BYTEIR

ByteIR: Towards Seamless Model Compilation Integration

Hongyu Zhu
ByteDance AML

Applied Machine Learning

ByteDance字节跳动

About Me

Hongyu Zhu

Received my PhD degree from University of Toronto (advisor: Gennady Pekhimenko)

Joined ByteDance AML group in 2022.3

Currently working on AI compiler and training optimizations

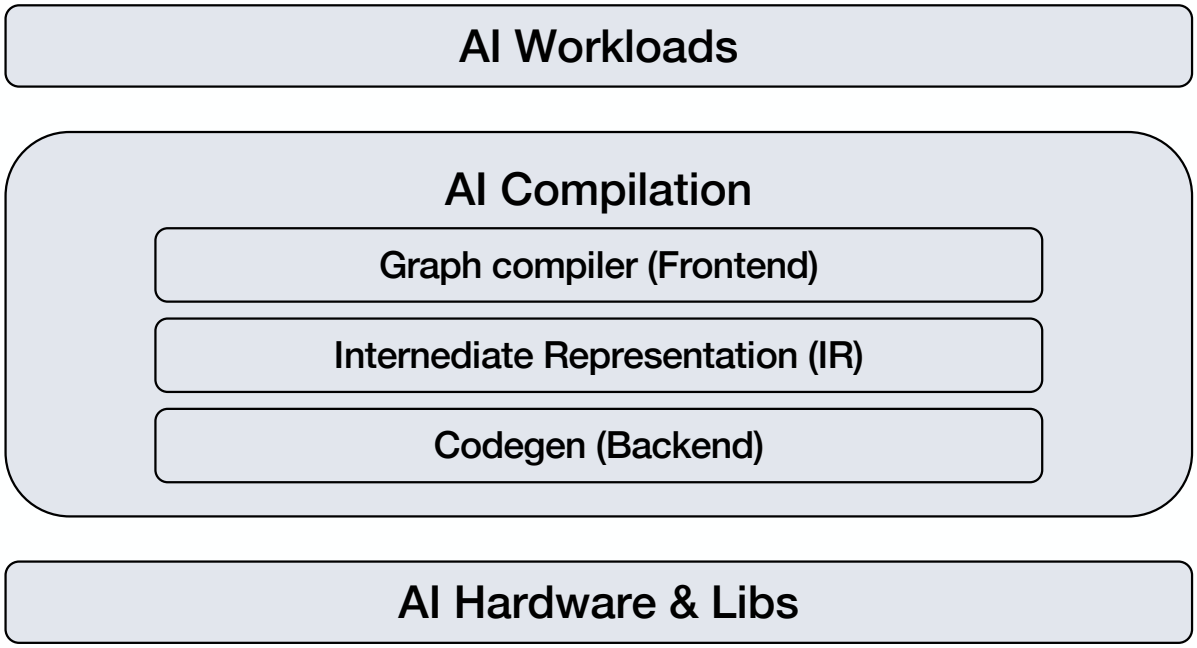
1. What is ByteIR and why?
2. Design + technical details
3. Example, demo, and performance

What is ByteIR

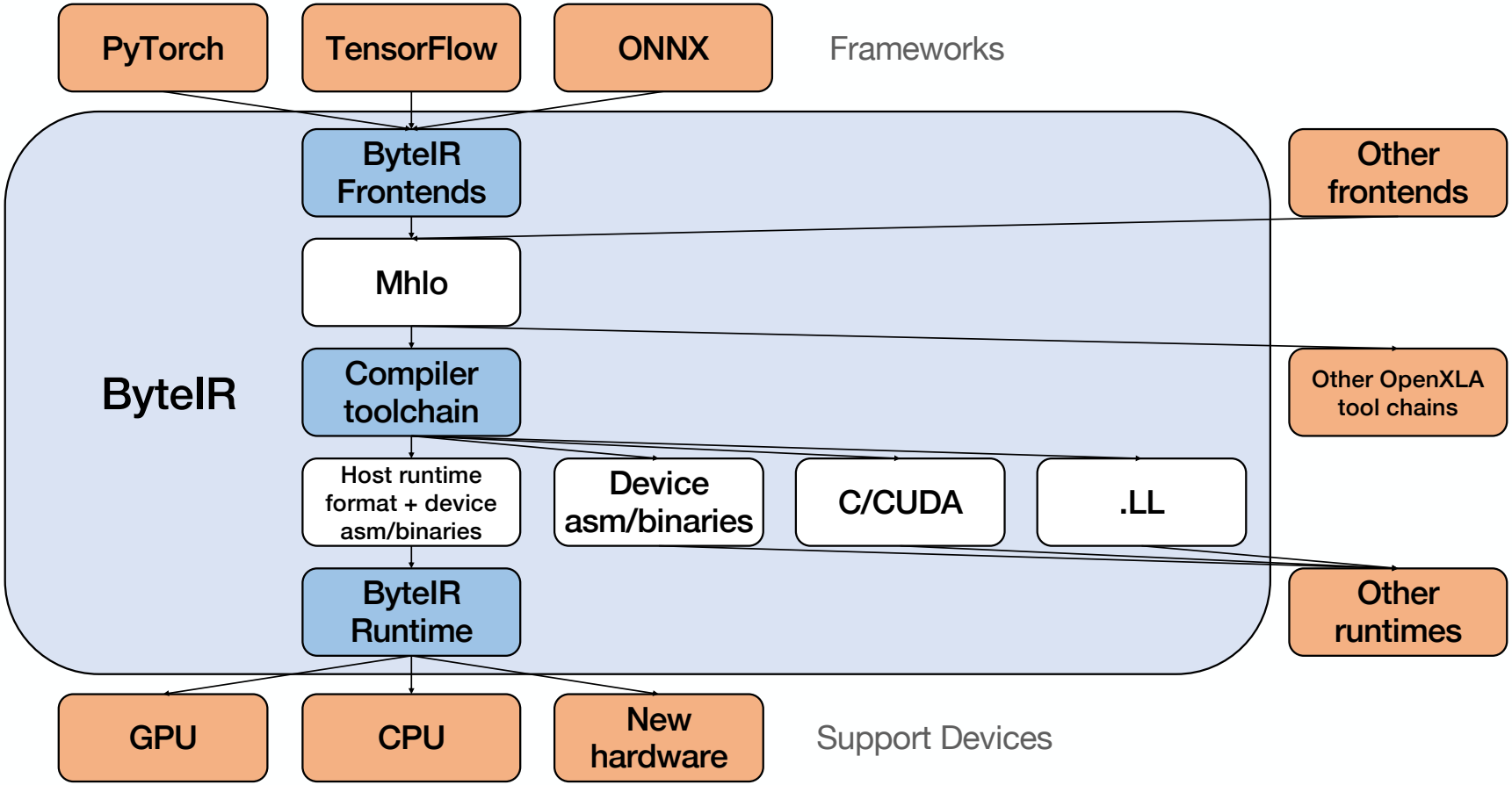
BYTEIR is NOT an IR

BYTEIR is our solution for framework-to-hardware compilation

AI Compilation: NN graph to HW



ByteIR Architecture



Why ByteIR

Extensibility

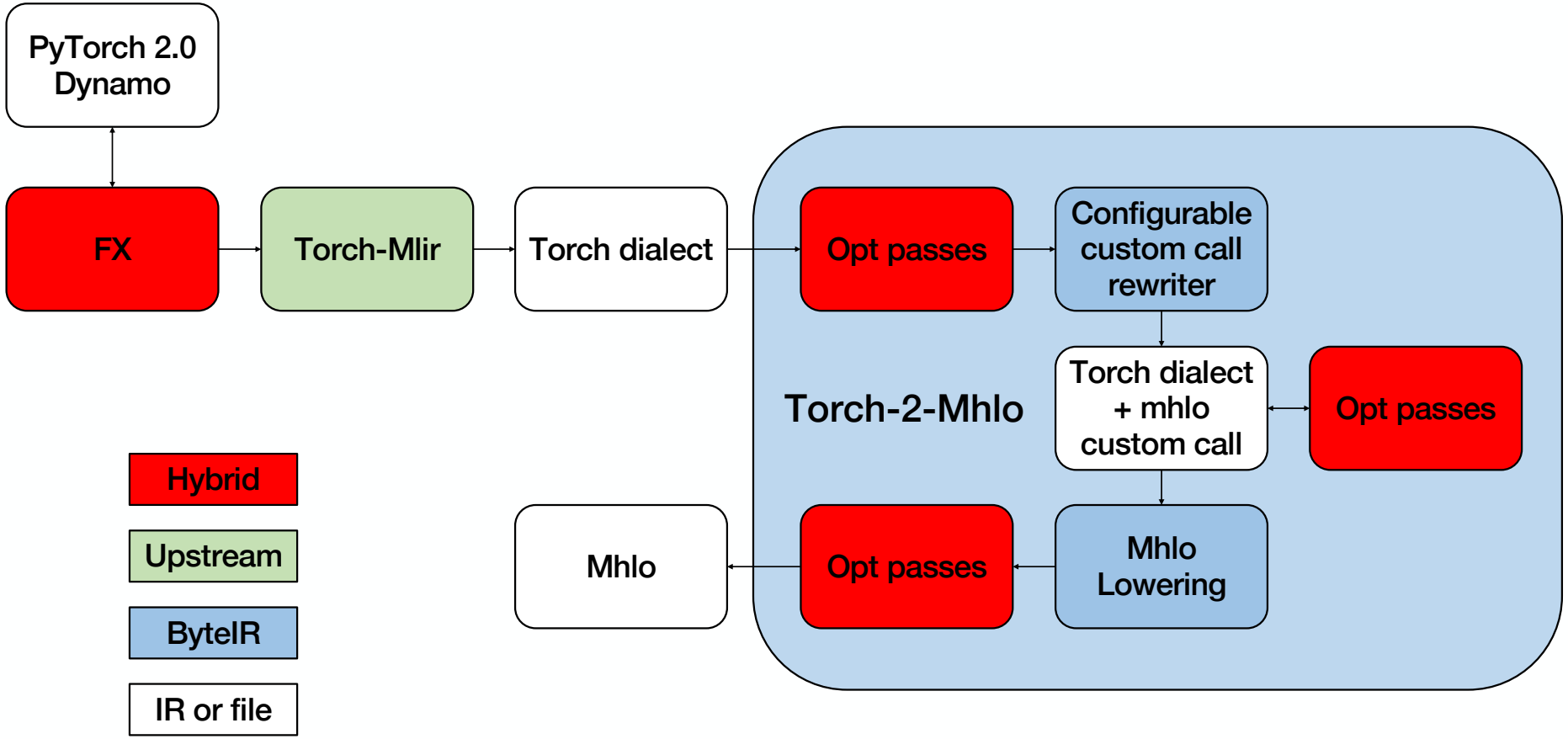
- 3 frontends + N backends
- Pluggable components

Performance

- Mixed optimization passes
- Heterogenous backends

1. What is ByteIR and why?
2. Design + technical details
3. Example, demo, and performance

ByteIR Frontend Overview - PyTorch Example



Torch-Frontend Lowering

torch

```
class Module(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.linear = nn.Linear(10, 20)
    def forward(self, x):
        return self.linear(x)
```

fx graph

```
def forward(self, arg0, arg1):
    mm = torch.ops.aten.mm.default(arg0, arg1);
arg0 = arg1 = None
    return mm
```

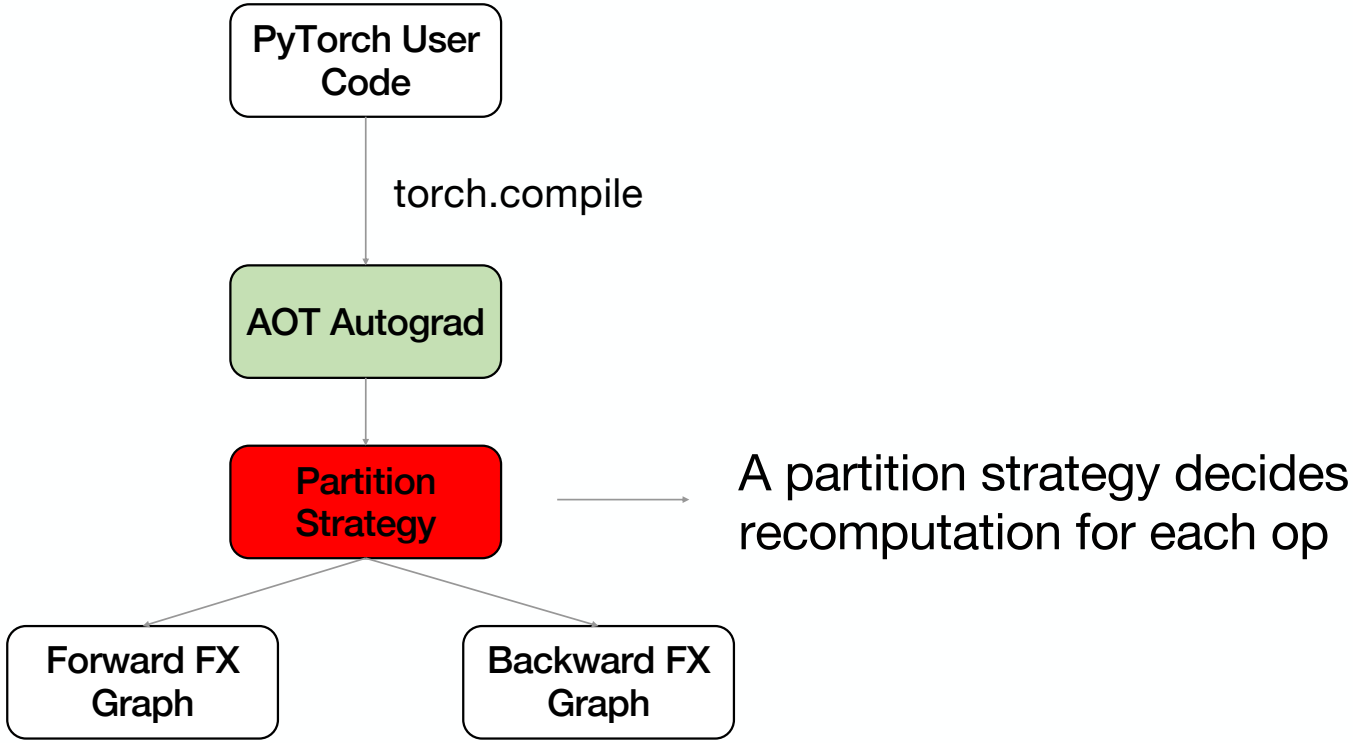
torch dialect

```
func.func @forward(%arg0: !torch.vtensor<[20,
10], f32>, %arg1: !torch.vtensor<[10, 20], f32>) -
> !torch.vtensor<[20, 20], f32> {
    %0 = torch.aten.mm %arg0,
%arg1 : !torch.vtensor<[20, 10],
f32>, !torch.vtensor<[10, 20], f32> -
> !torch.vtensor<[20, 20], f32>
    return %0 : !torch.vtensor<[20, 20], f32>
}
```

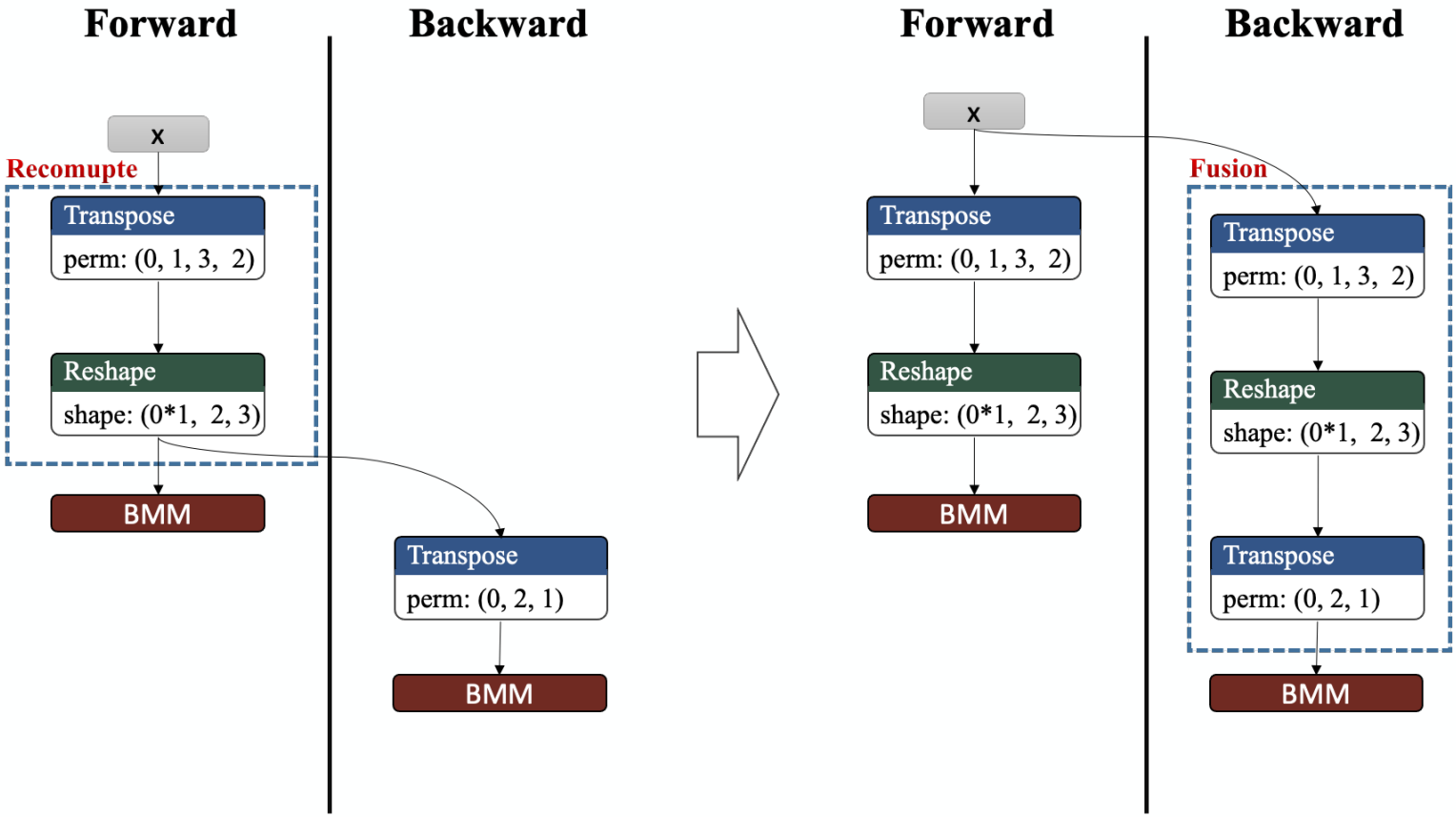
mhlo dialect

```
func.func @forward(%arg0: tensor<20x10xf32>,
%arg1: tensor<10x20xf32>) ->
tensor<20x20xf32> {
    %0 = "mhlo.dot"(%arg0, %arg1) :
tensor<20x10xf32>, tensor<10x20xf32> ->
tensor<20x20xf32>
    return %0 : tensor<20x20xf32>
}
```

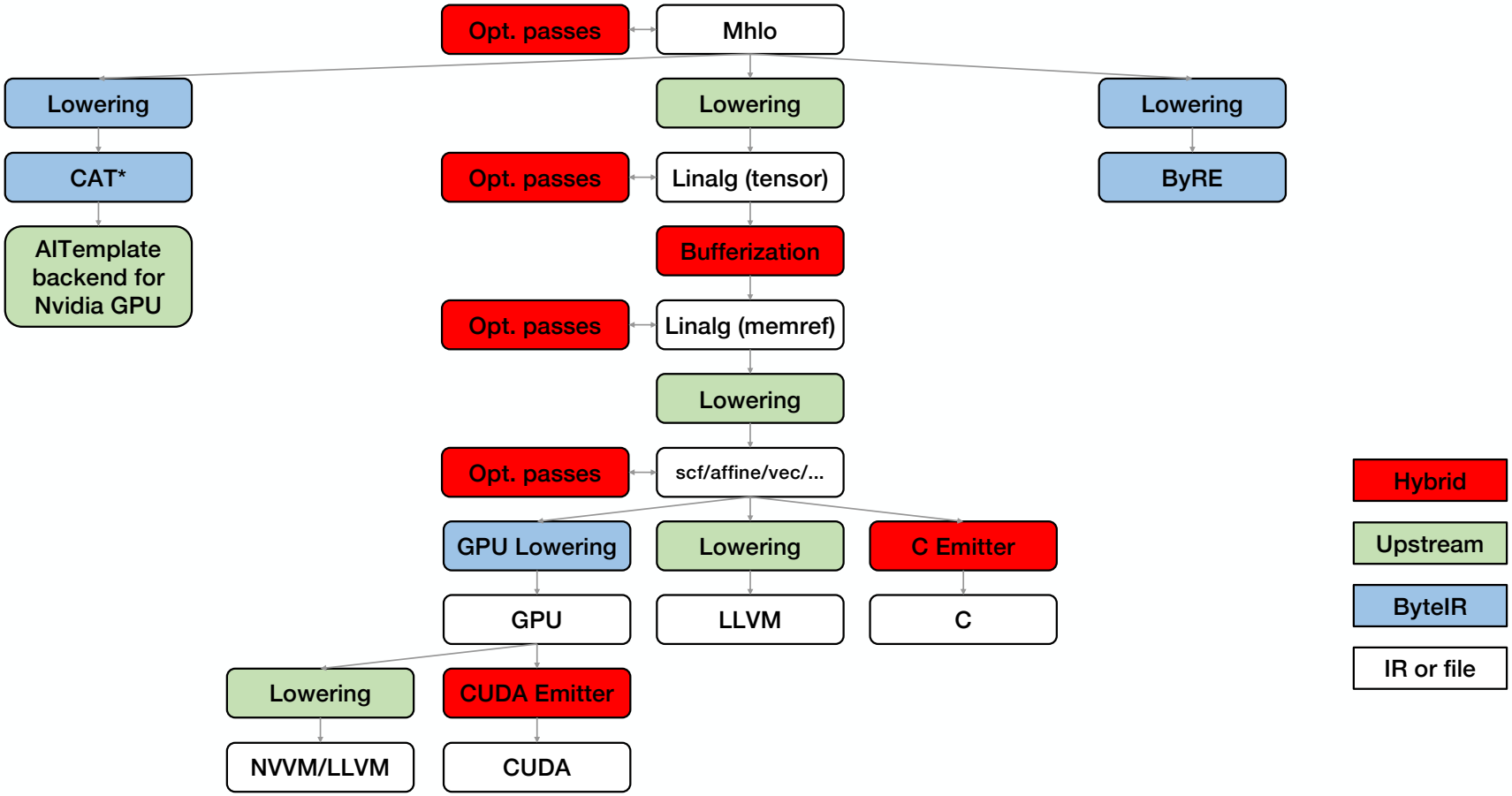
FW/BW Partition in FX



Transpose Elimination in FX

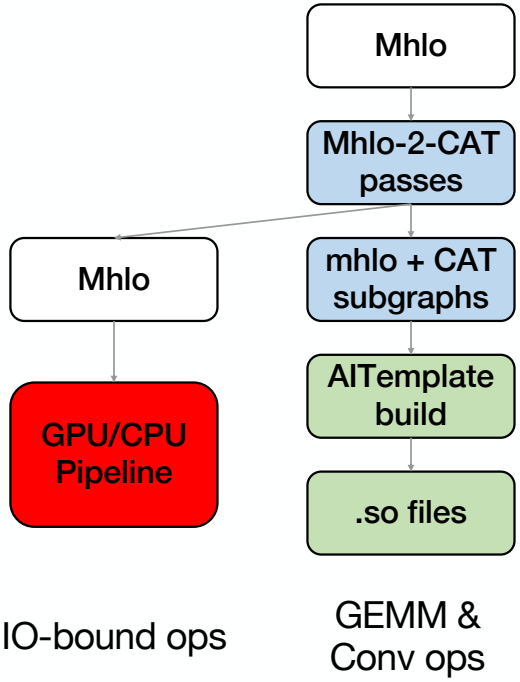


ByteIR Compiler Overview



Integrating AITemplate

We introduce **CAT** (Composable Algebra Template) dialect

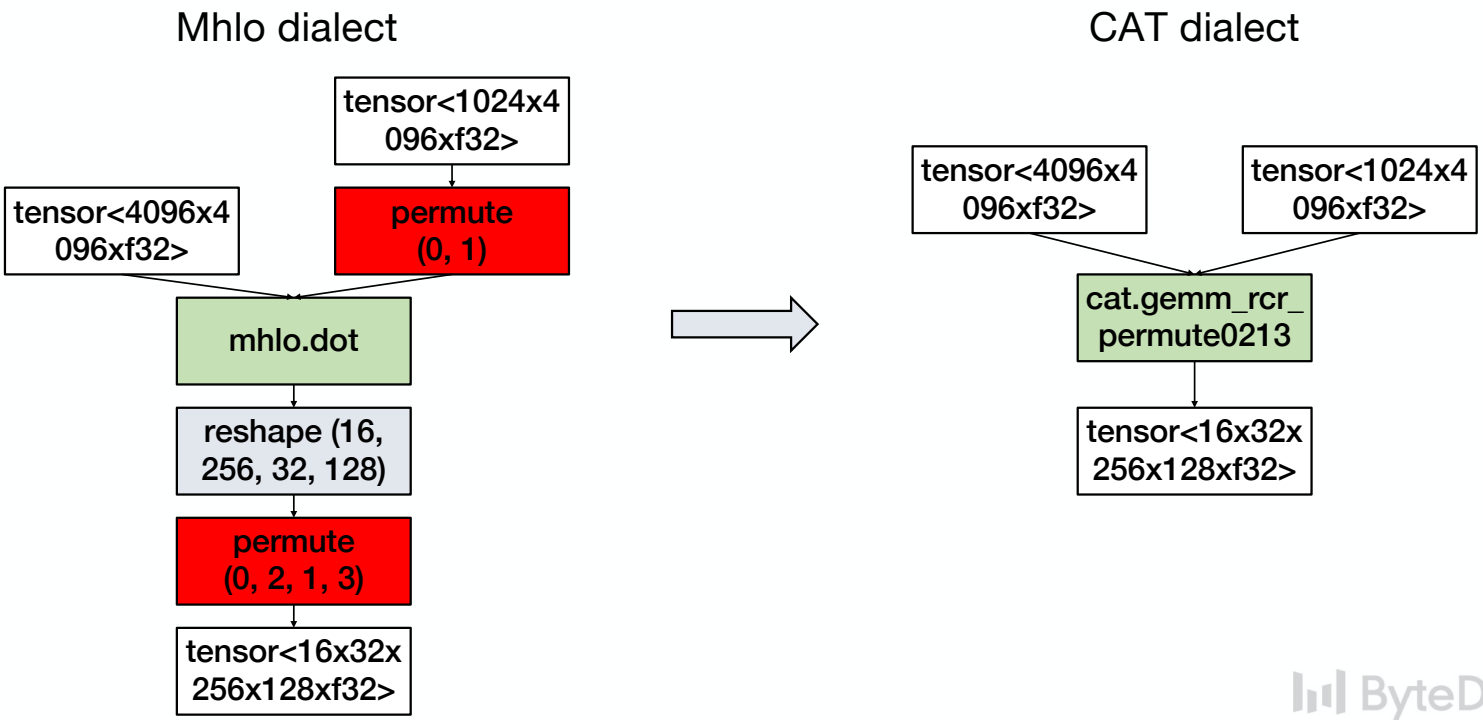


AITemplate: Meta's inference framework converting PyTorch NN to CUDA code (based on CUTLASS)

Mhlo-2-CAT Passes

Mhlo-2-CAT:

- Convert Mhlo ops to CAT ops (one CAT op corresponds to one AIT op)
- Eleminate redundant transpose/permuate ops



Linalg Tiling and Fusion

Mhlo op:

```
func.func @fuse_element(%arg0 ..., %arg1 ...)
    %0 = mhlo.some_elemwise_binary_1(%arg0, %arg1)
    %1 = mhlo.some_elemwise_binary_2(%0, %arg1)
    return %1
```

Linalg transformation

Linalg op:

```
func.func @fuse_element(%arg0 ..., %arg1 ...)
    %1 = linalg.generic {indexing_maps = ...,
           iterator_types = ...} ins(%arg0, %arg1) outs... {
    ^bb0(%in0, %in1, %out)
        %2 = arith.some_elemwise_binary_1(%in0, %in1)
        %3 = arith.some_elemwise_binary_2(%2, %in1)
        linalg.yield %3 ...
    }
    return %1
```

ByetIR's Linalg Extension

More ops:

- Alias, Diag, Scan, Scatter, Softmax, TopK
- support transformations of extended ops

Enhanced fusion transformations:

- producer-consumer & input-sharing fusion
- tiling along reduction axis correction
- intermediates as outputs within fusion
- intermediate tensor dim simplification
- map ops to generic ops conversion
- ...

Other introduced transformations:

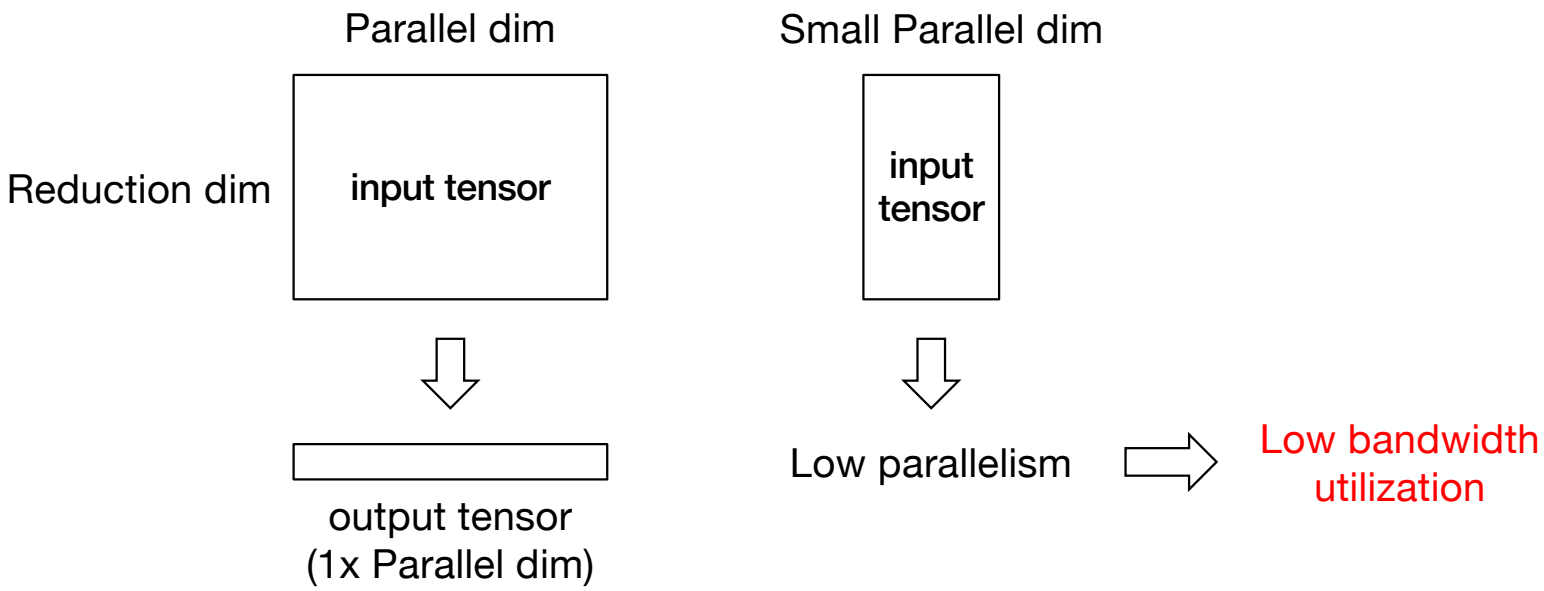
- Collapse dims transformation
- Fuse operands transformation
- ...

Benefits:

- Extreme IO-bound op fusion
- Lower overhead for fused ops (Exploit GPU DRAM bandwidth)

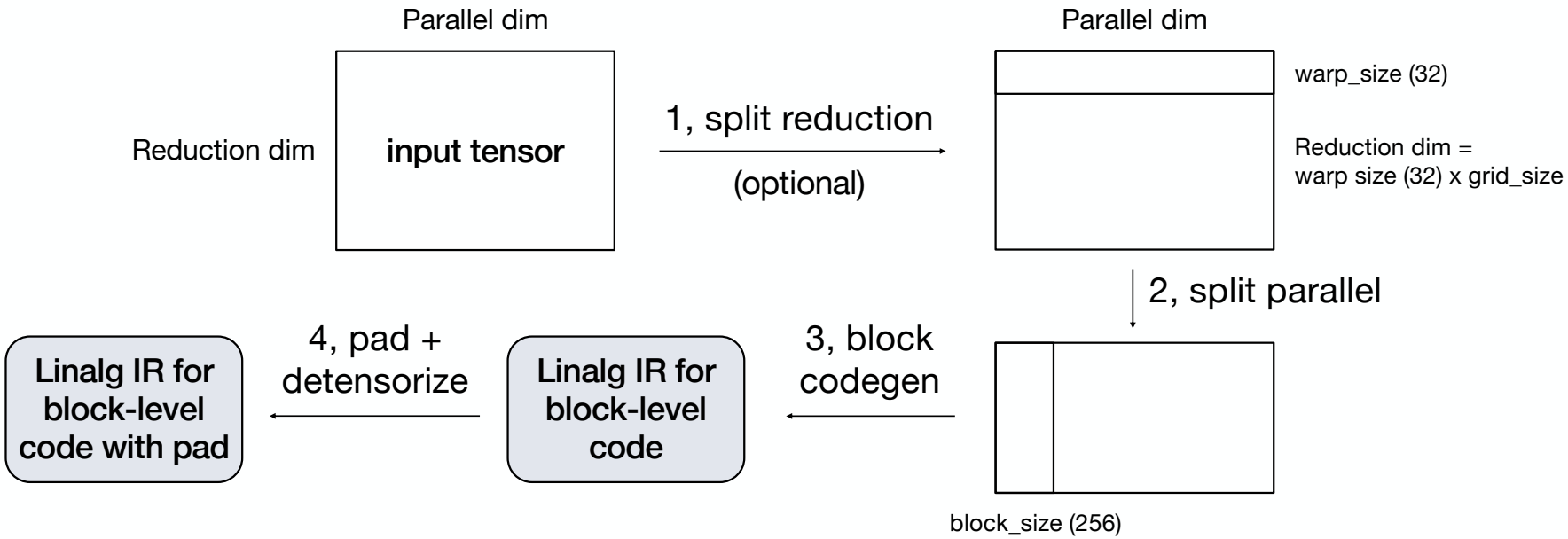
Reduce Op Optimization (Fusion)

Optimization 1: Fusing reduce op with producer ops



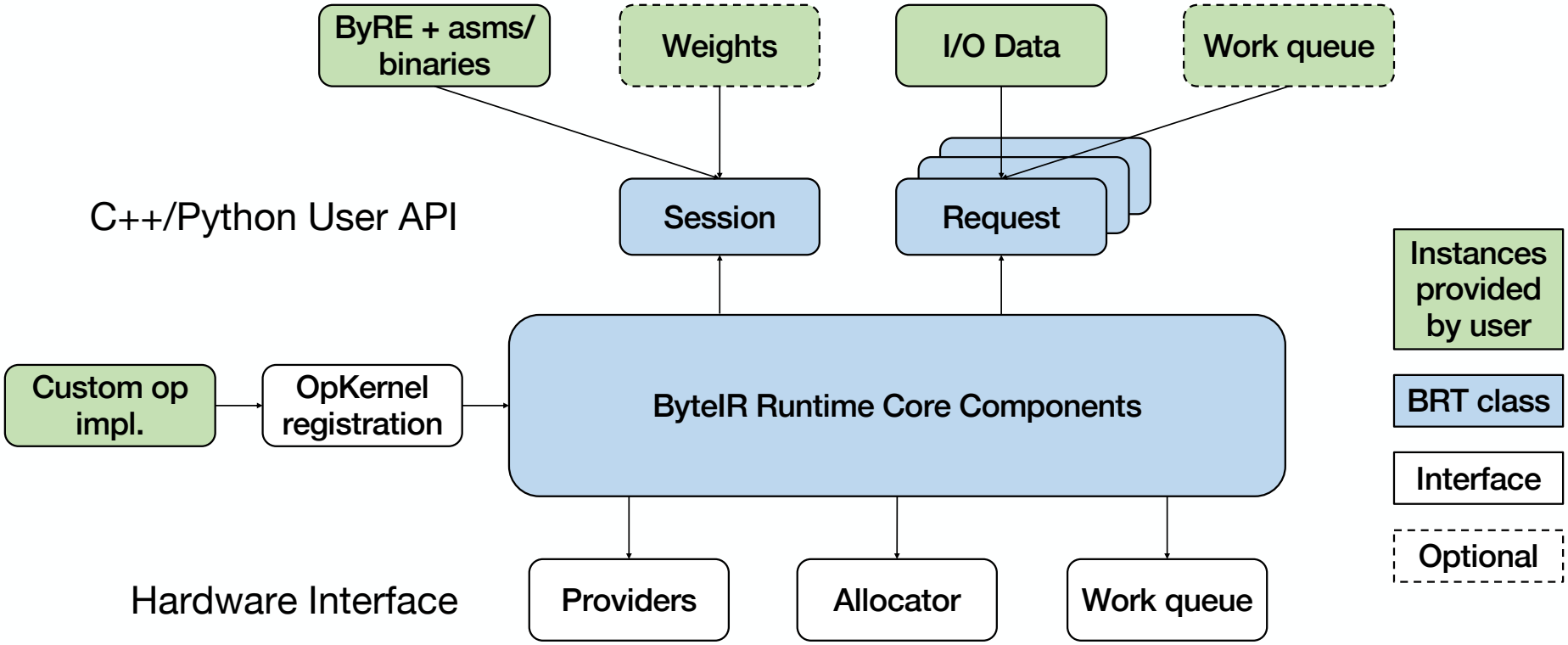
Reduce Op Optimization (Tiling)

Optimization 2: Parallelizing reduce dimension



We use utilize our LinalgExt transformations to achieve best tiling efficiency

ByteIR Runtime (BRT) Overview



BRT Interface for Hardware

Provider	A collection of op implementation • e.g., mm, d2h/h2d memcpy
Work Queue	Abstraction for execution order (like CUDAStream)
Allocator	Memory Allocate/Free

1. What is ByteIR and why?
2. Design + technical details
3. Example, demo, and performance

ByteDance字节跳动

ByteIR PyTorch Example

```
from byteir import byteir_compile_fx

model = make_model(model_name)

# 1, compile with byteir
optimized_model = torch.compile(model, backend=byteir_compile_fx)

# 2, execution as usual
data = make_data(optimized_model, model_name, device)
model.zero_grad(set_to_none=True)
with torch.cuda.amp.autocast(enabled=True, dtype=torch.float16):
    # forward compile
    loss = compute_loss(optimized_model, data)
    # backward compile
    loss.backward()
```

BRT PyTorch Example

```
import brt

session = brt.Session()
session.load(byre_model_path)
req = session.new_request_context(torch.cuda.current_stream())
inputs, outputs = [], []

# init input/output data
for offset in session.get_input_arg_offsets():
    inputs.append(torch.randn(session.get_static_shape(offset),
                              dtype=dtype, device="cuda"))
    req.bind_arg(offset, inputs[-1].data_ptr())
...

req.finish_io_binding()
req.run()
req.sync()
```

ByteIR Training & Inference Demo

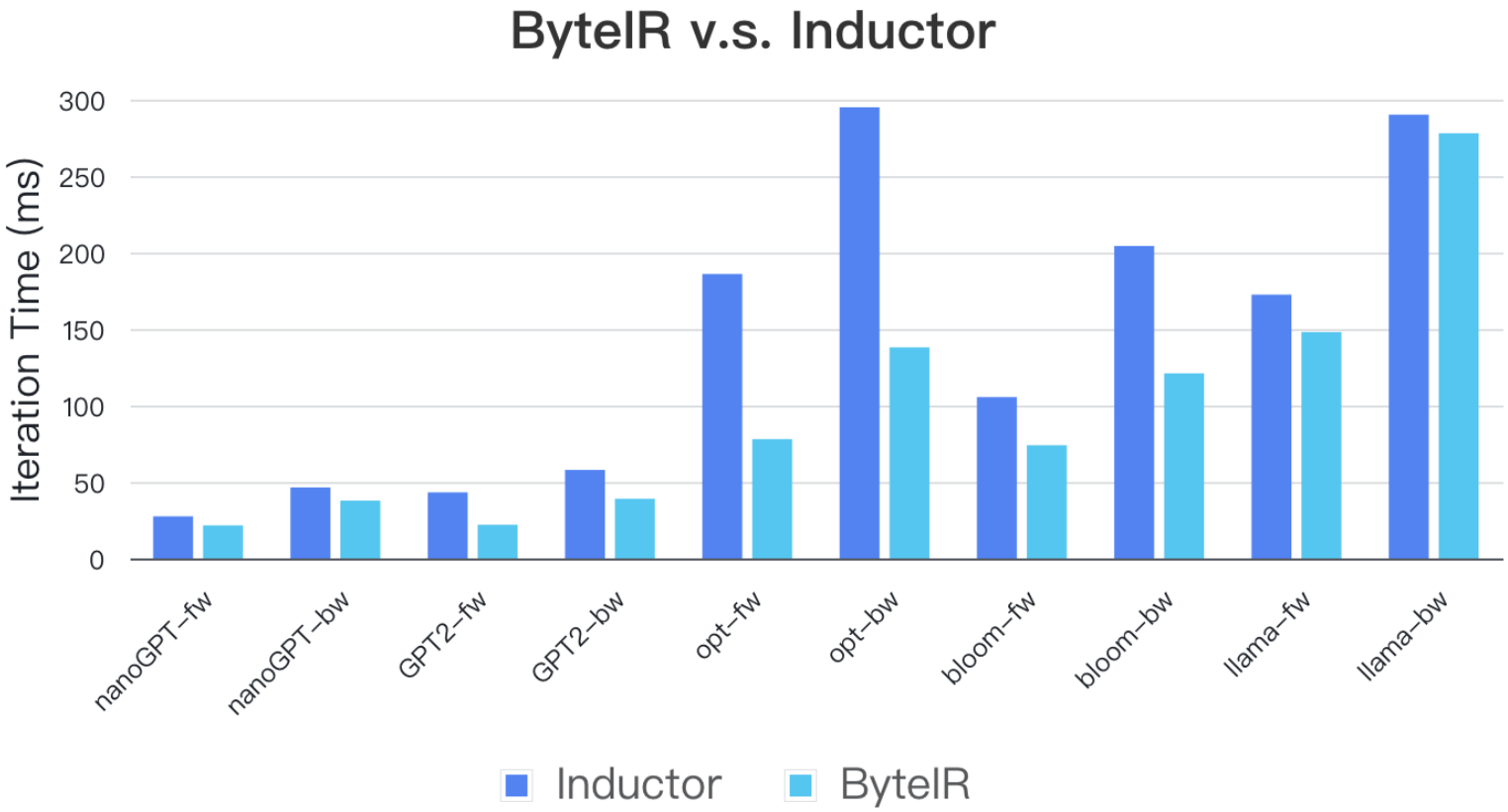
```
import transformers

llama_config = transformers.LlamaConfig(num_hidden_layers=4, ...)
llama_model = transformers.LlamaForCausalLM(config=self.config)

import torch_frontend
import byteir
# compile
fx_module = torch_frontend.preprocess_fx_graph(make_fx(llama_model))
mhlo_model = torch_frontend.compile(module, mhlo_file, ...)
# weights stored in mhlo.constant
byteir.compile(mhlo_file, byre_file, ...)

# run with brt given byre_file
import brt
...
```

Performance



Conclusion & Takeaways

We present ByteIR: a framework-to-hardware compiler solution that is

- Extensibility:
  - 3 frontends + N backends
  - pluggable components with C++/Python APIs
- High Performance: heterogenous backends

Website: https://byteir.ai

Github: https://github.com/bytedance/byteir

THANKS

ByteDance字节跳动

https://byteir.ai

my email: serailhydra@gmail.com
wechat: darkmoor_1121