Bring Your Own Codegen to TVM

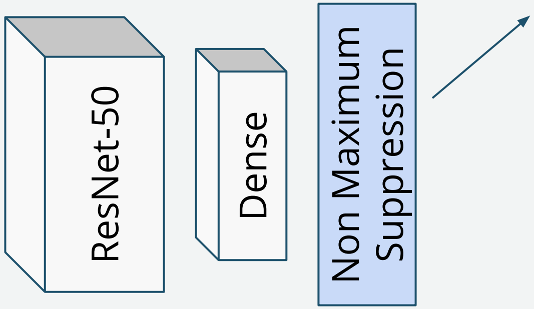
AWS AI

Presenter: Zhi Chen, Cody Yu
Amazon SageMaker Neo, Deep Engine Science

Considering You...

Design and manufacture a deep learning chip which achieves amazing performance on **widely-used operators (e.g. conv2d, dense, ReLU, etc)**

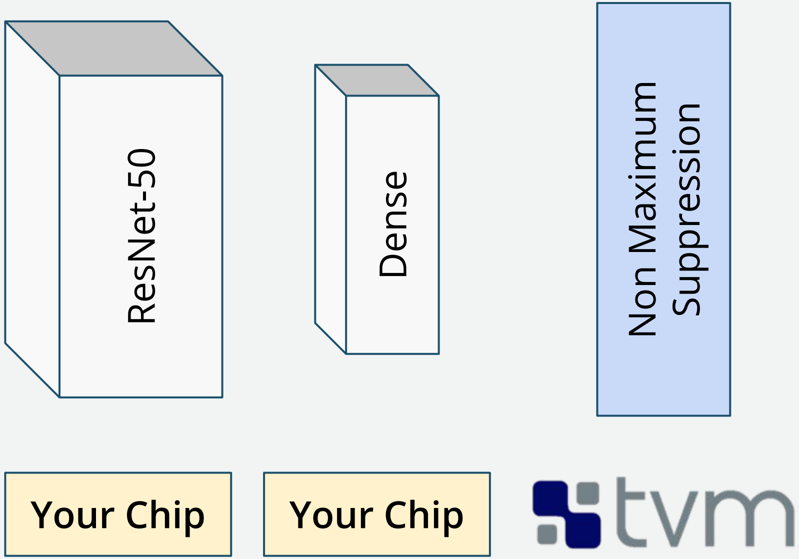
Now your customer wants to run a YOLO model, but...



Non Maximum Suppression (NMS) is too new to be supported by your chip

But NMS is supported by TVM!

Let TVM Be the Compiler of Your Chip



Your chip can run any models

Your compiler (TVM) supports multiple frontends
(e.g., TensorFlow, PyTorch, MXNet)

How Would That Look Like?

Example showcase: Intel MKL-DNN (DNNL) library

1. Import packages

   ```
   import numpy as np
   from tvm import relay
   ```

2. Load a pretrained network

   ```
   mod, params = relay.testing.mobilenet.get_workload(batch_size=1)
   ```

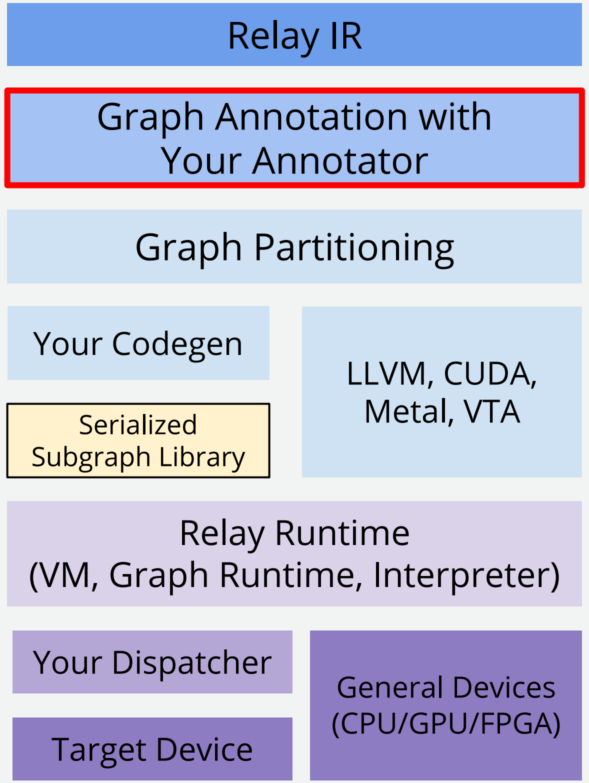
3. **Partition and build the network with an external codegen**

   ```
   mod = relay.build_extern(mod, “dnnl”)
   ```

4. Run the inference

   ```
   exe = relay.create_executor(“vm”, mod=mod, ctx=tvm.cpu(0))
   data = np.random.uniform(size=(1, 3, 224, 224)).astype(“float32”)
   out = exe.evaluate()(data, **params)
   ```

System Overview



Relay IR

Graph Annotation with Your Annotator

Graph Partitioning

Your Codegen

Serialized Subgraph Library

LLVM, CUDA, Metal, VTA

Relay Runtime (VM, Graph Runtime, Interpreter)

Your Dispatcher

Target Device

General Devices (CPU/GPU/FPGA)

Mark supported operators or subgraphs

1. Implement an operator-level annotator, OR
2. Implement a graph-level annotator

Option 1: Operator-Level Annotation

- Implement a Python template to indicate if an op can be supported by your codegen
- Template path: python/tvm/relay/op/contrib/**<your_codegen_name>**/extern_op.py
- Boolean functions in the template

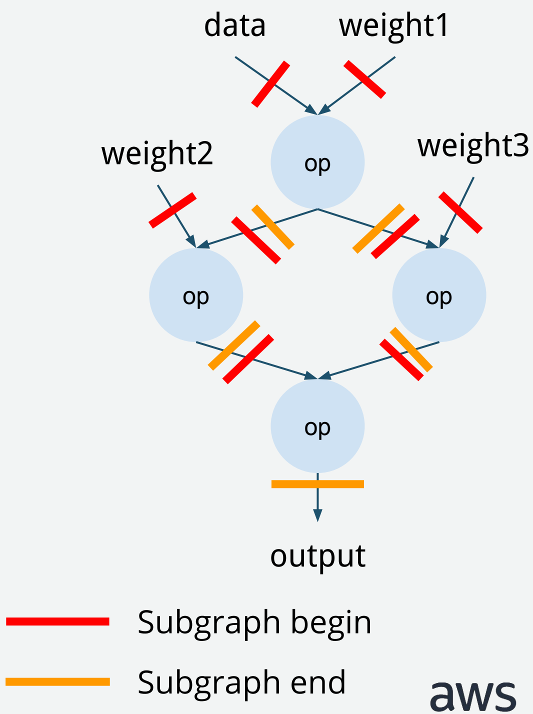
Relay operator name

Operator attributes and args (inputs) can be checked as well

```
def conv2d(attrs, args):
    return is_float32(args)
```

Return True/False for this op

After Annotation



Option 2: Graph-Level Annotation

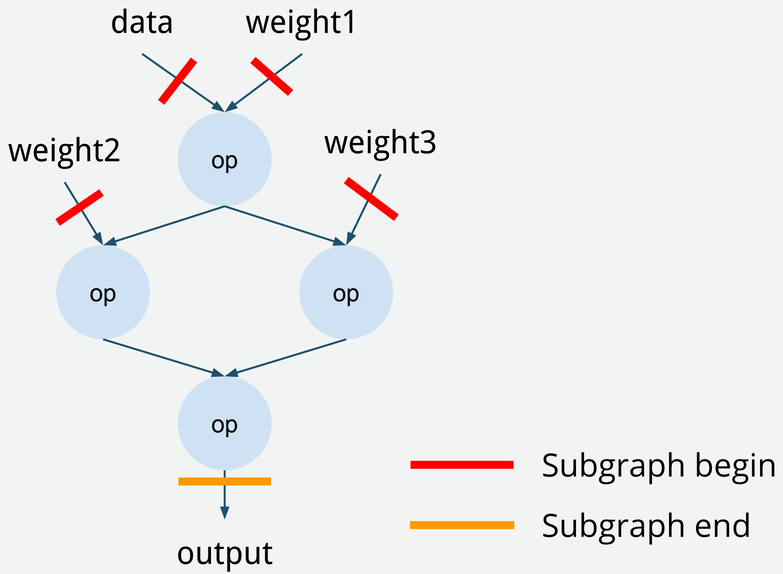
- Implement a Relay IR visitor to annotate a subgraph
- Module path:
  python/tvm/relay/op/contrib/**<your_codegen_name>**/graph_annotator.py
- Apply the annotator to a workload:

```
mod, params = relay.testing.mobilenet.get_workload(batch_size=1)
mod[‘main’] = MyAnnotator().visit(mod[‘main’])
mod = relay.build_extern(mod, “dnnl”)
```

Example: Annotate an Entire Graph

```
class WholeGraphAnnotator(ExprMutator):
    def __init__(self, target):
        super(WholeGraphAnnotator, self).__init__()
        self.target = target
        self.last_call = True

    def visit_call(self, call):
        curr_last = self.last_call
        self.last_call = False

        params = []
        for arg in call.args:
            param = super().visit(arg)
            if isinstance(param, relay.expr.Var):
                param = subgraph_begin(param, self.target)
            params.append(param)
        new_call = relay.Call(call.op, params, call.attrs)
        if curr_last:
            new_call = subgraph_end(new_call, self.target)
        return new_call
```

After Annotation



Comparison of Two Options

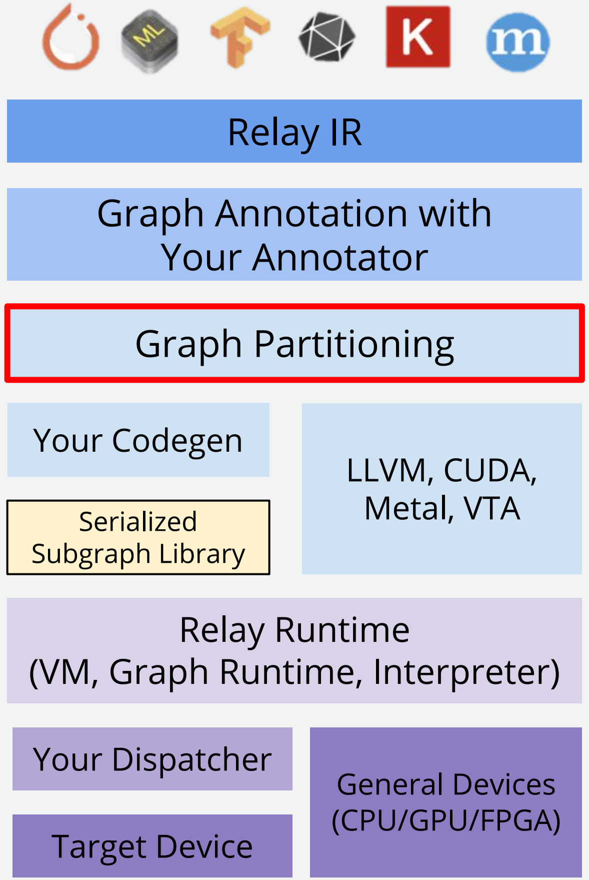
Op-level annotation

- Simple and easy to implement 👍
- One op per subgraph results in overhead 👎
  (working on an algorithm to merge annotated ops)

Graph-level annotation

- High flexibility and allow multiple ops in a subgraph 👍
- Relatively hard to implement 👎

System Overview



Relay IR

Graph Annotation with Your Annotator

Graph Partitioning

Your Codegen

Serialized Subgraph Library

LLVM, CUDA, Metal, VTA

Relay Runtime (VM, Graph Runtime, Interpreter)

Your Dispatcher

Target Device

General Devices (CPU/GPU/FPGA)

Mark supported operators or subgraphs

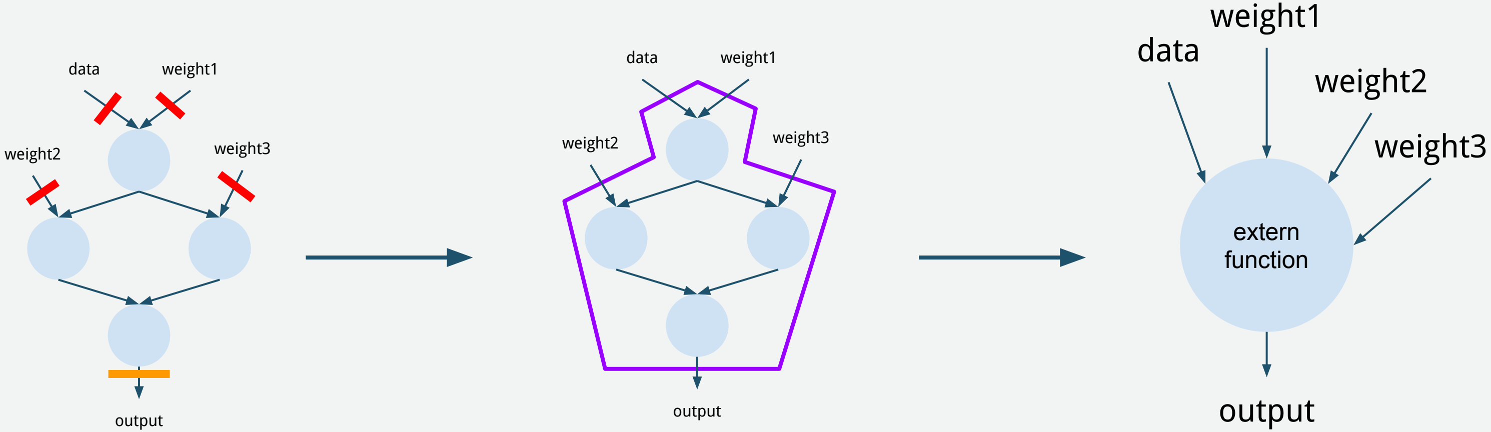
1. Implement extern operator functions, OR
2. Implement a graph annotator

Partition the Relay IR graph

- No user involvement

Graph Partitioning

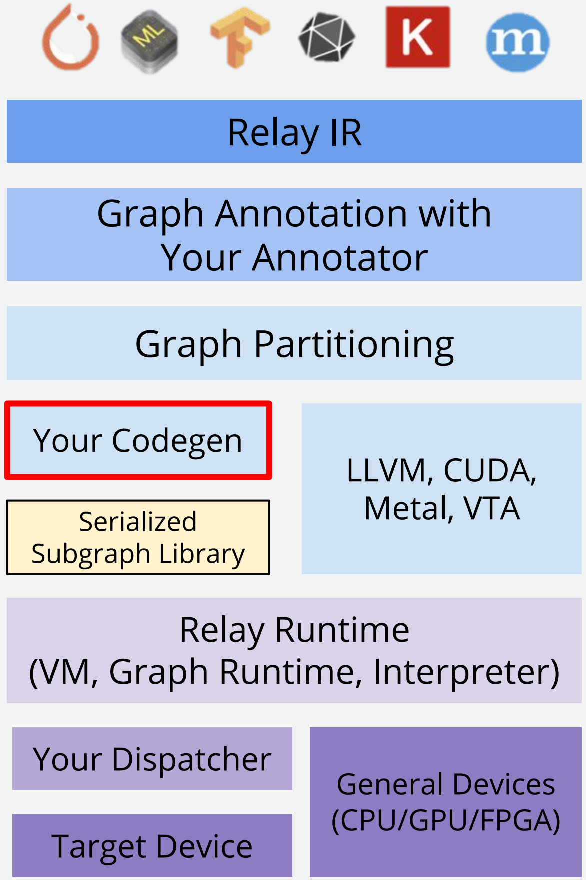
Use external functions to wrap annotated subgraphs



What are not supported yet?

- Duplicated inputs optimization (e.g., reused parameters)
- Multiple outputs (e.g., batch normalization)
- Subgraph merging (e.g., conv2d + ReLU)

System Overview


Relay IR
Graph Annotation with Your Annotator
Graph Partitioning
Your Codegen
Serialized Subgraph Library
LLVM, CUDA, Metal, VTA
Relay Runtime (VM, Graph Runtime, Interpreter)
Your Dispatcher
Target Device
General Devices (CPU/GPU/FPGA)

Mark supported operators or subgraphs

1. Implement extern operator functions, OR
2. Implement a graph annotator

Partition the Relay IR graph

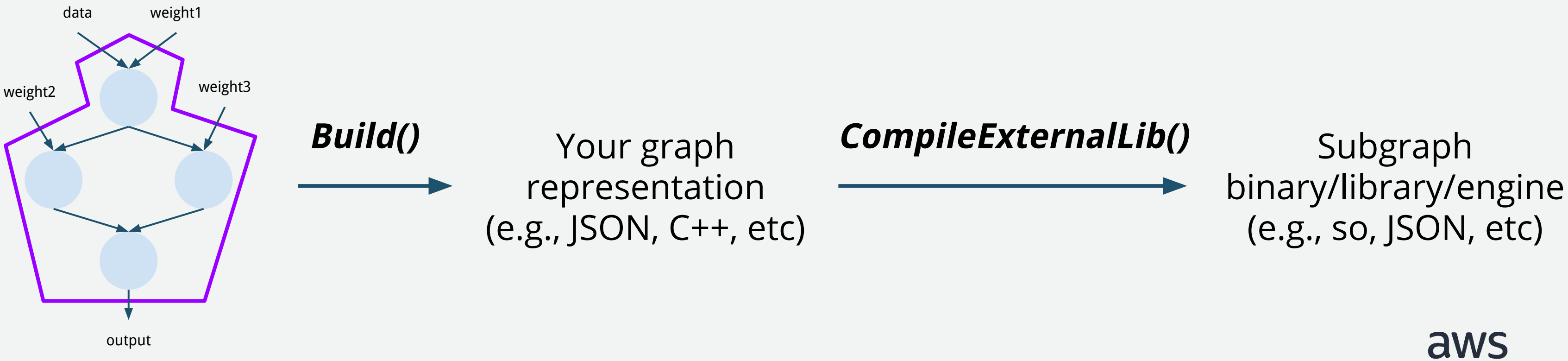
- No user involvement

Generate binary/library/engine for the subgraph

- Implement an IR visitor for codegen
- Implement the build logic

Implement the Codegen

- Implement a codegen class to accept subgraphs and build binary/library/engine for runtime dispatching
- Codegen path: src/relay/backend/contrib/**<your_codegen_name>**/codegen.cc
- Flow overview



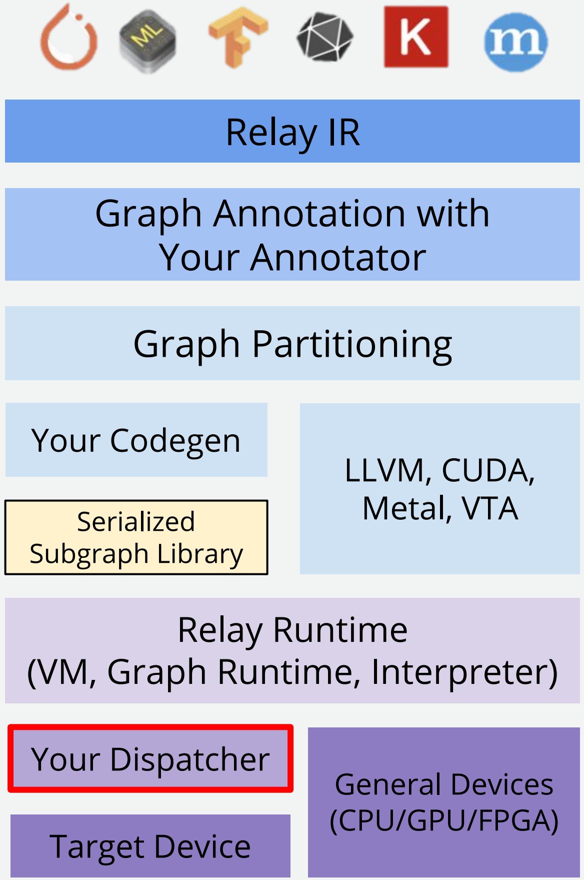
Build()

Your graph representation (e.g., JSON, C++, etc)

CompileExternalLib()

Subgraph binary/library/engine (e.g., so, JSON, etc)

System Overview



Relay IR

Graph Annotation with Your Annotator

Graph Partitioning

Your Codegen

Serialized Subgraph Library

LLVM, CUDA, Metal, VTA

Relay Runtime (VM, Graph Runtime, Interpreter)

Your Dispatcher

Target Device

General Devices (CPU/GPU/FPGA)

Mark supported operators or subgraphs

1. Implement extern operator functions, OR
2. Implement a graph annotator

Partition the Relay IR graph

- No user involvement

Generate binary/library/engine for the subgraph

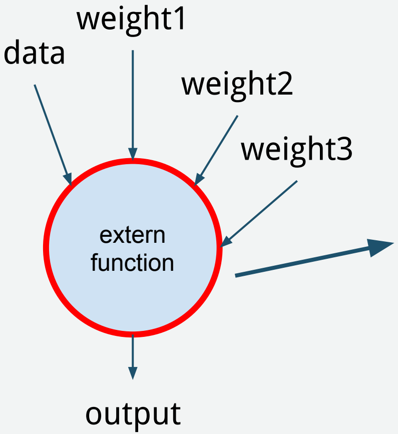
- Implement an IR visitor for codegen
- Implement the build logic

Dispatch generated binary/library/engine in runtime

- Implement a runtime packed function

Implement the Runtime Dispatcher

- Implement a TVM runtime module to dispatch the subgraph to the generated executable engine
- Runtime path:
  src/runtime/contrib/**<your_codegen_name>**/<your_codegen_name>.{h, cc}
- Overview



Relay runtime invokes your ***GetFunction()*** to execute the external function

Example: Dispatch Codegen Built Shared Library

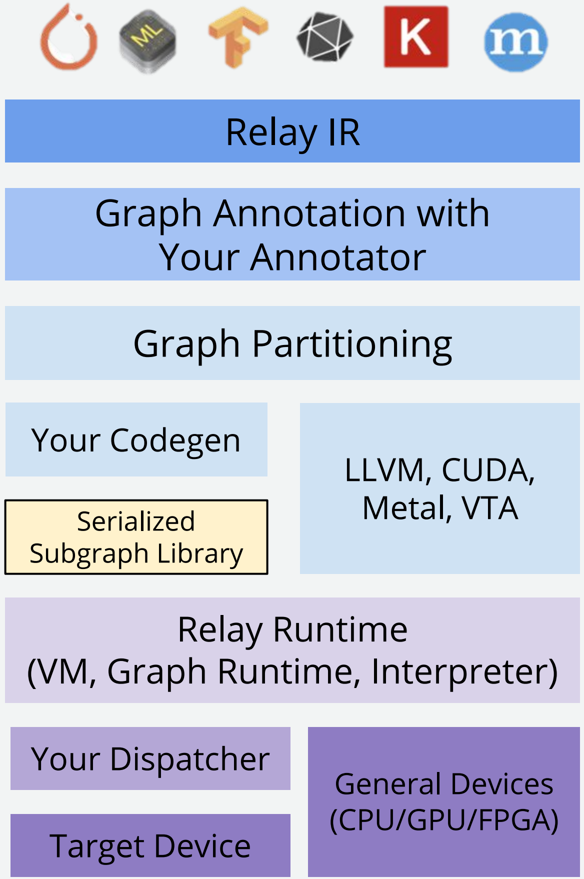
```
runtime::PackedFunc DNNLModule::GetFunction(
  const std::string& name, const std::shared_ptr<ModuleNode>& sptr_to_self) {
  if (name == "init") {
    return PackedFunc([sptr_to_self, this](TVMArgs args, TVMRetValue* rv) {
      this->Init(args[0]);
    });
  } else {
    std::string curr_id = GetSubgraphID(name);
    return PackedFunc([sptr_to_self, curr_id, this](TVMArgs args, TVMRetValue* rv) {
      auto out = reinterpret_cast<float*>(args[args.size() - 1]>data);

      std::string encoded_name = kDnnlPrefix + curr_id;
      auto func_s = reinterpret_cast<DnnlSubgraphFunc>(GetSymbol(encoded_name));

      DnnlPackedArgs packed_args;
      packed_args.data = reinterpret_cast<void**>(malloc(sizeof(float*) *
args.size()));
      for (int i = 0; i < args.size() - 1; ++i) {
        runtime::NDArray arg = args[i];
        packed_args.data[i] = reinterpret_cast<float*>(arg->data);
      }
      (*func_s)(packed_args, out); *rv = out;
});}}
```

Load the built shared library

Get the corresponding subgraph function

Execute the subgraph

Next Steps



Relay IR

Graph Annotation with Your Annotator

Graph Partitioning

Your Codegen

Serialized Subgraph Library

LLVM, CUDA, Metal, VTA

Relay Runtime (VM, Graph Runtime, Interpreter)

Your Dispatcher

Target Device

General Devices (CPU/GPU/FPGA)

- Send PRs to the upstream
- Improve graph partitioning
- An algorithm to merge supported operators

Thank You and Q&A

System Prototyping
https://github.com/apache/incubator-tvm/pull/4258

RFC
https://discuss.tvm.ai/t/bring-your-own-codegen-to-tvm/4501

Acknowledgement

aws

tvm

APACHE