# AOTInductor

Bin Bao (PyTorch Compiler, Meta)

# Objective

- Run PyTorch Model inference in a Python-less environment
- Take torch.export-ed IR and Ahead-Of-Time compile into a shared library

# Design Principle

- Follow JIT TorchInductor as much as possible
  - Keep 1:1 functionality mapping in codegen (explained later with examples)
  - Reduce errors
  - Leverage optimizations
- Support both GPU and CPU
  - Use GPU backend example in this talk

# Example

| Input Code | Output Triton Kernel | Output Python Wrapper (JIT) | Output cpp wrapper (AOT) |
|---|---|---|---|

```
class Model(torch.nn.Module):
    def __init__(self):
        super().__init__()

    def forward(self, x, y):
        a = torch.sin(x)
        b = torch.mm(a, y)
        c = torch.cos(b)
        return c
```

# Example

| Input Code | Output Triton Kernel | Output Python Wrapper (JIT) | Output cpp wrapper (AOT) |
|---|---|---|---|

```
@triton.jit
def triton_(in_ptr0, out_ptr0, xnumel, XBLOCK : tl.constexpr):
    xnumel = 100
    xoffset = tl.program_id(0) * XBLOCK
    xindex = xoffset + tl.arange(0, XBLOCK)[:]
    xmask = xindex < xnumel
    x0 = xindex
    tmp0 = tl.load(in_ptr0 + (x0), xmask)
    tmp1 = tl_math.sin(tmp0)
    tl.store(out_ptr0 + (x0), tmp1, xmask)


# Triton kernels will be compiled and stored as .cubin files by AOTInductor
```

# Example

| Input Code | Output Triton Kernel | Output Python Wrapper (JIT) | Output cpp wrapper (AOT) |
|---|---|---|---|

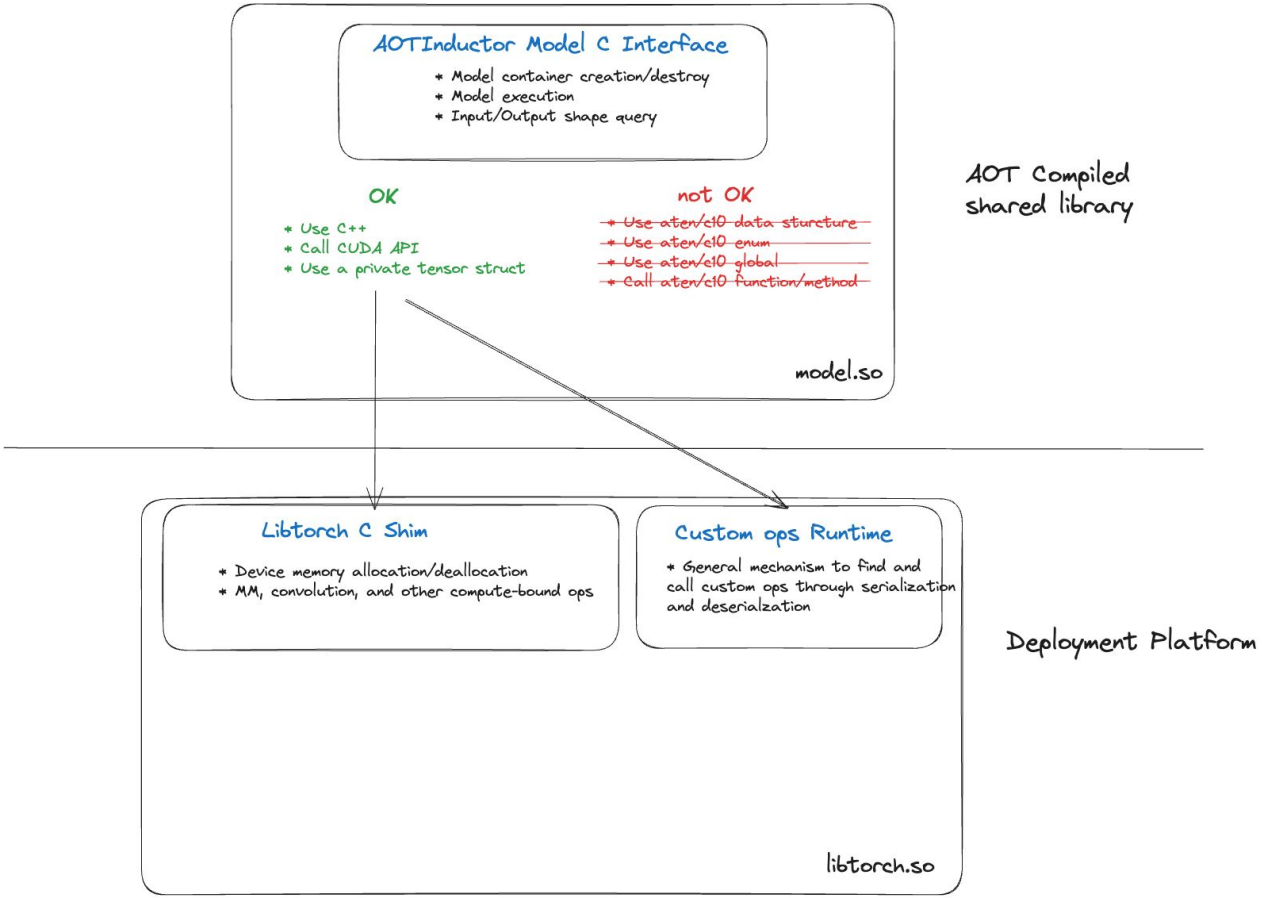
```
def call(args):
    …
    stream0 = get_cuda_stream(0)
    buf0 = empty_strided((64, 64), (64, 1), device='cuda', dtype=torch.float32)
    triton_poi_fused_sin_0.run(arg0_1, buf0, 4096, grid=grid(4096), stream=stream0)
    …
    extern_kernels.mm(buf0, arg1_1, out=buf1)
    …
    buf2 = buf1; del buf1  # reuse
    …
```

# Example

| Input Code | Output Triton Kernel | Output Python Wrapper (JIT) | Output cpp wrapper (AOT) |
|---|---|---|---|

```
at::cuda::CUDAStreamGuard stream_guard(at::cuda::getStreamFromExternal(stream, …);

auto buf0 = at::empty_strided({64L, 64L}, …);

if (triton_poi_fused_sin_0 == nullptr) {

     triton_poi_fused_sin_0 = loadKernel(…); // Triton kernels are compiled and stored as .cubin files

}...

launchKernel(triton_poi_fused_sin_0, …);

…

at::mm_out(buf1, buf0, arg1_1);

…

decltyp(auto) buf2 = buf1; buf1.reset(); // reuse

…
```

# ABI Compatibility Requirement



# C Shim

- Pass Tensor pointers across boundary
  - Ownership Management
    - Tensor storage: managed by reference counting
    - Tensor object: managed by unique pointer
- Stable C interface shim layer
  - Utility functions
  - Aten ops as fallback
- Custom ops
  - Serialization at compile time and deserialization at run time

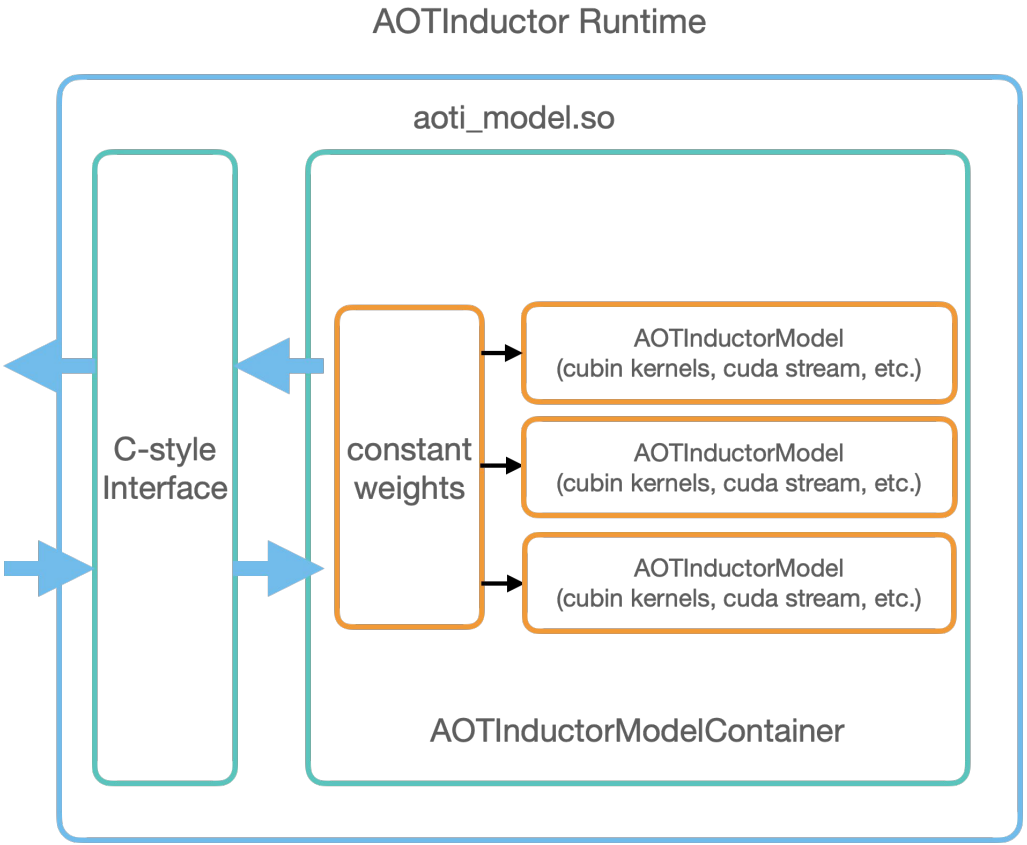
# Example

| Input Code | Output Triton Kernel | Output Python Wrapper (JIT) | Output cpp wrapper (AOT, ABI-compatible) |
|---|---|---|---|

```
AOTICudaStreamGuard stream_guard(stream, this->device_idx_);
…
AOTI_TORCH_ERROR_CODE_CHECK(aoti_torch_empty_strided(2, …, &buf0_handle));
RAIIAtenTensorHandle buf0(buf0_handle);
…
launchKernel(kernels.triton_poi_fused_sin_0, …, stream);
…
AOTI_TORCH_ERROR_CODE_CHECK(aoti_torch_cuda_mm_out(buf1,buf0,arg1_1));
…
auto buf2 = std::move(buf1);  // reuse
…
```

# Runtime Component

- Multithreaded model serving
- Weights loading and sharing



# Demo

- AOTInductor_example.ipynb