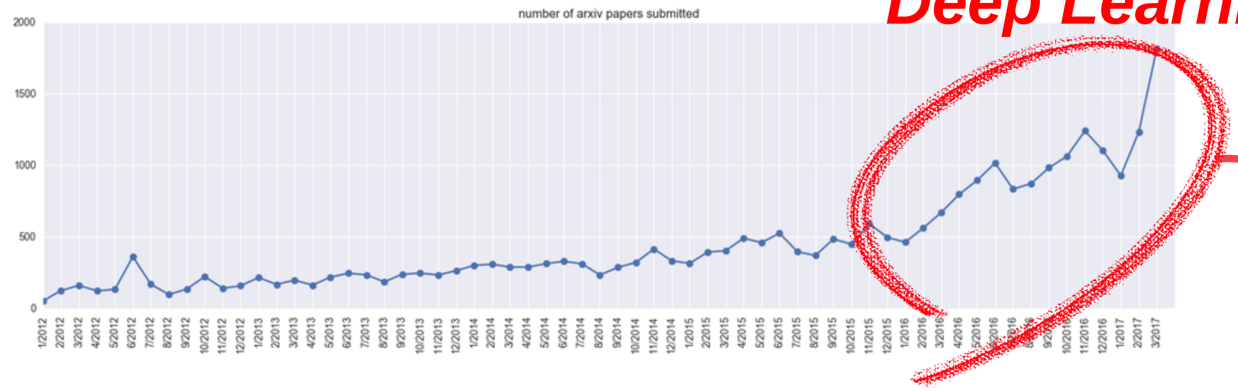
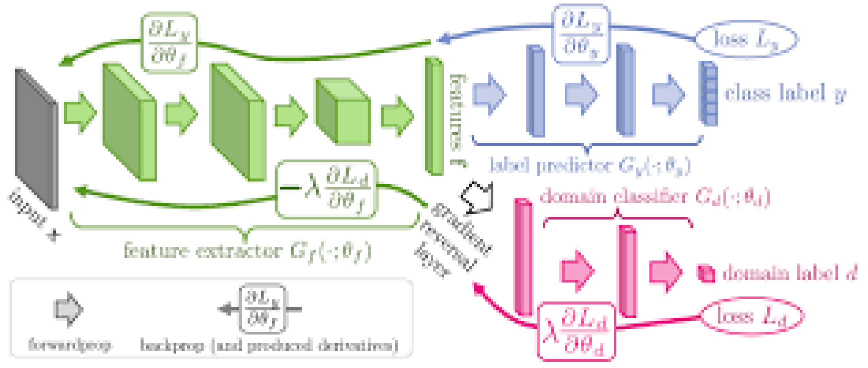
Neural Network Syntax Analyzer for Embedded Standardized Deep Learning

MyungJae Shin, Joongheon Kim*
Chung-Ang University
Aziz Mohaisen
University of Central Florida
Jaebok Park, Kyung Hee Lee
Electronics and Telecommunications Research Institute (ETRI)

Background



Deep Learning


$\frac{\partial L_y}{\partial \theta_f}$
$\frac{\partial L_y}{\partial \theta_y}$
loss L_y
class label y
features f
label predictor $G_y(\cdot;\theta_y)$
$-\lambda \frac{\partial L_d}{\partial \theta_f}$
domain classifier $G_d(\cdot;\theta_d)$
input x
feature extractor $G_f(\cdot;\theta_f)$
gradient reversal layer
domain label d
loss L_d
$\lambda \frac{\partial L_d}{\partial \theta_d}$
forwardprop
backprop (and produced derivatives)

Neural Network

TensorFlow™ PYTORCH Caffe2 K scikit learn

Background

- Neural network development platforms is diversified.
- Each platform forms a neural network in a different way.
- The different strength and weakness.

Which is best???

TensorFlow™ PYTORCH Caffe2 K scikit learn

Background

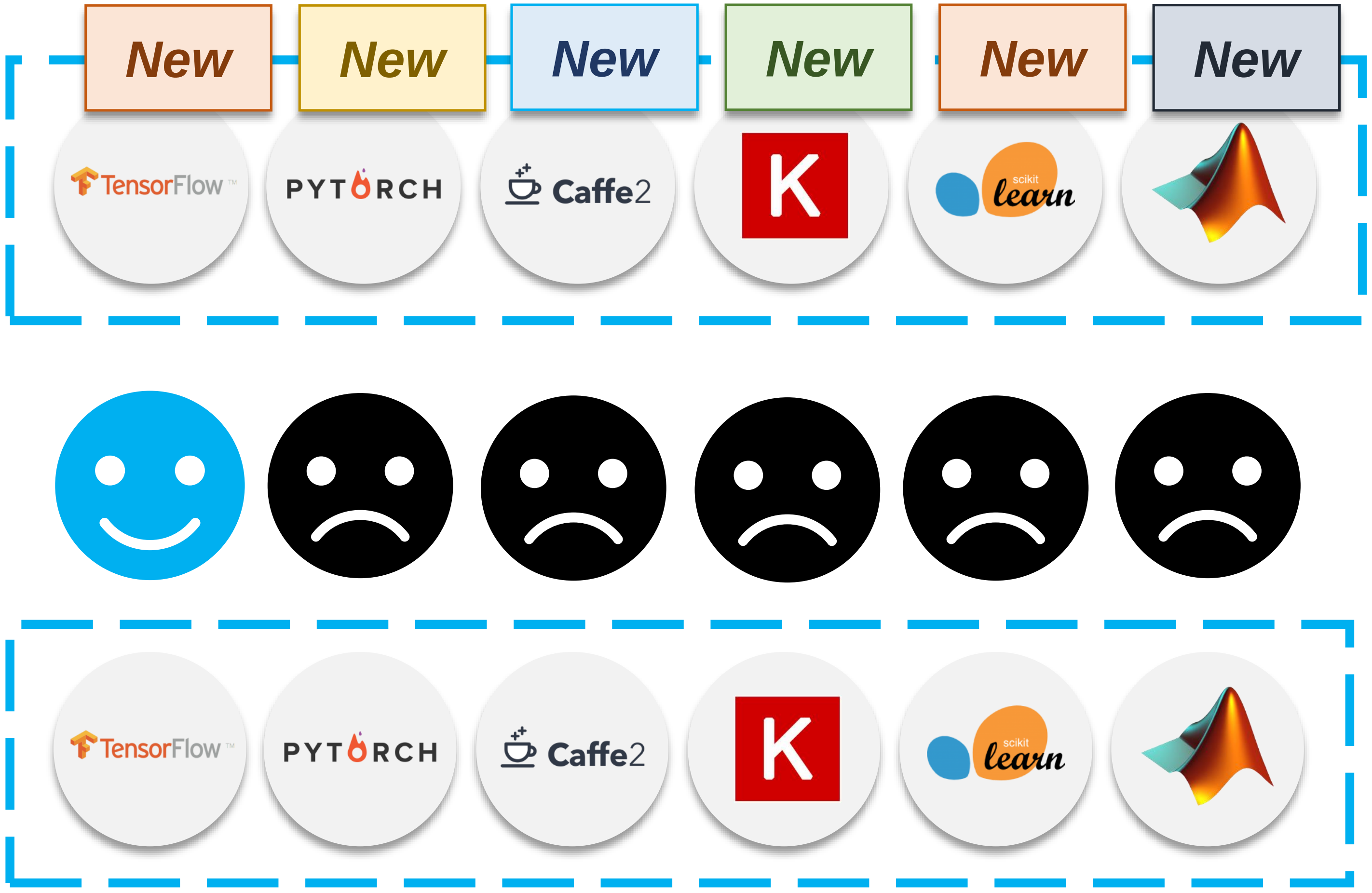
TensorFlow™

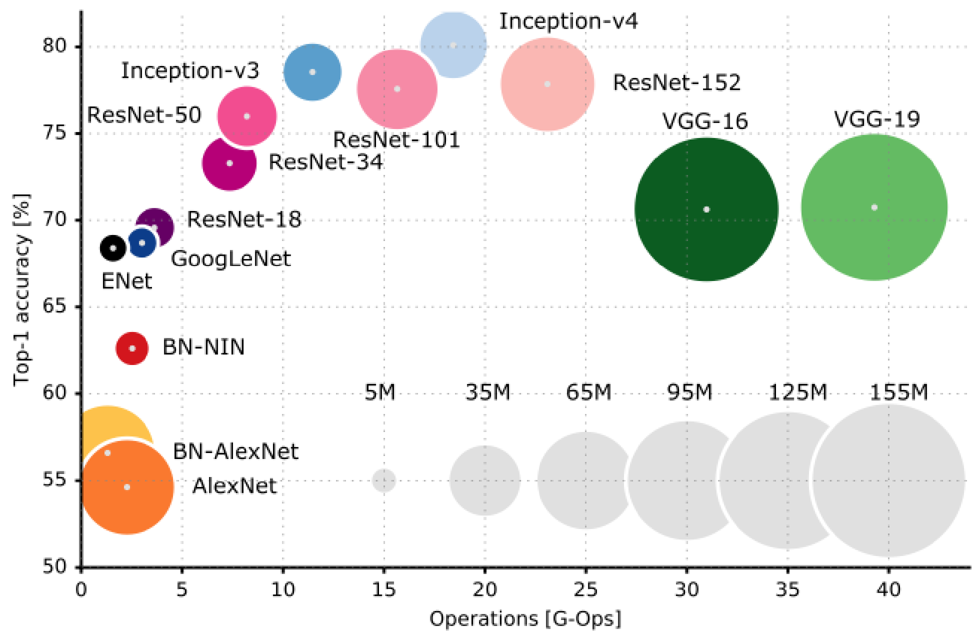
```
x = tf.placeholder( tf.float32 )
y = tf.placeholder( tf.float32 )
z = tf.placeholder( tf.float32 )
a = x * y
b = a + z
c = tf.reduce_sum(b)
with  tf.Session as sess:
values = {
            x: np.random.randn(3, 4)
            y: np.random.randn(3, 4)
            z: np.random.randn(3, 4)
}
```

PYTORCH

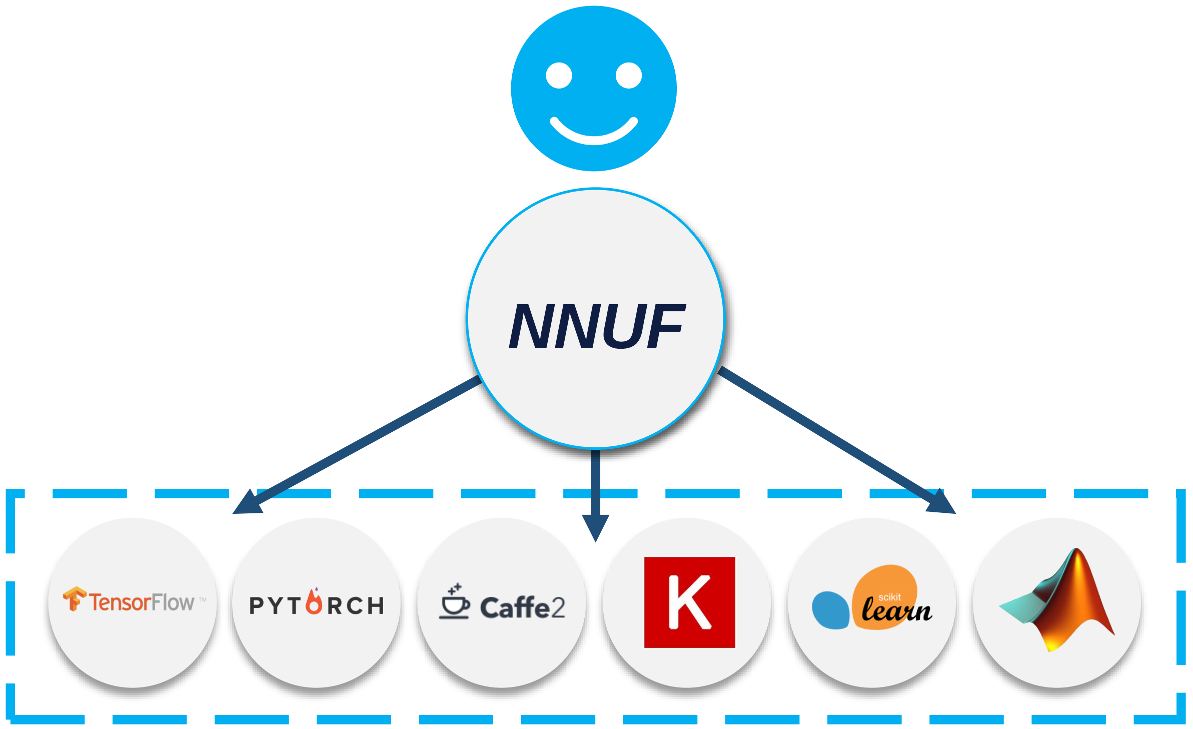
```
x = Variable( torch.randn(3,4).cuda(), requires_grad = True)
y = Variable( torch.randn(3,4).cuda(), requires_grad = True)
z = Variable( torch.randn(3,4).cuda(), requires_grad = True)
a = x * y
b = a + z
c = torch.sum(b)
```

Background


New
New
New
New
New
New


Inception-v4
Inception-v3
ResNet-152
ResNet-50
ResNet-101
ResNet-34
VGG-16
VGG-19
ResNet-18
GoogLeNet
ENet
BN-NIN
BN-AlexNet
AlexNet
5M
35M
65M
95M
125M
155M
Top-1 accuracy [%]
Operations [G-Ops]
80
75
70
65
60
55
50
0
5
10
15
20
25
30
35
40

MOTIVATION

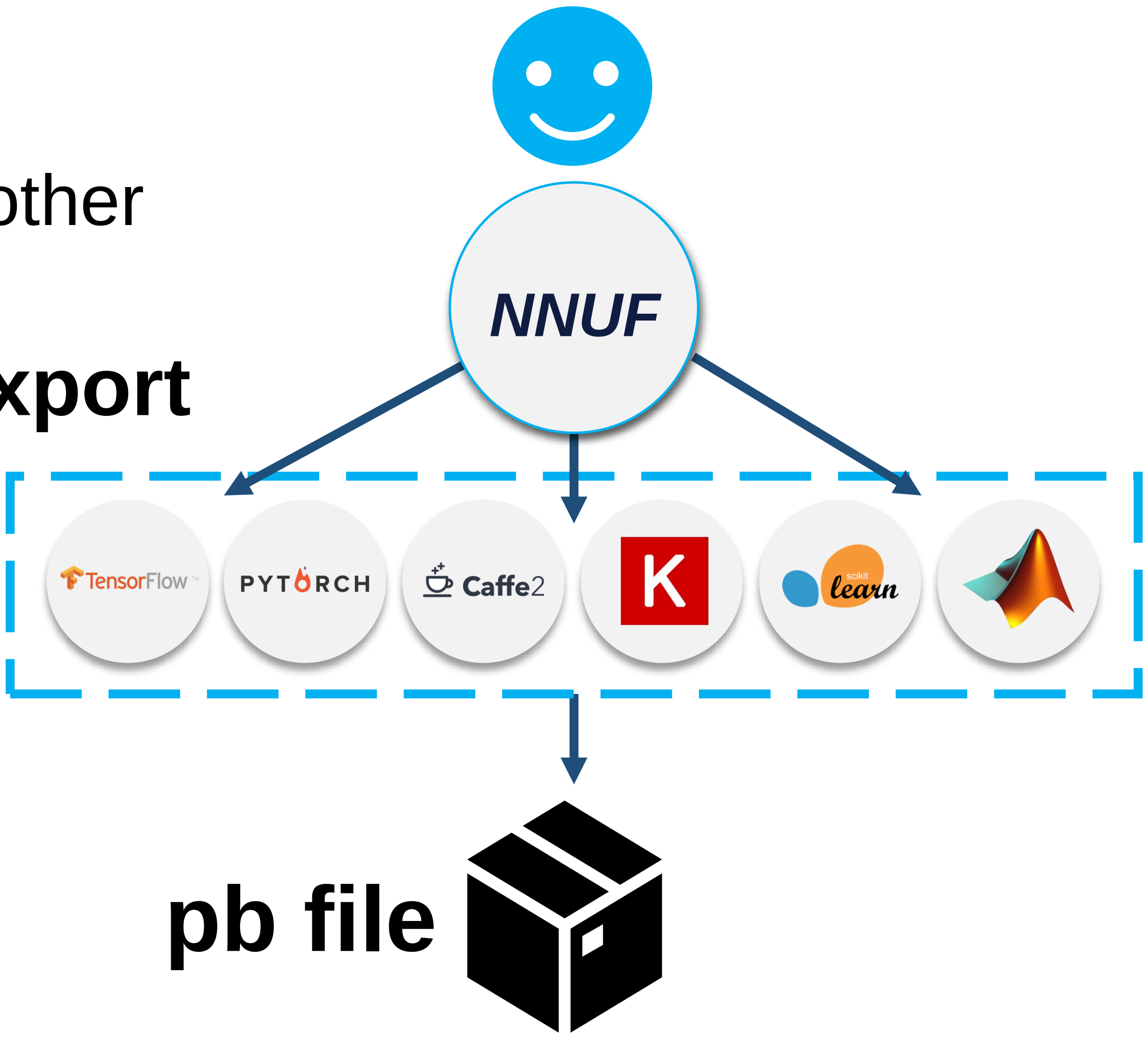


- **Standardized** deep learning computation
- **Platform independent** model configuration

NNEF OVERVIEW

The graph can be easily shared with other platform-dependent description

➔ **Protocol Buffer (Protobuf) export**



NNEF COMPONENT

Component 1

Define a NNUF term

```
// Neural network definition
graph CustomNet(inputVar) -> (output Var) {
    // Variables definition
    Input = reshape(inputVar, [-1, 28, 28, 1] );
    Kernel = variable(shape=[3, 3, 1, 64],
         label="conv1/kernel");
    Bias1 = variable(shape=[64], label="conv1/bias");
    // Operations definition
    conv1 = conv(input, filter=kernel1, strides=[1, 1,
         1, 1], padding="SAME");
    add1 = add(conv1, bias1);
    outputVar = relu(add1);
}
```

Component 2

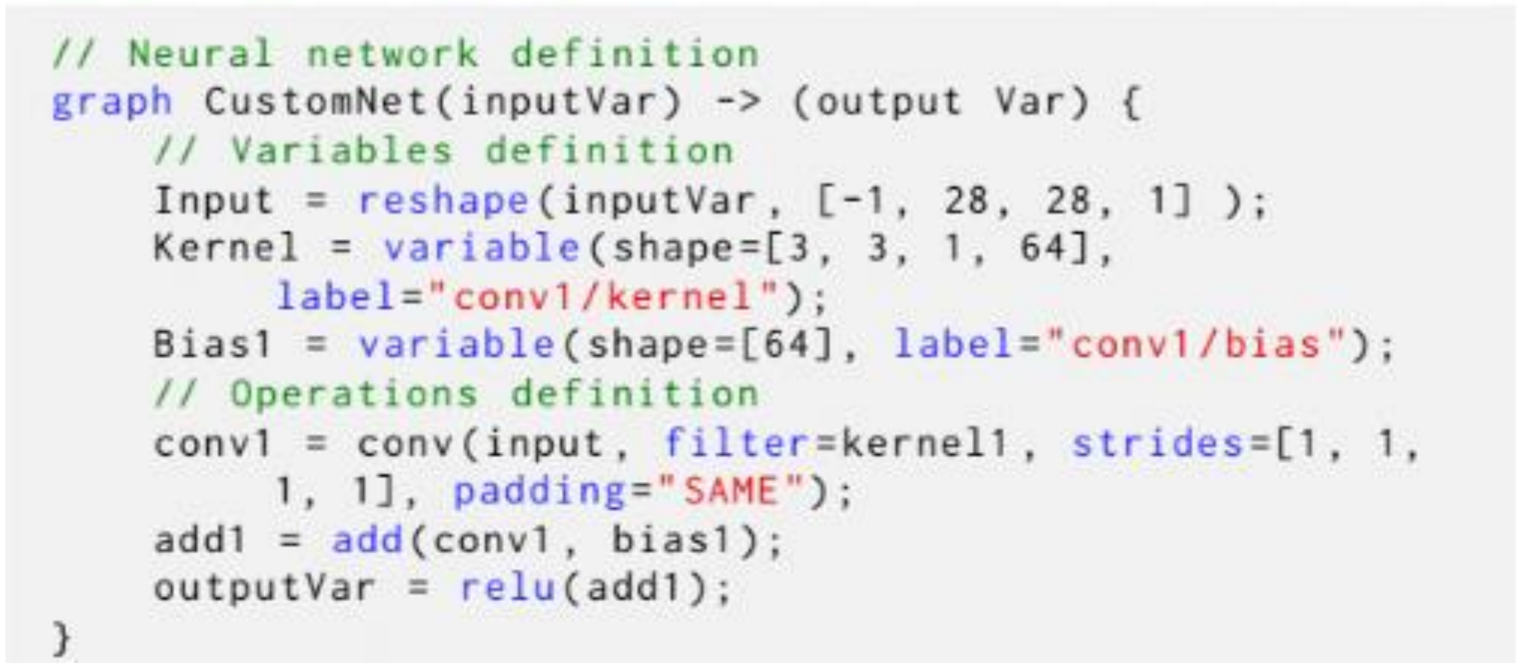
Design a parser

```
String argument():
{String arg, exp, name, res; Token id;}
{
  (
    ((id=<IDENTIFIER>)<ASSIGN>(exp=expression())){
      name = id.toString();
      switch(name) {
        case "label": res="name";arg+=res+"="+exp; break;
        case "filter": arg+=exp; break;
        case "size": res="shape="+exp; arg+=res; break;
        case "type": res="dtype=tf."+exp;arg+=res; break;
        default: arg+=name+"="+exp; break;
      }
    } | (exp = expression()){ arg += exp; }
  ) {return arg;}
}
```

NNUF PATTERN

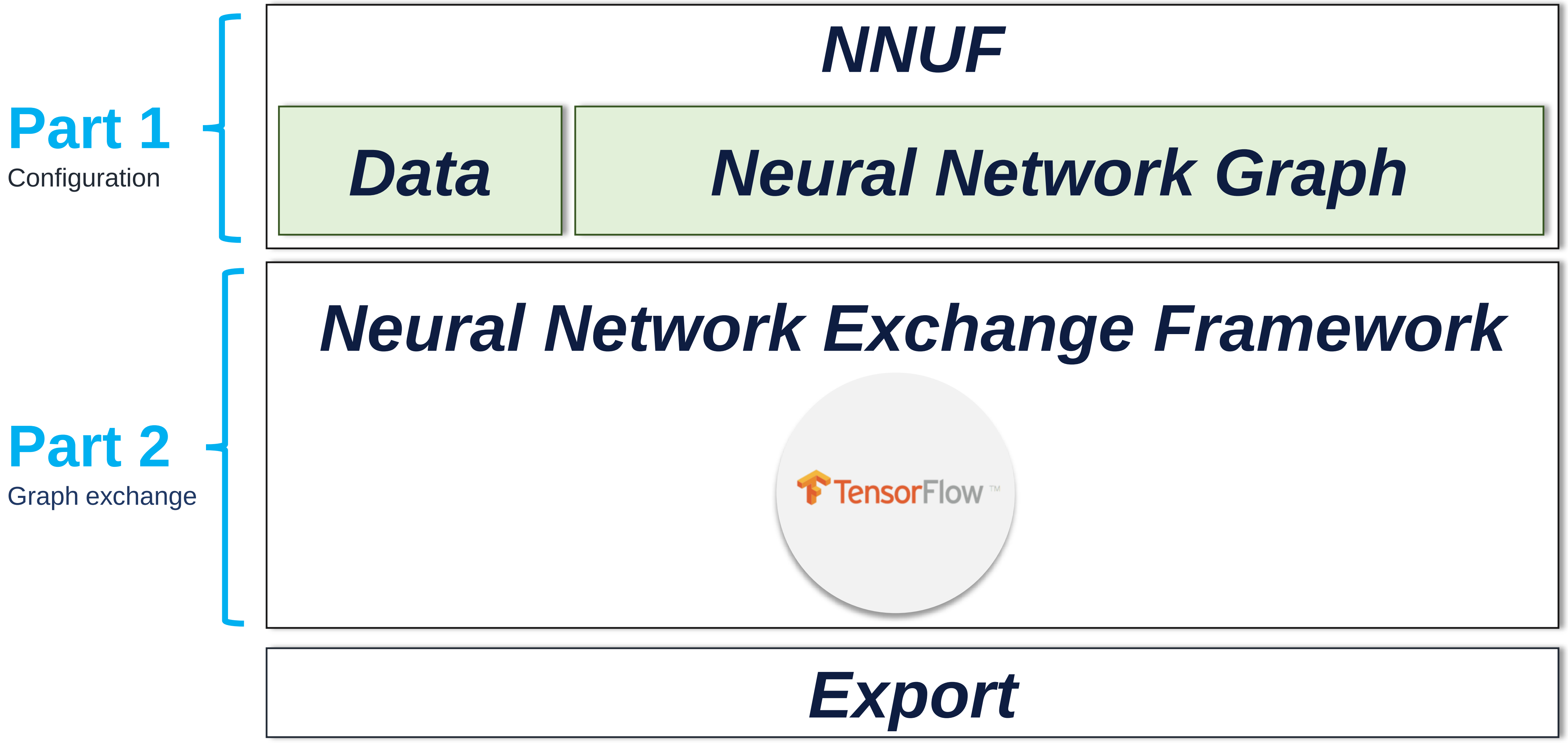
Text based human-readable format

- Intuitive description
- A collection of often used operations from which networks can be built.
- Independent of the implementation details of neural network platforms.

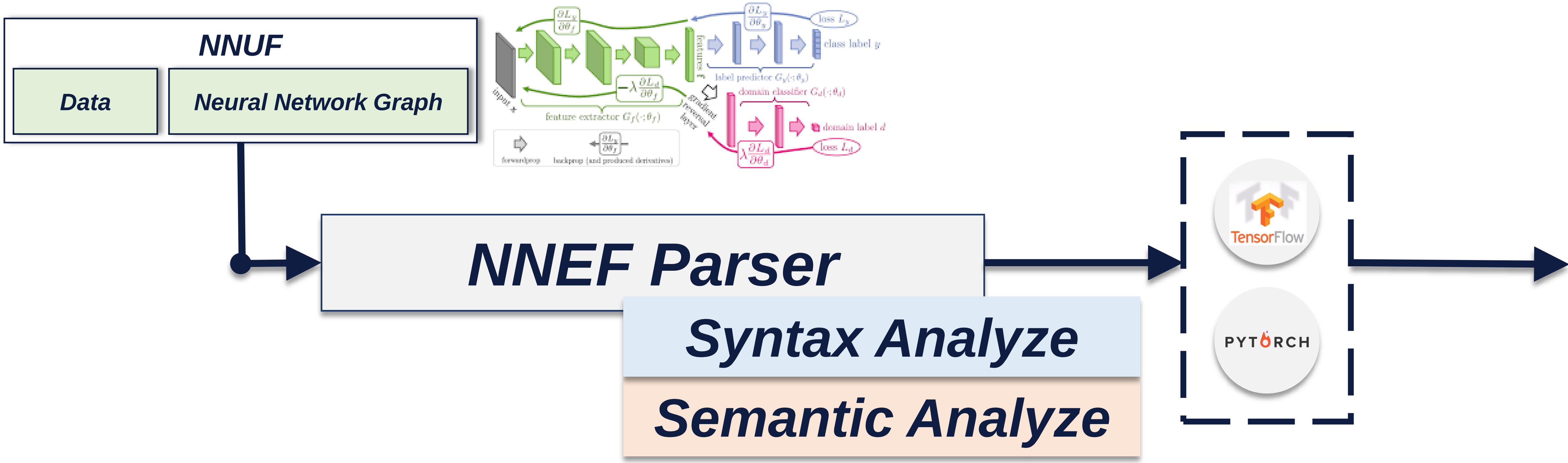


```
// Neural network definition
graph CustomNet(inputVar) -> (output Var) {
    // Variables definition
    Input = reshape(inputVar, [-1, 28, 28, 1] );
    Kernel = variable(shape=[3, 3, 1, 64],
        label="conv1/kernel");
    Bias1 = variable(shape=[64], label="conv1/bias");
    // Operations definition
    conv1 = conv(input, filter=kernel1, strides=[1, 1,
        1, 1], padding="SAME");
    add1 = add(conv1, bias1);
    outputVar = relu(add1);
}
```

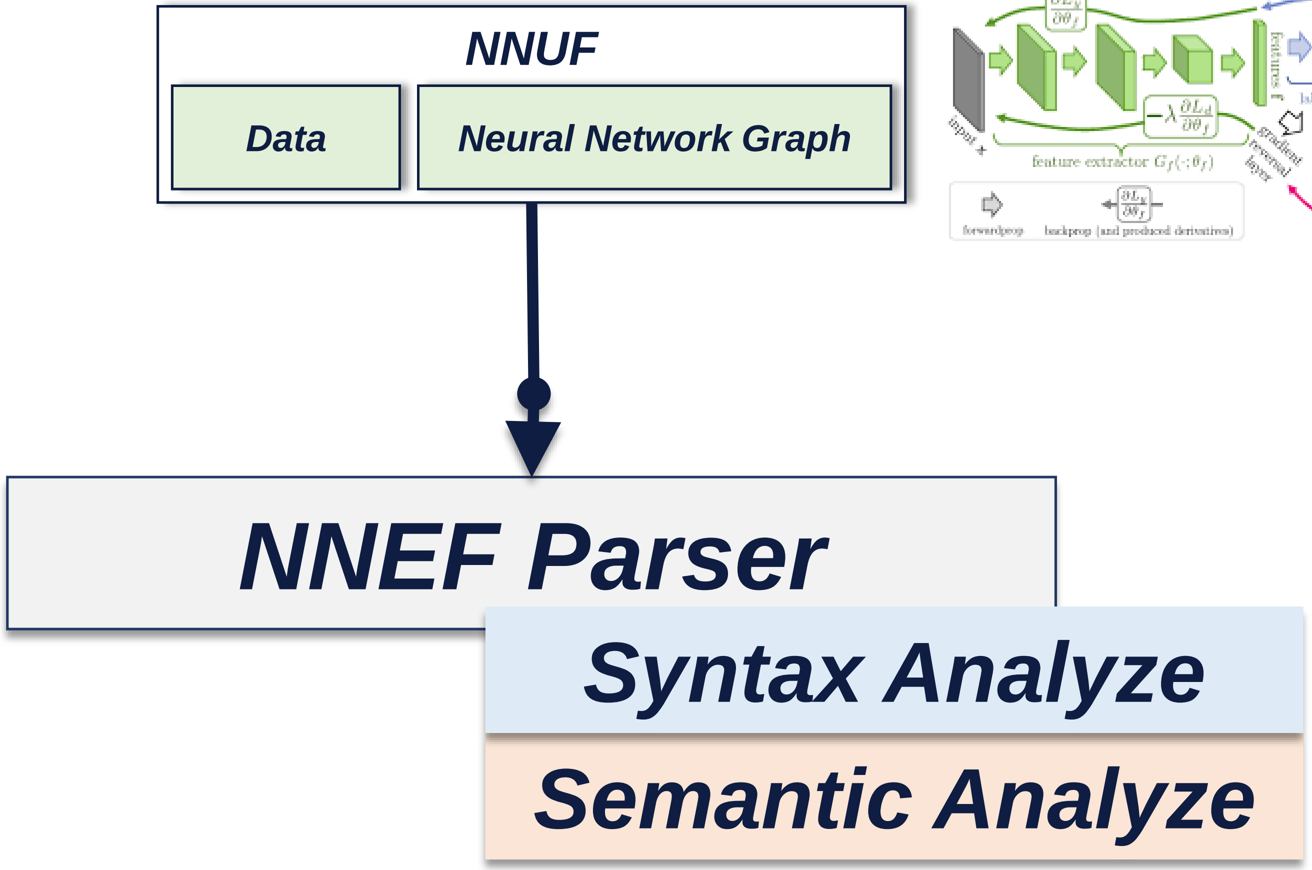
How to exchange framework?

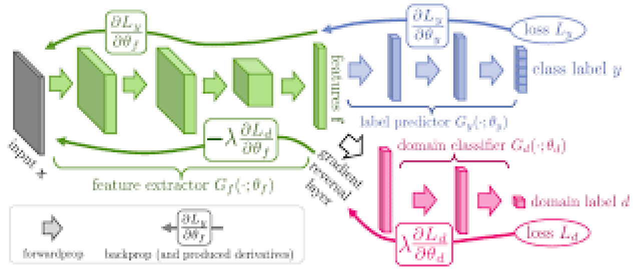


How to exchange framework?

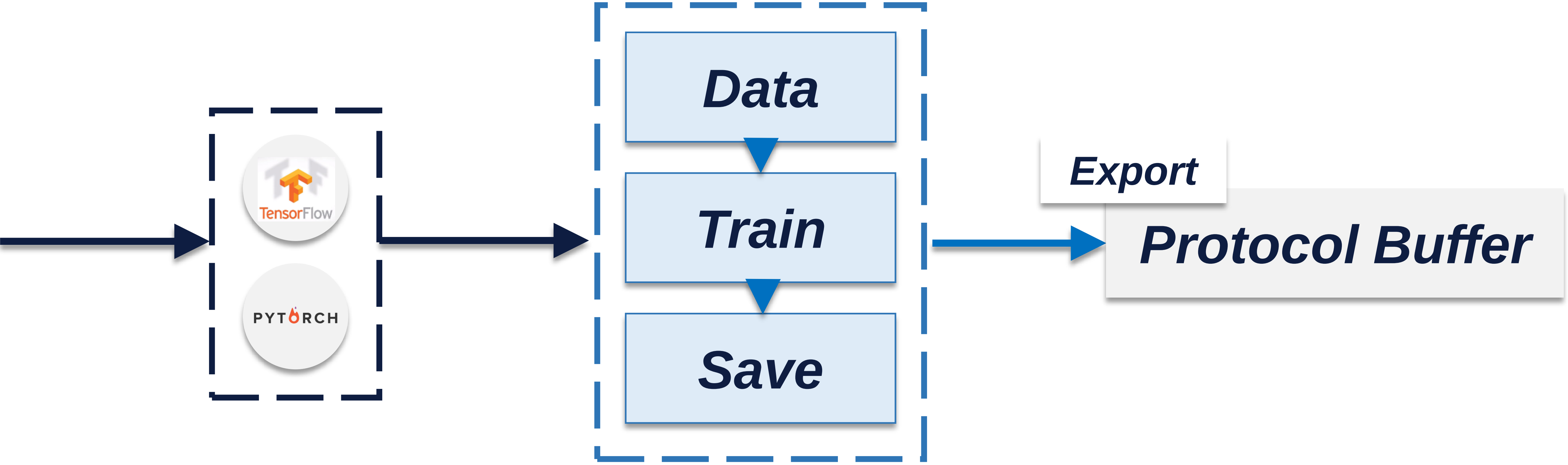

NNUF
Data
Neural Network Graph
feature extractor $G_f(\cdot;\theta_f)$
label predictor $G_y(\cdot;\theta_y)$
domain classifier $G_d(\cdot;\theta_d)$
class label y
domain label d
loss L_y
loss L_d
forwardprop
backprop (and produced derivatives)
NNEF Parser
Syntax Analyze
Semantic Analyze
TensorFlow
PYTORCH

How to exchange framework?




$\frac{\partial L_y}{\partial \theta_f}$
$\frac{\partial L_y}{\partial \theta_y}$
loss L_y
class label y
features f
label predictor $G_y(\cdot;\theta_y)$
$-\lambda \frac{\partial L_d}{\partial \theta_f}$
input x
feature extractor $G_f(\cdot;\theta_f)$
gradient reversal layer
domain classifier $G_d(\cdot;\theta_d)$
domain label d
loss L_d
$\lambda \frac{\partial L_d}{\partial \theta_d}$
forwardprop
backprop (and produced derivatives)

How to exchange framework?



Protocol Buffer

protobuf

Protocol Buffers

- Google's language-neutral, platform-neutral and extensible mechanism
- Serializing structured data
- Smaller, faster, and simpler
- Define how data to be serialized, then user can use special generated source code to easily
- Write and read your structured data to and from a variety of data streams and using a variety of languages

Parser

JavaCC

JavaCC™

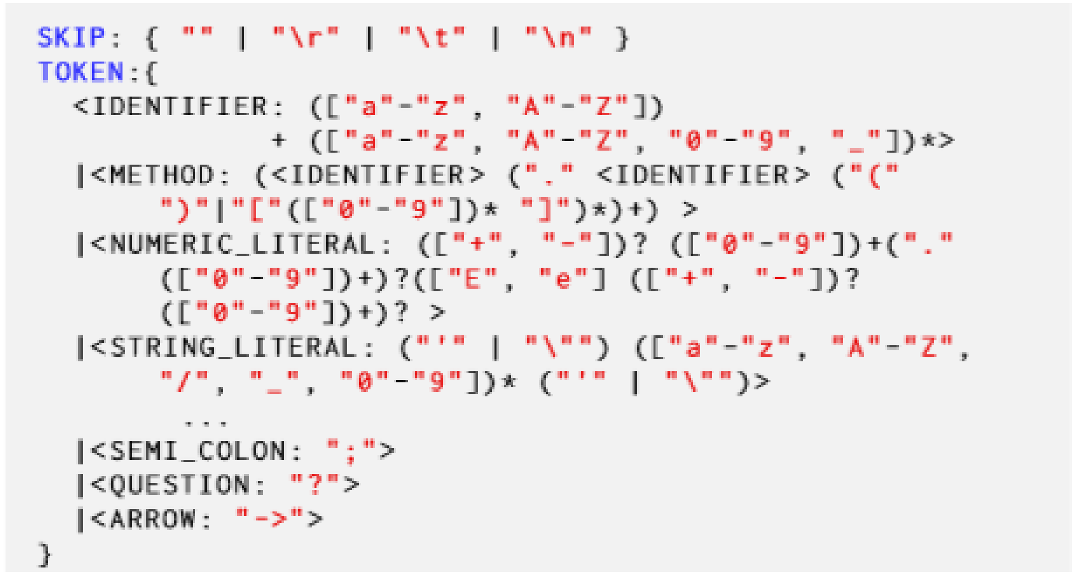
The Java Parser Generator

- Java Compiler Compiler™ (JavaCC™) is the most popular parser generator for use with Java™ applications.
- A parser generator is a tool that reads a grammar specification and converts it to a Java program that can recognize matches to the grammar.
- In addition to the parser generator itself, JavaCC provides other standard capabilities related to parser generation such as tree building

Parser

Top-Down Parser

- Start at the root of the parse tree and grow toward leaves.
  - The root node is labeled with the goal symbol of the grammar.
  - Graph definition by NNUF is the root node.
- Pick a production & try to match the input.
- Bad “pick” ⇒ Backtrack.
- Some grammars are backtrack-free.

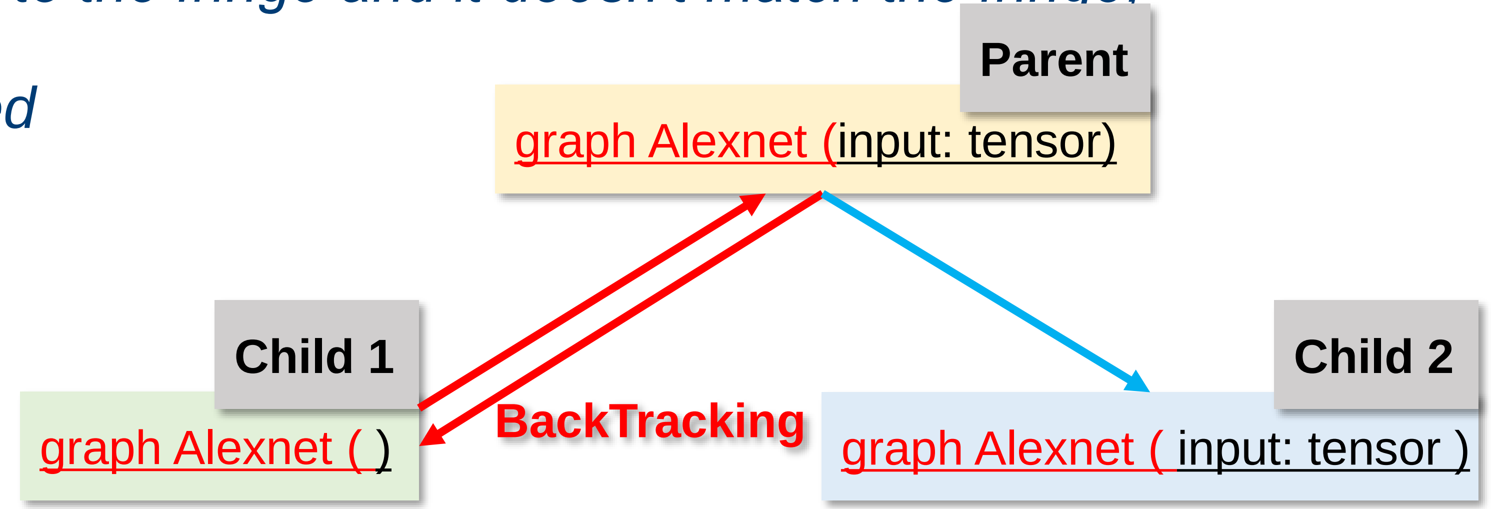


```
SKIP: { "" | "\r" | "\t" | "\n" }
TOKEN:{
  <IDENTIFIER: (["a"-"z", "A"-"Z"])
             + (["a"-"z", "A"-"Z", "0"-"9", "_"])*>
  |<METHOD: (<IDENTIFIER> ("." <IDENTIFIER> ("("
       ")"|"["(["0"-"9"])* "]")*)+) >
  |<NUMERIC_LITERAL: (["+", "-"])? (["0"-"9"])+("."
       (["0"-"9"])+)?(["E", "e"] (["+", "-"])?
       (["0"-"9"])+)? >
  |<STRING_LITERAL: ("'" | "\"") (["a"-"z", "A"-"Z",
       "/", "_", "0"-"9"])* ("'" | "\"")>
        ...
  |<SEMI_COLON: ";">
  |<QUESTION: "?">
  |<ARROW: "->">
}
```

Parser Example

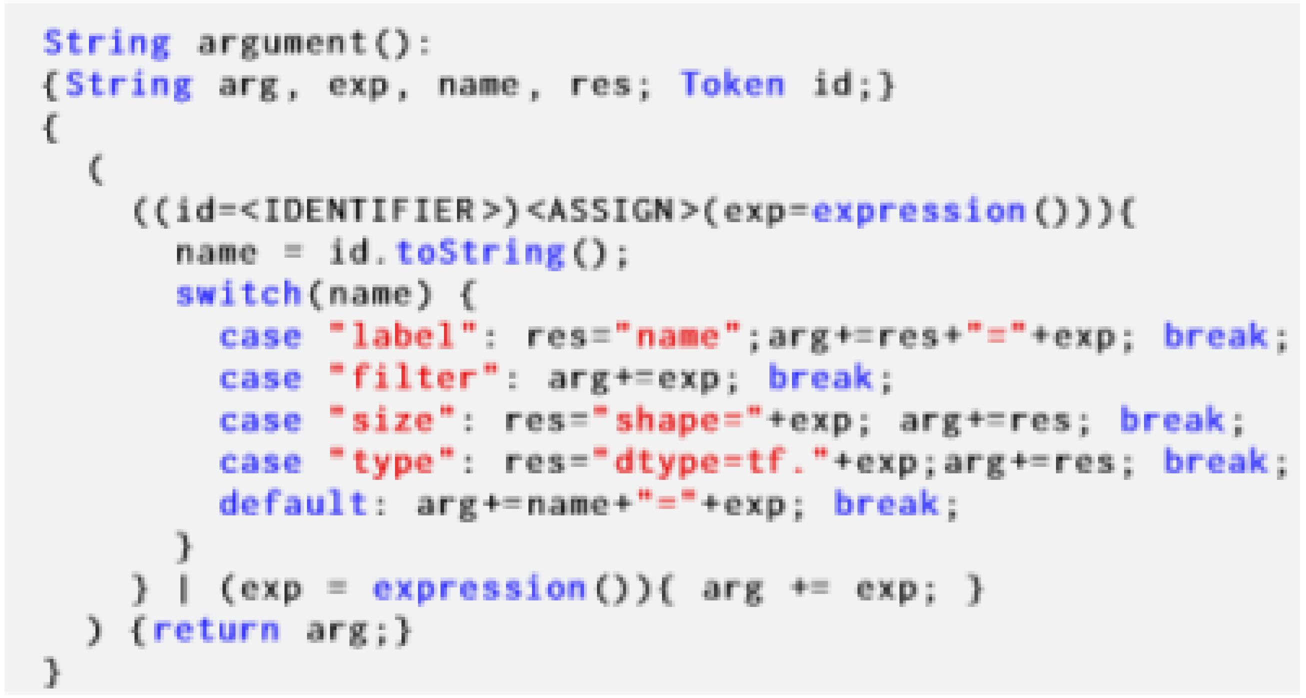
Top-down parsing algorithm:

- ***Construct*** *the root node of the parse tree.*
  - ***Root node is graph definition of NNUF.***
- *Repeat until lower fringe of the parse tree matches the input*
  - *At a node labeled A, select a production with A on its lhs and, for each symbol on its rhs, construct the appropriate child*
  - *When a terminal symbol is added to the fringe and it doesn't match the fringe, backtrack*
  - *Find the next node to be expanded*


Parent
graph Alexnet (input: tensor)
Child 1
graph Alexnet ( )
BackTracking
Child 2
graph Alexnet (input: tensor)

Parser

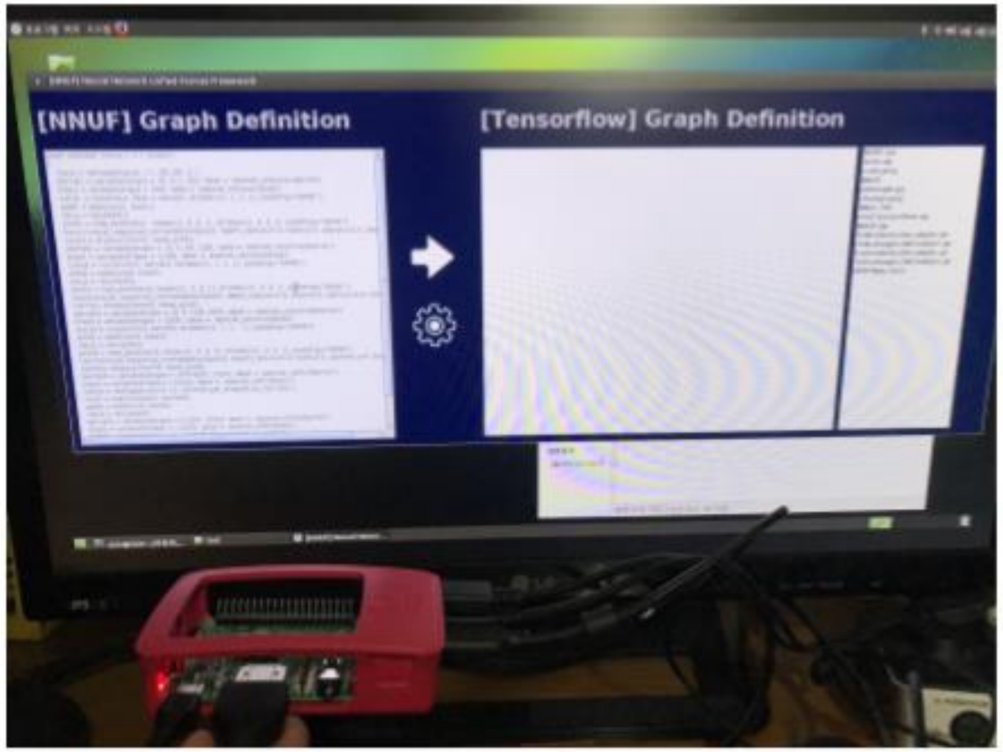
- **The NNEF can be easily revised according to the change of target version.**
  - Convert the structure of the ".jj" file in JavaCC
  - Available in Java syntax and compatible with Android Java application.



```
String argument():
{String arg, exp, name, res; Token id;}
{
  (
    ((id=<IDENTIFIER>)<ASSIGN>(exp=expression())){
      name = id.toString();
      switch(name) {
        case "label": res="name";arg+=res+"="+exp; break;
        case "filter": arg+=exp; break;
        case "size": res="shape="+exp; arg+=res; break;
        case "type": res="dtype=tf."+exp;arg+=res; break;
        default: arg+=name+"="+exp; break;
      }
    } | (exp = expression()){ arg += exp; }
  ) {return arg;}
}
```

Contribution

- Makes the neural network configuration and platform independent.
- Build a framework for platform-independent-language working in embedded device.
- A very fast transform rate in Raspberry Pie system.



Conclusion

- Currently, NNEF can be used to configure **complex convolutional neural network** such as **Alexnet**.
- In **Raspberry Pi** system, NNUF parser transform the model to target format within about **0.86** seconds on average.
- Detail information of NNEF tool to be publically accessible soon at ETRI.
- https://youtu.be/l_iEq6yyALI

Future Work

- Another exchange target framework (Caffe2 etc)
- More detail data manipulating system

Contact

- For questions and comments, contact : joongheon@gmail.com