Hewlett Packard
Enterprise

Chapel Con 2025

Iain Moncrief

August 15, 2024

Quick background

Chapel

- HPC language developed at HPE Cray
- Parallel and distributed
- For-free parallelism
- Performant operations on large distributed arrays

- Suite of machine learning tools
- Industry standard
- Python based

Introduction

- Applying Chapel to machine learning
  - Chapel does not have an existing machine learning library
  - Utilize Chapels array programming features

- Can Chapel be used for machine learning? (2023 Internship)
  - Yes, but best if it’s done using a library.

- Can we make a ML library in Chapel? (2024 internship)

Goals for a Chapel ML library

- Familiar syntax, names, and operations
  - PyTorch-like interface
- Easy to reproduce ML code
- Existing model loading
- GPU enabled

Use Cases

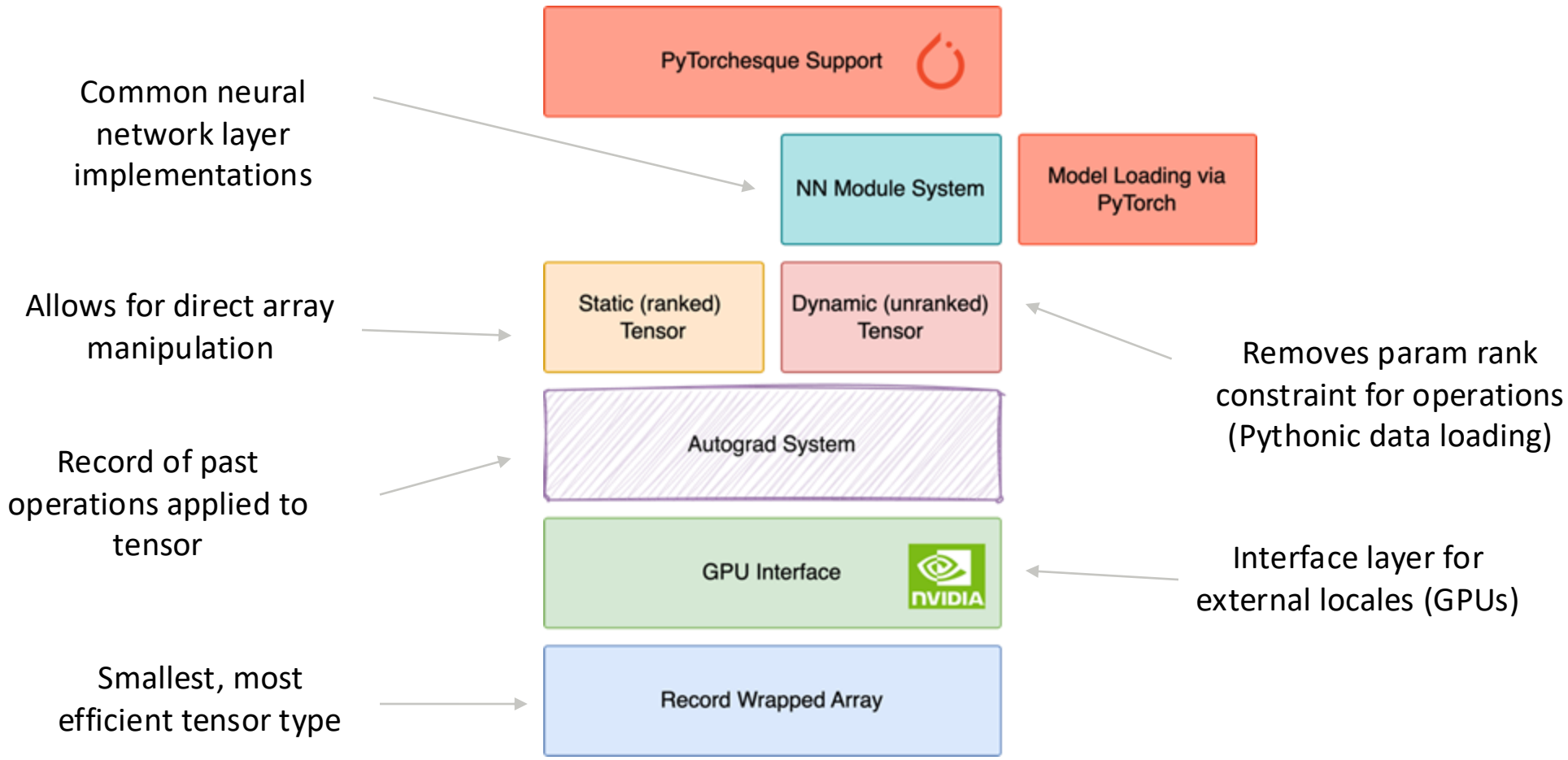
- A data scientist…
  - trains or downloads (off the shelf!) a model via PyTorch,
  - loads the model using Chapel,
  - then runs inference on a massive dataset leveraging distributed parallelism.
- A team with an application in Chapel wants to integrate an existing ML model.

What did I do? (ChAI 1.0)

- Developed a collection of ML tools to support…
  - Low-level ML programming (user defined tensor operations, layer types, and optimizers),
  - High-level utilities (pre-defined layer types and common data operations),
  - Defining and loading models,
  - Distributed inference (running the model across many compute nodes)
  - Utilities for PyTorch compatibility and interplay with Chapel

- Packaged into a library called **ChAI** (short for Chapel AI)

Architecture



Integration with PyTorch backend (libtorch)

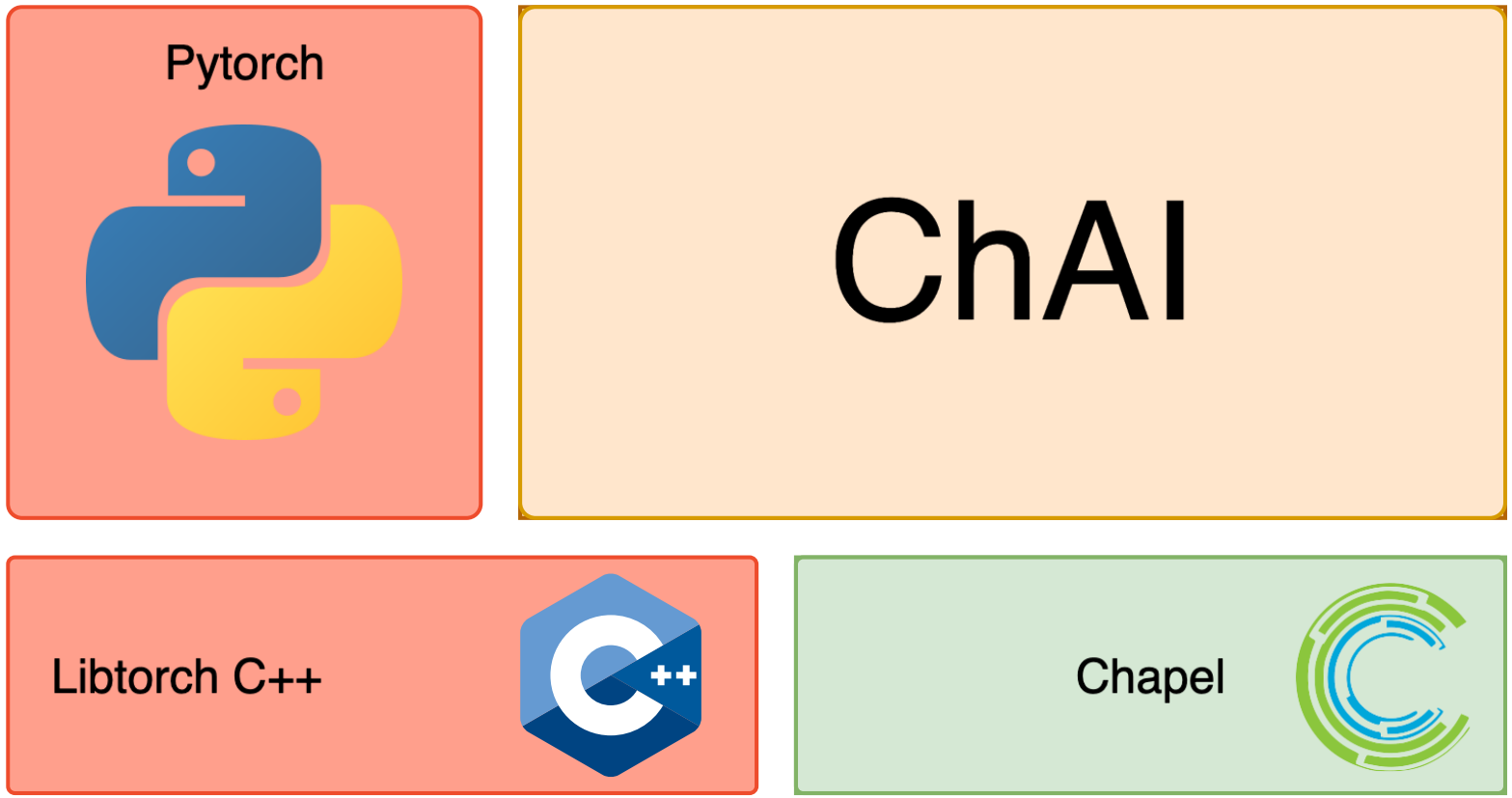
Why PyTorch?

- one of the most popular ML libraries
- vast collection of very efficient tensor operations
- exists an implementation for anything you might need
- faster low-level tensor operations

What you get

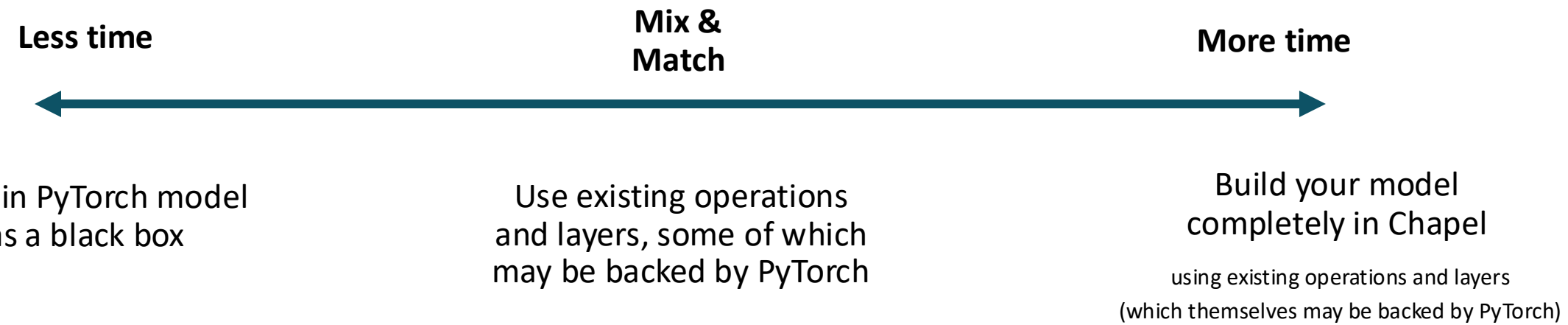
- Easily wrap any low-level tensor computation
- Load **any** pretrained PyTorch model into Chapel as a black box function
  - Supports .pth, .pt, and custom weight file format
- Access to any PyTorch function, don’t need to implement it in Chapel

Philosophy with ChAI with PyTorch



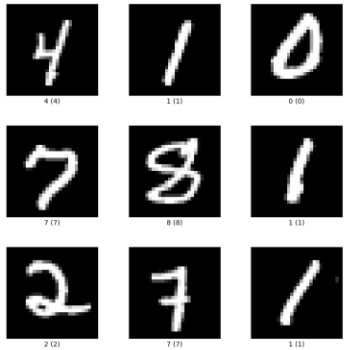
Different ways to use ChAI

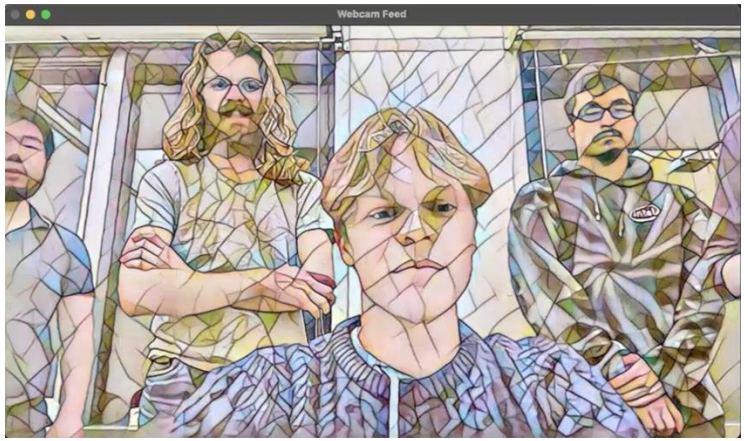
- Operations can rely on either a Chapel kernel or a low level PyTorch implementation
  - If you don't want to implement an operation in Chapel, then you can just call the PyTorch one instead
- Intended to be a high-level library that can be a wrapper for low-level Chapel or PyTorch
  - High level control of efficient low-level operations

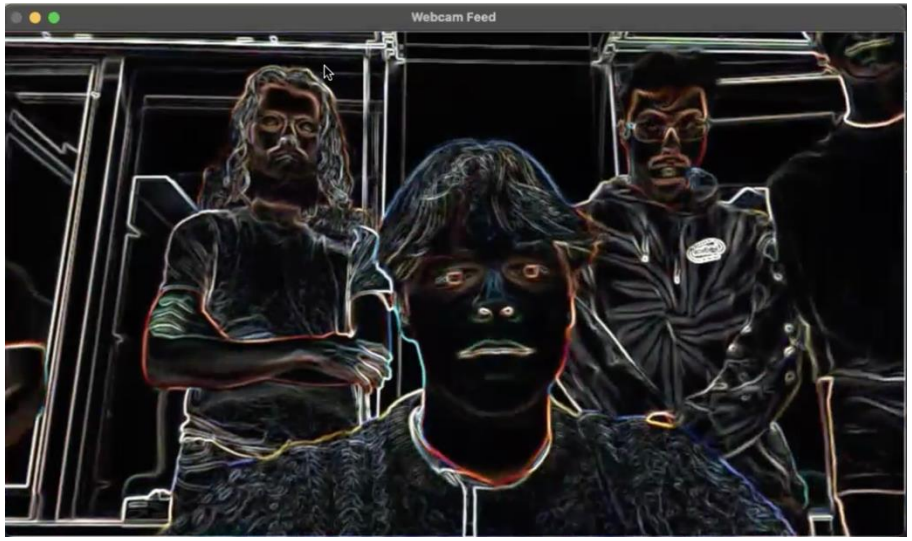


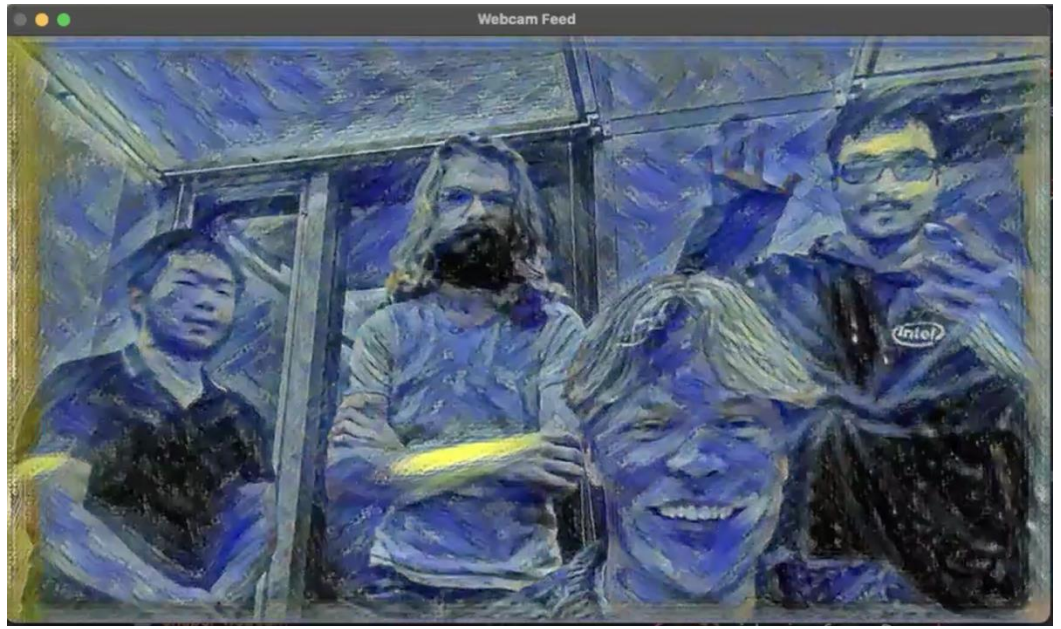
Cool examples

- Off-the-shelf MNIST model (digit classification)
- Resnet (general image classification)
- Style Transfer
- Sobel convolution (edge detection)









State of ChAI

- Supports PyTorch (libtorch) backend for low level tensor operations
- A sufficiently broad set of fundamental tensor operations (reshape,squeeze,shrink,dilate,pad,…)
- Multiple user entry points to interface
  - User may choose their level of control and type restriction
- Supports arbitrary model loading from .pt, .pth, and custom format
- Many neural network layer types
  - Several basic layer types implemented in Chapel
  - Ability to use any PyTorch layer type
- Many examples in repository
- CMake build system (tricky to get right)

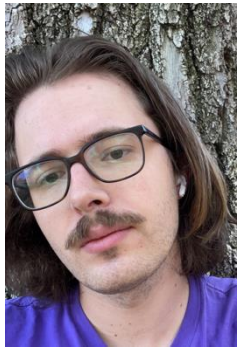
Continued work…

ChAI

- Discussing eligibility to become a supported Chapel module
- Further develop functionality
- Improve documentation and build system

Jade

Daniel



Brandon

Michelle

Engin

Contact Info
Iain Moncrief
moncrief@oregonstate.edu
github.com/iainmon/ChAI