

# 计算图 与控制流



ZOMI

BUILDING A BETTER CONNECTED WORLD

Ascend & MindSpore



www.hiascend.com  
www.mindspore.cn

# 关于本内容

## 1. 内容背景

- 计算图基础介绍

## 2. 具体内容

- 计算图（数据流图）：AI系统化问题 – 计算图的提出
- 计算图和自动微分：深度学习与微分 - 回顾自动微分 – 计算图表达自动微分
- 图的调度和执行：图优化 – 图调度与执行
- 图与控制流：控制流 – 动态图 – 静态图 – 动静统一
- 计算图的挑战与未来

# AI框架：可编程系统

算法应用

计算机视觉

自然语言处理

信号处理

推荐搜索

...

## AI框架核心

数据处理

开发接口

调试调优

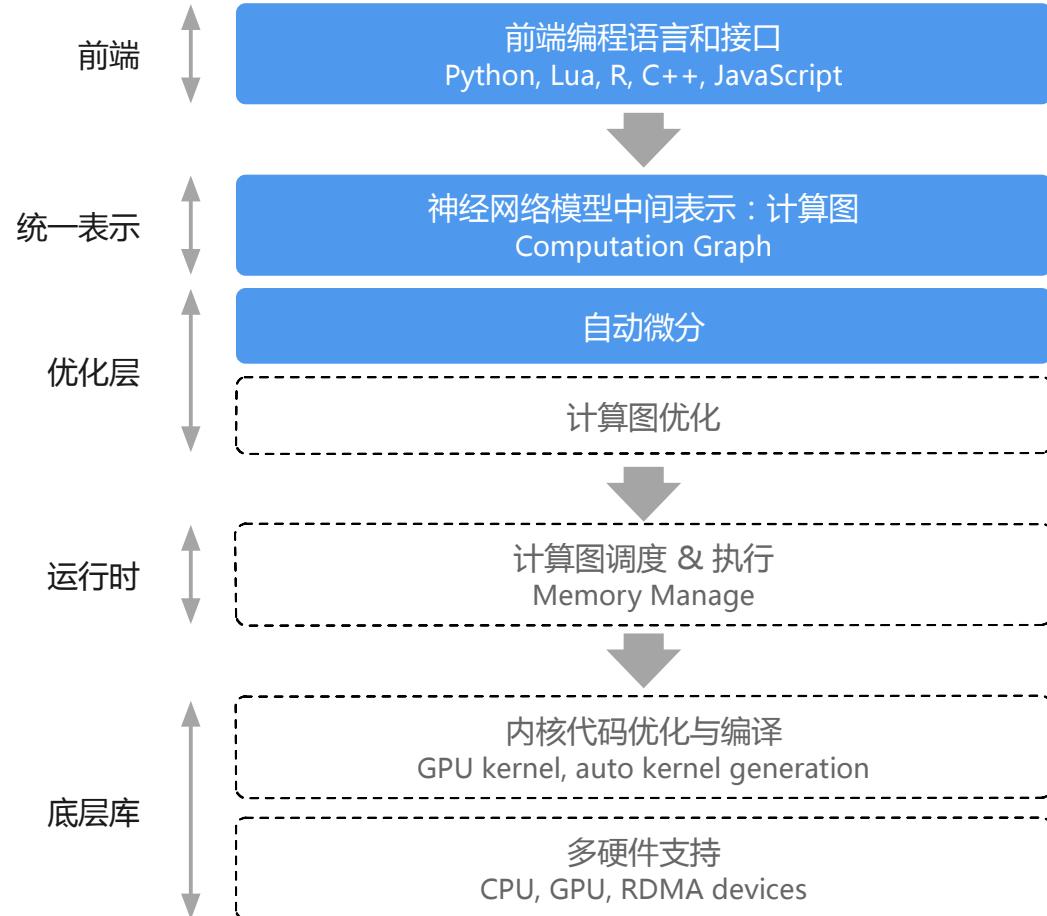
编译&执行

推理部署

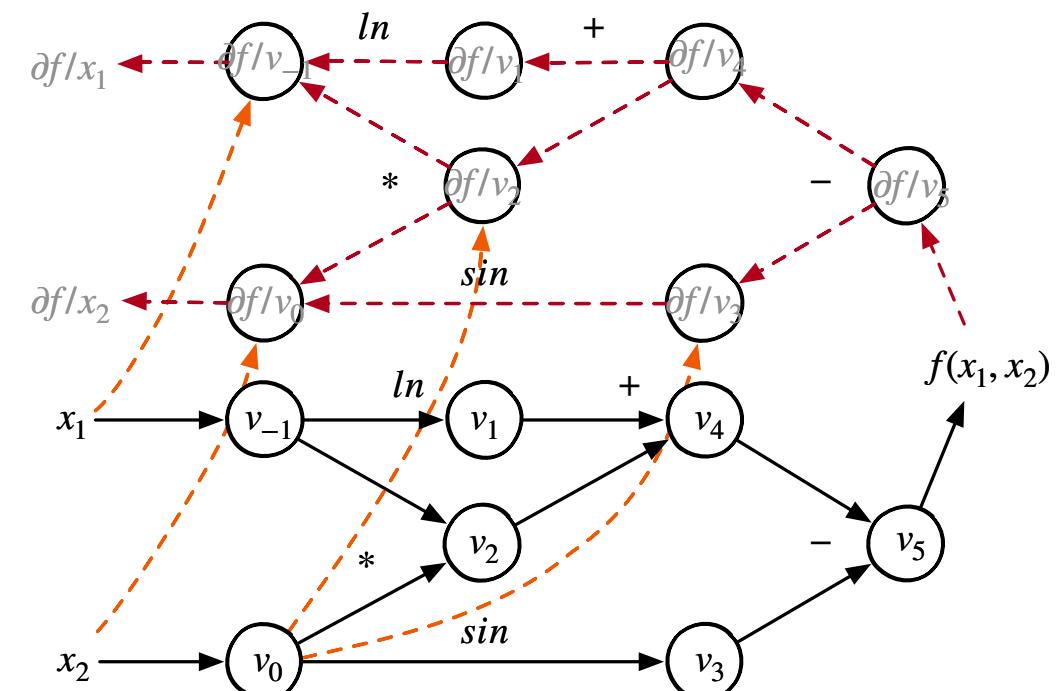
芯片使能



# AI框架从深度学习中抽象出计算图



DAG 有向无环图



# AI框架引入对动态控制流的支持

[https://github.com/huggingface/transformers/blob/main/src/transformers/models/decision\\_transformer/modeling\\_decision\\_transformer.py](https://github.com/huggingface/transformers/blob/main/src/transformers/models/decision_transformer/modeling_decision_transformer.py)

```
def forward(self, hidden_states, encoder_hidden_states,
            output_attentions):
    if encoder_hidden_states is not None:
        query = self.q_attn(hidden_states)
        key, value = self.c_attn(encoder_hidden_states)
    else:
        query, key, value = self.c_attn(hidden_states)

    query = self._split_heads(query, self.num_heads, self.head_dim)
    key = self._split_heads(key, self.num_heads, self.head_dim)
    value = self._split_heads(value, self.num_heads, self.head_dim)

    attn_output = self._merge_heads(attn_output, self.num_heads)
    attn_output = self.c_proj(attn_output)
    attn_output = self.resid_dropout(attn_output)
    if output_attentions:
        outputs += attn_weights
    return outputs
```

## Decision Transformer: Reinforcement Learning via Sequence Modeling

Lili Chen<sup>\*1</sup>, Kevin Lu<sup>\*1</sup>, Aravind Rajeswaran<sup>2</sup>, Kimin Lee<sup>1</sup>,  
Aditya Grover<sup>2</sup>, Michael Laskin<sup>1</sup>, Pieter Abbeel<sup>1</sup>, Aravind Srinivas<sup>†1</sup>, Igor Mordatch<sup>†,3</sup>

<sup>\*</sup>equal contribution <sup>†</sup>equal advising

<sup>1</sup>UC Berkeley <sup>2</sup>Facebook AI Research <sup>3</sup>Google Brain

{lilichen, kzl}@berkeley.edu

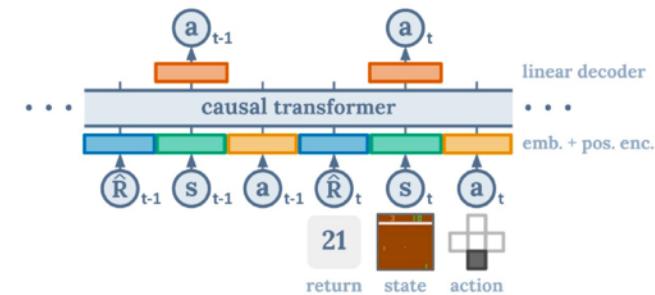


Figure 1: Decision Transformer architecture<sup>1</sup>. States, actions, and returns are fed into modality-specific linear embeddings and a positional episodic timestep encoding is added. Tokens are fed into a GPT architecture which predicts actions autoregressively using a causal self-attention mask.



Figure 2: Illustrative example of finding shortest path for a fixed graph (left) posed as reinforcement learning. Training dataset consists of random walk trajectories and their per-node returns-to-go (middle). Conditioned on a starting state and generating largest possible return at each node, Decision Transformer sequences optimal paths.

# 控制流解决方案：3种方案

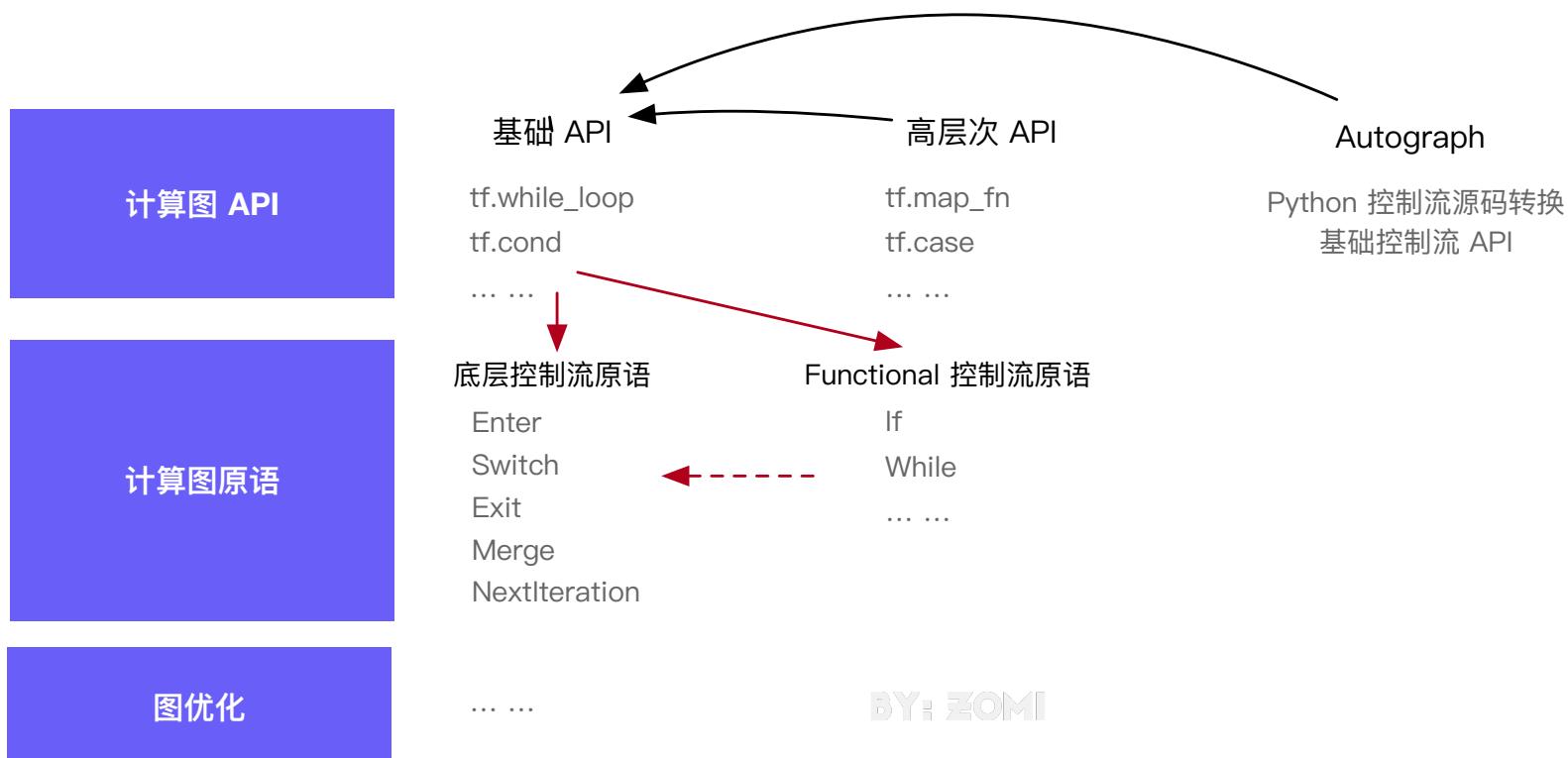
- 后端对控制流语言结构进行原生支持，计算图中允许数据流和控制流的混合；
- 复用前端语言的控制流语言结构，用前端语言中的控制逻辑驱动后端数据流图的执行；
- 后端对控制流语言结构解析成子图，对计算图进行延伸；



# 静态图：向数据流图中添加控制流原语（I）

- 声明式编程计算前获得计算图的统一描述，使得编译期器能够全局优化
- 执行流无需在前端语言与运行时反复切换，可以有更高的执行效率

**控制流原语**  
第一等级  
( first-class )



# 静态图：向数据流图中添加控制流原语（II）

```
for(i = 0; i < 10; i++) {
    for(j = 0; j < 10; j++) {
        ...
    }
}
```

```
i = tf.constant(0)
j = tf.constant(0)
a = lambda i: tf.less(i, 10)
b = lambda i: (tf.add(i, 1), )
c = lambda i: (tf.add(j, 1), )
y = tf.while_loop(a, b, [j])
r = tf.while_loop(c, y, [i])
```

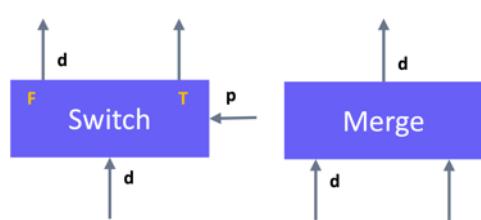
[https://www.tensorflow.org/api\\_docs/python/tf/while\\_loop](https://www.tensorflow.org/api_docs/python/tf/while_loop)

使用TF 2.0 `tf.while_loop()`

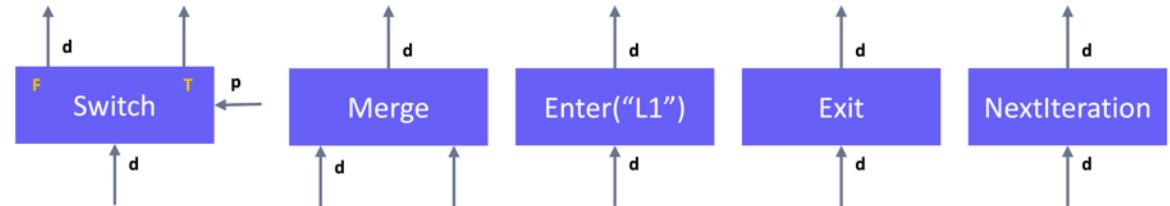
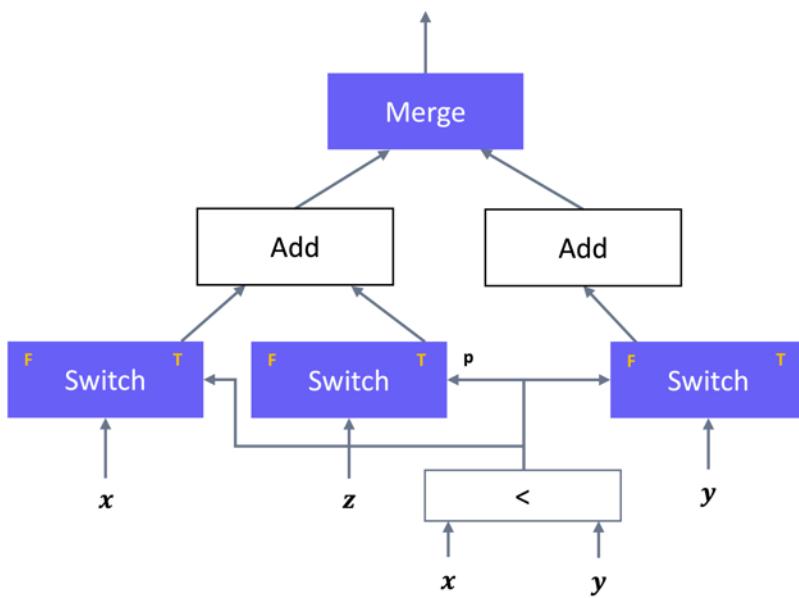
```
execution frame: ←----- 具有全局唯一的名字作为标识符
{
    key(ops_1): value(input_addr, output_addr, ops_attr),
    key(ops_2): value(input_addr, output_addr, ops_attr),
    ...
    execution frame: ←----- 保存着执行算子所需的上下文信息
    {
        key(ops_1): value(input_addr, output_addr, ops_attr),
        key(ops_2): value(input_addr, output_addr, ops_attr),
        ...
    }
}
```

←----- 嵌套Execution frame，可并发优化

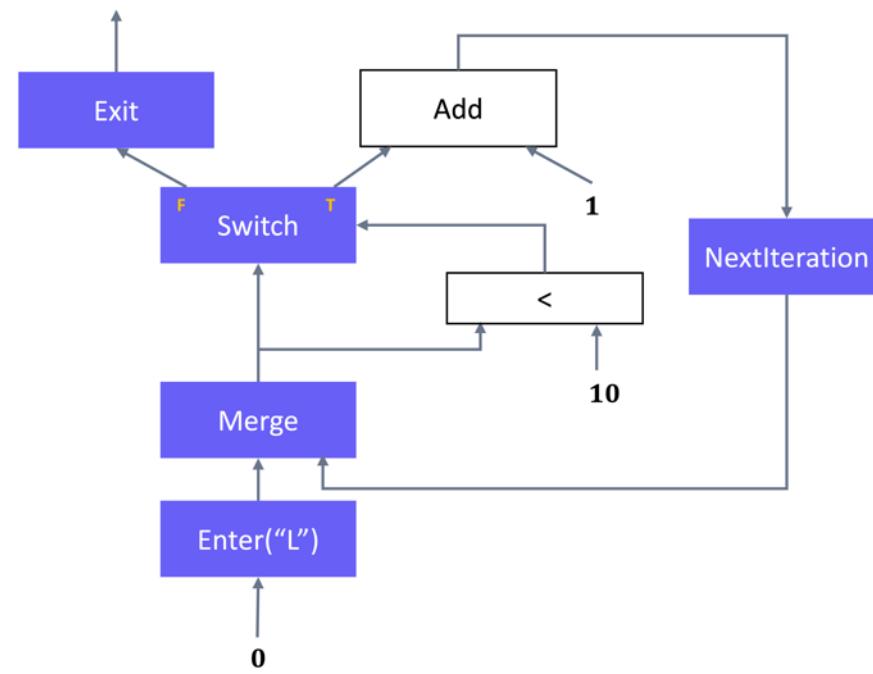
# 静态图：向数据流图中添加控制流原语 ( III )



`tf.cond(x < y, lambda: tf.add(x, z), lambda: tf.square(y))`



`tf.while_loop(lambda i: i < 10, lambda i: tf.add(i, 1), [0])`



# 静态图：向数据流图中添加控制流原语

- 声明式编程计算前获得计算图的统一描述，使得编译期器能够全局优化
- 执行流无需在前端语言与运行时反复切换，可以有更高的执行效率



控制流原语

第一等级  
( first-class )

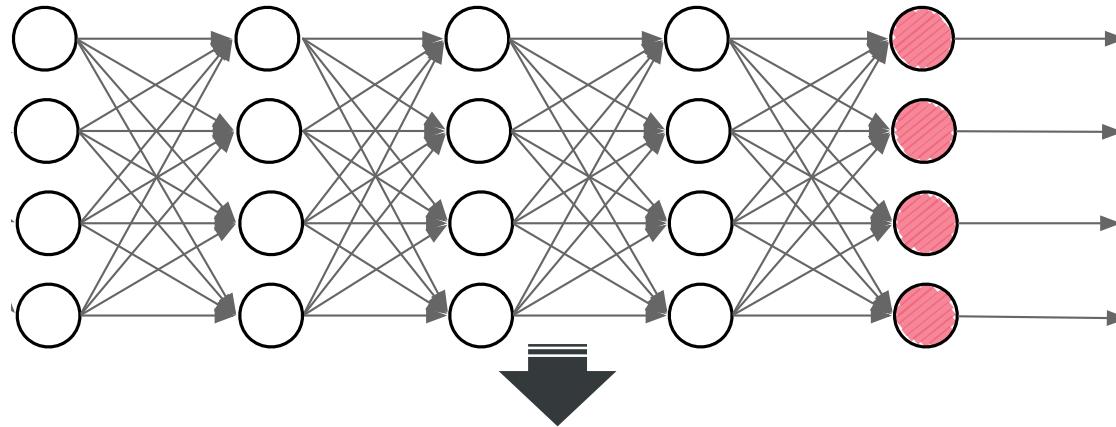
## 缺点

- 控制流原语语义设计首要服务于运行时系统的并发执行模型，与深度学习概念差异大；
- 对控制流原语进行再次封装，以控制流API的方式供前端用户使用，导致计算图复杂；

## 优点

- 向计算图中引入控制流原语利于编译期得到全计算过程描述，发掘运行时效率提升点；
- 解耦宿主语言与执行过程，加速运行时执行效率；

## 动态图：复用宿主语言控制流语句



```
def foo1(x: Tensor, y: Tensor, z: Tensor) -> Tensor:
    if x < y:
        s = x + y
    else:
        s = torch.square(y)
    return s
```

AI框架不维护全局神经网络算法描述  
( 计算图 )

神经网络变成一段Python代码  
( 模型即代码 )

后端张量计算以库的形式提供  
使能芯片执行计算



## 动态图：复用宿主语言控制流语句

### 优点

- 用户能够自由地使用前端宿主的控制流语言，即时输出张量计算的求值结果
- 定义神经网络计算就像是编写真正的程序

### 缺点

- 用户易于滥用前端语言特性，带来复杂的性能问题
- 执行流会在语言边界来回跳转，带来严重运行时开销
- 控制流和数据流被严格地隔离在前端语言和后端语言，跨语言优化困难

## 静态图：源码解析对计算图展开和转换子图

- 计算图能够表达的控制直接展开，如for(xxx)展开成带顺序的计算图；
- 通过创建子图进行表示，运行时时候动态选择子图执行，如 if else；

### 优点

- 用户能够一定程度自由地使用前端宿主的控制流语言；
- 解耦宿主语言与执行过程，加速运行时执行效率；
- 计算图在编译期得到全计算过程描述，发掘运行时效率提升点；

### 缺点

- 硬件不支持控制方式下，执行流会仍然会在语言边界跳转，带来运行时开销；
- 部分宿主的控制流语言不能表示，带有一定约束性；

# 动态图转换为静态图



Auto-graph



PT JIT

- **基于追踪Trace**：直接执行用户代码，记录下算子调用序列，将算子调用序列保存为静态图，执行中脱离前端语言环境，由运行时按照静态图逻辑执行；
- **基于源代码解析**：以宿主语言的抽象语法树（AST）为输入，转化为内部语法树，经过别名分析，SSA（static single value assignment），类型推断等Pass，转换为计算图表示；

# 动态图转换为静态图



Auto-graph



- **基于追踪Trace**：直接执行用户代码，记录下算子调用序列，将算子调用序列保存为静态图，执行中脱离前端语言环境，由运行时按照静态图逻辑执行；

## 优点

- 能够更广泛地支持宿主语言中的各种动态控制流语句；

## 缺点

- 执行场景受限，只能保留程序有限执行轨迹并线性化，静态图失去源程序完整控制结构；

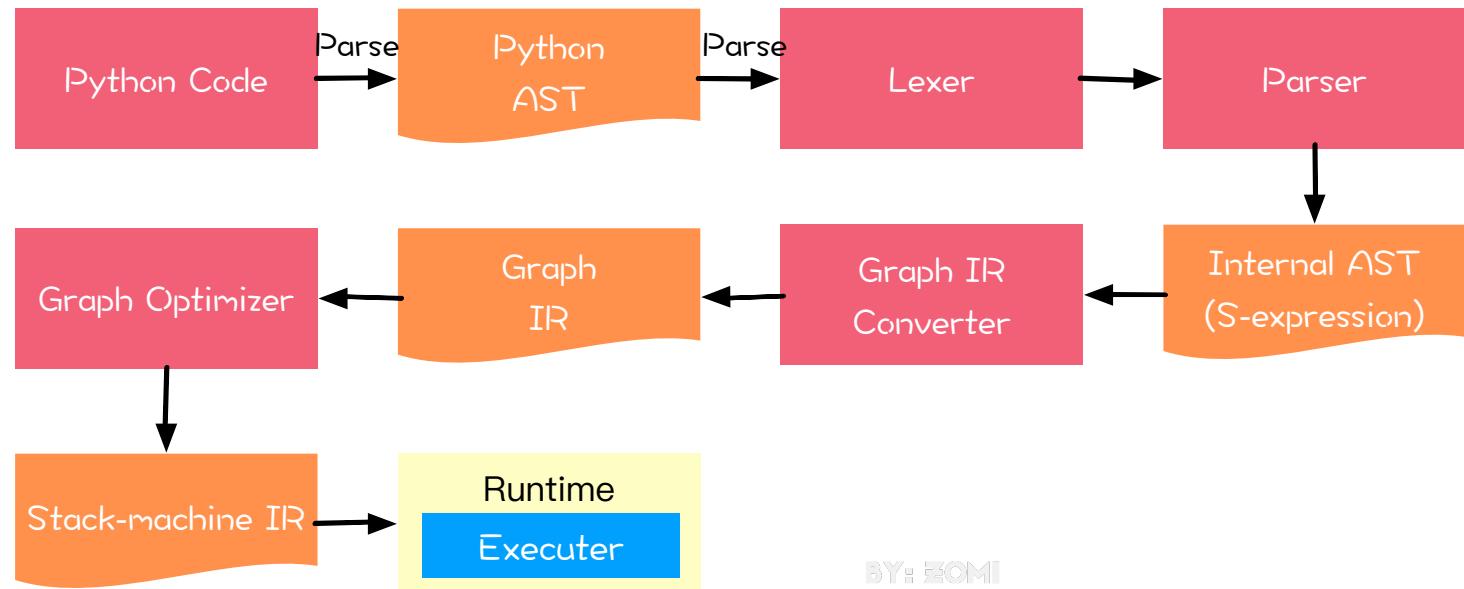
# 动态图转换为静态图



- **基于源代码解析**：以宿主语言的抽象语法树（AST）为输入，转化为内部语法树，经过别名分析，SSA（static single value assignment），类型推断等Pass，转换为计算图表示；

```
@torch.jit.script
def foo1(x: Tensor, y: Tensor, z: Tensor):
    if x < y:
        s = x + y
    else:
        s = torch.square(y)
    return s

@torch.jit.script
def foo2(s: Tensor):
    for i in torch.range(10):
        s += i
    return s
```



# 动态图转换为静态图



Auto-graph



PT JIT

- **基于源代码解析**：以宿主语言的抽象语法树（AST）为输入，转化为内部语法树，经过别名分析，SSA（static single value assignment），类型推断等Pass，转换为计算图表示；

## 优点

- 能够更广泛地支持宿主语言中的各种动态控制流语句

## 缺点

- 后端实现和硬件实现会对静态图表示进行限制和约束，多硬件需要切分多后端执行逻辑；
- 宿主语言的控制流语句并不总是能成功映射到后端运行时系统的静态图表示；
- 遇到过度灵活的动态控制流语句，运行时会退回到由前端语言跨语言调用驱动后端执行；

# Summary

1. 控制流采用不同设计，AI框架为声明式编程的静态图，以及命令式编程的动态图；
2. 静态图统一DL表示利于编译优化和执行加速，但是灵活性和易用性受限；
3. 动态图灵活复用宿主语言中的控制流原语，但是缺乏性能优化阶段；
4. 基于追踪Trace或基于源代码解析可将动态图转换为静态图，兼顾易用性和性能；



BUILDING A BETTER CONNECTED WORLD

THANK YOU

Copyright©2014 Huawei Technologies Co., Ltd. All Rights Reserved.

The information in this document may contain predictive statements including, without limitation, statements regarding the future financial and operating results, future product portfolio, new technology, etc. There are a number of factors that could cause actual results and developments to differ materially from those expressed or implied in the predictive statements. Therefore, such information is provided for reference purpose only and constitutes neither an offer nor an acceptance. Huawei may change the information at any time without notice.