PyGlove: Symbolic Programming for AutoML

Daiyi Peng, Xuanyi Dong, Esteban Real, Mingxing Tan, Yifeng Lu, Hanxiao Liu, Gabriel Bender, Adam Kraft, Chen Liang, Quoc V. Le
AutoML helps human experts in pushing the state-of-the-art results on ImageNet, with more accurate and more efficient model architectures (e.g., Tan & Le, ICML, 2019)
AutoML: the progress

Google AutoML Tables beats >90% of daily submissions for approximately the first two weeks of the Kaggle Days competition. (Cloud Next ’19)
AutoML: the process
AutoML: the process

E.g: Training a classification model on ImageNet
AutoML: the process

E.g: Training a classification model on ImageNet

E.g: A list of operations to try in a residual block of the model

Child Program

Search Algorithm

Search Space
AutoML: the process

E.g: Training a classification model on ImageNet

E.g: A list of operations to try in a residual block of the model

E.g: A reinforcement learning algorithm
**AutoML: the process**

E.g: A list of operations to try in a residual block of the model

- **Search Space**
- **Child Program**
- **Search Algorithm**

E.g: Training a classification model on ImageNet

E.g: A reinforcement learning algorithm

1. Make sampling decisions
2. Mutate the program based on the decisions
AutoML: the process

1. Make sampling decisions
2. Mutate the program based on the decisions

Search Space

Child Program

Search Algorithm

E.g: Training a classification model on ImageNet

E.g: A list of operations to try in a residual block of the model

E.g: A reinforcement learning algorithm
AutoML: programming challenges
Example 1: coupling between child program and search space

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

A class in the child program
Example 1: coupling between child program and search space

A class in the child program

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

A class that couples with a search space

```python
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 methods

```python
class MobileModel(object):
    def call(self, inputs):
        ...
        layer = MBConv()
        ...
        return Sequential([layer, ...])
```

A class in the child program
Example 2: coupling in Efficient NAS methods

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

A class in the child program
```

```
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, ...])

A class that couples with a search algorithm
```
Example 2: coupling in Efficient NAS methods

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

A class in the child program
```

```
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, ...])
```

```
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, ...])
```

A class in the child program

Search Space

Child Program

Search Algorithm
The fluidity of couplings

Change search space

Search space A with a blackbox search algorithm  
Search space B with a blackbox search algorithm

Change search algorithm

Blackbox search algorithm X on any search space  
Blackbox search algorithm Y on any search space
The fluidity of couplings

Efficient NAS algorithm Y on search space B

Change search space & search algorithm

Efficient NAS algorithm X on search space A
What if?

Fixed coupling
What if?

Fixed coupling

Dynamic coupling
Symbolic Programming for AutoML
Symbolize new classes

Symbolize: Make regular program symbolically programmable

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

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

Symbolize existing classes

Hyper-parameters are like the studs of a LEGO brick
Symbolic objects are mutable

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

Program parts are not only compositional, but also ...
Symbolic objects are mutable

Program parts are not only compositional, but also can be modified programmatically.
Programming interfaces are provided for symbolic manipulation

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

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

Clone trainer and replace all the Conv layers into MaxPools
From a static program to a search space

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))]),
)

A static child program

A search space
From a static program to a search space

A static child program

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

A search space

```python
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))]))
)```
From a static program to a search space

```python
class SearchableStacked:
    def call(self, inputs, hps):
        if hps.op == 'identity':
            op = Identity()
        elif hps.op == 'max_pool':
            op = MaxPool(...)
        elif hps.op == 'Conv':
            op = Conv(...)
        return [op] * hps.repeats
```

A class that blends the child program and the search space

```python
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))])
)
```

Search space via composition
Search expressed as a for-loop

```python
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
Search expressed as a for-loop

```python
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 (1)

```python
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**
How **sample** works (2)

```
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

Abstract Search Space

input

Search Algorithm
How **sample** works (3)

```
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**
- Choice \( \in \{0, 1, 2\} \)
- Float \( \in [1e-6, 1e-3] \)

**Abstract Search Space**

**Abstract Child Program**
How **sample** works (4)

```
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

Abstract Search Space

Abstract Child Program

Child Program

Search Algorithm
Supporting Efficient NAS using rewrite

```python
hyper_trainer = Trainer(
    model=MobileNet(
        layers=[
            oneof([MBConv((3, 3), 3),
                   MBConv((5, 5), 3),
                   MBConv((7, 7), 3),
                   ...
            ]),
            ...
        ]),

    ...

)

A regular search space
```
Supporting Efficient NAS using rewrite

hyper_trainer = Trainer(
    model=MobileNet(
        layers=[
            oneof([
                MBConv((3, 3), 3),
                MBConv((5, 5), 3),
                MBConv((7, 7), 3),
                ...
            ]),
            ...
        ]),
        ...
    )
)

super_trainer = SuperNetworkTrainer(
    model=MobileNet(
        layers=[
            Switch([
                MBConv((3, 3), 3),
                MBConv((5, 5), 3),
                MBConv((7, 7), 3),
                ...
            ], index=algo.next_decision()),
            ...
        ]),
        ...
    )
)

A regular search space

A super-program required by the efficient NAS algorithm
Supporting Efficient NAS using rewrite

```python
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, ...])
```

```python
super_trainer = SuperNetworkTrainer(model=MobileNet(layers=[
    Switch([MBConv((3, 3), 3), MBConv((5, 5), 3), MBConv((7, 7), 3)],
    selected=hps.op_choice0),
    ...]
), index=algo.next_decision(),
    ...]
), ...
```

A class that blends the child program, search space and search algorithm together

A super-program required by the efficient NAS algorithm
Expressing complex search spaces (1)

NAS-Bench-101
Expressing complex search spaces (1)

\[ C(N, 2) = \frac{N \times (N - 1)}{2} \text{ edge positions} \]

NAS-Bench-101
Expressing complex search spaces (1)

\[
\text{ModelSpec}(
\text{nodes}=[\text{oneof}(\text{range}(K))]^N, \\
\text{edges}=[\text{oneof}([0, 1])]^N \cdot (N-1)/2)
\]

\[
C(N, 2) = N \cdot (N - 1) / 2 \text{ edge positions}
\]

Independently sampling \(C(N, 2)\) edge positions results in \(\text{NAS-Bench-101}\)
Expressing complex search spaces (2)

The i’th FPN Node

NAS-FPN: a FPN (Feature Pyramid) node
Expressing complex search spaces (2)

The i’th FPN node

NAS-FPN: a FPN (Feature Pyramid) node
Expressing complex search spaces (2)

FpnNode(
    type=oneof(['sum', 'attention']),
    level=3,
    input_offsets=manyof(
        2, range(NUM_PRE_NODES),
        distinct=True,
        sorted=True)
)

Multiple choices with constraints.

NAS-FPN: a FPN (Feature Pyramid) node
Expressing complex search spaces

TuNAS: A residual layer
Expressing complex search spaces

Residual\(\text{oneof}(\text{InvertedBottleneck}(\text{filters=oneof}([32, 48, 64]), \text{kernel=oneof}([3, 5, 7]), \text{expansion=oneof}([3, 6])), \text{Zero()})))\)

TuNAS: A residual layer
Expressing complex search spaces

InvertedBottleneck

Residual

Residual(oneof(
    InvertedBottleneck(
        filters=oneof([32, 48, 64]),
        kernel=oneof([3, 5, 7]),
        expansion=oneof([3, 6]),
        Zero()]))

TuNAS: A residual layer

Downgrades the Residual to Identity when selected

Active only when the InvertedBottleneck is selected
Create search space programmatically

```python
def relax_filters(k, v, parent):
    if isinstance(parent, Conv):
        if k == 'filters':
            return oneof([v//2, v, v*2])
    return v

hyper_trainer = trainer.clone().rebind(relax_filters)
```

Creating an architectural search space by relaxing the filters of all Conv layers in current model into a set of options.
Create search space programmatically

```python
def relax_filters(k, v, parent):
    if isinstance(parent, Conv):
        if k == 'filters':
            return oneof([v//2, v, v*2])
    return v

hyper_trainer = trainer.clone().rebind(relax_filters)
```

Creating an architectural search space by relaxing the filters of all Conv layers in current model into a set of options.
Switch between search algorithms

```python
for trainer, feedback in sample(
    search_space=hyper_trainer,
    algorithm=RandomSearch()):
    reward = trainer.train()
    feedback(reward)
```
Switch between search algorithms

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

RS → Bayesian

for trainer, feedback in sample(search_space=hyper_trainer, algorithm=Bayesian()):
    reward = trainer.train()
    feedback(reward)
Switch between search algorithms

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

RS → Bayesian
```

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

Bayesian → TuNAS

```python
tunas.rewrite(hyper_trainer, Reinfonce()).search()
```
Exploring 3 search spaces and 3 search algorithms

<table>
<thead>
<tr>
<th>#</th>
<th>Search space</th>
<th>Search algorithm</th>
<th>Lines of codes</th>
<th>Search cost</th>
<th>Train cost</th>
<th>Test accuracy</th>
<th># of MAdds</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(static)</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>1</td>
<td>73.1</td>
<td>300M</td>
</tr>
<tr>
<td>2</td>
<td>(static) $\rightarrow S_1$</td>
<td>RS</td>
<td>+23</td>
<td>25</td>
<td>1</td>
<td>73.7 (↑ 0.6)</td>
<td>299M</td>
</tr>
<tr>
<td>3</td>
<td>$S_1$</td>
<td>RS $\rightarrow$ Bayesian</td>
<td>+1</td>
<td>25</td>
<td>1</td>
<td>73.9 (↑ 0.8)</td>
<td>305M</td>
</tr>
<tr>
<td>4</td>
<td>$S_1$</td>
<td>Bayesian $\rightarrow$ TuNAS</td>
<td>+1</td>
<td>1</td>
<td>1</td>
<td>74.2 (↑ 1.1)</td>
<td>301M</td>
</tr>
<tr>
<td>5</td>
<td>(static) $\rightarrow S_2$</td>
<td>TuNAS</td>
<td>+10</td>
<td>1</td>
<td>1</td>
<td>73.3 (↑ 0.2)</td>
<td>307M</td>
</tr>
<tr>
<td>6</td>
<td>$S_1, S_2 \rightarrow S_3$</td>
<td>TuNAS</td>
<td>+1</td>
<td>2</td>
<td>1</td>
<td>73.8 (↑ 0.7)</td>
<td>303M</td>
</tr>
</tbody>
</table>

It requires only a few lines of change to iterate on both search spaces and search algorithms.
Expressing complex search flows

```python
for trainer, feedback in sample(
    search_space=hyper_trainer,
    algorithm=PPO(),
    partition_fn=None):
    reward = trainer.train()
    feedback(reward)
```

Regular search flow
for trainer, feedback in sample(
    search_space=hyper_trainer,
    algorithm=PPO(),
    partition_fn=None):
    reward = trainer.train()
    feedback(reward)

Expressing complex search flows

<table>
<thead>
<tr>
<th>Search type</th>
<th>for-loop pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joint</td>
<td>for($x$, $f_x$) : ...</td>
</tr>
<tr>
<td>Separate</td>
<td>for($x_1$, $f_{x_1}$) : ...</td>
</tr>
<tr>
<td></td>
<td>for($x_2$, $f_{x_2}$) : ...</td>
</tr>
<tr>
<td>Factorized</td>
<td>for($x_1$, $f_{x_1}$) :</td>
</tr>
<tr>
<td></td>
<td>for($x_2$, $f_{x_2}$) : ...</td>
</tr>
</tbody>
</table>

Regular search flow

Complex search flow
Expressing complex search flows: An example

```python
def factorized_search(search_space):
    for edge_space, ops_feedback in pg.sample(
        search_space, RegularizedEvolution(),
        trials=300, partition_fn=lambda v: v.hints == OP_HINT):
        rewards = []
        for example, edges_feedback in pg.sample(
            edge_space, RegularizedEvolution(), trials=20):
            reward = nasbench.get_reward(example)
            edges_feedback(reward)
            rewards.append(reward)
        ops_feedback(top5_average(rewards))
```

Outer loop search for nodes, whose examples are sub-spaces for edges

Inner loop search for edges on fixed nodes

---

Factorized  
\[
\text{for} (x_1, f_{x_1}), \quad \text{for} (x_2, f_{x_2}) : \ldots
\]
Expressing complex search flows: An example

def factorized_search(search_space):
    for edge_space, ops_feedback in pg.sample(
        search_space, RegularizedEvolution(),
        trials=300, partition_fn=lambda v: v.hints == OP_HINT):
        rewards = []
        for example, edges_feedback in pg.sample(
            edge_space, RegularizedEvolution(), trials=20):
            reward = nasbench.get_reward(example)
            edges_feedback(reward)
            rewards.append(reward)
        ops_feedback(top5_average(rewards))

Allows the search algorithm of the outer loop to see only the node sub-space
Exploring 3 search flows on NAS-Bench-101

Search performances (mean) and their standard deviations with 500 runs.
AutoML needs Symbolic Programming
Thank you!
Thank you!