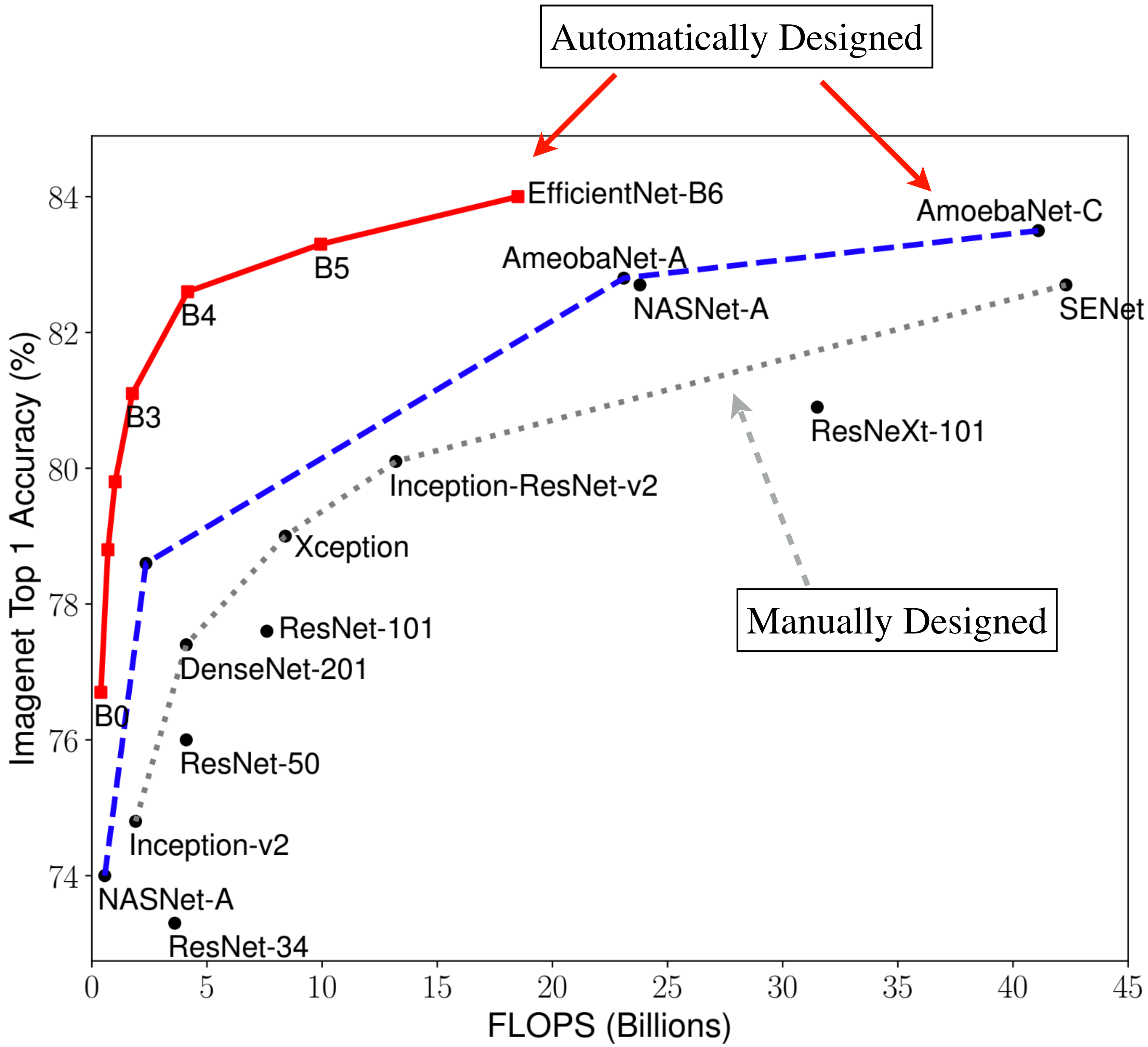
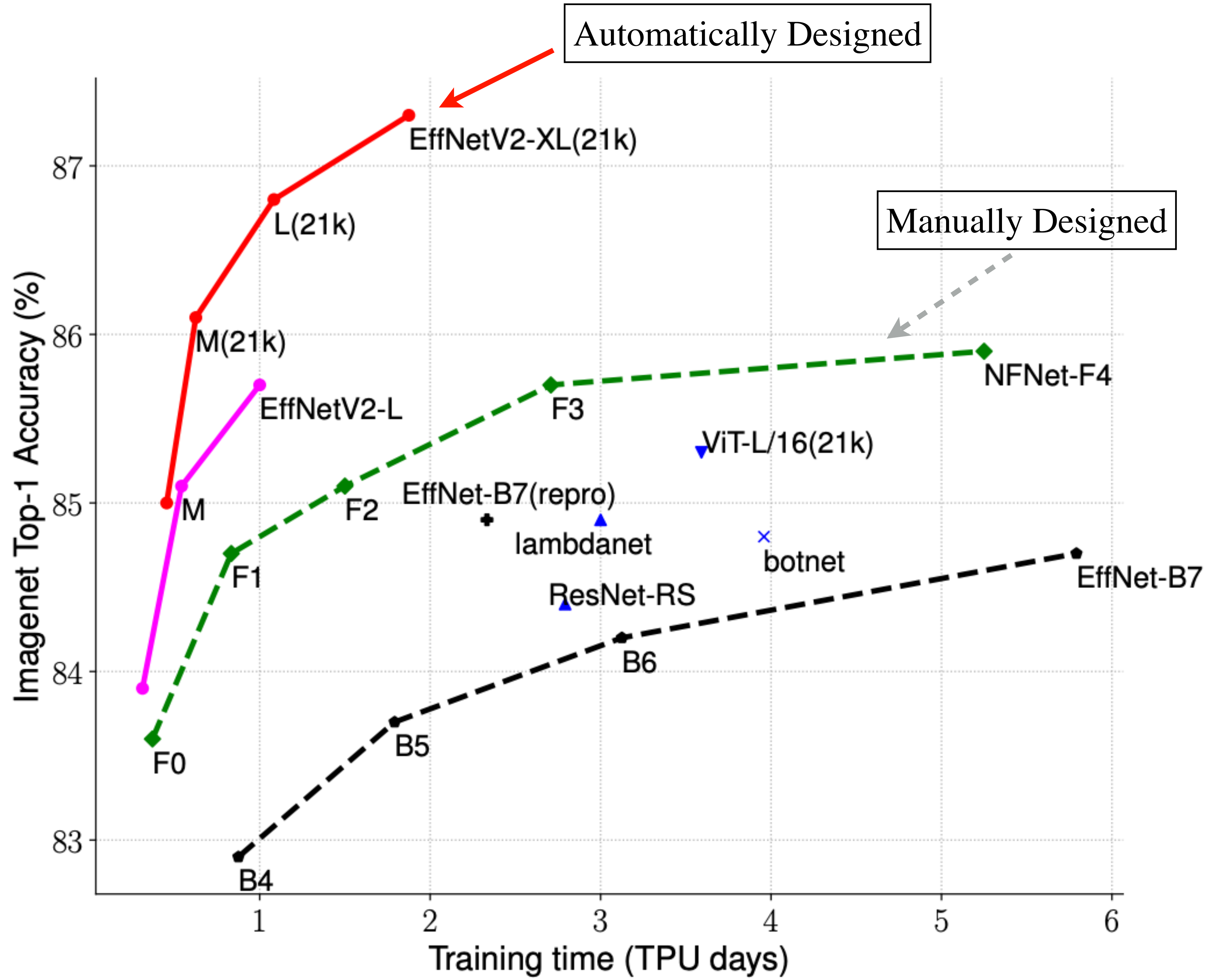
Extending the Search from Architecture to Hyperparameter, Hardware, and System

Xuanyi Dong
http://xuanyidong.com

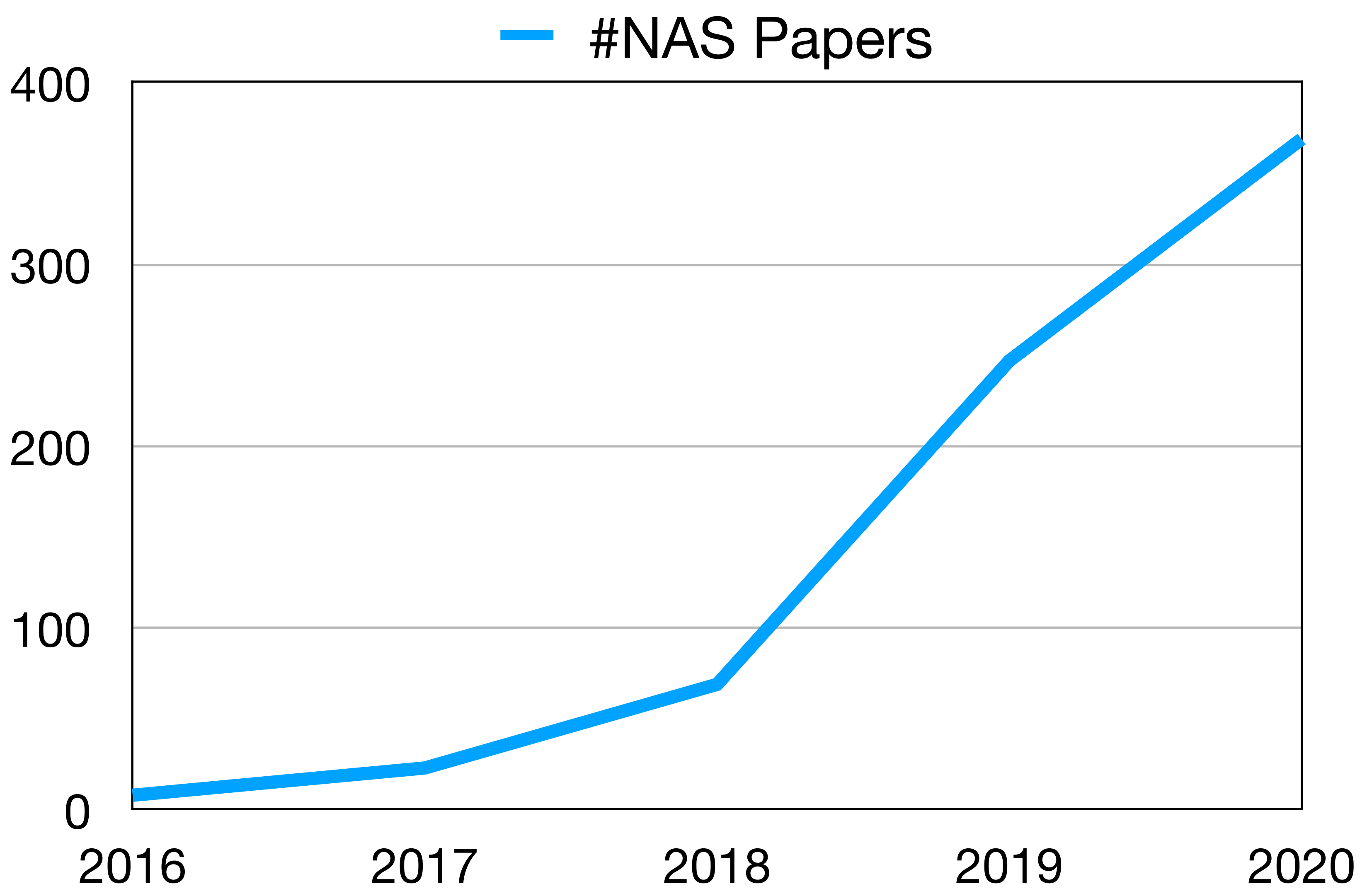
Auto-Architecture vs. Manual Architecture



Auto-Architecture vs. Manual Architecture

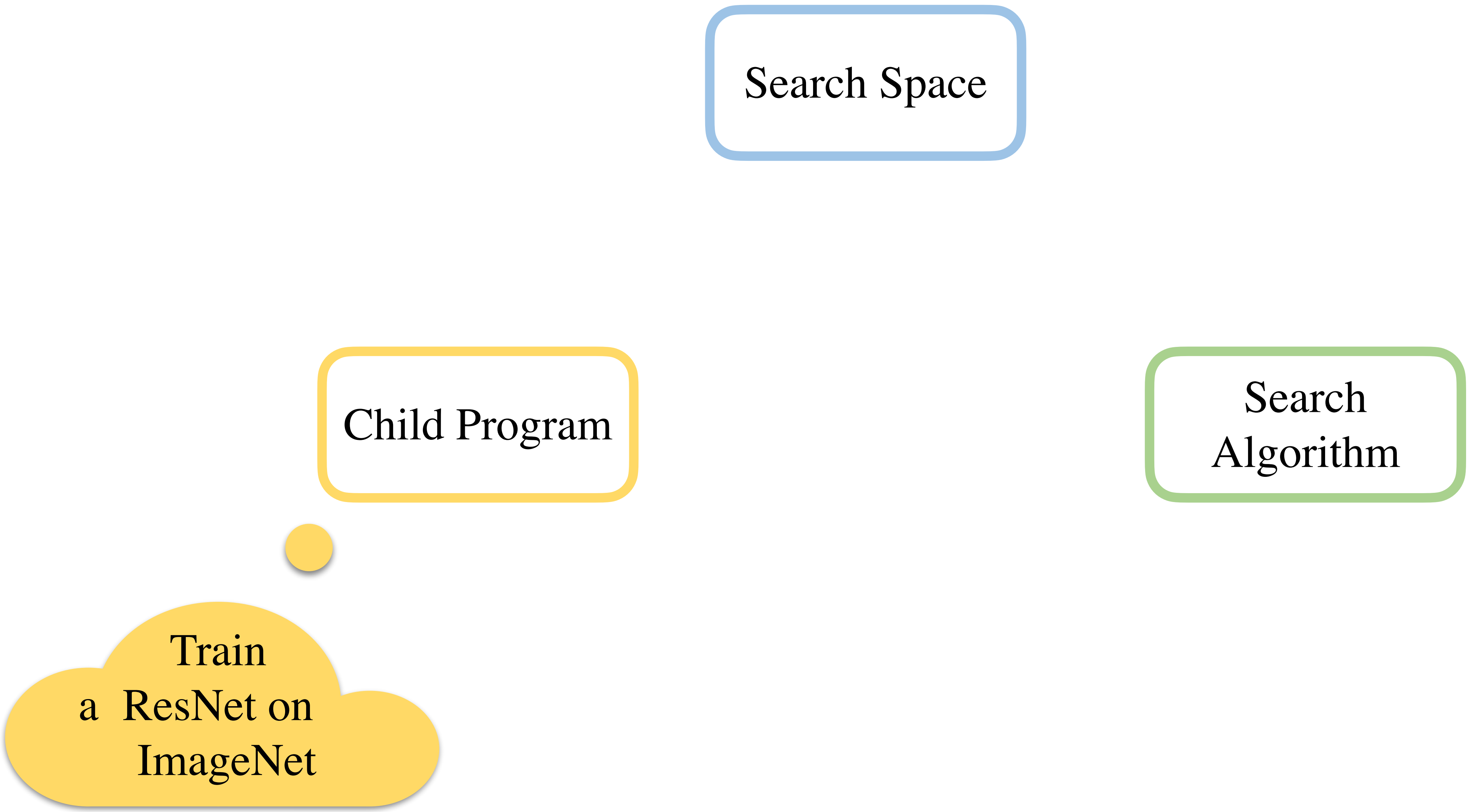


Neural Architecture Search Grows Fast

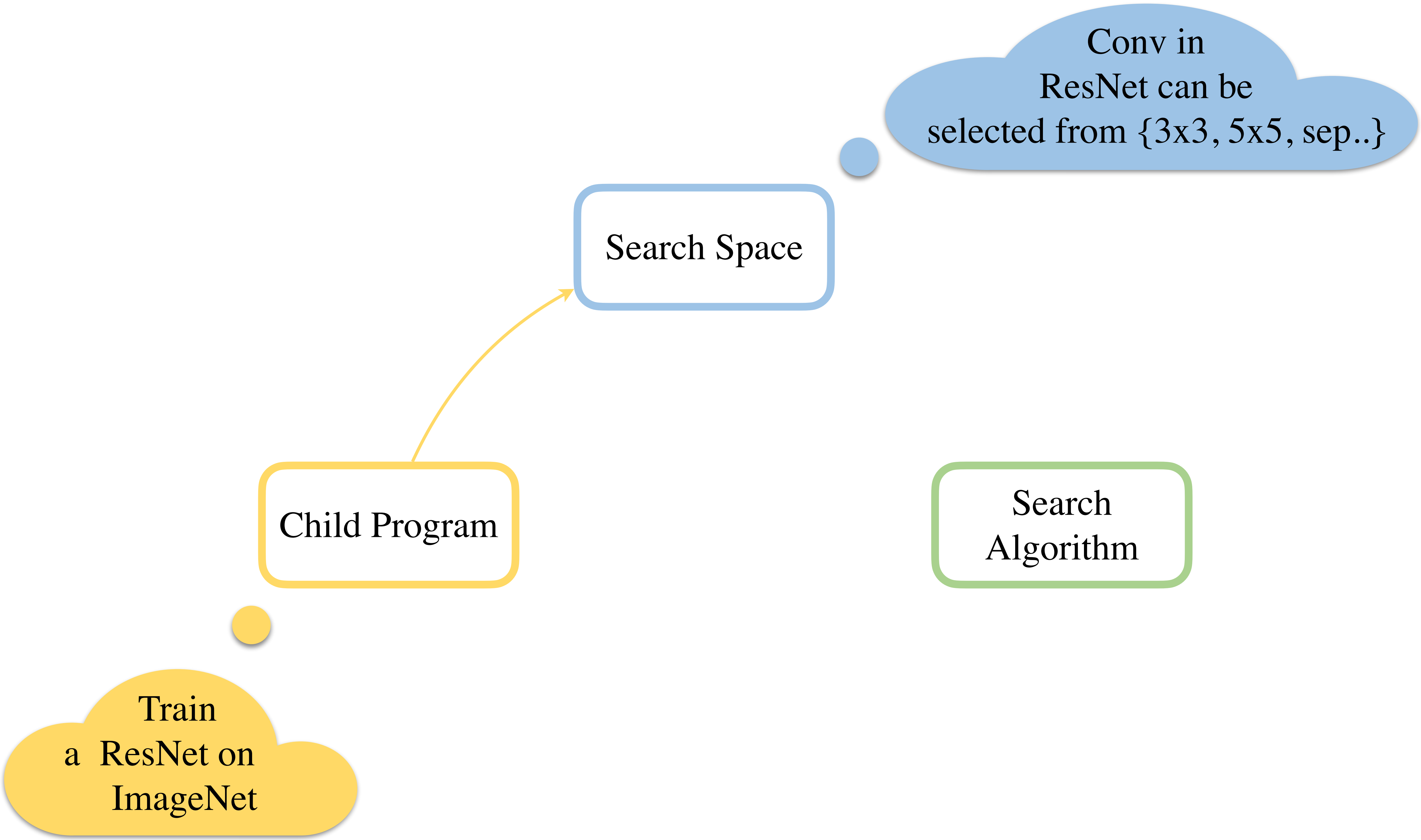


The number of NAS papers rapidly increases.

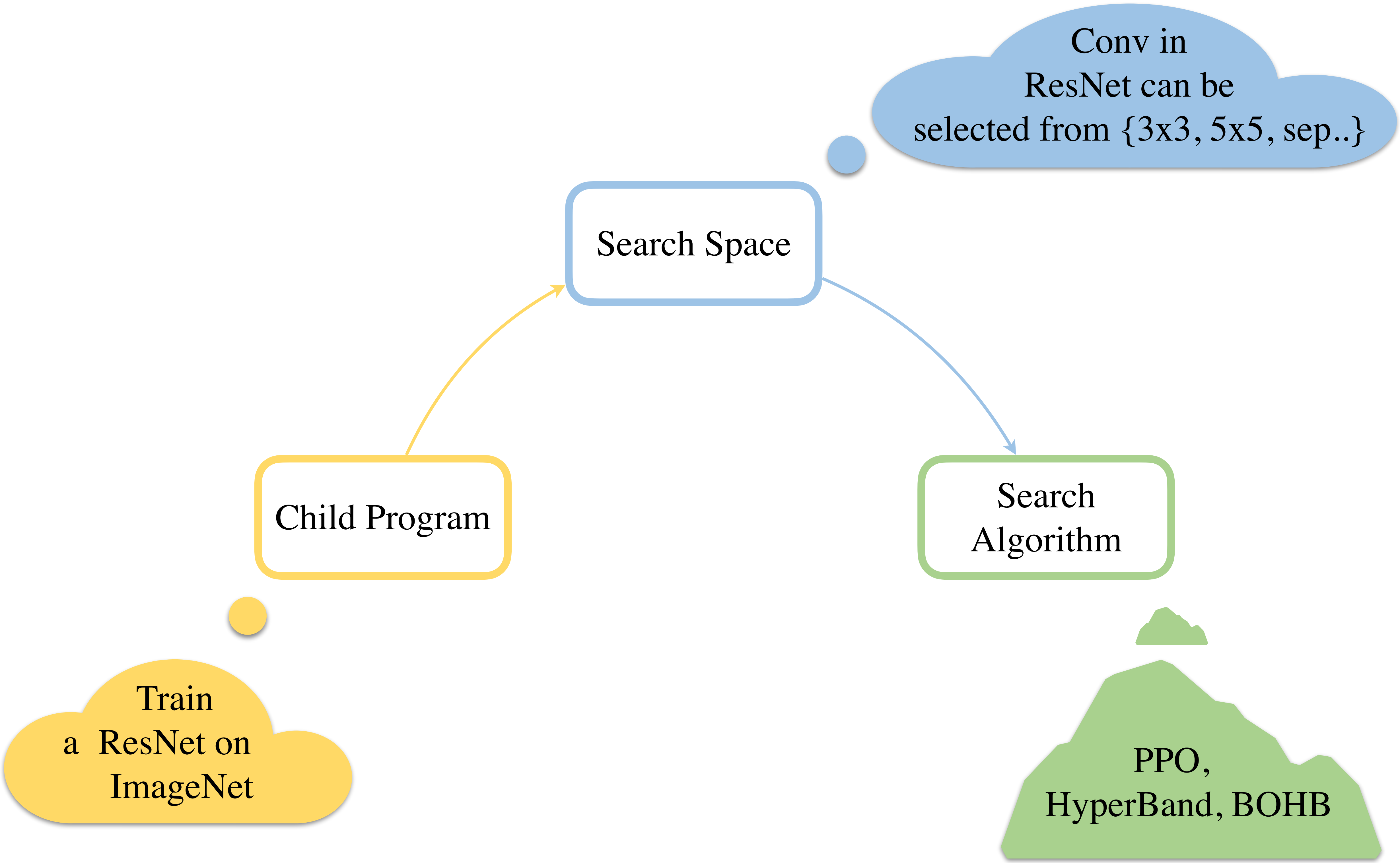
What is Neural Architecture Search (NAS)?



What is Neural Architecture Search (NAS)?

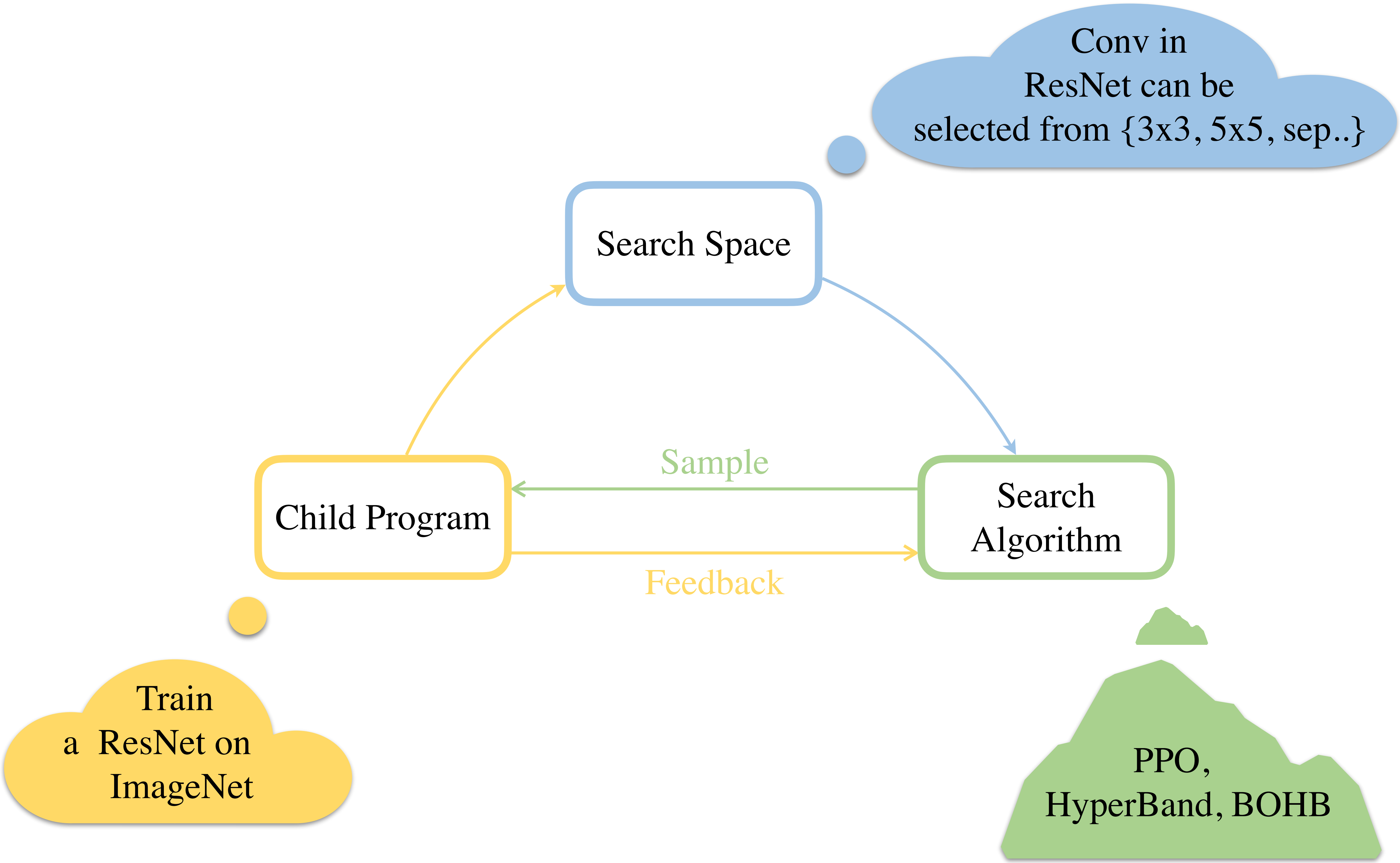


What is Neural Architecture Search (NAS)?



PyGlove: Symbolic Programming for Automated Machine Learning, NeurIPS 2020

What is Neural Architecture Search (NAS)?



What is important to NAS?

Search Space

NASNet Search Space

MBConv Search Space

Search Algorithm

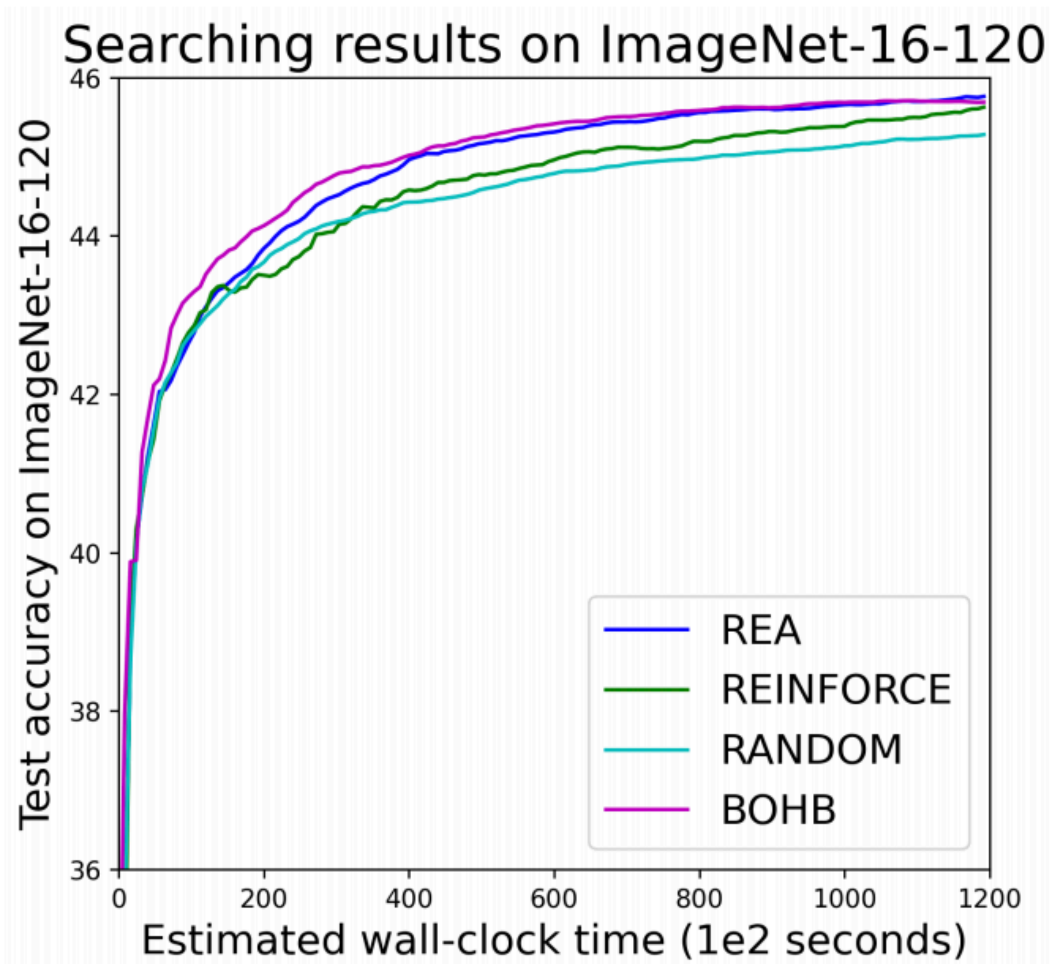
BayesOpt
PPO, REINFORCE
Evolution
Differentiable Search

NAS's performance is saturated - Search Space

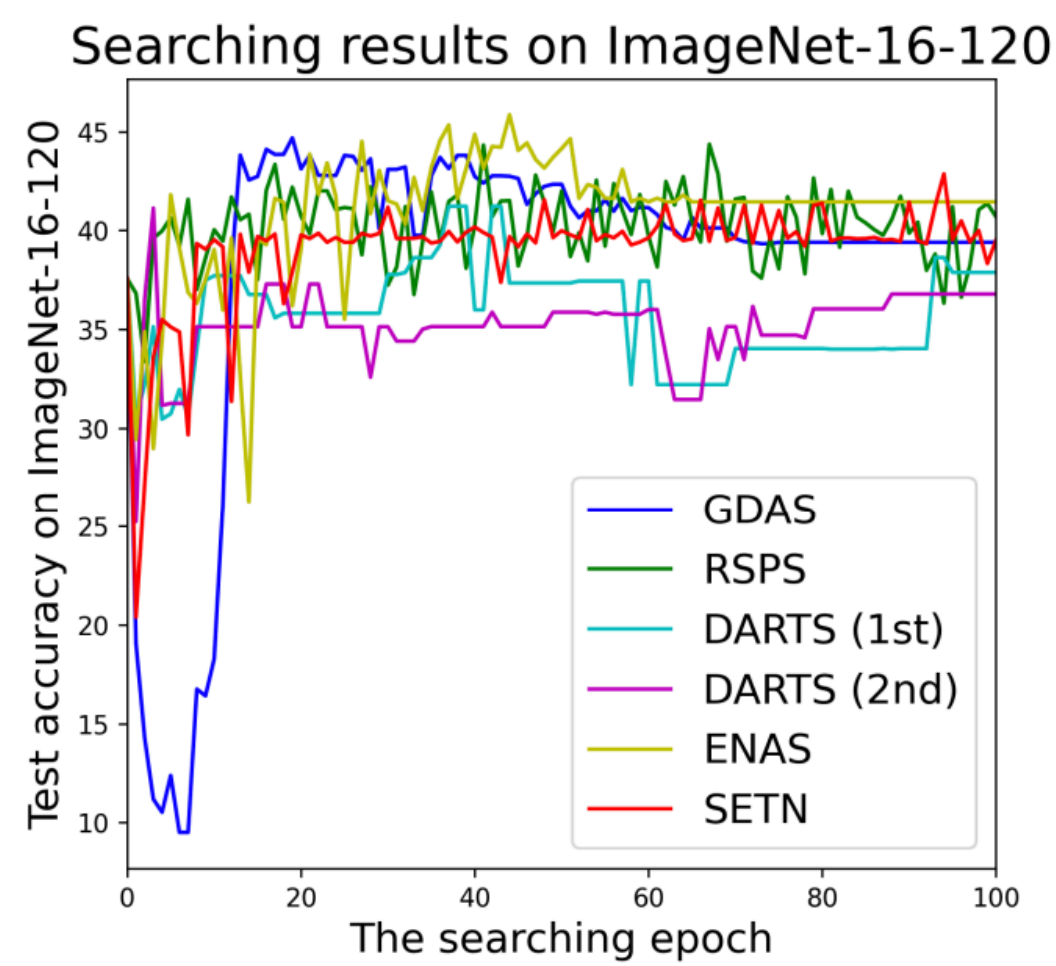
Search Space	Methods		CIFAR-10		CIFAR-100		ImageNet-16-120	
	Type	Name	validation	test	validation	test	validation	test
Topology Search Space $\mathcal{S}_t$	Multi-trial	REA	91.25±0.31	94.02±0.31	72.28±0.95	72.23±0.84	45.71±0.77	45.77±0.80
		REINFORCE	91.12±0.25	93.90±0.26	71.80±0.94	71.86±0.89	45.37±0.74	45.64±0.78
		RANDOM	91.07±0.26	93.86±0.23	71.46±0.97	71.55±0.97	45.03±0.91	45.28±0.97
		BOHB	91.17±0.27	93.94±0.28	72.04±0.93	72.00±0.86	45.55±0.79	45.70±0.86
	Weight Sharing	RSPS	87.60±0.61	91.05±0.66	68.27±0.72	68.26±0.96	39.73±0.34	40.69±0.36
		DARTS (1st)	49.27±13.44	59.84±7.84	61.08±4.37	61.26±4.43	38.07±2.90	37.88±2.91
		DARTS (2nd)	58.78±13.44	65.38±7.84	59.48±5.13	60.49±4.95	37.56±7.10	36.79±7.59
		GDAS	89.68±0.72	93.23±0.58	68.35±2.71	68.17±2.50	39.55±0.00	39.40±0.00
		SETN	90.00±0.97	92.72±0.73	69.19±1.42	69.36±1.72	39.77±0.33	39.51±0.33
		ENAS	90.20±0.00	93.76±0.00	70.21±0.71	70.67±0.62	40.78±0.00	41.44±0.00
		ResNet	90.86	93.91	70.50	70.89	44.10	44.23
		Optimal	91.61	94.37 (94.37)	73.49	73.51 (73.51)	46.73	46.20 (47.31)
Size Search Space $\mathcal{S}_s$	Multi-trial	REA	90.37±0.20	93.22±0.16	70.23±0.50	70.11±0.61	45.30±0.69	45.94±0.92
		REINFORCE	90.25±0.23	93.16±0.21	69.84±0.59	69.96±0.57	45.06±0.77	45.71±0.93
		RANDOM	90.10±0.26	93.03±0.25	69.57±0.57	69.72±0.61	45.01±0.74	45.42±0.86
		BOHB	90.07±0.28	93.01±0.24	69.75±0.60	69.90±0.60	45.11±0.69	45.56±0.81
	Weight Sharing	channel-wise interpolation	90.71±0.00	93.40±0.00	70.30±0.00	70.72±0.00	44.73±0.00	47.17±0.00
		masking + Gumbel-Softmax	90.41±0.10	93.14±0.13	70.30±0.00	70.72±0.00	45.71±0.39	46.38±0.27
		masking + sampling	89.73±0.37	92.78±0.30	69.67±0.22	70.11±0.33	44.70±0.60	45.11±0.76
		Largest Candidate	90.71	93.40	70.30	70.72	44.73	47.17
		Optimal	90.71	93.40 (93.65)	70.92	70.12 (71.34)	46.73	45.10 (47.40)

NATS-Bench: Benchmarking NAS Algorithms for Architecture Topology and Size, IEEE TPAMI 2021

NAS's performance is saturated - Search Algorithm



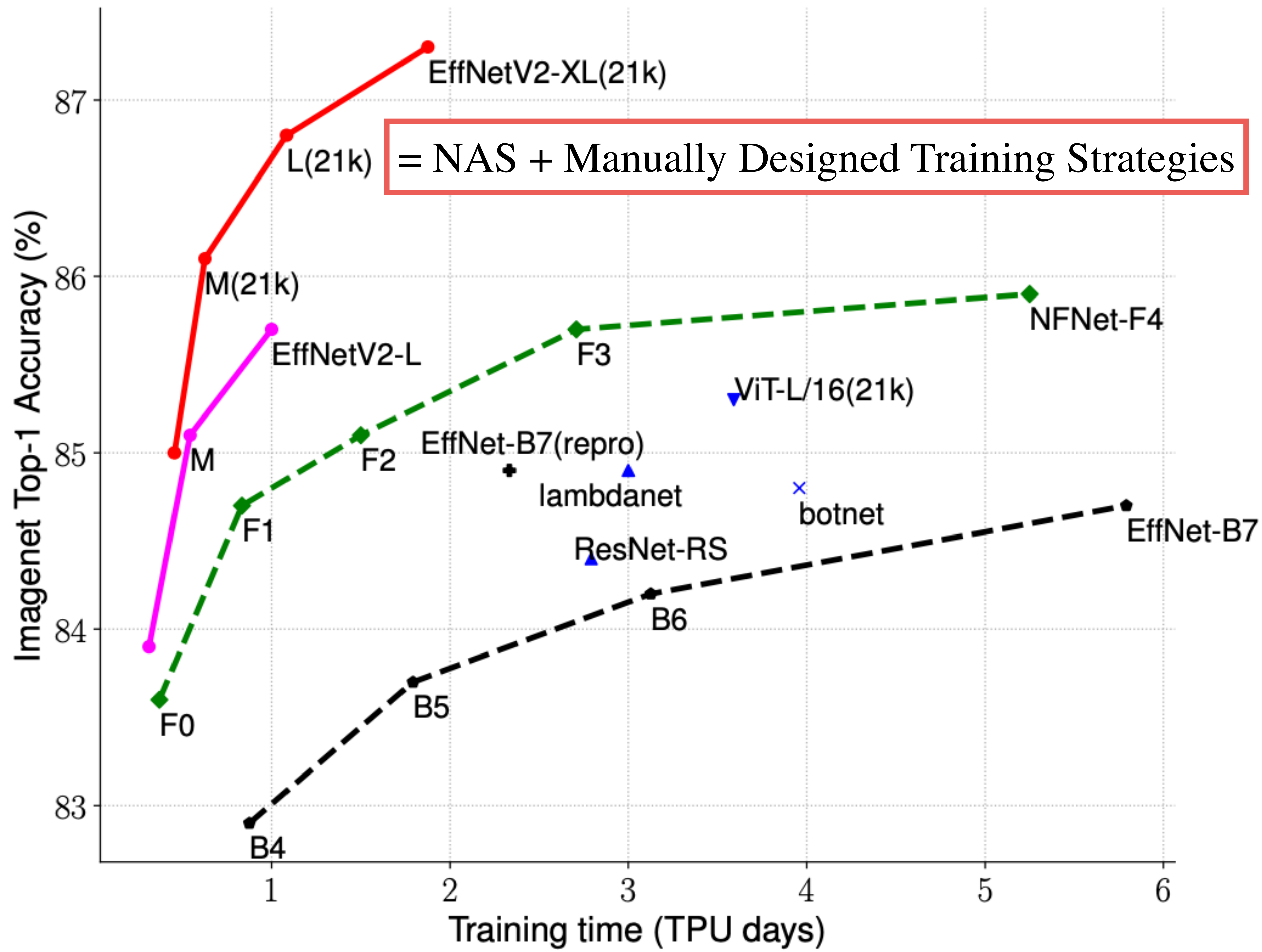
Multi-trial Search



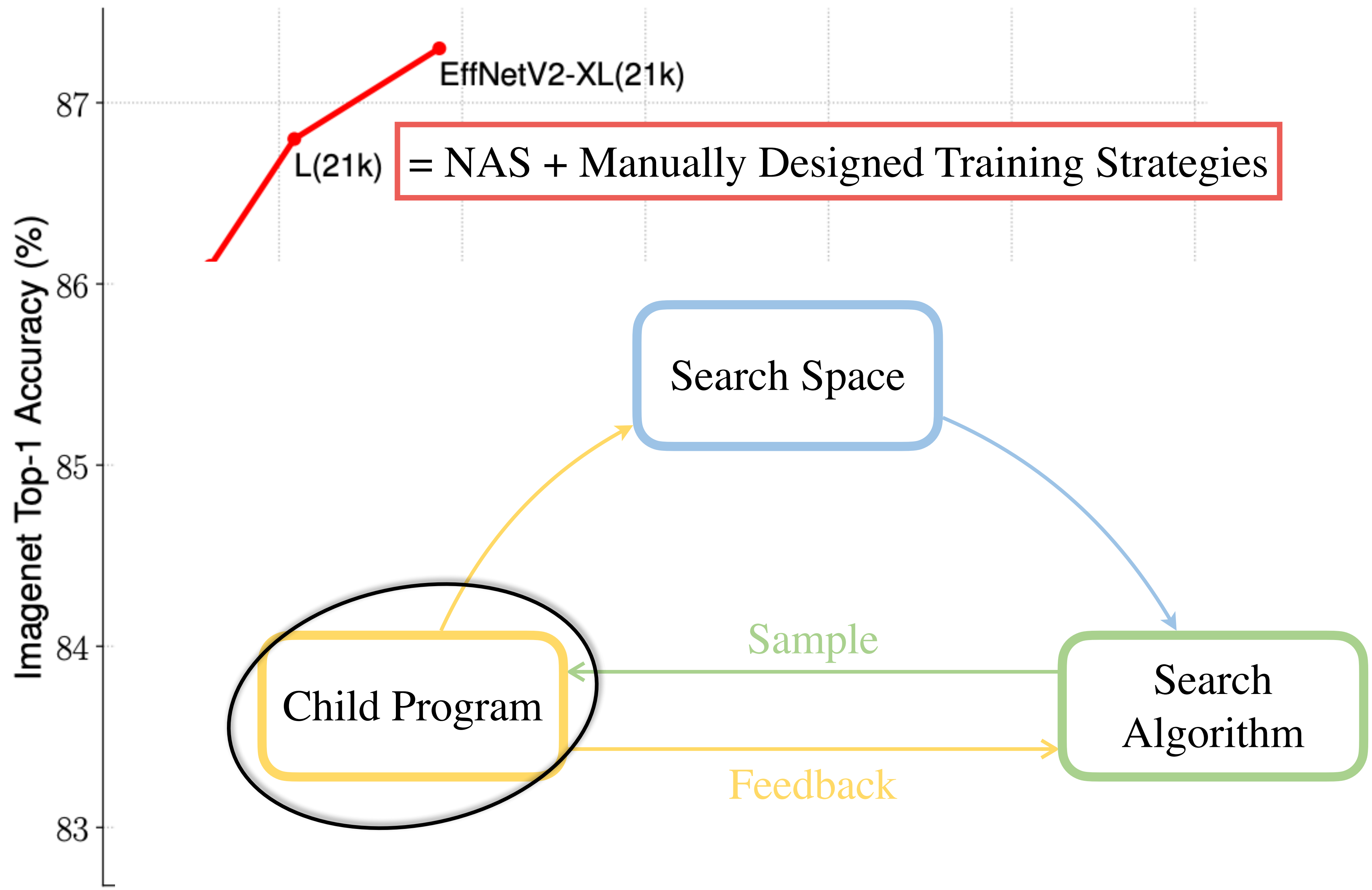
Weight-sharing Search

NATS-Bench: Benchmarking NAS Algorithms for Architecture Topology and Size, IEEE TPAMI 2021

NAS is sub-optimal



NAS is sub-optimal



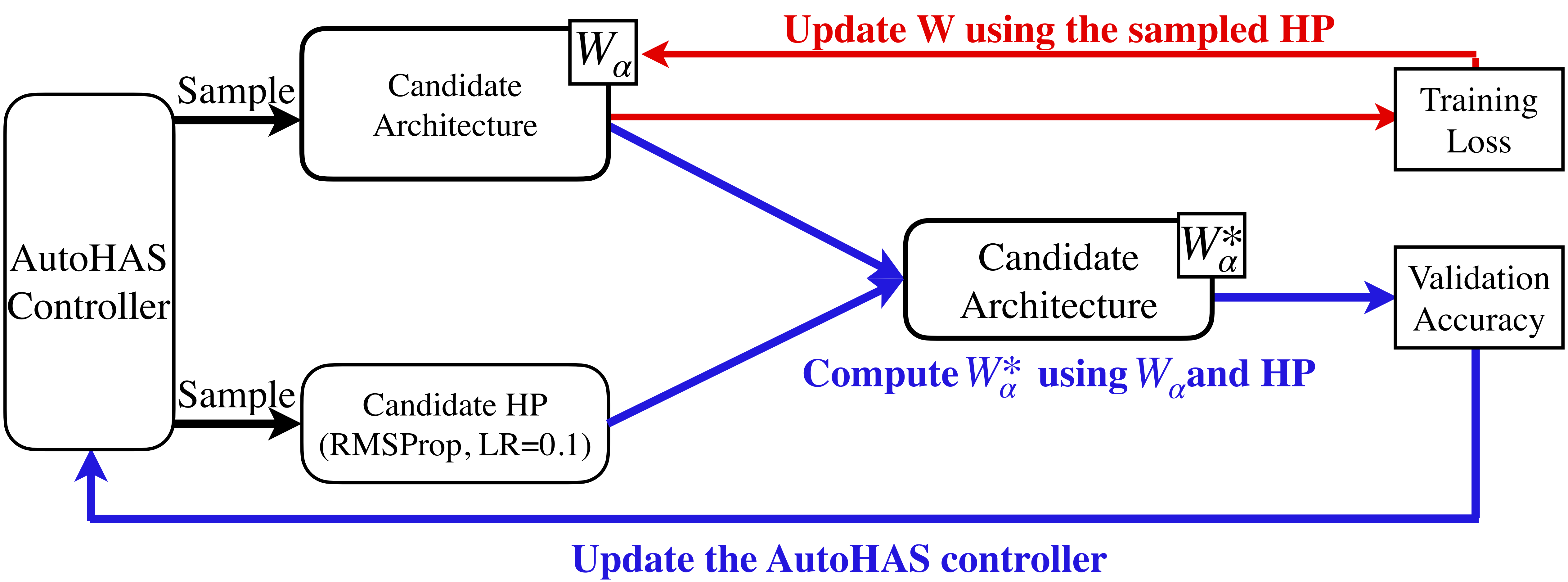
NAS is sub-optimal

	Model-1	Rank	Model-2
HP-1 (LR=5.5, L2=1.5e-4)	56.9%	>	55.6%
HP-2 (LR=1.1, L2=8.4e-4)	54.7%	<	56.2%

AutoHAS: Efficient and Joint Search

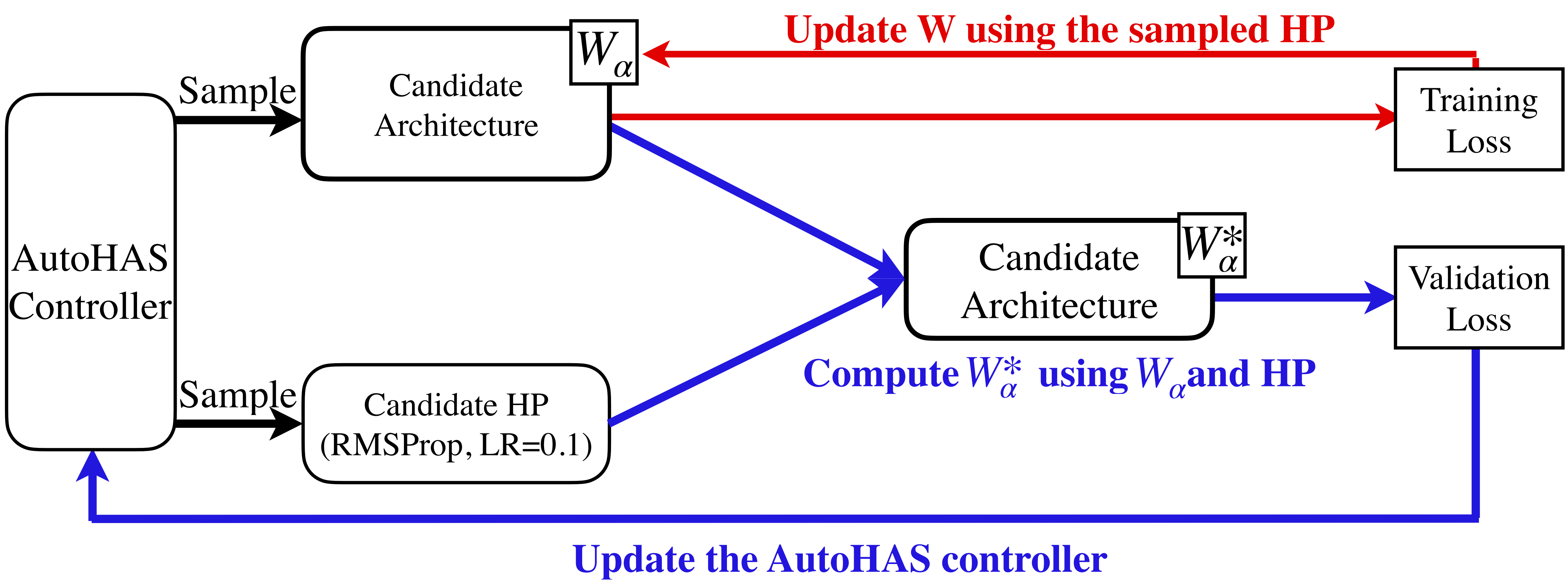
	learning rate	weight decay	augmentation	dropout	architecture	**efficient**
Bayesian	√	√	√	√	√	×
RL or Evolution	√	√	√	√	√	×
PBT	√	√	√	√	×	×
Gradient Descent on LR	√	×	×	×	×	√
Hypergradient	×	√	√	×	√	√
NAS (Weight Sharing)	×	×	×	×	√	√
AutoHAS	√	√	√	√	√	√

AutoHAS: Efficient and Joint Search



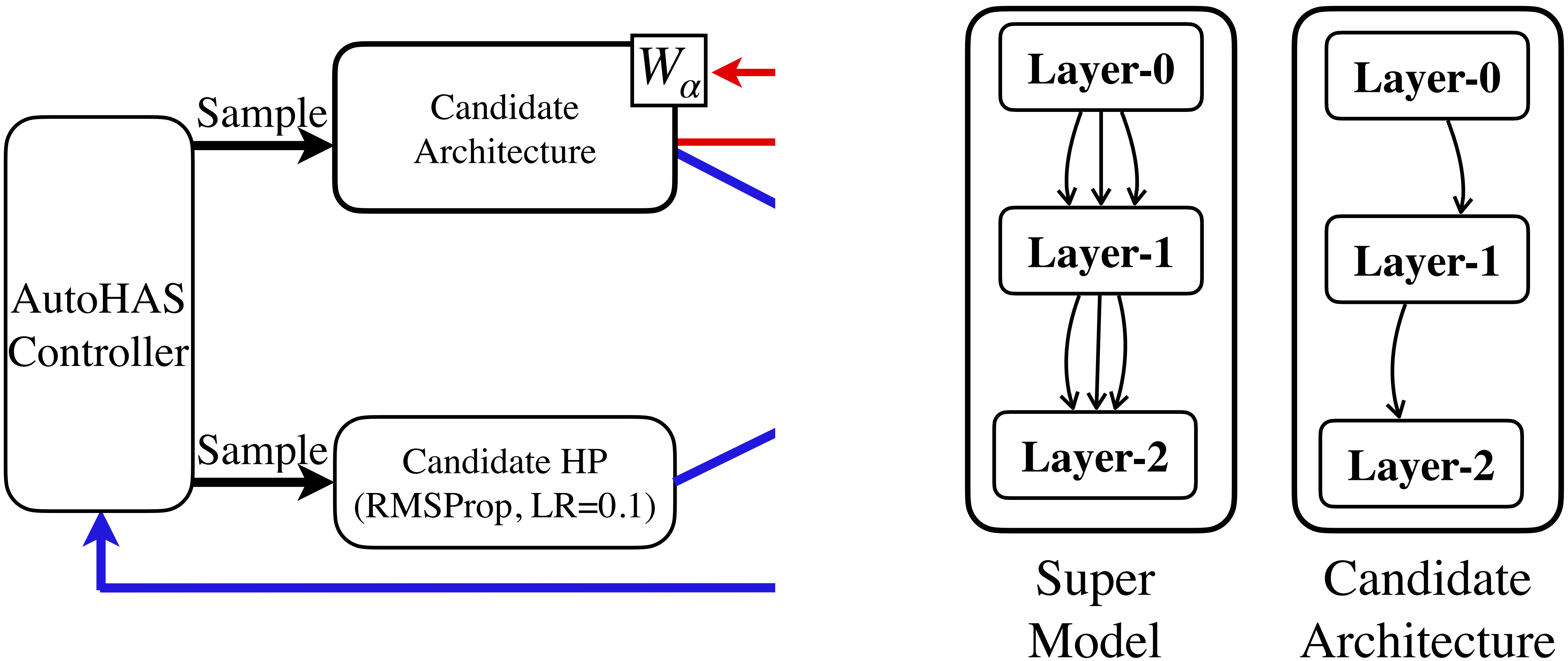
Integrate REINFORCE into AutoHAS

AutoHAS: Efficient and Joint Search



Integrate Differentiable Search into AutoHAS

AutoHAS: Weight Sharing



AutoHAS: Differentiable vs. REINFORCE

	#Params (MB)	#FLOPs (M)	Accuracy (%)	Search Cost Memory (GB)	Search Cost Time (TPU Hour)
Baseline model	1.5	35.9	50.96	1.0	44.8
AutoHAS (Differentiable)	1.5	36.1	52.17	6.1	92.8
AutoHAS (REINFORCE)	1.5	36.3	53.01	**1.8**	**54.4**

Search for architecture, learning rate, weight decay

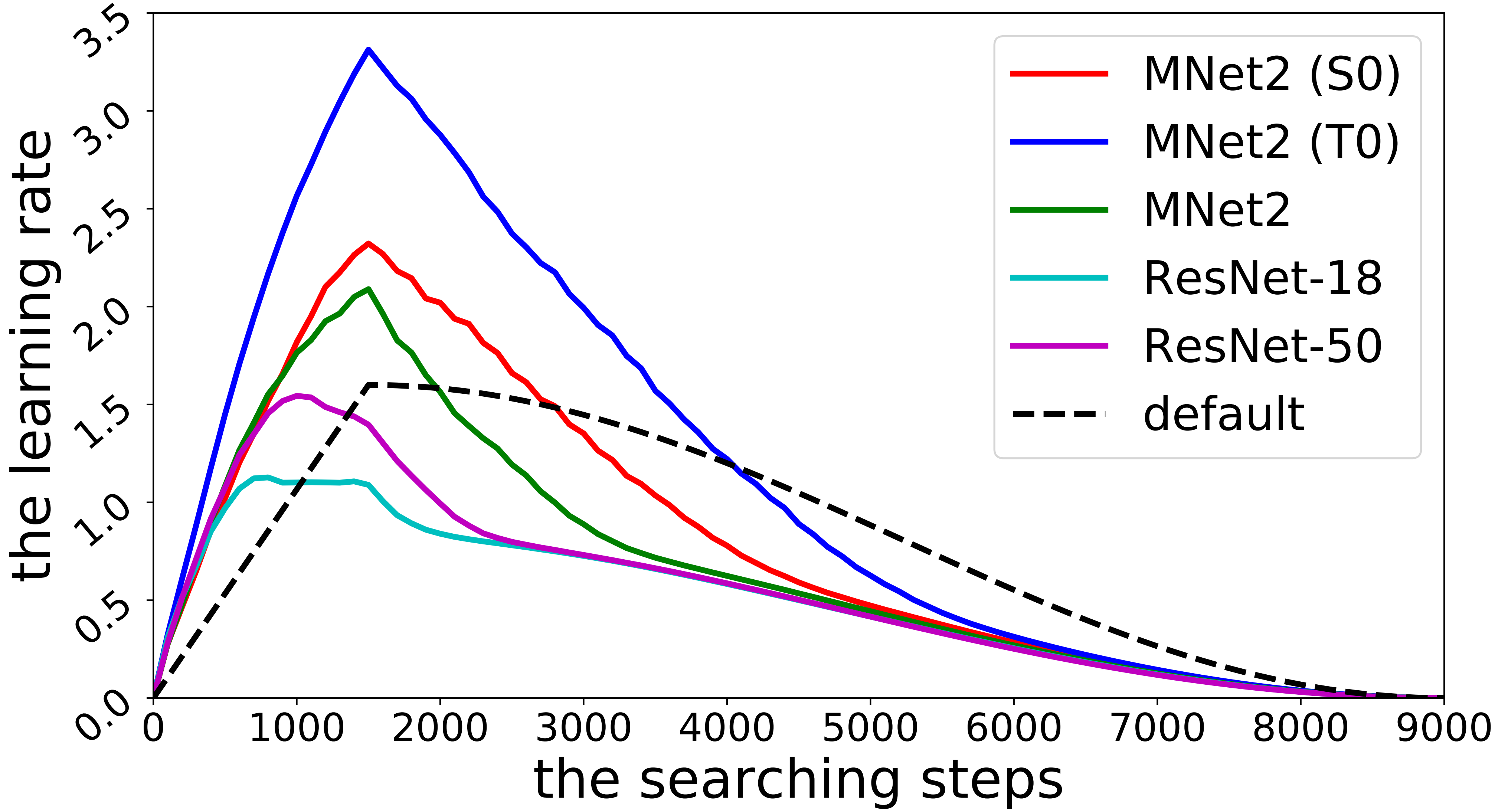
AutoHAS improves SoTA models

Model	Method	#Params (M)	#FLOPs (M)	Top-1 Accuracy (%)
ResNet-50	Human	25.6	4110	77.20
	AutoHAS	25.6	4110	**77.83** (+0.63)
EfficientNet-B0	NAS	5.3	398	77.15
	AutoHAS	5.2	418	**77.92** (+0.77)

AutoHAS-discovered Hyperparameters

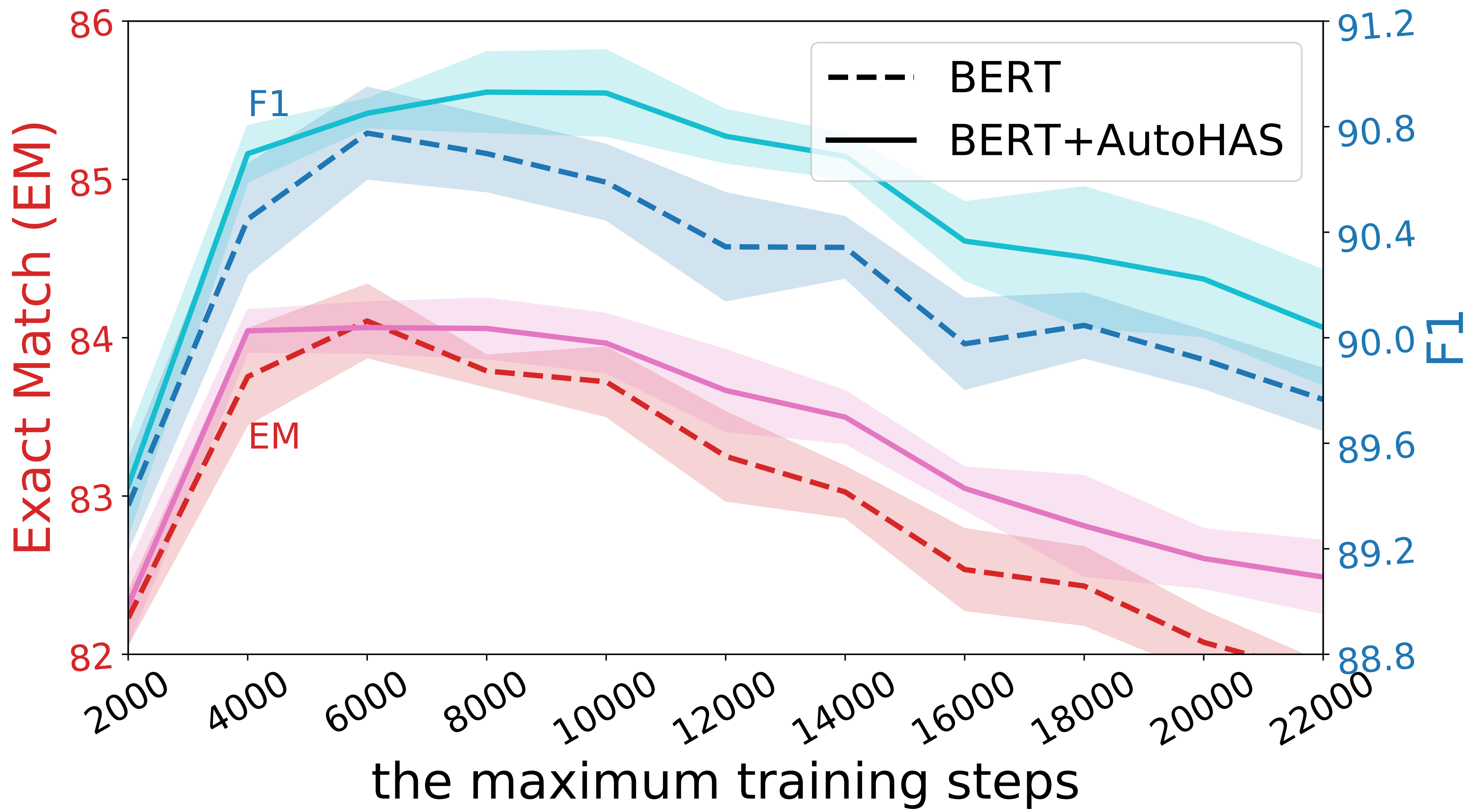
Search for drop-path ratio in EfficientNet

AutoHAS-discovered Hyperparameters



Search for learning rate for MobileNet and ResNet

AutoHAS-discovered Hyperparameters



Search for learning rate and weight decay for BERT

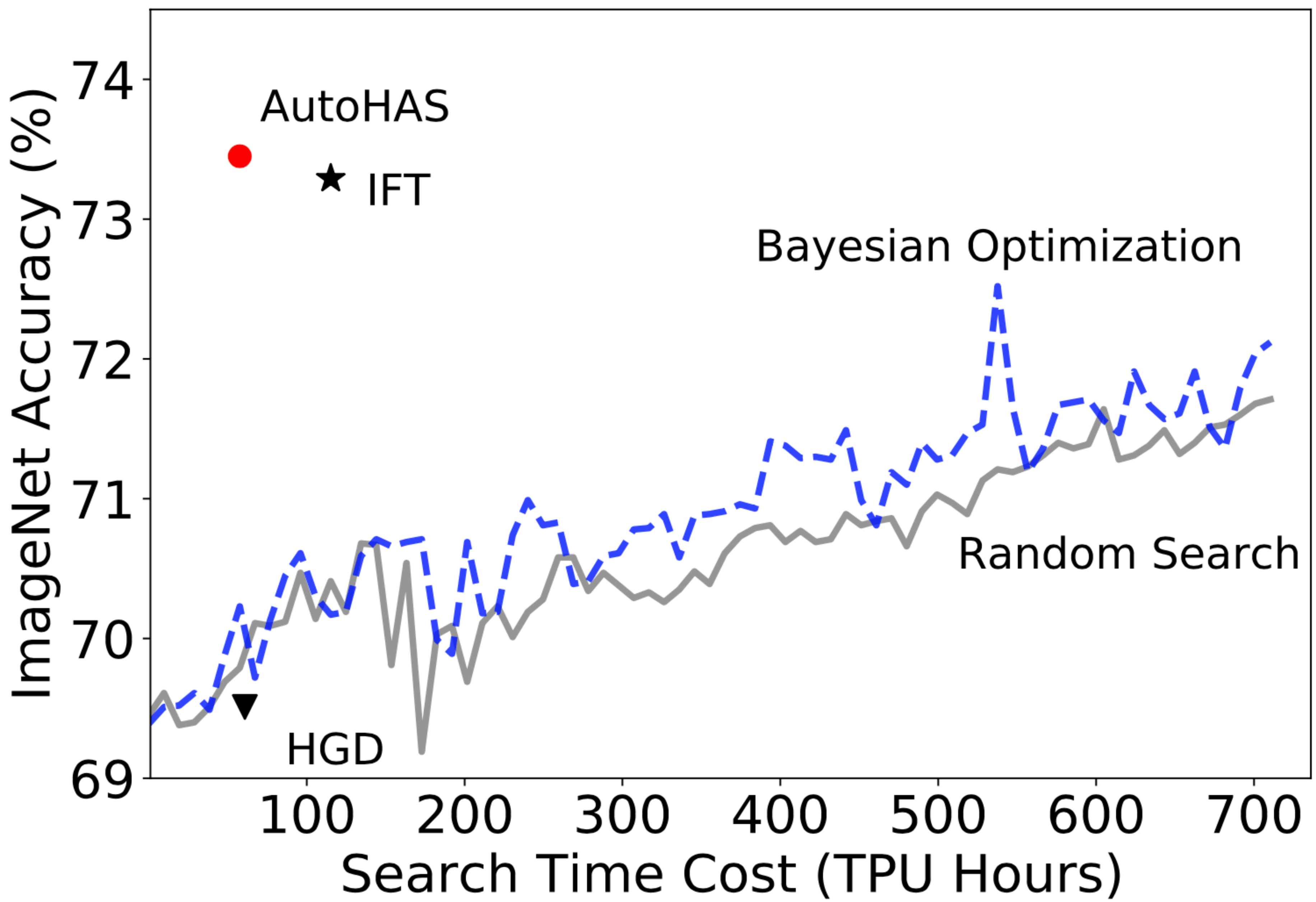
AutoHAS works on different datasets

Method	Search Space	CIFAR-10	CIFAR-100	Stanford Cars	Oxford Flower	SUN-397
MobileNetV2 (baseline)		94.1	76.3	83.8	74.0	46.3
AutoHAS	Weight Decay	95.0	77.8	89.0	84.4	49.1
AutoHAS	MixUp	94.1	77.0	85.2	79.6	47.4
AutoHAS	Arch	94.5	76.8	84.1	76.4	46.3
AutoHAS	MixUp + Arch	94.4	77.4	84.8	78.2	47.3
AutoHAS	Weight Decay + MixUp	**95.0** (**+0.9**)	**78.4** (**+2.1**)	**89.9** (**+6.1**)	**84.4** (**+10.4**)	**50.5** (**+4.2**)

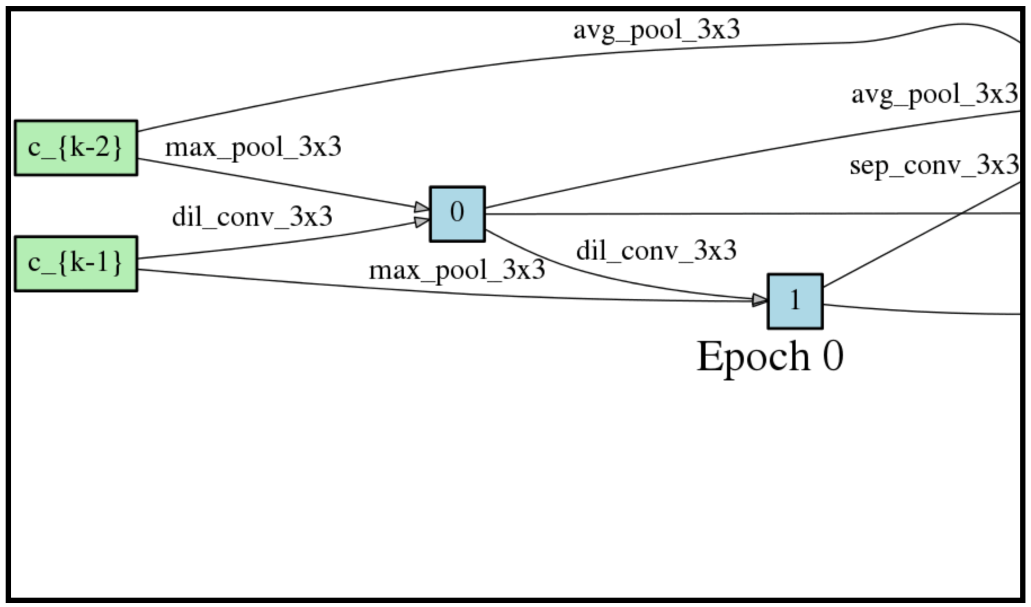
AutoHAS works on different datasets

Method	Search Space	Image Classification Top-1 Accuracy (%) CIFAR-10	CIFAR-100	Stanford Cars	Oxford Flower	SUN-397
MobileNetV2 (baseline)		94.1	76.3	83.8	74.0	46.3
AutoHAS	Weight Decay	95.0	77.8	89.0	84.4	49.1
AutoHAS	MixUp	94.1	77.0	85.2	79.6	47.4
AutoHAS	Arch	94.5	76.8	84.1	76.4	46.3
AutoHAS (Joint)	MixUp + Arch	94.4	77.4	84.8	78.2	47.3
AutoHAS (Sequential)	MixUp + Arch	94.4	77.6	85.5	79.6	48.3
AutoHAS (Joint)	Weight Decay + MixUp	**95.0** (**+0.9**)	**78.4** (**+2.1**)	89.9	84.4	50.5
AutoHAS (Sequential)	Weight Decay + MixUp	94.9	78.2	**90.5** (**+6.8**)	**85.4** (**+11.4**)	**50.8** (**+4.5**)

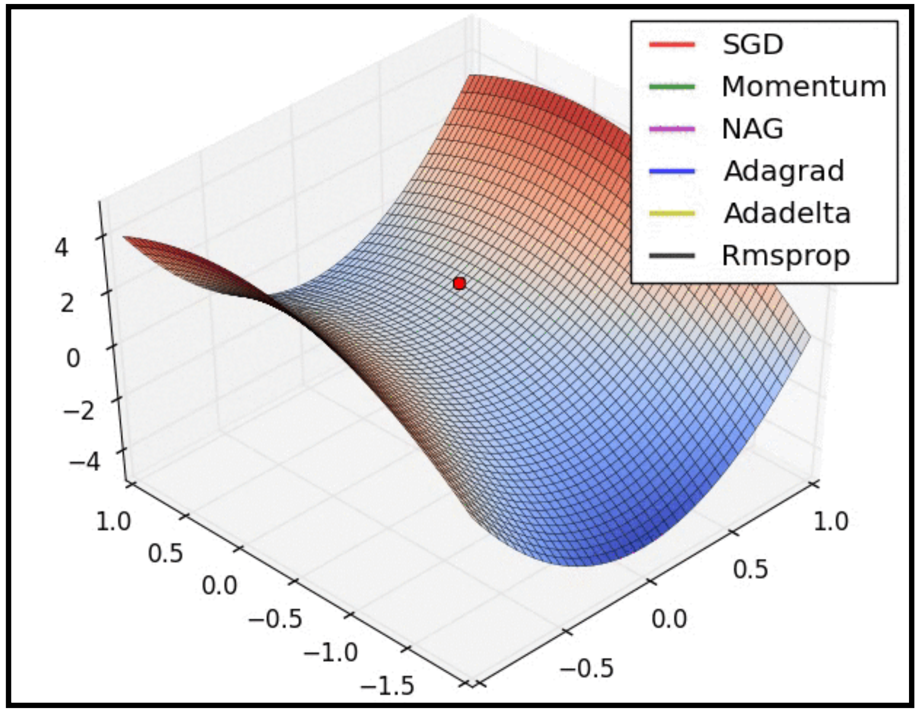
AutoHAS vs. other HPO methods



What's else?

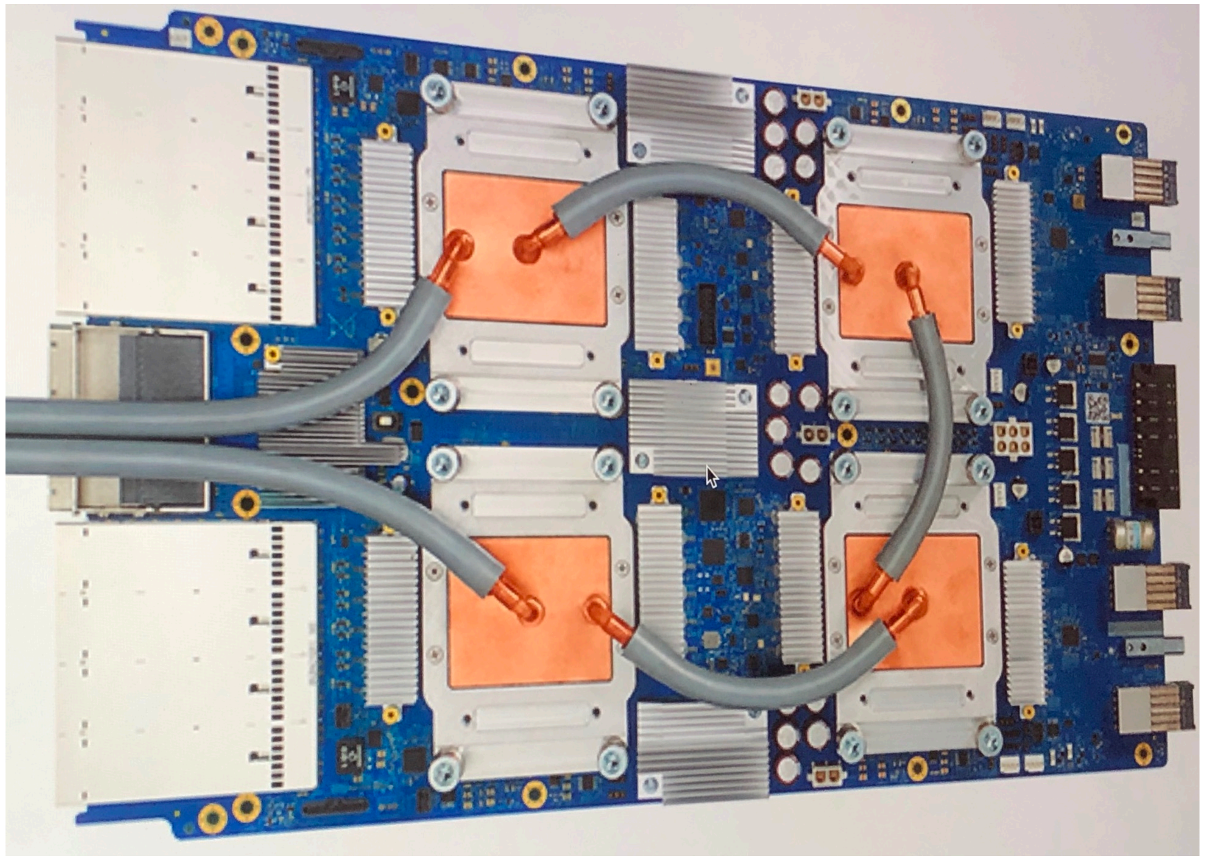


avg_pool_3x3
avg_pool_3x3
c_{k-2}
max_pool_3x3
sep_conv_3x3
dil_conv_3x3
0
c_{k-1}
dil_conv_3x3
max_pool_3x3
1
Epoch 0

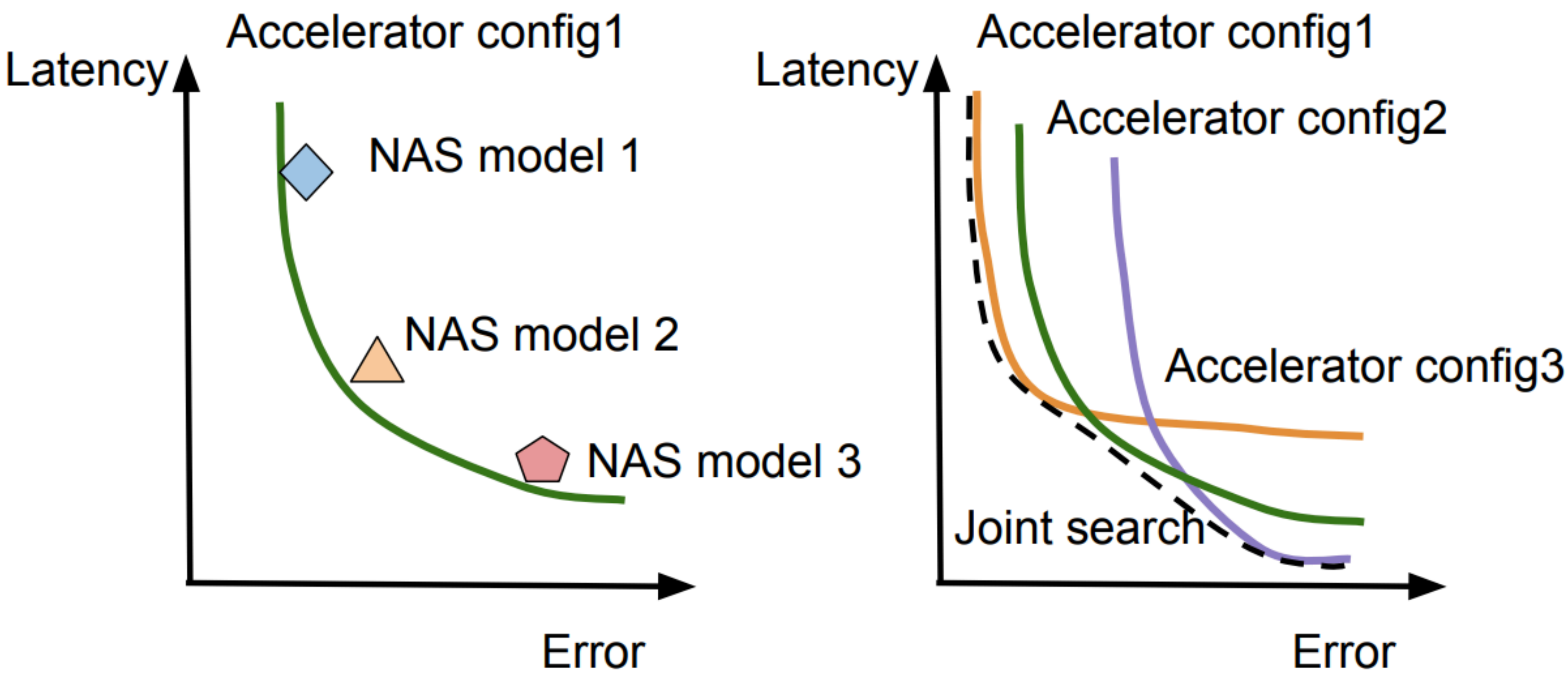


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop

What's else?

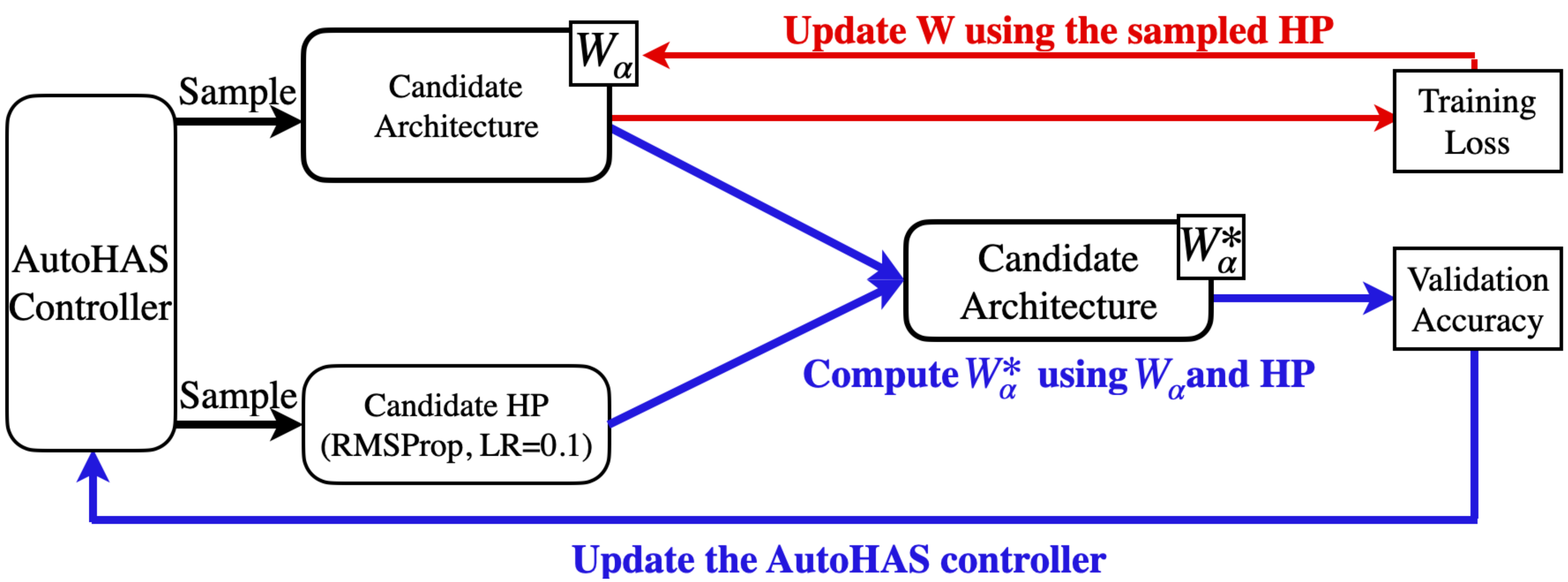


NAHAS: Better Pareto Frontier



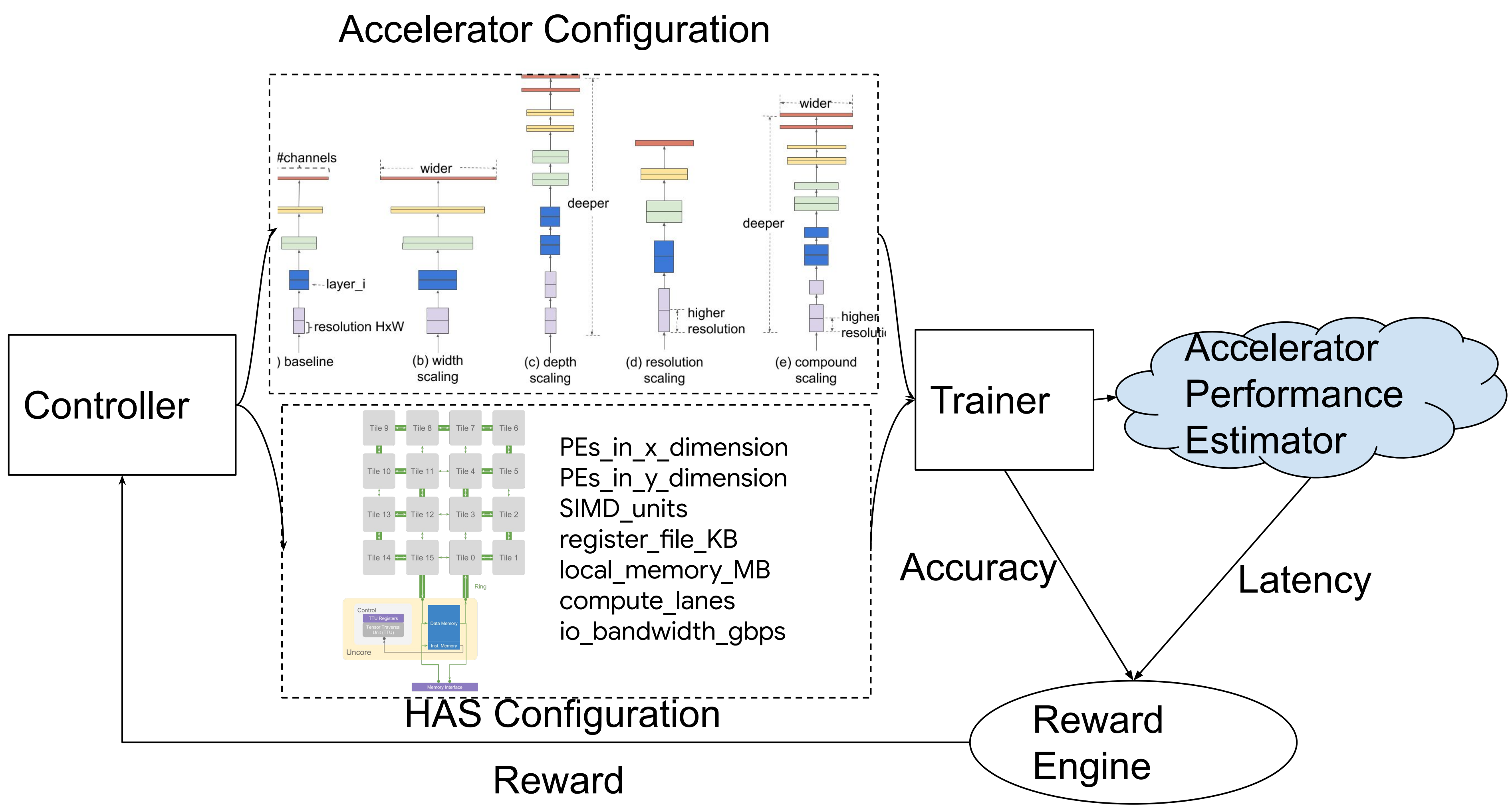
Rethinking Co-design of Neural Architectures and Hardware Accelerators, arXiv 2021.02

Joint Architecture and Accelerator Search



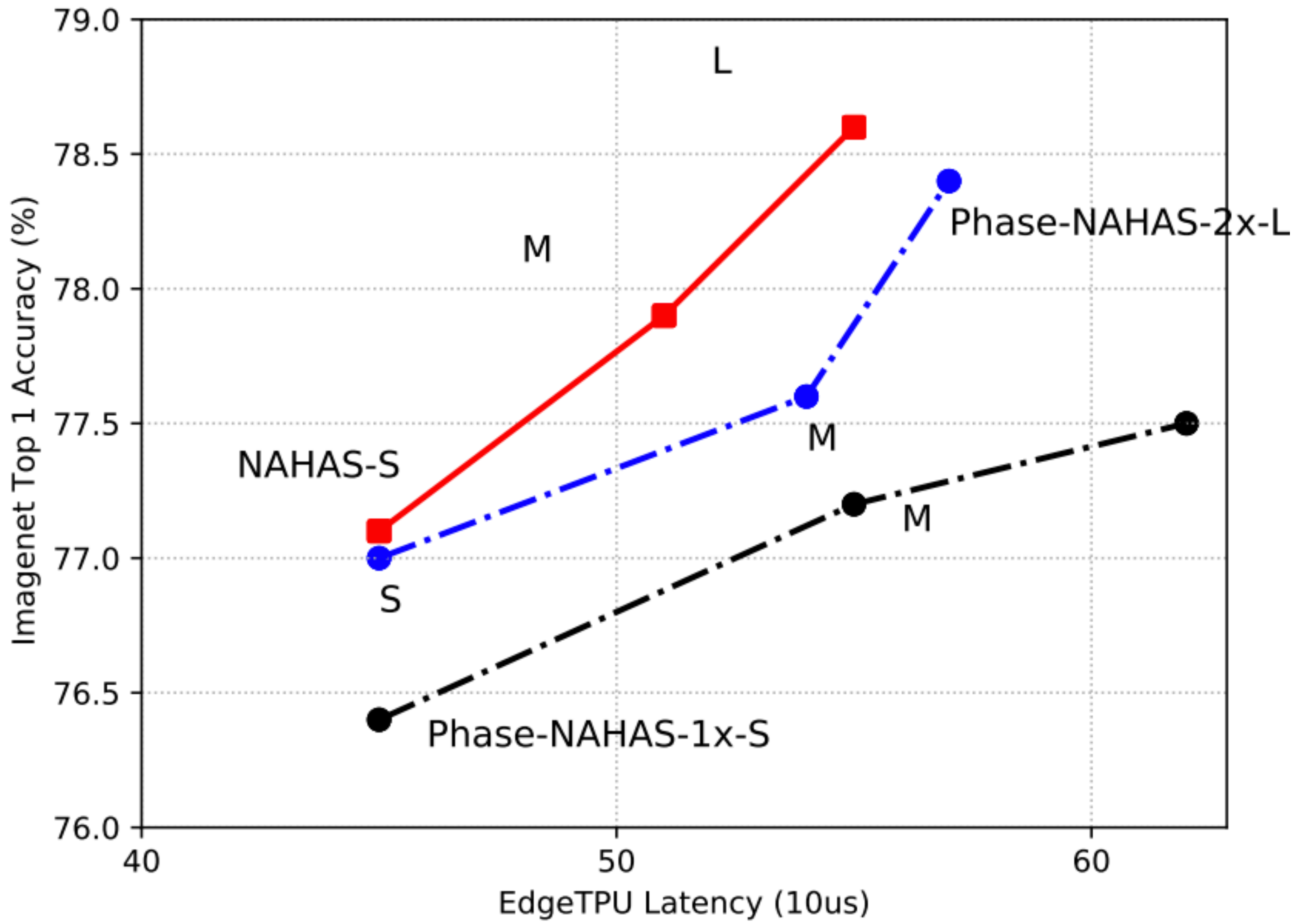
AutoHAS cannot handle hardware design

Joint Architecture and Accelerator Search



Rethinking Co-design of Neural Architectures and Hardware Accelerators, arXiv 2021.02

Joint Search vs. Phase Search



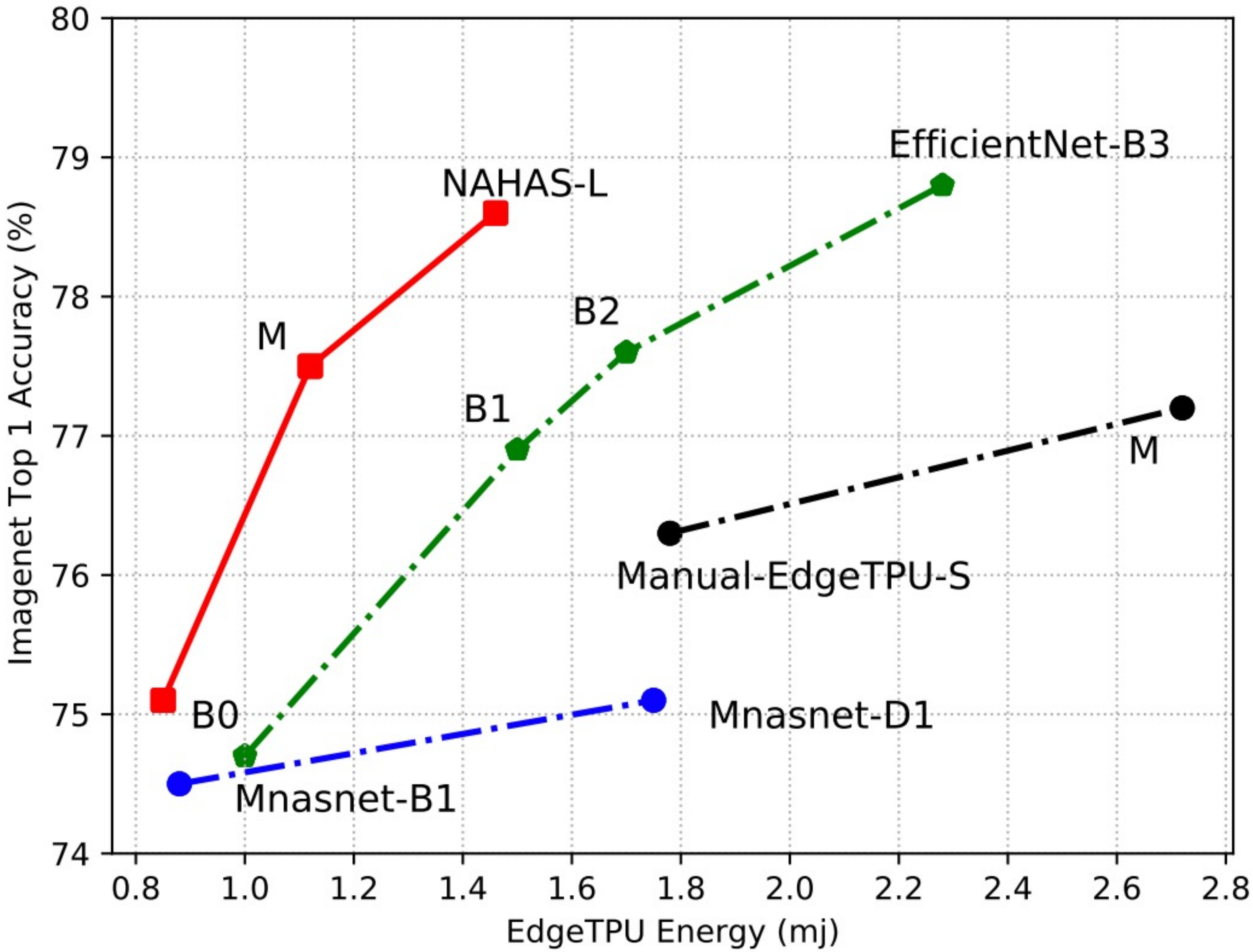
Rethinking Co-design of Neural Architectures and Hardware Accelerators, arXiv 2021.02

Multi-trial vs. One-shot

Model	Top-1 Acc.	Latency in **ms** (Ratio-to-best)	Energy in **mJ** (Ratio-to-best)
EfficientNet-B0 (Tan & Le, 2019) wo SE/Swish	74.7%	0.35 (1.17x)	1.00 (1.64x)
MobileNetV2 (Sandler et al., 2018)	74.4%	0.30 (1.00x)	0.70 (1.15x)
MnasNet-B1 (Tan et al., 2019)	74.5%	0.41 (1.37x)	0.88 (1.44x)
ProxylessNAS (Cai et al., 2019)	74.8%	0.42 (1.40x)	0.98 (1.61x)
Manual-EdgeTPU-small	76.2%	0.42 (1.40x)	1.78 (2.91x)
IBN-only fixed accelerator	74.6%	0.38 (1.27x)	0.82 (1.34x)
IBN-only NAHAS multi-trial	74.9%	**0.30**	0.75 (1.23x)
IBN-only NAHAS oneshot	**76.5%**	0.35 (1.17x)	**0.61**
EfficientNet-B1 (Tan & Le, 2019) wo SE/Swish	76.9%	0.51 (1.04x)	1.50 (1.53x)
MnasNet-D1 (Tan et al., 2019)	75.1%	0.55 (1.12x)	1.75 (1.78x)
Fixed accelerator multi-trial w fused-IBN	75.3%	0.52 (1.06x)	1.32 (1.35x)
IBN-only NAHAS multi-trial	77.4%	0.52 (1.06x)	1.50 (1.53x)
[illegible]	[illegible]	[illegible]	[illegible]
IBN-only NANAS oneshot	76.8%	**0.49**	**0.98**
EfficientNet-B3 (Tan & Le, 2019) wo SE/Swish	78.8%	0.72 (1.12x)	2.28 (1.56x)
Manual-EdgeTPU-medium	77.2%	**0.62**	2.72 (1.86x)
MobilenetV3 w SE	76.8%	1.44 (2.32x)	4.00 (2.74x)
Fixed accelerator multi-trial w fused-IBN	78.2%	0.74 (1.19x)	1.75 (1.20x)
NAHAS multi-trial w fused-IBN	**79.5%**	0.72 (1.12x)	**1.46**

Rethinking Co-design of Neural Architectures and Hardware Accelerators, arXiv 2021.02

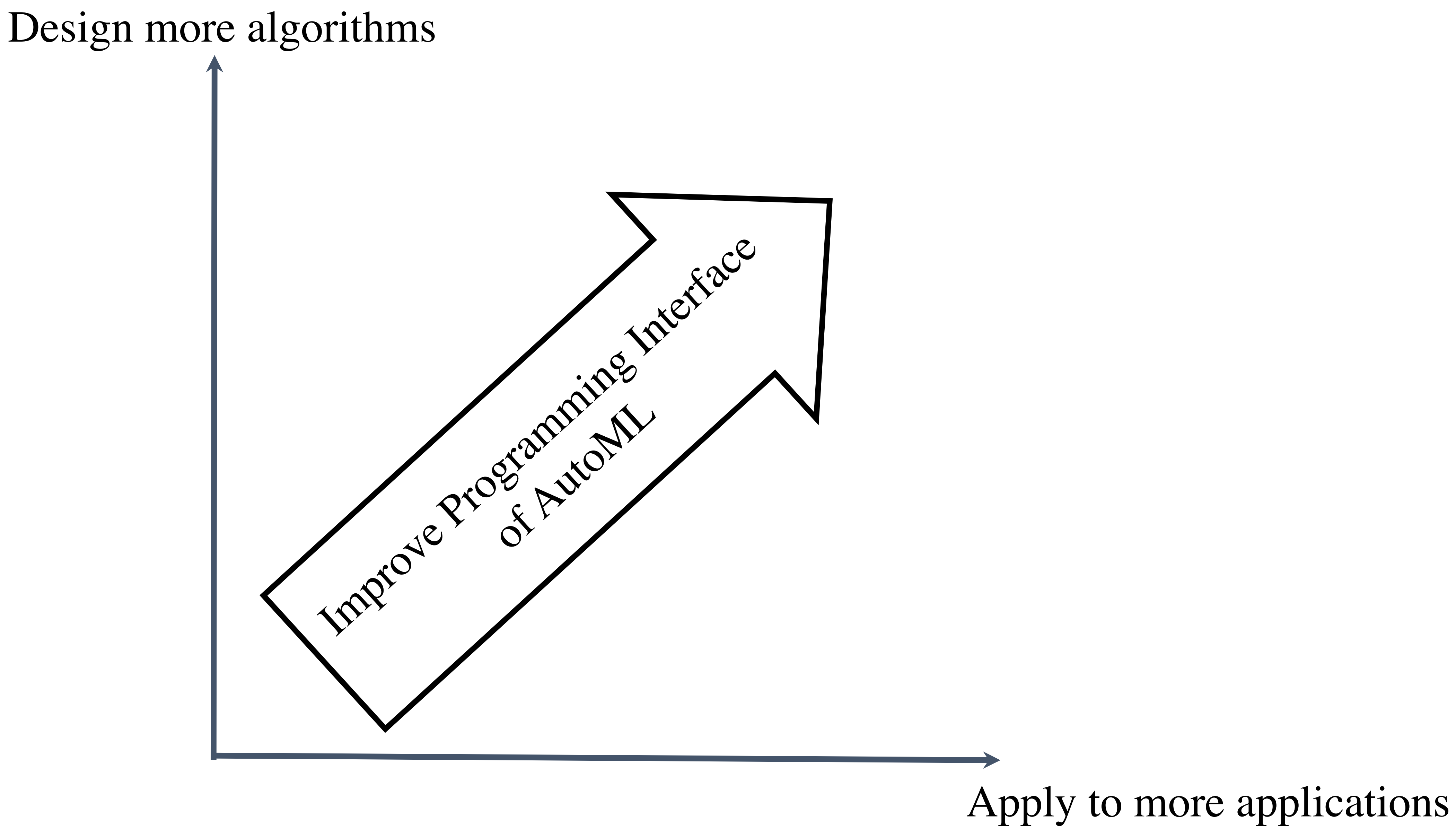
Co-design Improves > 1% ImageNet Accuracy



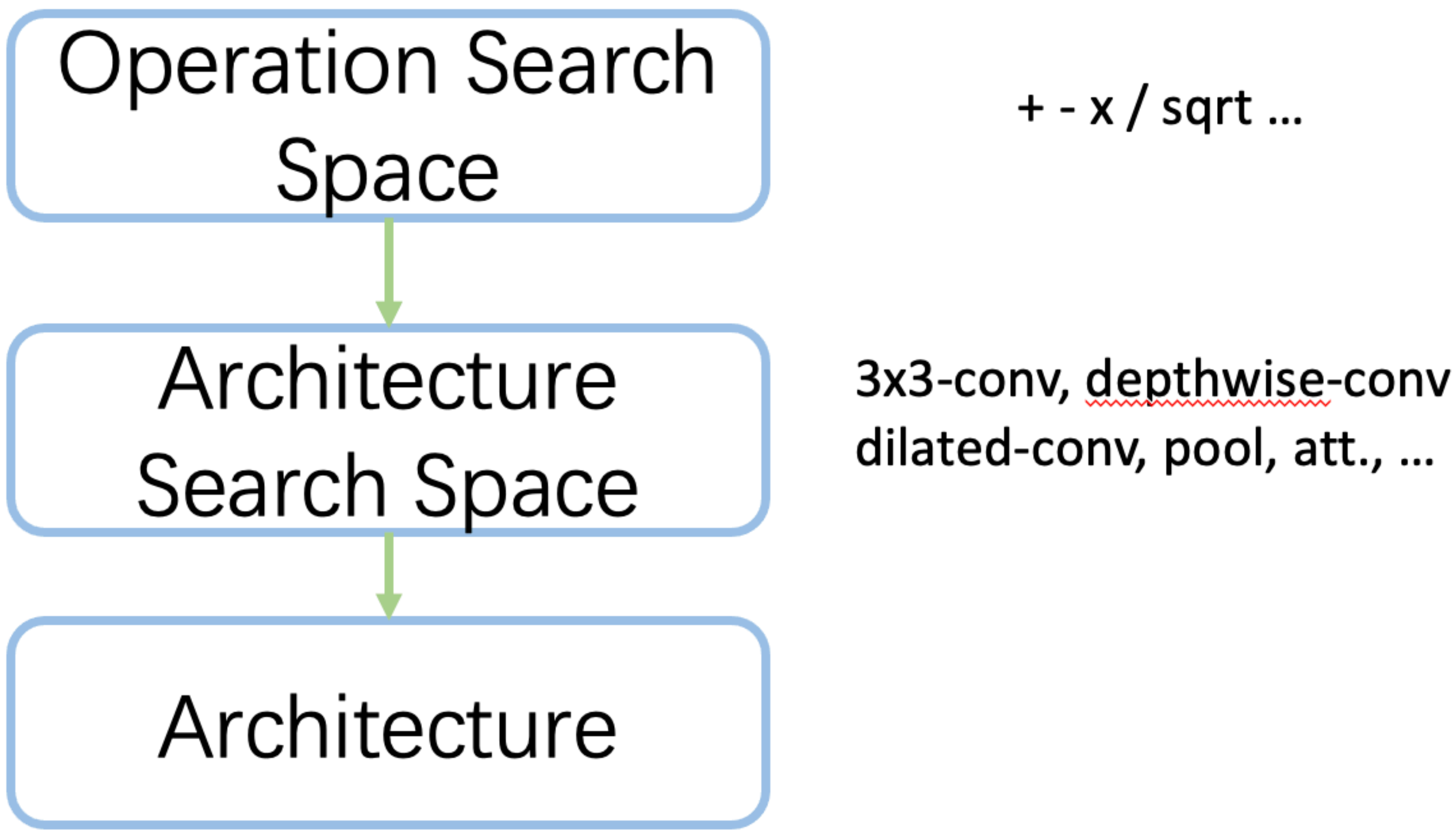
Rethinking Co-design of Neural Architectures and Hardware Accelerators, arXiv 2021.02

AutoML: System Design

Target: Scale AutoML horizontally and vertically

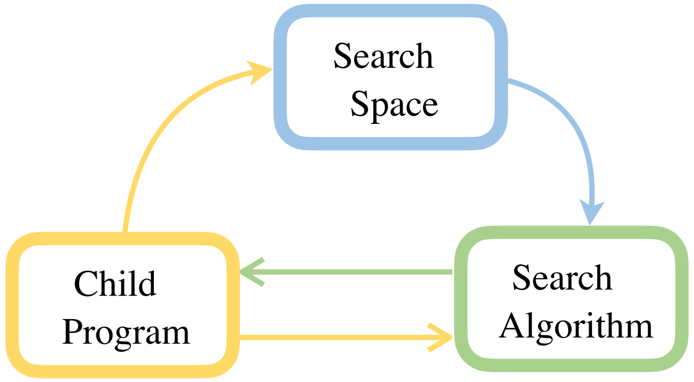


Example 0: Triple-level Search



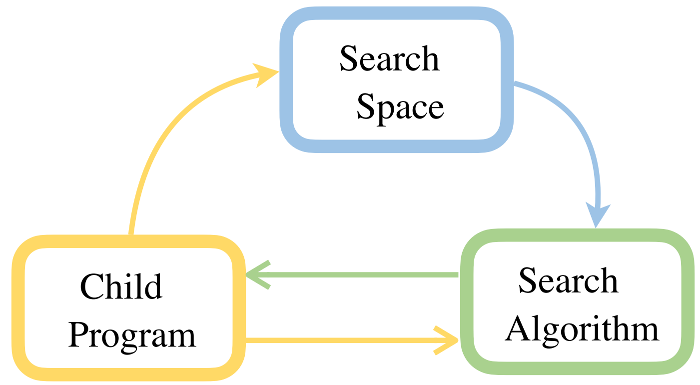
PyGlove: Symbolic Programming for Automated Machine Learning, NeurIPS 2020

Example 1: Coupling between CP and SS




```
class ResidualBlock:
  def call(self, inputs):
    op = Conv(...)
    return Add(
      inputs, op(inputs))
```

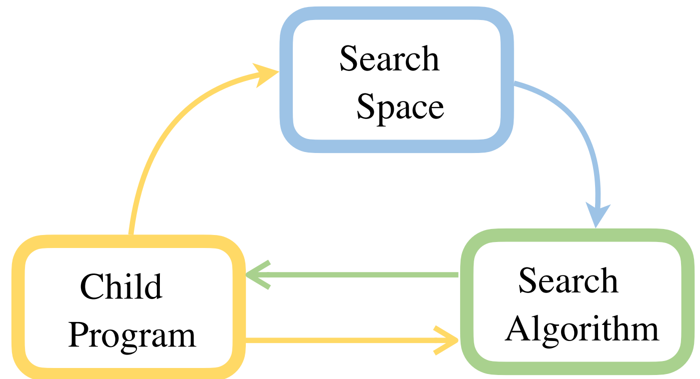
Example 1: Coupling between CP and SS




```
class ResidualBlock:
  def call(self, inputs):
    op = Conv(...)
    return Add(
      inputs, op(inputs))
```

```
class SearchableResidualBlock:
  def call(self, inputs, hps):
    if hps.op_type == 'conv':
       op = Conv(...)
    elif hps.op_type == 'dense':
       op = Dense(...)
    elif ...
    return Add(inputs, op(inputs))
```

Example 2: Coupling in Efficient NAS



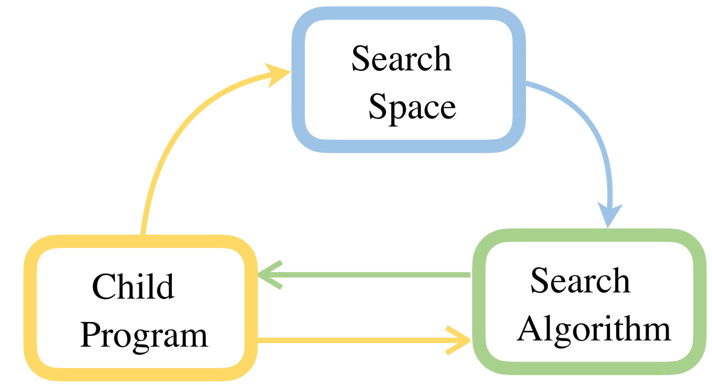

```
class MobileModel(object):

  def call(self, inputs):
    …
    layer = MBConv()
     ...
    return Sequential(
              [layer, ...])
```

ENAS

```
class EnasMobileModel(object):

  def call(self, inputs, hps):
    …
    layer = Switch([
      MBConv((3, 3), 3)
      MBConv((5, 5), 3)
      MBConv((7, 7), 3)
        ...
    ], selected=hps.op_choice0)
      ...
    return Sequential([layer, ...])
```

PyGlove: Symbolic Programming for Automated Machine Learning, NeurIPS 2020

Example 2: Coupling in Efficient NAS




```
class MobileModel(object):

  def call(self, inputs):
    …
    layer = MBConv()
     ...
    return Sequential(
              [layer, ...])
```

DARTS

```
class EnasMobileModel(object):
```

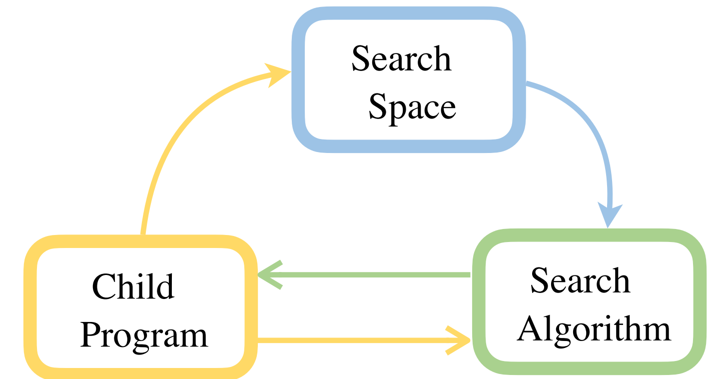
```
class DartsMobileModel(object):

  def call(self, inputs, hps):
     …
    layer = WeightedSum([
      MBConv((3, 3), 3)
      MBConv((5, 5), 3)
      MBConv((7, 7), 3)
        ...
    ], weights=hps.op_weights0)
      ...
    return Sequential(
        [layer, ...])
```

))

...])

PyGlove: Symbolic Programming for Automated Machine Learning, NeurIPS 2020

Example 2: Coupling in Efficient NAS




```
class MobileModel(object):

  def call(self, inputs):
     …
    layer = MBConv()
     ...
    return Sequential(
               [layer, ...])
```

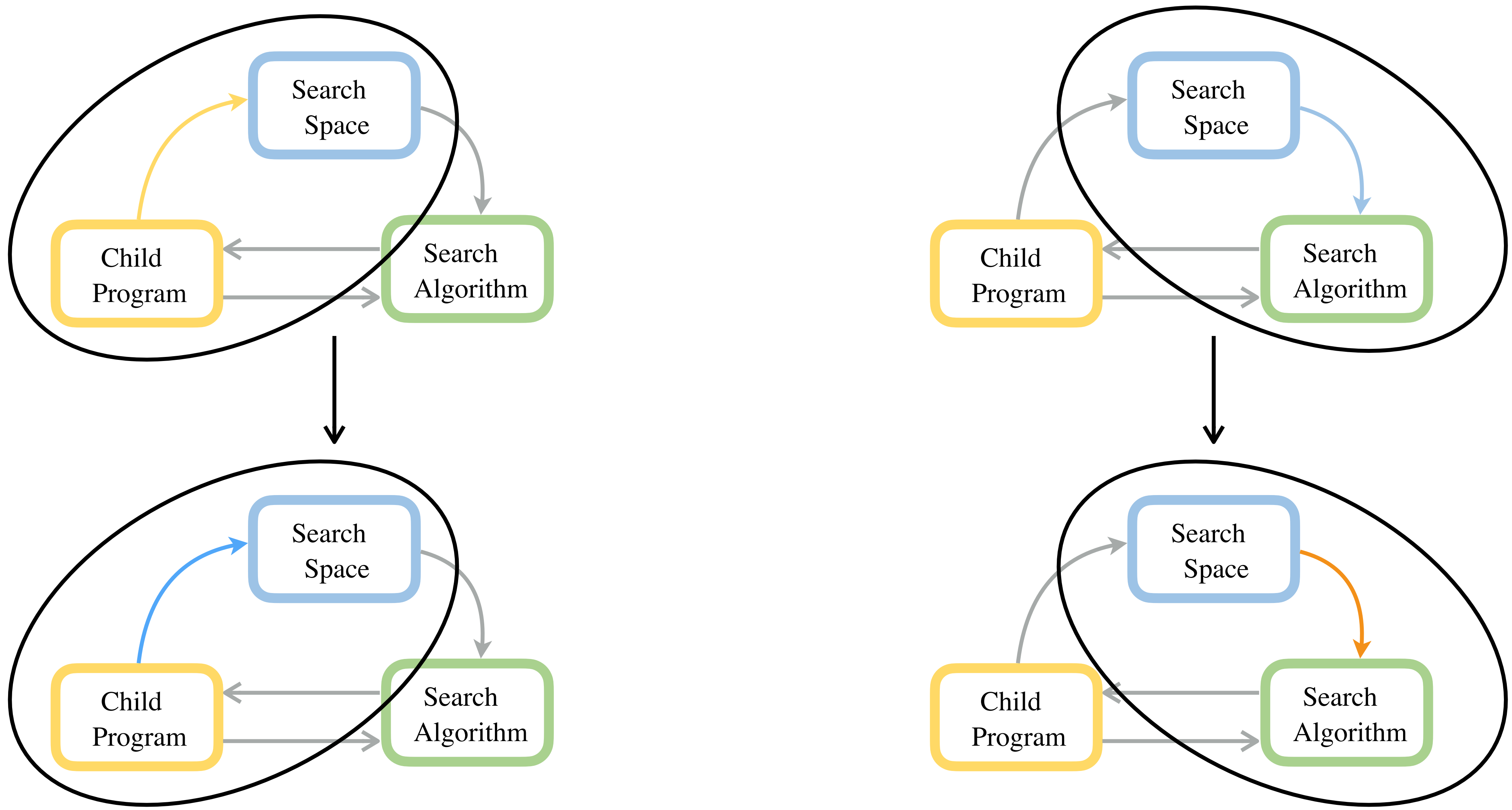
DARTS

```
class EnasMobileModel(object):
```

```
class DartsMobileModel(object):

  def call(self, inputs, hps):
     …
    layer = WeightedSum([
      MBConv((3, 3), 3)
      MBConv((5, 5), 3)
      MBConv((7, 7), 3)
        ...
    ], weights=hps.op_weights0)
      ...
    return Sequential(
        [layer, ...])
```

PyGlove: Symbolic Programming for Automated Machine Learning, NeurIPS 2020

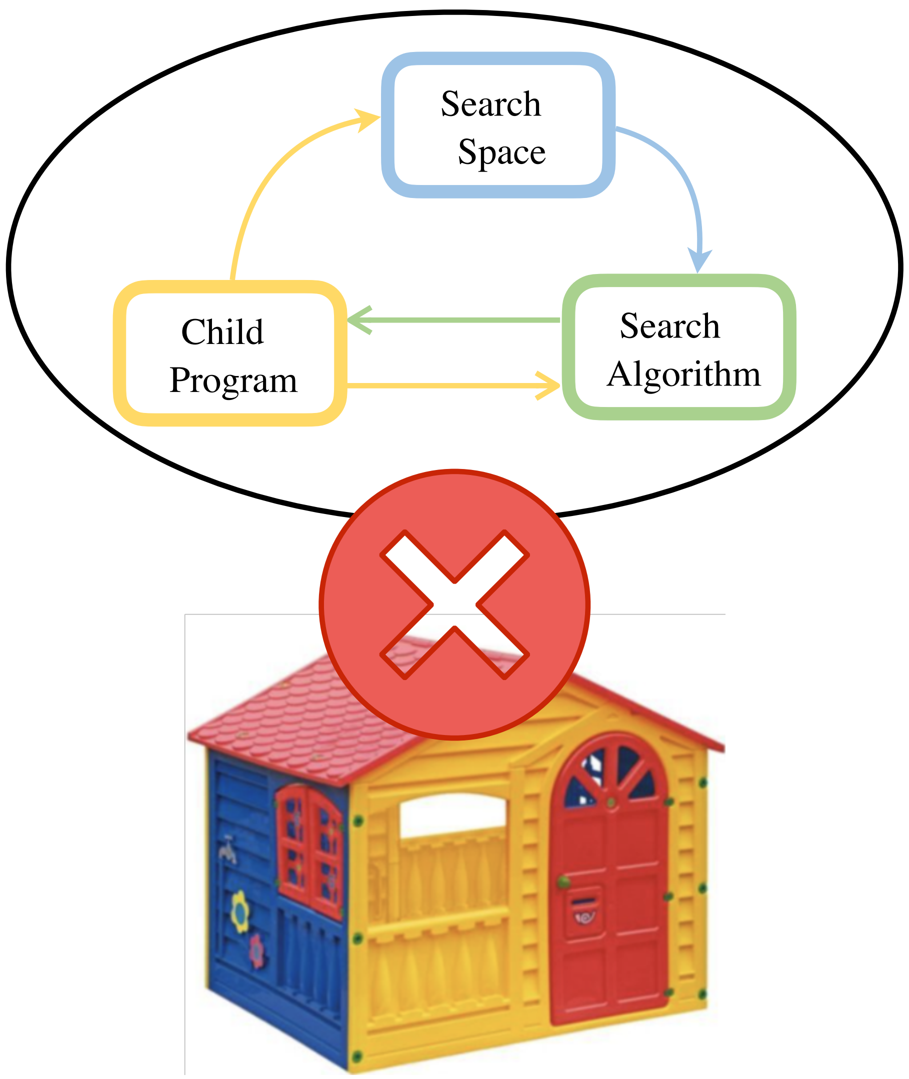
The Fluidity of Couplings



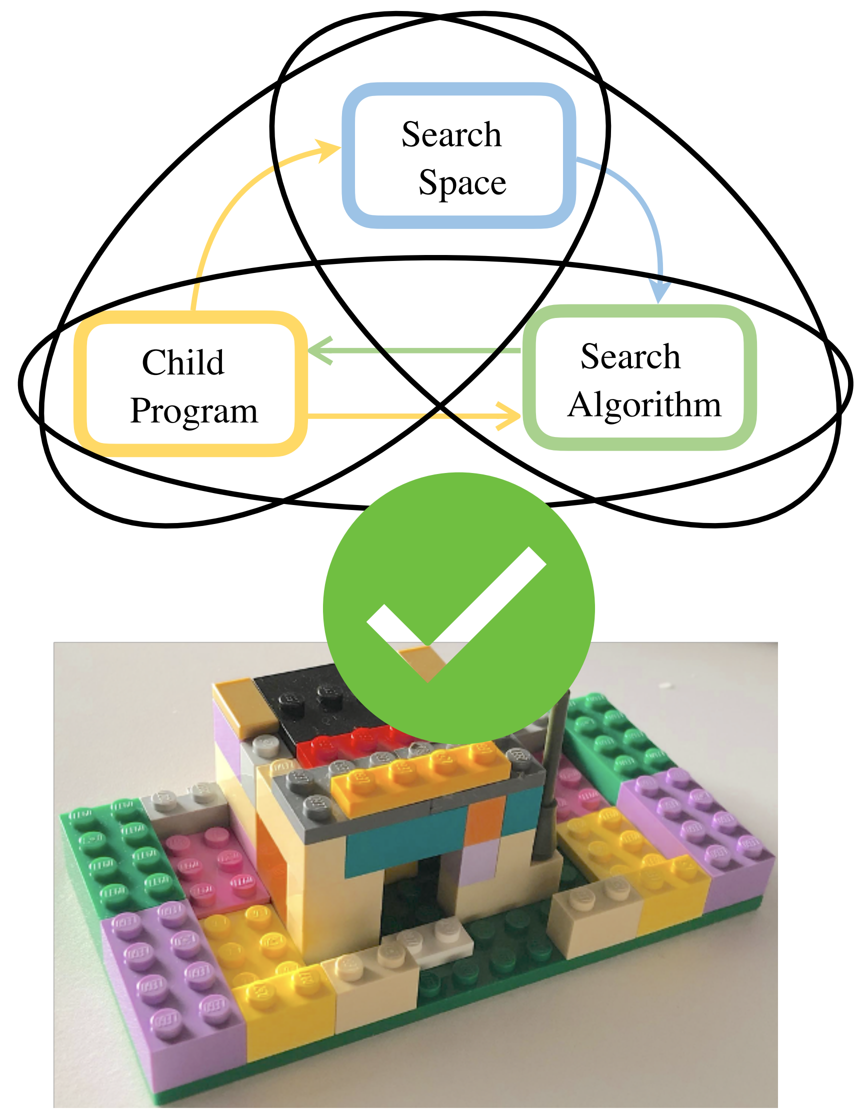
Change Search Space

Change Search Algorithm

What if?



Fixed Coupling

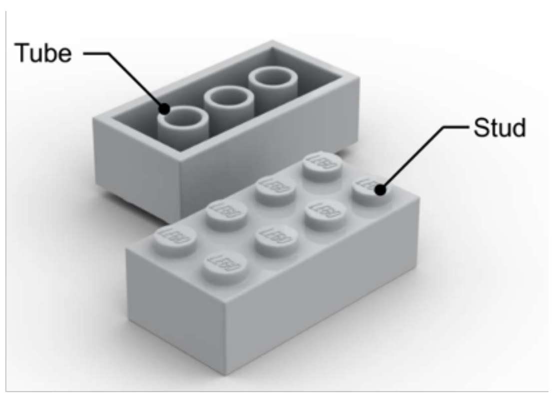


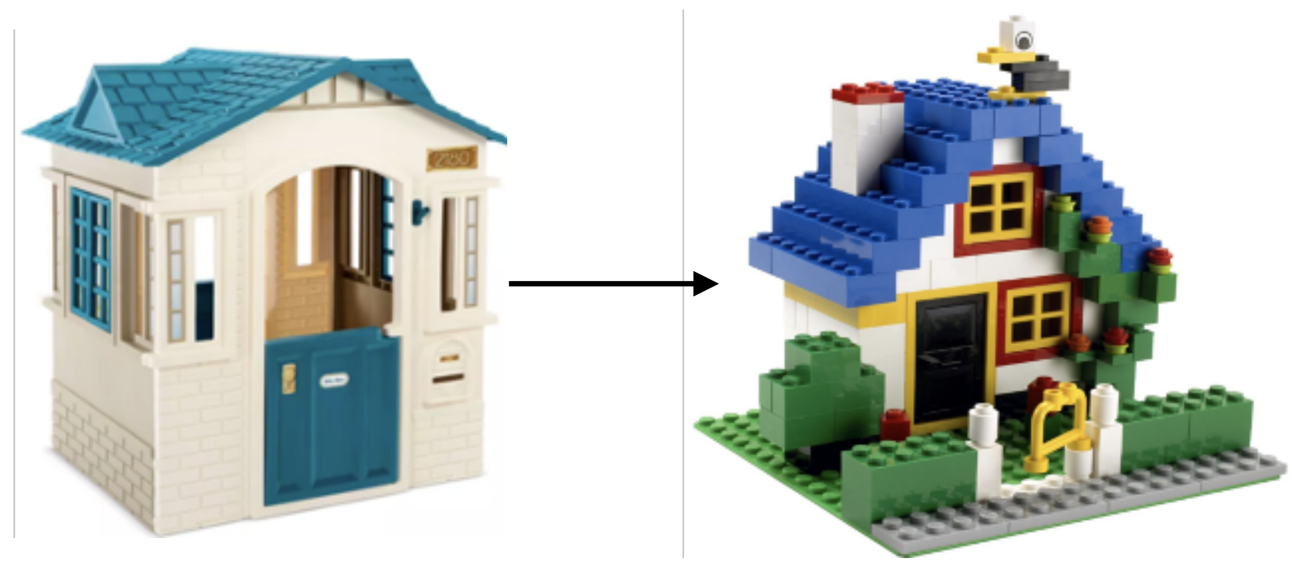
Dynamic Coupling

PyGlove: Symbolic Programming for Automated Machine Learning, NeurIPS 2020

Symbolic Programming for AutoML

Simple and unified interfaces





Fixed

Symbolic

PyGlove: Symbolic Programming for Automated Machine Learning, NeurIPS 2020

Symbolize: Make regular program symbolically programmable

```
Conv = symbolize(
    tf.keras.layers.Conv2D)
```

Symbolize existing classes

```
@symbolize
class Trainer(object):
  def __init__(
    self, model, optimizer):
     ...
  def train(self):
    return trainer_impl(
      Self.optimizer,
self.model)
```

Symbolize new classes

Hyper-parameters are like the studs of a LEGO brick

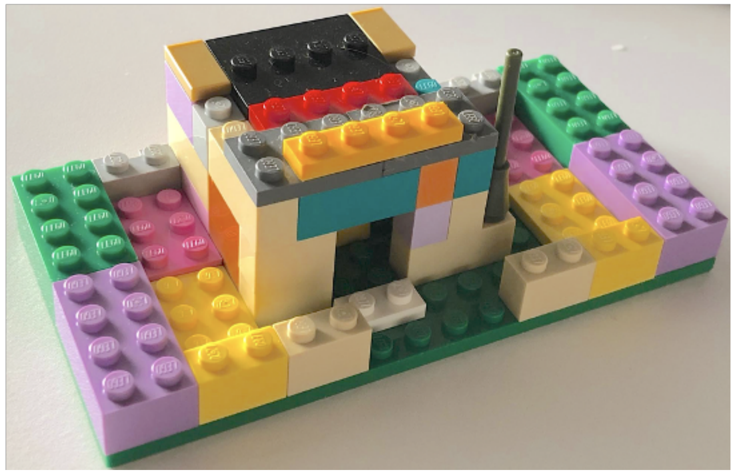
PyGlove: Symbolic Programming for Automated Machine Learning, NeurIPS 2020

Symbolic objects are mutable

```
Trainer(
  model=Stacked(
    op=Conv(8, (3, 3)),
    repeats=2),
  optimizer=Adam(2e-2)
)
```

```
Trainer(
  model=Stacked(
    op=MaxPool((3, 3)),
    repeats=2),
  optimizer=Adam(2e-2)
)
```

Program parts are not only compositional, but also can be modified programmatically.



Programming interfaces are provided for symbolic manipulation

```
def swap(k, v, parent):
  if isinstance(v, Conv):
    return MaxPool(v.kernel)

trainer.clone().rebind(swap)
```

Clone trainer and replace all the Conv layers into MaxPools

From Static Program to Search Space

```
Trainer(
  model=Stacked(
    op=Conv(8, (3, 3)),
    repeats=3),
  optimizer=Adam(2e-2)
)
```

Static Child Program

```
Trainer(
  model=Stacked(
    op=oneof([
      Identity(),
      MaxPool((3, 3)),
      Conv(oneof([4, 8]), (3, 3))]),
    repeats=3)),
  optimizer=
    oneof([
      Adam(2e-2),
      RMSProp(floatv(1e-6, 1e-3))])
)
```

Search Space

From Static Program to Search Space

```
trainer = Trainer(
  model=Stacked(
    op=Conv(8, (3, 3)),
    repeats=3),
  optimizer=Adam(2e-2)
)
```

```
hyper_trainer = Trainer(
  model=Stacked(
    op=oneof([
      Identity(),
      MaxPool((3, 3)),
      Conv(oneof([4, 8]), (3, 3))]),
    repeats=3)),
  optimizer=
    oneof([
      Adam(2e-2),
      RMSProp(floatv(1e-6, 1e-3))])
)
```

Static Child Program

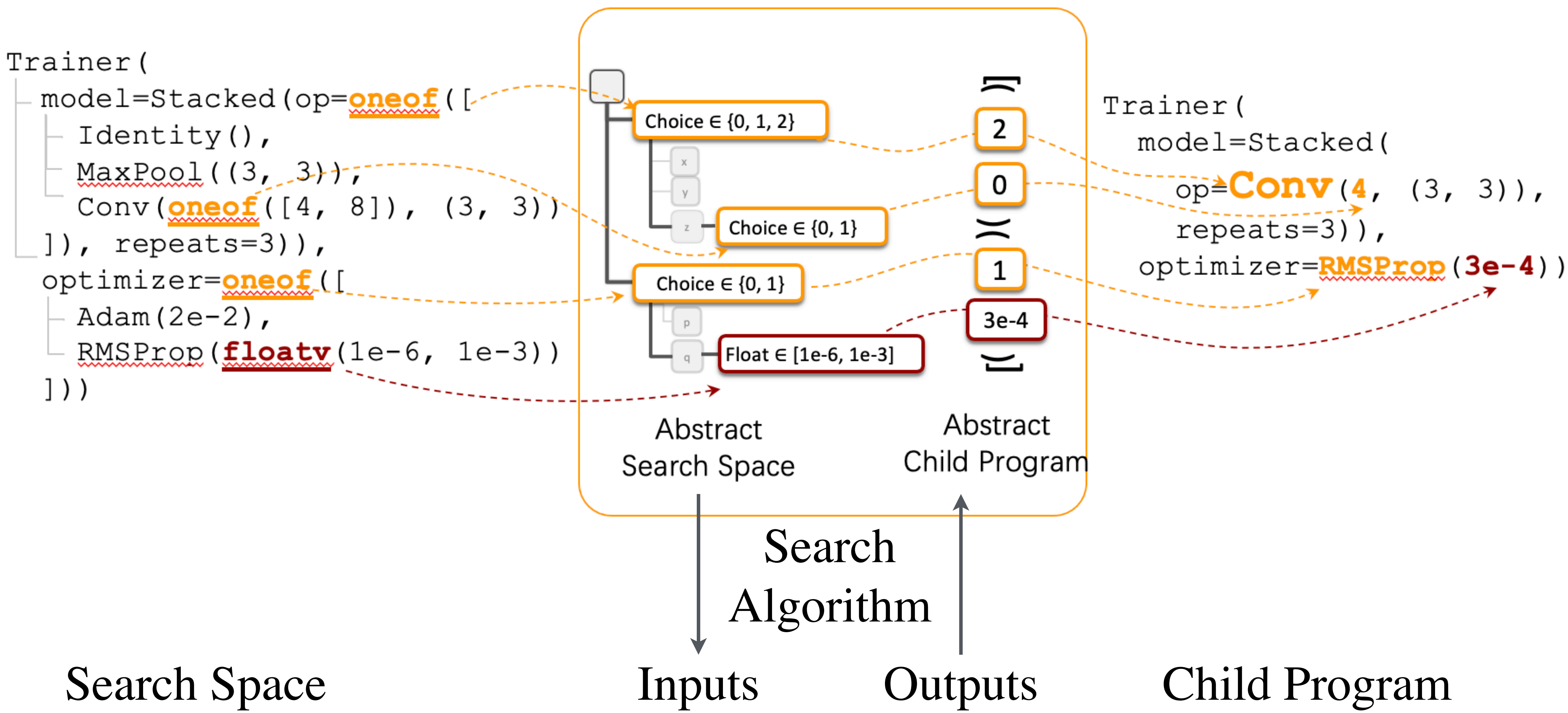
Search Space

Search Expressed as a For-loop

```
for trainer, feedback in sample(
    search_space=hyper_trainer,
    algorithm=PPO()):
  reward = trainer.train()
  feedback(reward)
```

Search as a feedback loop with sampled child programs

How Sample Works?



PyGlove: Symbolic Programming for Automated Machine Learning, NeurIPS 2020

AutoML: System Design

3 Search Spaces:
- *S1*: Search the kernel size & expansion factor of the inverted bottleneck units in MobileNetV2
- *S2*: Search the output filters of the inverted bottlenecks units in MobileNetV2
- *S3*: *S1* + *S2*

3 Search Algorithms:
- *RS: Random Search*
- *Bayesian: Bayesian Optimisation*
- *TuNAS: Efficient Search Algorithm*

#	Search space	Search algorithm	**Lines of codes**	Search cost	Train cost	Test accuracy	# of MAdds
1	$(static)$	N/A	N/A	N/A	1	73.1	300M
2	$(static) \rightarrow \mathcal{S}_1$	RS	**+23**	25	1	73.7 ($\uparrow$ 0.6)	299M
3	$\mathcal{S}_1$	RS $\rightarrow$ Bayesian	**+1**	25	1	73.9 ($\uparrow$ 0.8)	305M
4	$\mathcal{S}_1$	Bayesian $\rightarrow$ TuNAS	**+1**	1	1	74.2 ($\uparrow$ 1.1)	301M
5	$(static) \rightarrow \mathcal{S}_2$	TuNAS	**+10**	1	1	73.3 ($\uparrow$ 0.2)	307M
6	$\mathcal{S}_1, \mathcal{S}_2 \rightarrow \mathcal{S}_3$	TuNAS	**+1**	2	1	73.8 ($\uparrow$ 0.7)	303M

PyGlove lets you use ~10 LoCs to switch between different spaces and algorithms

PyGlove: Symbolic Programming for Automated Machine Learning, NeurIPS 2020

Take Away

The Future of AutoDL：

– Symbolic Programming Based Infra

– Architecture -> Hyperparameter

– Architecture -> Hardware