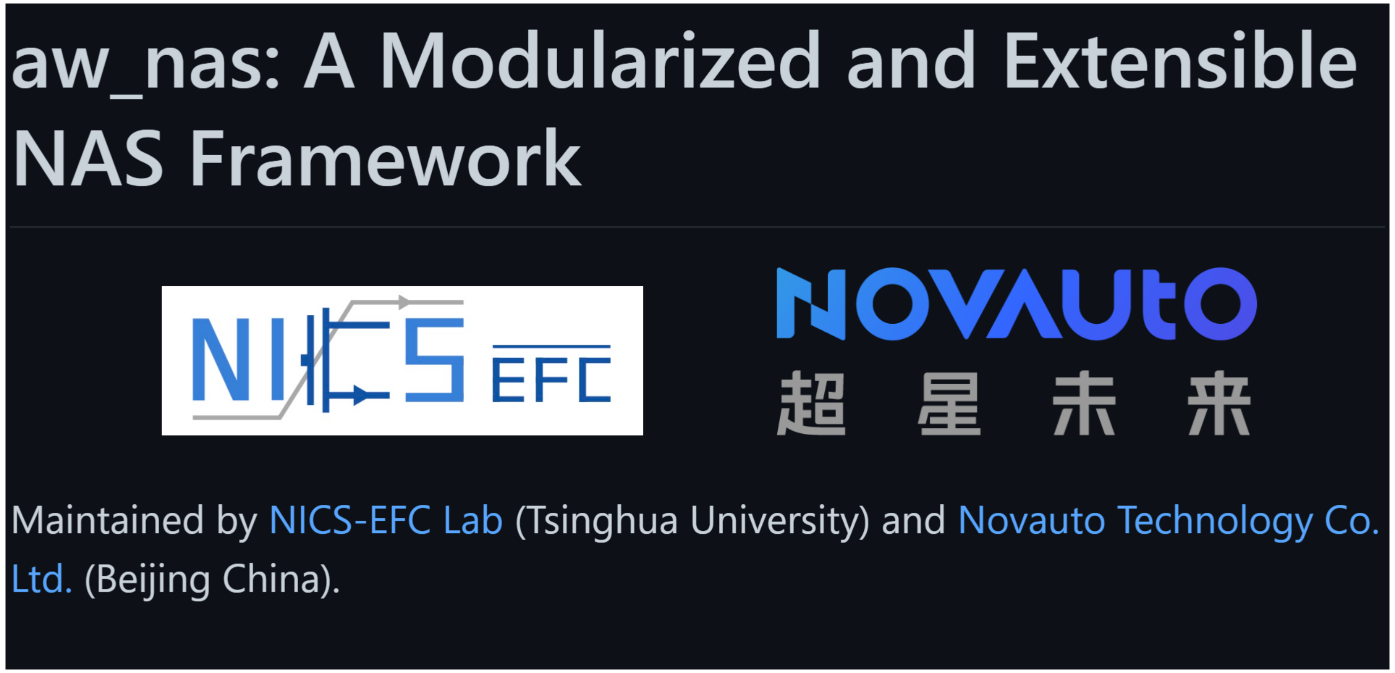
NAS Frameworks

https://github.com/automl/NASLib

Archai: Platform for Neural Architecture Search

https://github.com/microsoft/archai

NAS Frameworks



aw_nas: A Modularized and Extensible NAS Framework

Maintained by NICS-EFC Lab (Tsinghua University) and Novauto Technology Co. Ltd. (Beijing China).

PyGlove: Manipulating Python Programs

https://github.com/walkerning/aw_nas

https://github.com/google/pyglove

Microsoft

Is NAS solved?

No fully general solution yet but useful successes!

Still, lots of domain knowledge injection into the process.

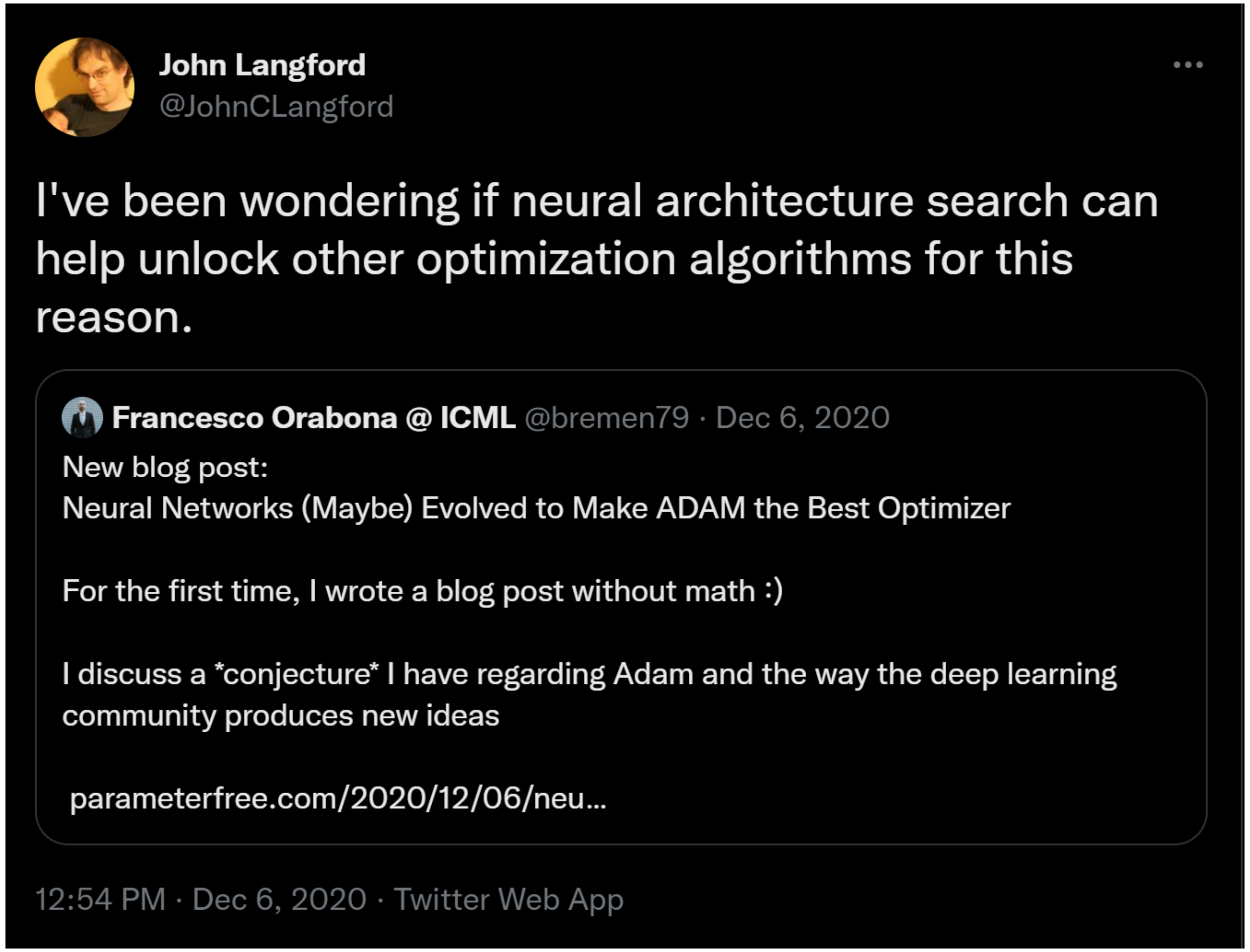
Tricks and tips needed for vision datasets are completely different from language or speech datasets (to be SOTA).

Need diverse benchmarks/tasks and rigorous reporting.

Hyperparameters are set to magic constants.

Open Problem 1: Optimizers and Learning Rate Schedules

- Are we handicapped by current optimizers?
- Rank of architectures in NAS-Bench-101 varies drastically on switching optimizer!
  - HW: Try this on other benchmarks!



John Langford
@JohnCLangford

I've been wondering if neural architecture search can help unlock other optimization algorithms for this reason.

Francesco Orabona @ ICML @bremen79 · Dec 6, 2020
New blog post:
Neural Networks (Maybe) Evolved to Make ADAM the Best Optimizer

For the first time, I wrote a blog post without math :)

I discuss a *conjecture* I have regarding Adam and the way the deep learning community produces new ideas

parameterfree.com/2020/12/06/neu...

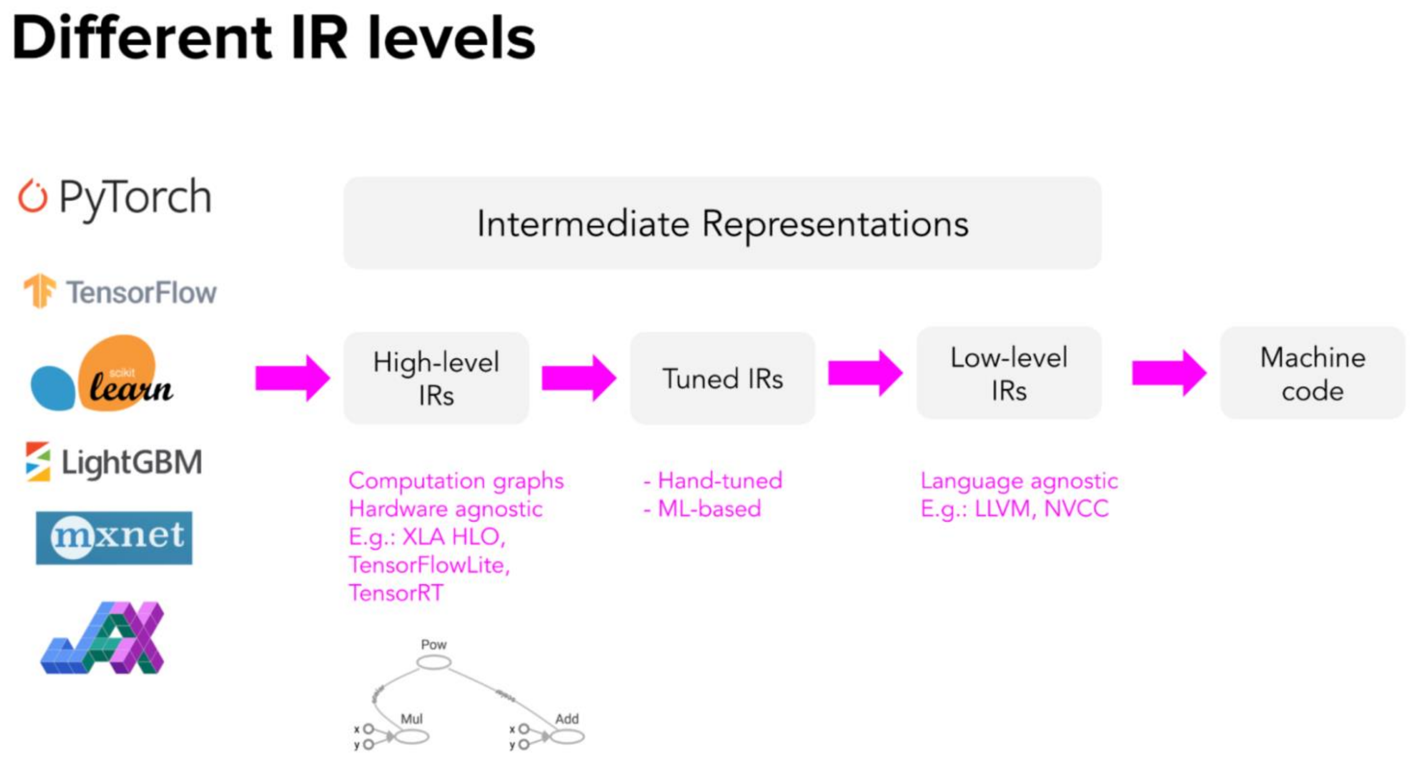
12:54 PM · Dec 6, 2020 · Twitter Web App

Tweet Dec 2020

Open Problem 2: Deep Learning Compilers!

"Pattern matching" for common manually designed architectures!

Compilers in the inner search loop will detect and optimize operator combinations that are commonly used!



Source: huyenchip

Questions?

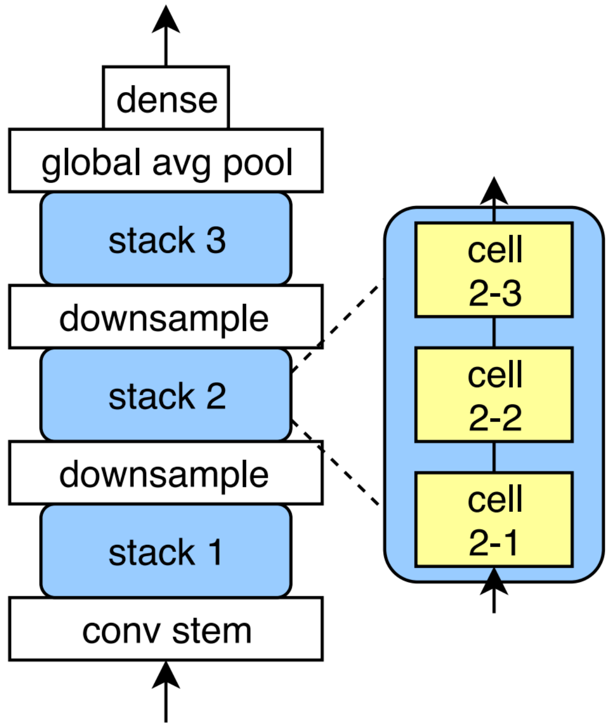
Appendix

Benchmarks to the Rescue: Tabular

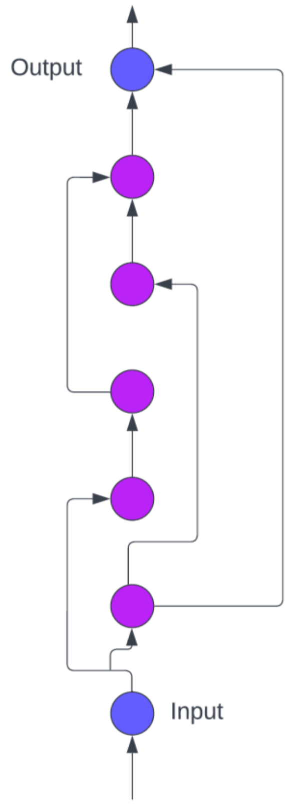
Arch ID	Training Error	Validation Error	Training Duration	Validation Duration	Test Error
0000001	0.15	0.18	632	10	0.21
0000002	0.33	0.41	515	8	0.46
0000003	0.28	0.22	585	11	0.23
…	…	…	…	…	…

Train *every* architecture in the search space.
Save all logs and data in a table.
NAS can be run on a laptop!

NAS-Bench-101



Skeleton

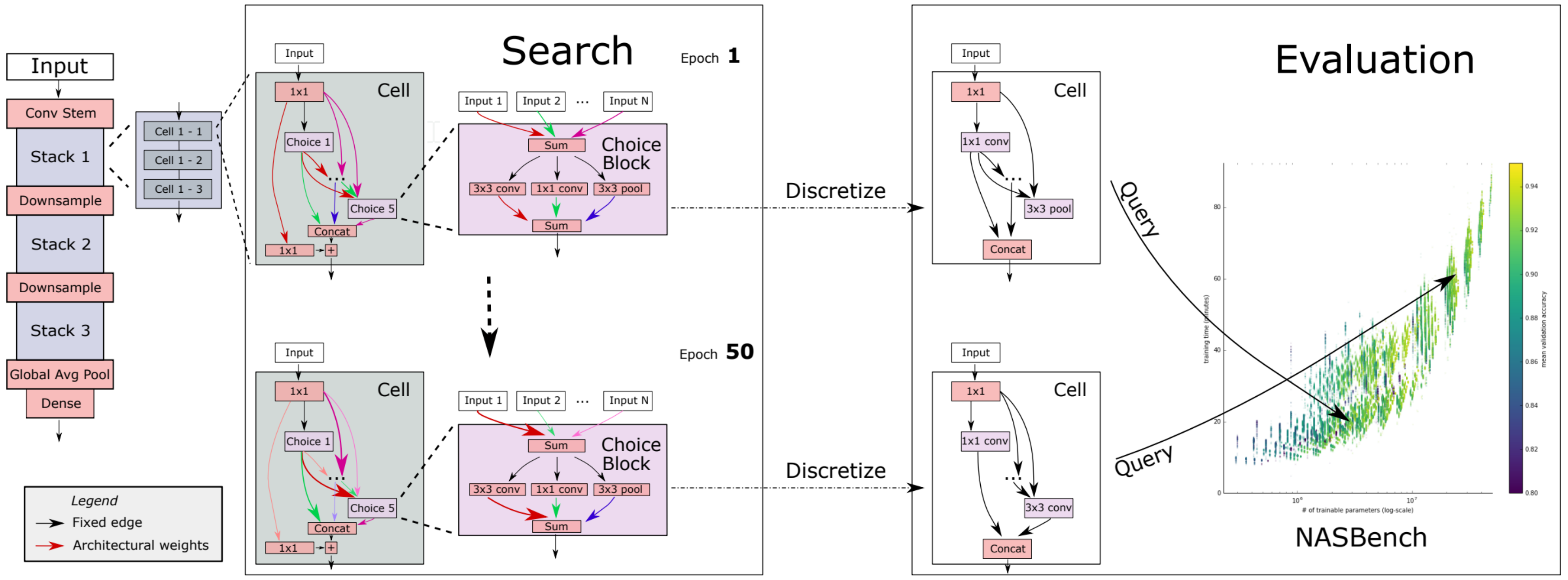


Example Cell

- Maximum 7 nodes per cell.
- Nodes are one of 3 operators.
  - 3x3 conv
  - 1x1 conv
  - 3x3 max pool
- Edges are tensors.
- Max edges 9 in a cell.
- 423k unique architectures.
- Trained on CIFAR10.
  - 4,12,36,108 epochs

NAS-Bench-101: Towards Reproducible Neural Architecture Search, Ying et al., ICML 2019

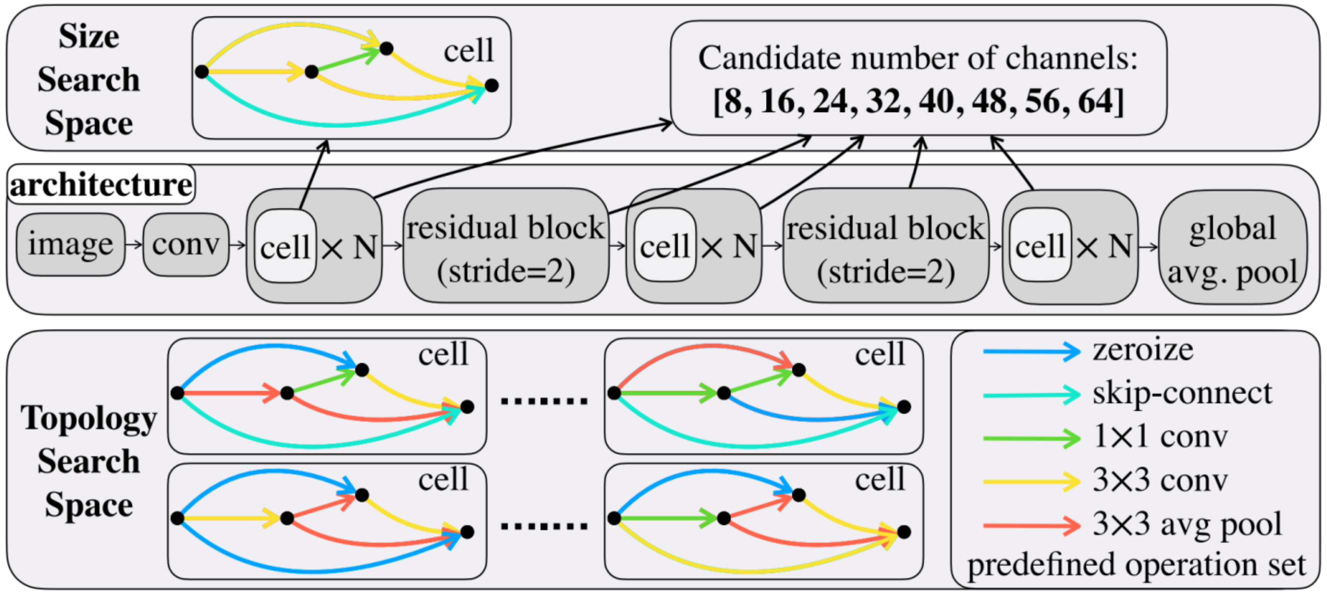
NAS-Bench-1Shot1



- Not possible to evaluate one-shot (weight-sharing) methods on NAS-Bench-101.
- Search space does not contain all possible cells (edges restricted <=9).
- NAS-Bench-1Shot1 defines a new search space.
  - Reuses 101 to allow for one-shot methods to be evaluated.

NAS-Bench-1Shot1: Benchmarking and Dissecting One-shot Neural Architecture Search,
Zela et al., ICLR 2020

NATS-Bench (NAS-Bench-201)



- One-shot compatible.
- Two search spaces:
  - Topological space: 6.5k unique
  - Size space: 32.8k unique
- 3 datasets:
  - CIFAR10
  - CIFAR100
  - ImageNet16-120
- Trained with 12, 20, 90 epochs.

NATS-Bench: Benchmarking NAS Algorithms for Architecture Topology and Size, Dong et al., TPAMI 2021