上海创智学院
Shanghai Innovation Institute

TVM FFI

Open ABI and FFI for Machine Learning Systems

THE APACHE SOFTWARE FOUNDATION

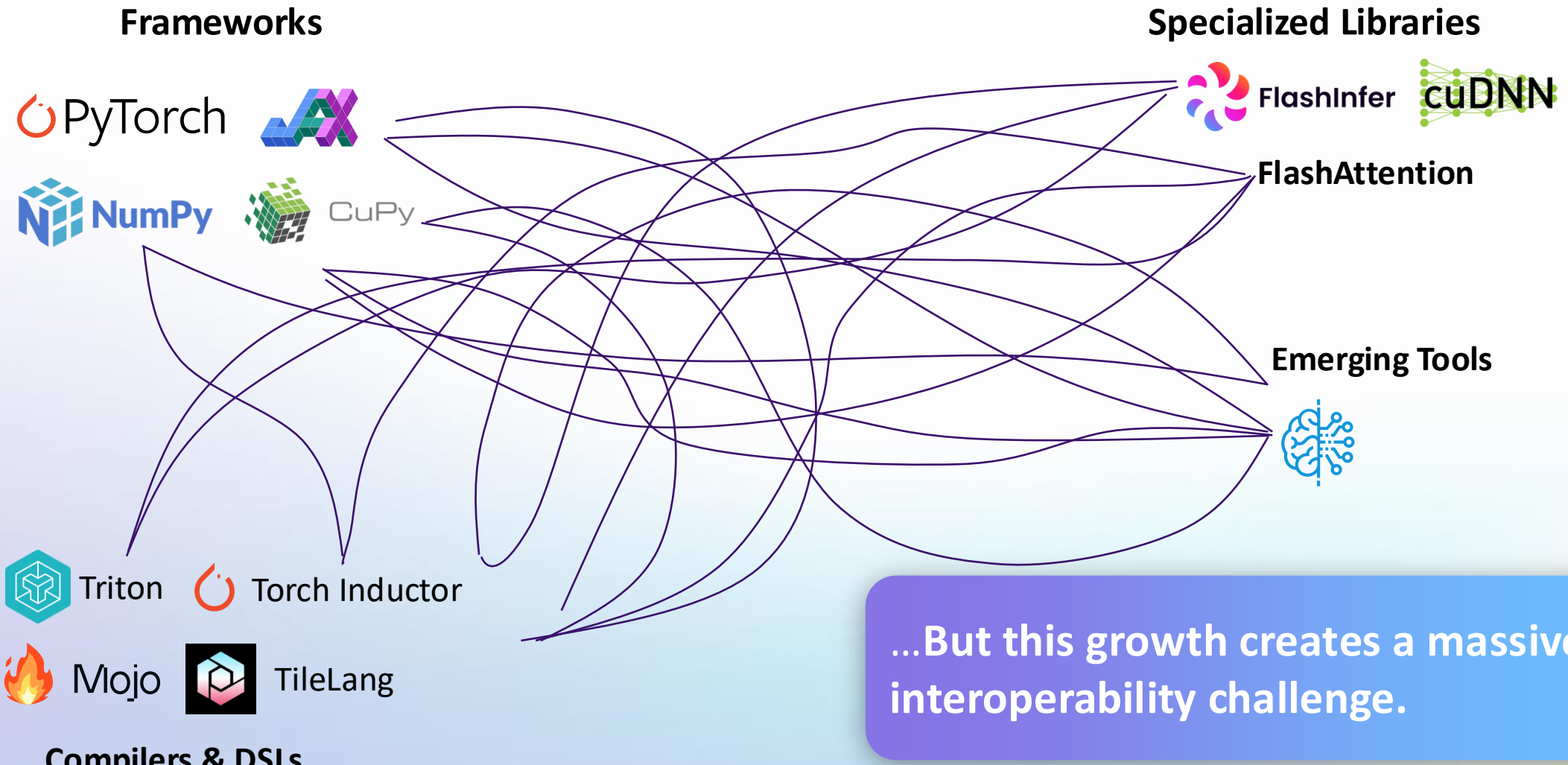
tvm

Siyuan Feng | 2025.12.27

SHANGHAI INNOVATION INSTITUTE

The ML Ecosystem is Exploding with Innovation..

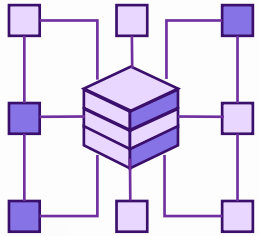
上海创智学院
Shanghai Innovation Institute



...But this growth creates a massive interoperability challenge.

The Interop Challenge Lives at the Core: ABI and FFI

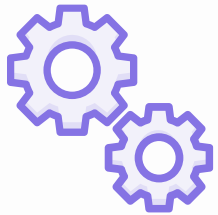
上海创智学院
Shanghai Innovation Institute



ABI- Application Binary Interface

What it is: Defines how data is stored in memory and what happens when a function is called.

The Problem: "You can't just pass a `torch.Tensor` pointer and expect it to work as a `cupy.NDArray`



FFI - Foreign Function Interface

What it is: The mechanism for cross-language communication, essential for ML (e.g., Python calling C++).

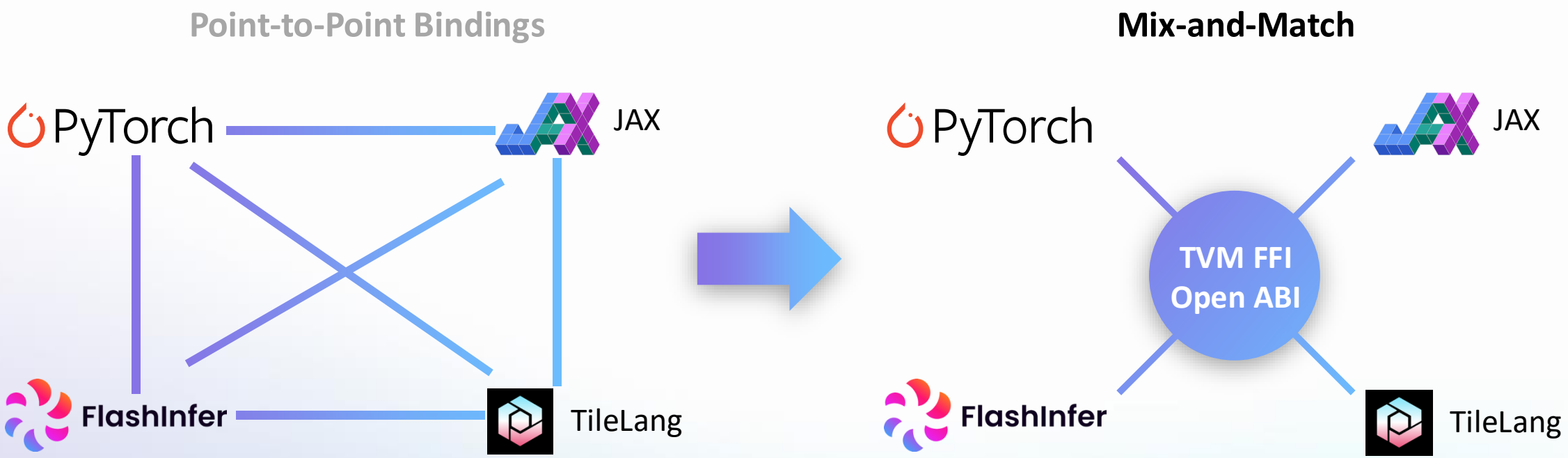
The Problem: Every DSL library, and framework needs custom runtime bindings for every environment, leading to an N*M explosion of fragile connectors.

Our take: Diversity is fundamental and here to stay.

We need sustainable interoperability, not another framework.

From N*M Point-to-Point Chaos to a Universal Hub

上海创智学院
Shanghai Innovation Institute



TVM FFI is an **open, standalone ABI and FFI for machine learning.** It's not another compiler or framework; it's the common ground that lets all other components amplify each other.

The Bedrock: A Stable C ABI

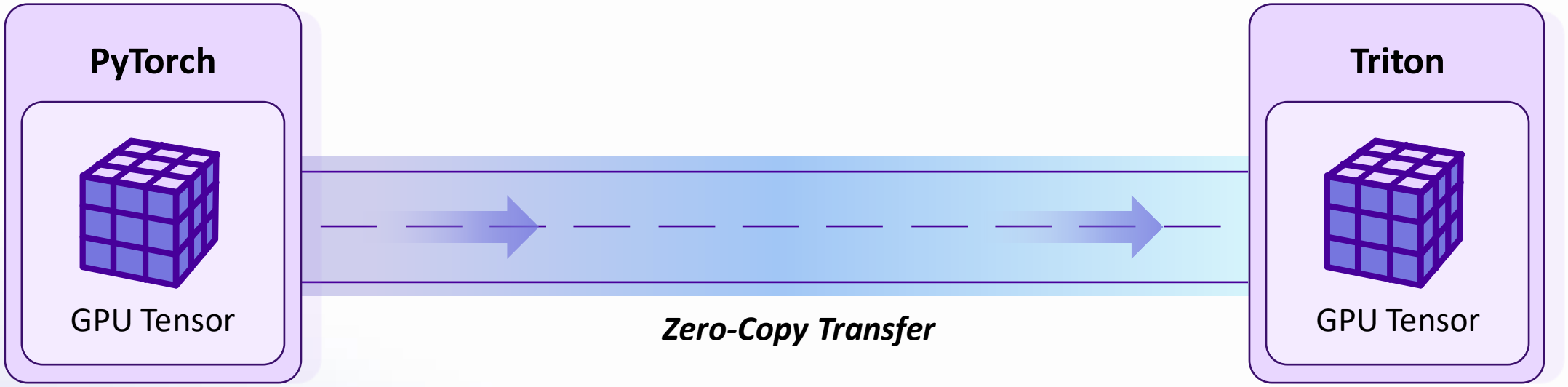
上海创智学院
Shanghai Innovation Institute

C is the *lingua franca* of programming languages. Building on it ensures maximum compatibility and long-term stability.

- **Decouples from Python:** Your compiled library is agnostic to Python's ABI and version changes.
- **Decouples from Frameworks:** Your library doesn‘t need to link against PyTorch, JAX, etc.
- **Enables True Portability:** Naturally enables cross-language bindings because every language speaks C
- **Minimal & Stable:** Provides a minimal set of stable C APIs, making it a reliable target for compilers, runtimes, and libraries.

The Data Highway: Zero-Copy Tensors with DLPack

上海创智学院
Shanghai Innovation Institute



For ML systems, performance is everything. DLPack is the community-driven standard that allows massive GPU tensors to be shared between frameworks without expensive memory copies.

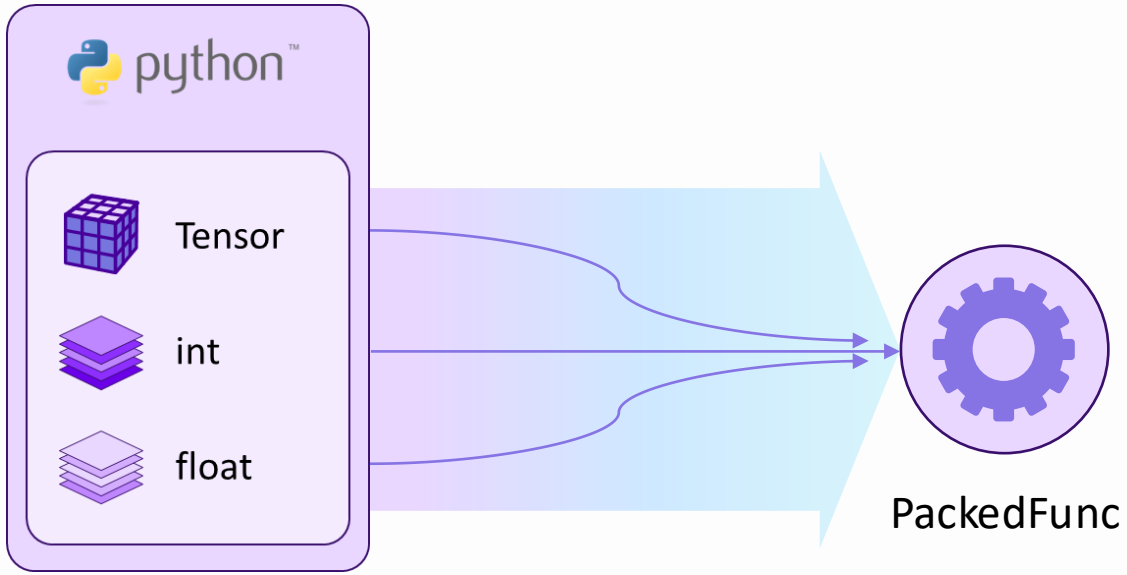
PyTorch JAX NumPy CuPy

First-class support for `torch.Tensor`. It's automatically and transparently converted for zero-copy transfer, and the CUDA stream context is carried over seamlessly. It feels like a native PyTorch function call.

The Engine: Low-Overhead Packed Calls

上海创智学院
Shanghai Innovation Institute

A single, standard C function signature ('TVMFFISafeCallType') is used for all functions in a "type-erased" way. This avoids the need for declaring and JIT-ing a unique shim for every FFI function.



Flexibility: Easily handles calls from dynamic languages (Python) and static languages (C++, Rust).

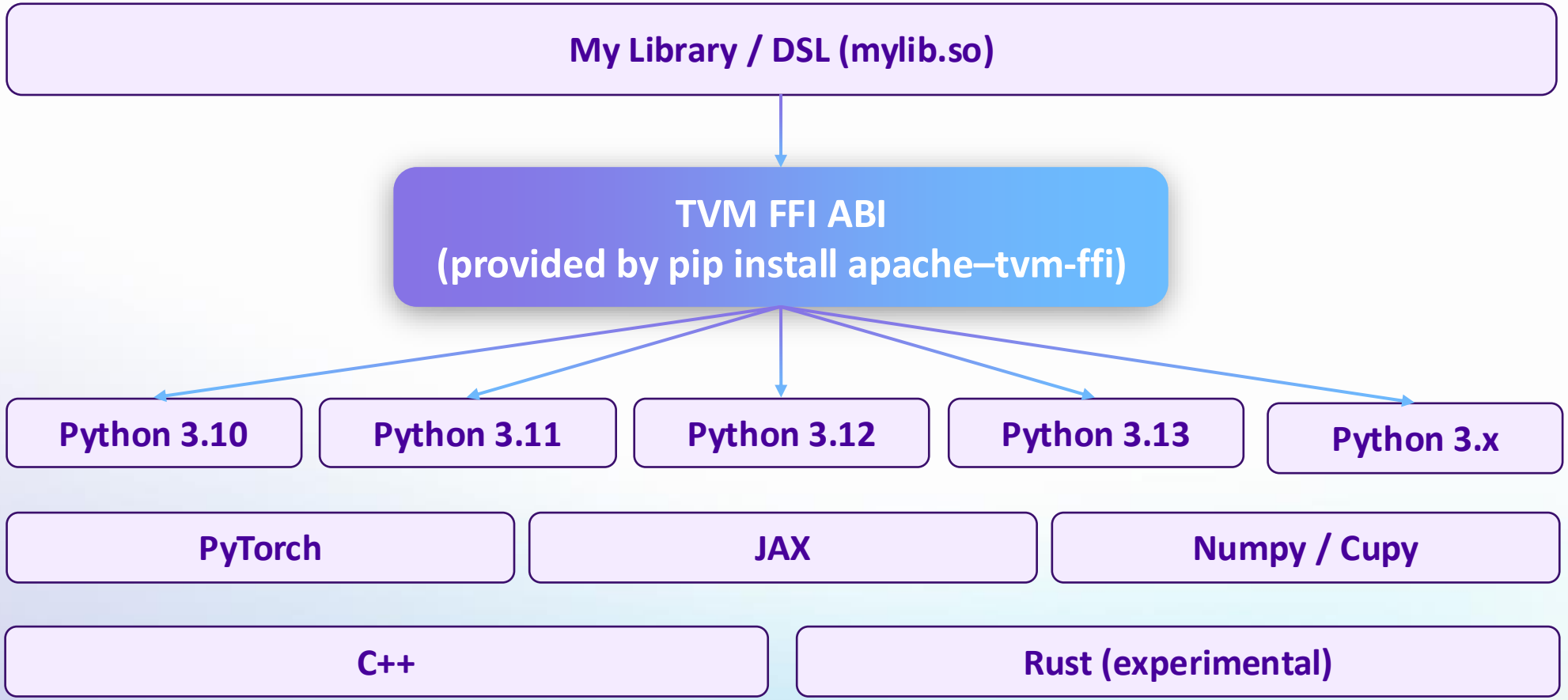
Performance: low overhead for Python-to-C++ calls.

Efficiency: Arguments are packed on the stack, avoiding heap allocation for calls.

Robustness: Built-in, thread-local error handling that propagates exceptions and tracebacks across the FFI boundary.

The Ultimate Payoff: Ship One Wheel

上海创智学院
Shanghai Innovation Institute



One compiled library. Works across all frameworks and Python versions.

Minimal example for Libraries

上海创智学院
Shanghai Innovation Institute

Libraries

```cpp
#include <tvm/ffi/container/tensor.h>
#include <tvm/ffi/function.h>

void AddOne(
  tvm::ffi::TensorView x,
  tvm::ffi::TensorView y,
) {
  int64_t n = x.size(0);
  float* x_data = static_cast<float*>(x.data_ptr());
  float* y_data = static_cast<float*>(y.data_ptr());
  for (int64_t i = 0; i < n; ++i) {
    y_data[i] = x_data[i] + 1;
  }
}

TVM_FFI_DLL_EXPORT_TYPED_FUNC(add_one_cpu, AddOne);
```

Python Caller

```python
import tvm_ffi
mod = tvm_ffi.load_module("add_one_cpu.so")

import torch
x = torch.tensor([1, 2, 3, 4, 5])
y = torch.empty_like(x)
mod.add_one_cpu(x, y)
print(y)
```

C++ Caller

```cpp
#include <tvm/ffi/container/tensor.h>
#include <tvm/ffi/extra/module.h>

int main() {
  ffi::Tensor x = Alloc1DTensor({1, 2, 3, 4, 5});
  ffi::Tensor y = Alloc1DTensor({0, 0, 0, 0, 0});
  // Load shared library `add_one_cpu.so`
  ffi::Module mod =
ffi::Module::LoadFromFile("add_one_cpu.so");
  // Look up `add_one_cpu` function
  ffi::Function add_one_cpu = mod-
>GetFunction("add_one_cpu").value();
  // Call the function
  add_one_cpu(x, y);
}
```

Minimal C ABI as for DSL Compilers

上海创智学院
Shanghai Innovation Institute

```c
#include <tvm/ffi/c_api.h>
#include <tvm/ffi/extra/c_env_api.h>

TVM_FFI_DLL_EXPORT int __tvm_ffi_add_one_cpu(
  void* handle, const TVMFFIAny* args,
  int32_t num_args, TVMFFIAny* result
) {
  // Step 1.1. Extract `x := args[0]`
  DLTensor* x;
  if (args[0].type_index == kTVMFFIDLTensorPtr)
    x = (DLTensor*)(args[0].v_ptr);
  else if (args[0].type_index == kTVMFFITensor)
    x = (DLTensor*)(args[0].v_c_str + sizeof(TVMFFIObject));
  else {
    TVMFFIErrorSetRaisedFromCStr("ValueError", "Expects a Tensor input");
    return -1;
  }
  // Step 1.2. Extract `y := args[1]`
  DLTensor* y;
  if (args[1].type_index == kTVMFFIDLTensorPtr)
    y = (DLTensor*)(args[1].v_ptr);
  else if (args[1].type_index == kTVMFFITensor)
    y = (DLTensor*)(args[1].v_c_str + sizeof(TVMFFIObject));
  else {
    TVMFFIErrorSetRaisedFromCStr("ValueError", "Expects a Tensor output");
    return -1;
  }
  // Step 2. Perform add one: y = x + 1
  for (int64_t i = 0; i < x->shape[0]; ++i) {
    ((float*)y->data)[i] = ((float*)x->data)[i] + 1.0f;
  }
  // Step 3. Return error code 0 (success)
  return 0;
}
```

```c
int Run(DLTensor* x, DLTensor* y) {
  int ret_code = 0;
  TVMFFIAny call_args[3] = {};
  TVMFFIAny mod = {.type_index = kTVMFFINone, .v_obj = NULL};
  TVMFFIAny func = {.type_index = kTVMFFINone, .v_obj = NULL};
  TVMFFIAny none = {.type_index = kTVMFFINone};  // ignore the return value

  // Step 1. Load module
  // Equivalent to:
  //    mod = tvm::ffi::Module::LoadFromFile("/add_one_cpu.so")
  call_args[0] = (TVMFFIAny){.type_index = kTVMFFIRawStr, .v_c_str = "add_one_cpu.so"};
  call_args[1] = (TVMFFIAny){.type_index = kTVMFFISmallStr, .v_int64 = 0};
  if ((ret_code = TVMFFIFunctionCall(fn_load_module, call_args, 2, &mod))) goto _RAII;

  // Step 2. Get function `add_one_cpu` from module
  // Equivalent to:
  //    func = mod->GetFunction("add_one_cpu", /*query_imports=*/false).value()
  call_args[0] = (TVMFFIAny){.type_index = mod.type_index, .v_obj = mod.v_obj};
  call_args[1] = (TVMFFIAny){.type_index = kTVMFFIRawStr, .v_c_str = "add_one_cpu"};
  call_args[2] = (TVMFFIAny){.type_index = kTVMFFIBool, .v_int64 = 0};
  if ((ret_code = TVMFFIFunctionCall(fn_get_function, call_args, 3, &func))) goto _RAII;

  // Step 3. Call function `add_one_cpu(x, y)`
  // Equivalent to:
  //    func(x, y)
  call_args[0] = (TVMFFIAny){.type_index = kTVMFFIDLTensorPtr, .v_ptr = x};
  call_args[1] = (TVMFFIAny){.type_index = kTVMFFIDLTensorPtr, .v_ptr = y};
  if ((ret_code = TVMFFIFunctionCall(func.v_ptr, call_args, 2, &none))) goto _RAII;

_RAII:
  if (mod.type_index >= kTVMFFIObject) TVMFFIObjectDecRef(mod.v_obj);
  if (func.type_index >= kTVMFFIObject) TVMFFIObjectDecRef(func.v_obj);
  if (none.type_index >= kTVMFFIObject) TVMFFIObjectDecRef(none.v_obj);
  return ret_code;
}
```

Inline Kernel and Agentic Flow

上海创智学院
Shanghai Innovation Institute

```python
import torch
import tvm_ffi.cpp

mod = tvm_ffi.cpp.load_inline(
  cuda_sources=r"""
    __global__ void AddOneKernel(float* x, float* y, int n) {
      // Kernel Impl ...
    }

    void add_one_cuda(tvm::ffi::TensorView x, tvm::ffi::TensorView y) {
      TVM_FFI_ICHECK(x.ndim() == 1) << "x must be a 1D tensor";
      ... // implementation of a library function
      int64_t nthread_per_block = 256, n = x.size(0);
      int64_t nblock = (n + nthread_per_block - 1) / nthread_per_block;
      cudaStream_t stream = static_cast<cudaStream_t>(
          TVMFFIEnvGetStream(x.device().device_type, x.device().device_id));
      AddOneKernel<<<nblock, nthread_per_block, 0, stream>>>(
        static_cast<float*>(x.data_ptr()), static_cast<float*>(y.data_ptr()), n
      );
    }
  """,
  functions=["add_one_cuda"],
)

x = torch.tensor([1, 2, 3, 4, 5], dtype=torch.float32).cuda()
y = torch.empty_like(x)
mod.add_one_cuda(x, y)
```

Directly compile and load inline a module in python.

Useful for coding agents to generate kernels in the loop.

Kernel Implementation

Host code for launching kernel

Automatically capture current cuda stream

Function call with native torch tensor

CuteDSL Integration: Native TVM-FFI

上海创智学院
Shanghai Innovation Institute

```python
@cute.kernel
def device_add_one(a: cute.Tensor, b: cute.Tensor):
    for i in range(a.shape[0]):
        b[i] = a[i] + 1

@cute.jit
def add_one(a: cute.Tensor, b: cute.Tensor):
    """b = a + 1"""
    device_add_one(a, b).launch(grid=(1, 1, 1), block=(1, 1, 1))

def main():
    a_torch = torch.arange(10, dtype=torch.float32, device="cuda")
    b_torch = torch.zeros(10, dtype=torch.float32, device="cuda")
    a_cute = from_dlpack(a_torch, enable_tvm_ffi=True).mark_layout_dynamic()
    b_cute = from_dlpack(b_torch, enable_tvm_ffi=True).mark_layout_dynamic()
    compiled_add_one = cute.compile(add_one, a_cute, b_cute, options="--enable-tvm-ffi")

    os.makedirs("./build", exist_ok=True)
    object_file_path = "./build/add_one.o"
    lib_path = "./build/add_one.so"
    compiled_add_one.export_to_c(object_file_path, function_name="add_one")

    shared_libs = cute.runtime.find_runtime_libraries(enable_tvm_ffi=True)
    cmd = ["gcc", "-shared", "-o", lib_path, object_file_path, *shared_libs]
    subprocess.run(cmd, check=True)
    print(f"Successfully created shared library: {lib_path}")

if __name__ == "__main__":
    main()
```

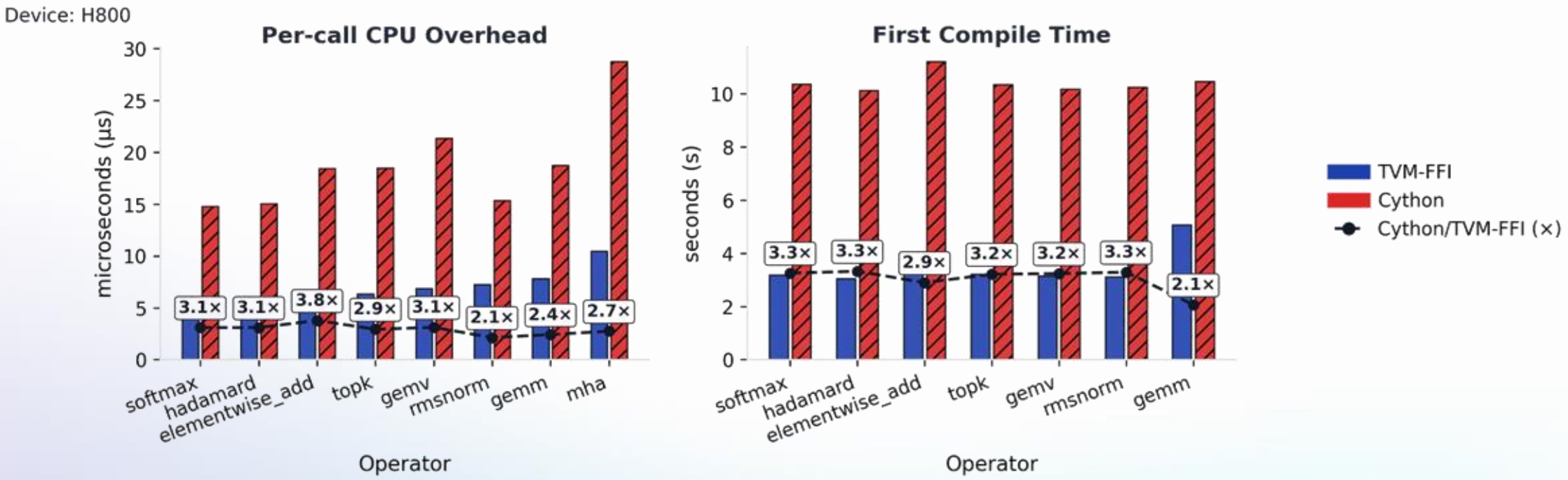
Enable TVM-FFI by adding options

Support both AOT and JIT mode

Interoperability with ML Frameworks

TileLang Integration: TVM-FFI + TVM Host Codegen

上海创智学院
Shanghai Innovation Institute



Migrate tensor checks (dim, shape, stride, etc.) from Python to C++.

TVM-FFI can generate and load cubin file instead of compiling into so.

A Unified Foundation for the Entire ML Stack

上海创智学院
Shanghai Innovation Institute

For Kernel Libraries
Ship a single package to support multiple frameworks, Python versions, and languages, (e.g., FlashInfer)

For Kernel DSLs
A reusable ABI for JIT and AOT kernel exposure to runtimes. (e.g., Triton, TileLang)

For Agentic Coding
Directly compile and load inline a module in Python.

An Open Convention, Evolving with the Community

上海创智学院
Shanghai Innovation Institute

TVM FFI is an open project that draws tte ML Systems community. TVM FFI is an open project that draws on the collective wisdom of the ML Systems community.

Acknowledged Insights From

Active Collaborations & Adopters

TileLang

CuteDSL

This isn't a top-down mandate; it’s a bottom-up collaboration to build shared infrastructure.

Amplify the ML Systems Ecosystem.

上海创智学院
Shanghai Innovation Institute

Get Started Now:

```
pip install apache-tvm-ffi
```

```
pip install torch-c-dlpack-ext # if torch <= 2.9
```

THE APACHE SOFTWARE FOUNDATION

tvm

Explore the Project:

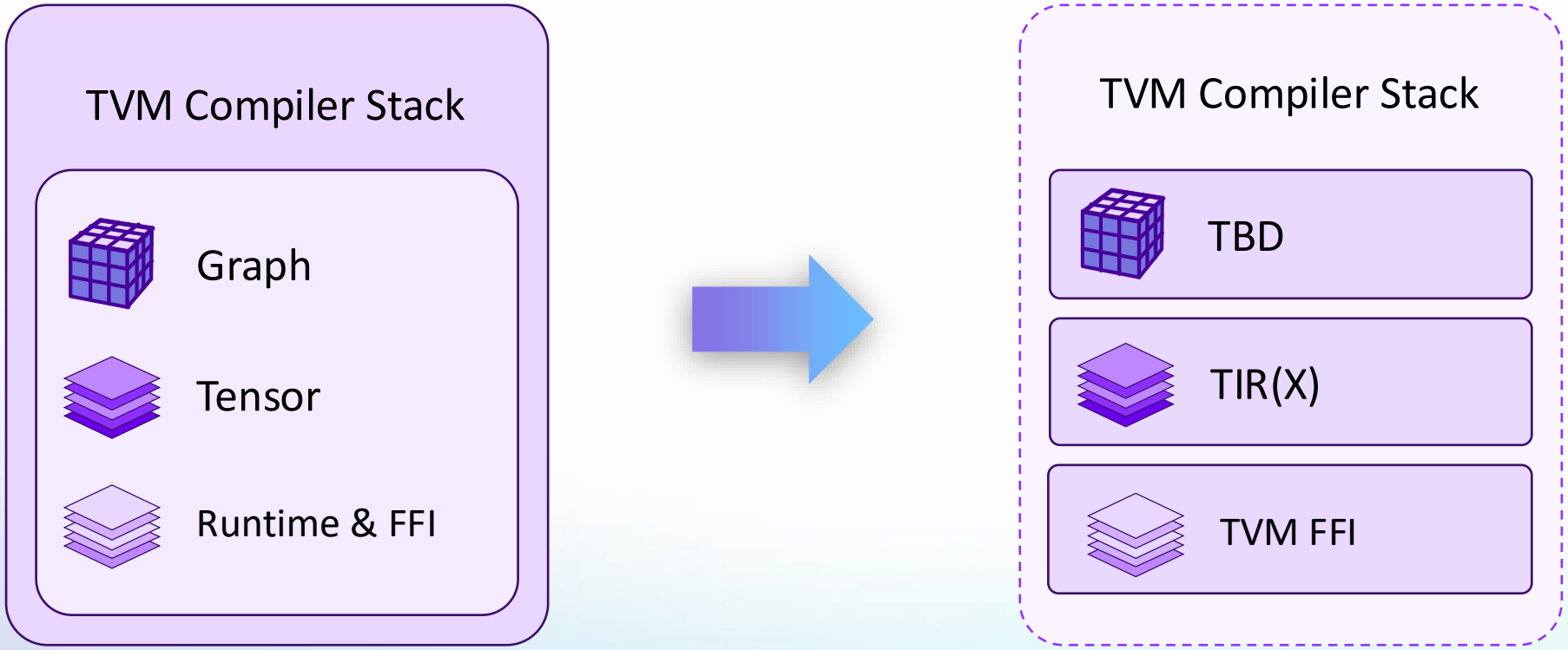
GitHub: https://github.com/apache/tvm-ffi

Quick Start : https://tvm.apache.org/ffi/get_started/quickstart.html

An open ABI is the foundation for the next wave of innovation in AI infrastructure

One More Thing…

上海创智学院
Shanghai Innovation Institute



TVM is transforming from an end-to-end AI compiler into the core infrastructure for AI compilers and ML systems.

Different from MLIR

上海创智学院
Shanghai Innovation Institute

- **Top-Down Approach**: evolved from a mature, end-to-end compiler
- **Out-of-Box Usage**: each components are self-contained as a part of compiler
- **End-to-end Example**: Remaining Apache TVM as an e2e compiler example

TVM's Family

- **Bottom-Up Trajectory:** built as a flexible framework for compiler design.
- **Flexible to Customized:** everything in MLIR should be written as a customized dialect
- **Compiler Framework:** Focus on providing framework rather than a compiler

MLIR

上海创智学院
Shanghai Innovation Institute

上海创智学院
Shanghai Innovation Institute

Thanks