PYTORCH CONFERENCE 2022

AGENDA Friday, Dec 2, 2022

View Bios | **Website**

8:00 - 9:00 am CST **Registration & Check In**		
9:00 - 10:20 am CST **Welcome, Keynote & Technical Talks** Host: Christian Keller		
9:00-9:10 am	Aparna Ramani | Meta AI	Welcome & Opening Remarks
9:10-9:20 am	Soumith Chintala | Meta AI	Keynote
9:20-9:30 am	Peng Wu | Meta AI	PyTorch Core: Technical Talks
9:30-9:40 am	Jason Ansel | Meta AI	
9:40-9:50 am	Michael Suo | Meta AI	
9:50-10:00 am	Horace He | Meta AI	
10:00-10:10 am	Anjali Sridhar | Meta AI	PyTorch Distributed: Towards Large-scale Distributed Training
10:10-10:20 am	Dennis van der Staay | Meta AI	TorchRec: PyTorch Domain Library for Recommendation Systems
10:20 - 10:30 am **Break**		
10:30-10:40 am	Tristan Rice | Meta AI	MultiPy: Python Models Scaled in C++ Services
10:40-10:50 am	Ankita De | Meta AI	Introduction to TorchMultimodal
10:50-11:00 am	Vincent Moens | Meta AI	Introduction to TorchRL
11:00-11:10 am	Raziel Alvarez Guevara | Meta AI	PyTorch Mobile: Past, Present, & Future

11:30 - 12:30 pm **Lunch** Virtual Lunch Livestream Talks*	
12:30 - 3:00 pm **Networking & Posters**	
1:00 - 3:00 pm **Breakout Sessions**	
1:00-1:30 pm	Joe Bowser | Adobe PyTorch Mobile on Android using C++
	Nikita Namjoshi & Eric Dong | Google Cloud Operationalize Distributed Training with PyTorch on Google Cloud
1:30-2:00 pm	Raghu Ganti | IBM Scaling PyTorch FSDP for Training Foundation Models on IBM Cloud
	Kiuk Chung & Pooja Maknikar | Amazon Research to Production at Amazon Scale With PyTorch
2:00-2:30 pm	Thomas Chaton | Lightning AI Lightning: From Models to Production AI Applications
	Parinita Rahi & Razvan Tanase | Microsoft Azure Container for PyTorch: An Optimized Container for Large Scale Distributed Training Workloads
2:30-3:00 pm	Jiong Gong | Intel What is New in Intel Extension for PyTorch

3:00 - 4:10 pm **Community & Partner Talks**		
3:00-3:10 pm	Joe Isaacson | Meta AI Geeta Chauhan | Meta AI	State of PyTorch
3:10-3:20 pm	Jeff Daily | AMD	Getting Started With PyTorch on AMD GPUs

3:20-3:30 pm	Uros Lipovsek | Amazon	Scaling Models With PyTorch on AWS Trainium
3:30-3:40 pm	Hagay Lupesko | MosaicML	Supercharge Your Model Training With MosaicML Composer
3:40-3:50 pm	Kulin Seth | Apple	Accelerate PyTorch Training on Mac Platforms Using MPS backend
3:50-4:00 pm	Lysandre Debut & Sylvain Gugger | Hugging Face	Run Very Large Models With Consumer Hardware Using 🤗Transformers and 🤗 Accelerate
4:00-4:10 pm	Zain Rizvi | Meta AI	How and why to Become a Contributor to PyTorch
4:10 - 4:20 pm **Break**		
4:20 - 5:00 pm **PyTorch Future & Closing**		
4:20-4:25 pm	Chris Mattman | NASA JPL	PyTorch on Mars
4:25-4:50 pm	PyTorch & The Linux Foundation	Panel Discussion: Maintainers & LF
4:50-4:53 pm	Video	PyTorch: A Look Back
4:53-5:00 pm	Refik Anadol & Christian Burke | Refik Anadol Studio	Machine Hallucinations
5:00-5:05 pm	Christian Keller | Meta AI	Closing Remarks
5:00 - 6:00 pm **Happy Hour** Sponsored by Google Cloud		

*VIRTUAL LIVESTREAM LUNCH TALKS

Tal Baumel | Microsoft

Title: Federated Multilingual Models for medical Transcript Analysis
Abstract: Text Analytics for Health is a healthcare-oriented NLP service and a part of Azure Cognitive Services that enables developers to process and extract insights from unstructured medical data. Developing large language models in the medical domain surfaces several issues in data and training methodology that are unique to the domain. First, to ensure high quality, it is required to expose the model to diverse medical specialties and languages. Such data is not publicly accessible and owned by worldwide healthcare organizations. Second, medical data is often secured by many data trust boundary restrictions and cannot be transferred and accessed freely between organizations. Last, to avoid privacy violation and data leakage data must be anonymized and de-identified prior to any usage in modeling.

Clément Chadebec | INRIA

Title: Pythae: Unifying Generative Autoencoder Implementations in PyTorch
Abstract: Pythae is a library that implements some of the most common (Variational) Autoencoder models under a unified implementation. In particular, it provides the possibility to perform benchmark experiments and comparisons by training the models with the same autoencoding neural network architecture. The feature make your own autoencoder allows you to train any of these models with your own data and own Encoder and Decoder neural networks. It integrates experiment monitoring tools such wandb and mlflow and allows model sharing and loading from the HuggingFace Hub in a few lines of code.

Min Jean Cho | Intel

Title: Scaling inference on CPUs with TorchServe
Abstract: We will introduce key optimizations to boosting out-of-box performance on CPUs with TorchServe, showcase performance boost via Intel® Extension for PyTorch with performance numbers and profiling, and how to enable key optimizations to TorchServe with easy-to-use API.

Robin Lobel | TorchStudio

Title: TorchStudio, an AI training assistant for PyTorch
Abstract: TorchStudio is an open source training assistant for PyTorch.
It can be used as a standalone software or connect to any Python IDE.
Easily explore, visualize, build, format, analyze datasets and models from the PyTorch ecosystem or your own, using a graphical interface providing realtime visual feedback at every step.
Train, monitor and compare performances using local hardware or any remote server in a couple clicks.

Li Ning | AWS

Title: TorchServe CPP Backend
Abstract: TorchServe is a PyTorch model serving solution. Internally, it is divided into two parts: frontend for model management, and backend for model loading and prediction. TorchServe's default Python backend allows users to easily plug in a model's pre and post processing, and also serves PyTorch's eager mode and torchscripted models. However, this backend limits TorchServe to further performance optimizations due to Python's restrictions. TorchServe's CPP backend is a new feature implemented in C++. It not only allows users to plug in model pre and post processing as Python backend does, but also builds the foundation for GPU utilization and concurrency optimization, even providing the flexibility to be embedded in an edge device.

Jimmy Whitaker | Pachyderm

Title: Automating PyTorch: Using TorchX to Make Data Centric ML Workflows
Abstract: TorchX makes running PyTorch on different platforms trivial. But when looking to automate jobs when new data comes in, the process becomes more tedious. In this talk, we,ÄôIl show you how to connect model training to a versioned data source, making data changes the driving force in your ML development.

PRESENTATION ABSTRACTS

Breakout Sessions

Joe Bowser | Adobe

Title: PyTorch Mobile on Android using C++
Abstract: This talk will be about some of the explorations that Adobe conducted with bringing existing and new ML features in our products to the Android platform using PyTorch Mobile. This talk will briefly touch on the challenges of porting models, why we choose to use C++ instead of Java or Kotlin on Android, the reasons we decided to investigate PyTorch Mobile, and the lessons learned when comparing the framework against competitors.

Nikita Namjoshi & Eric Dong | Google

Title: Operationalize Distributed Training with PyTorch on Google Cloud
Abstract: Training time is a key bottleneck for machine learning development. To speed up training of large models, many engineering teams are adopting distributed training using scale-out clusters of accelerators. However, distributed training at scale brings its own set of challenges. As a result, there's a demand for tools and frameworks to help scale large ML models, and take them to production in a reliable and repeatable way. In this session, we show how to run distributed training of PyTorch models on Google Cloud with Vertex AI. Along the way, we'll also explore how to introduce MLOps practices to improve the velocity and reliability of training these models in production.

Raghu Ganti | IBM

Title: Scaling PyTorch FSDP for Training Foundation Models on IBM Cloud
Abstract: IBM Research kicked off a strategic initiative for AI in 2022 for frictionless development and deployment of foundation models for enterprise use cases in Hybrid Cloud. These use-cases target multiple modalities such as NLP, timeseries, weather, chemistry, tabular data, cybersecurity with model sizes from 100s of millions to 100s of billions of parameters. This talk will present results from our joint work with PyTorch Distributed team on scaling training using FSDP on commodity Ethernet, which demonstrates 90% GPU efficiency across 100s of GPUs for 10B+ parameter models. The Research infrastructure in IBM Cloud houses

1400+ state of the art 80GB A100 GPUs (8 per node) with multiple 100G ethernet-based network interfaces per node which are managed by Kubernetes. We will also touch on our extensions to K8s with MCAD scheduler that provides job queuing, gang scheduling, prioritization, and quota management (https://github.com/IBM/multi-cluster-app-dispatcher) and integrate it with TorchX for seamless launching of jobs using Python APIs. We developed a multi NIC CNI (https://github.com/foundation-model-stack/multi-nic-cni) that discovers all available network interfaces and handles them as a single NIC pool. We support a single pane observability for the full stack using a desktop CLI (https://github.com/project-codeflare/codeflare-cli).

Thomas Chaton | Lightning AI

Title: Lightning: From Models to Production AI Applications
Abstract: In this presentation, we'll explore the process of going from building and training PyTorch models to embedding those models in fully-functional AI apps and products. The Lightning framework allows you to do this without handling DIY infrastructure, cost management, scale, and other common issues. We'll explore how to organize PyTorch code into standard, interoperable components using the Lightning Trainer and Lightning Module. Then, we'll go over the key components of building tailored AI apps with PyTorch: Lightning Flow and Works.

Parinita Rahi & Razvan Tanase | Microsoft

Title: Azure Container for PyTorch: An Optimized Container for Large Scale Distributed Training Workloads
Abstract: PyTorch is a popular open source machine learning framework. AzureML has launched a new container for simplifying set up and ease of acceleration of deep learning jobs with PyTorch. This container brings together PyTorch and Microsoft technologies for large scale distributed training and is used by many teams at Microsoft to develop the models used in Office, Bing, Dynamics, etc. In this talk we will share details of the container and show you how easy it is to get started with your PyTorch training workflows in Azure.

Jiong Gong | Intel

Title: What is new in Intel Extension for PyTorch
Abstract: We will show the new features and optimizations in Intel Extension for PyTorch, including into quantization, ease-of-use APIs, and ecosystem integration (such as Hugging Face).

Kiuk Chung & Pooja Maknikar | Amazon

Title: Research to Production at Amazon Scale With PyTorch
Abstract: The M5 team at Amazon builds state-of-the-art semantic representations of Amazon-specific entities such as products, shopping sessions, and reviews which are used by various machine learning systems across Amazon. Due to the scale, dynamic, and diverse nature of Amazon's AI use-cases, M5's AI platform was designed to enable high velocity research and experimentation paired with low-friction productization. With PyTorch being the deep learning library of choice, we've built a Research to Production (R2P) stack servicing over a hundred teams at Amazon. In this session we dive into the journey of M5 from zero to success and discuss the challenges of using PyTorch on our infrastructure, the solutions we came up with, and lessons learned.

Community and Partner Talks

Joe Isaacson & Geeta Chauhan | Meta AI

Title: State of PyTorch
Abstract: It takes a village to build an open-source framework. PyTorch has added many great features and has seen explosive growth in 2022, all thanks to our awesome community of contributors. This talk gives a run through of who builds PyTorch, ways to get involved in the development, and examples of it being used in industry.

Jeff Daily | AMD

Title: Getting Started With PyTorch on AMD GPUs
Abstract: This talk will cover everything a developer would need to know to get started quickly using PyTorch on AMD GPUs. The presentation will lay the foundation by introducing the ROCm open software platform as well as HIP (Heterogeneous Interface for Portability), AMD's dedicated GPU programming environment. Next, we will cover building and installing ROCm PyTorch from source or wheels and how to port existing PyTorch applications and workloads to AMD GPUs. Lastly, we will conclude with recent performance results.

Uros Lipovsek | Amazon

Title: Scaling Models With PyTorch on AWS Trainium
Abstract: The trend of increasing model size brings new challenges from model, software and hardware perspective. AWS released the Trainium chip to address this opportunity, aided by PyTorch and distributed training algorithms such as Fully

sharded data parallel. Trainium offers high network bandwidth and accelerator memory, common bottlenecks for large models.

Hagay Lupesko | MosaicML

Title: Supercharge Your Model Training With MosaicML Composer
Abstract: Learn how to make your PyTorch model training faster and more efficient by easily leveraging dozens of algorithmic optimization methods available within MosaicML Composer. In this session, we will go through recipes for training canonical CV and NLP models with 4x speedups, dive in to understand specific algorithmic optimizations, and showcase how you can apply these methods to optimize your PyTorch training workflows.

Kulin Seth | Apple

Title: Accelerate PyTorch Training on Mac Platforms Using MPS Backend
Abstract: With PyTorch v1.12 we introduced GPU-accelerated Machine Learning training on Mac platforms using the Metal Performance Shaders (MPS) backend. We will provide details about the MPS backend architecture, and how current operations are accelerated on the GPU using Metal. We will also discuss how developers can accelerate their own operations by targeting the MPS backend, as well as debugging tools & tricks for Mac. Finally we will discuss advantages of training on Apple Silicon and conclude with performance results on popular benchmarks.

Lysandre Debut & Sylvain Gugger | Hugging Face

Title: Run Very Large Models With Consumer Hardware Using 🤗 Transformers and 🤗 Accelerate
Abstract: Come and deep dive with Hugging Face maintainers into the inner workings of Transformers and Accelerate to run very large PyTorch models such as BLOOM and OPT-176B on accessible hardware. We'll understand how to efficiently load big models without maxing out your RAM and GPU memory, take advantage of various numerical precision types, and perform inference even if the model is distributed across GPUs, CPUs, or even with some weights offloaded onto the disk. We think that very large models shouldn't be constrained to clusters and believe that PyTorch can be leveraged efficiently to make them accessible to anyone.

Zain Rizvi | Meta AI

Title: How and why to Become a Contributor to PyTorch
Abstract: We'll discuss the process of contributing to PyTorch. Starting with the benefits of contributing, for both business and individuals, we'll move on to how you propose ideas and get your changes reviewed, and how to ask for help if you get stuck. We'll finally cover tips for advanced contributors and describe custom tools we've built in-house to make contributor's lives easier.

POSTER DIRECTORY

POSTER DIRECTORY

12:30-1:45 pm	Group 1
1:45-3:00 pm	Group 2

COMPUTER VISION

(GROUP 1)

A1	Enabling State-of-the-art Interpretability for Medical Imaging Using PyTorch

LIBRARIES

(GROUP 1)

B1	TorchUnmix: Automatic Stain Unmixing and Augmentation for Histopathology Images in PyTorch
B2	Scalable Training and Inference With Ray AIR
B3	AutoMAD: Mixed Mode Autodiff for PyTorch Models
B4	xFormers: Building Blocks for Efficient Transformers

B5	linear_operator - Structured Linear Algebra in PyTorch
B6	Declarative Machine Learning with Ludwig: End-to-end Machine Learning Pipelines Using Simple and Flexible Data-driven Configurations
B7	Generalized Shapes: Block Sparsity, MaskedTensor, NestedTensor
B8	Betty: An Automatic Differentiation Library for Generalized Meta Learning
B9	Functorch: Composable Function Transforms in Pytorch
B10	Large-Scale Neural Solvers for Partial Differential Equations
B11	PyTorch Video: A Deep Learning Library for Video Understanding
B12	Model Preparation for Federated Learning and Device Computation
B13	Constrained Optimization in PyTorch With Cooper
B14	Two Dimensional Parallelism Using Distributed Tensors
B15	PyTorch Tabular: A Framework for Deep Learning With Tabular Data
B17	Better Transformer: Accelerating Transformer Inference in PyTorch
B18	PiPPy: Automated Pipeline Parallelism for PyTorch

OPTIMIZATION

(G1 GROUP 1)
(C1 - C4 GROUP 2)

C1	Practical Guide on PyTorch Inference Using AWS Inferentia
C2	PyG Performance Optimization for CPU
C3	Quantization in PyTorch 2.0 Export
C4	Torch-TensorRT: A Compiler for Accelerating PyTorch Inference Using TensorRT
G1	Accelerating Inference with PyTorch by Leveraging Graph Fusions With oneDNN Graph (GROUP 1)

OTHER

(GROUP 2)

D1	Back to Python: Extending PyTorch without touching C++
D2	Functionalization in PyTorch
D3	Walmart Search: Serving Models at a Scale on TorchServe

PRODUCTION

(H1 GROUP 1)
(E1 - E4 GROUP 2)

E1	TorchX: From Local Development to Kubernetes and Back
E2	Training at Scale using Fully Sharded Data Parallel (FSDP) with PyTorch/XLA
E3	FSDP Production Readiness
E4	Orchestrating Pytorch Workflows With Kubeflow Pipelines and TorchX
H1	A Community- led and OSS Ecosystem of ML Compiler and Infrastructure Projects

TOOLS

(GROUP 2)

F1	Squeezing GPU Memory Usage in PyTorch
F2	"Brainchop": In Browser MRI Volumetric Segmentation and Rendering
F3	TorchBench: Quantifying PyTorch Performance During the Development Loop
F4	Democratizing AI for Biology with OpenFold

POSTER ABSTRACTS

COMPUTER VISION

Dinkar Juyal, Syed Asher Javed, Harshith Padigela, Amaro Taylor-Weiner, Limin Yu, Aaditya Prakash, Logan Kilpatrick, Anand Sampat | PathAI

Enabling State-of-the-art Interpretability for Medical Imaging Using PyTorch

PathAI is a Boston based company focussed on improving patient care using AI powered pathology. We heavily use PyTorch for building our ML systems, specifically training and deploying models on large gigapixel pathology images. In this case study, we highlight our use of PyTorch to build, experiment and deploy Additive Multiple Instance Learning (MIL) models. Additive MIL is a novel MIL technique built using PyTorch Lightning which allows end-to-end learning from millions of pixels while providing granular interpretability of spatial heatmaps. These models allow for the exact computation of the extent to which each smaller region in the gigapixel-sized image contributes to the final model prediction. This enables class-wise excitatory and inhibitory contributions to be visualized on top of the pathology image. This informs the practitioners of model failures and guides the pathologists to areas of interest. All this is made possible due to PyTorch's rapid research-to-prototype-to-deployment iteration cycle.

LIBRARIES

Erik Hagendorn | AbbVie

TorchUnmix: Automatic Stain Unmixing and Augmentation for Histopathology Images in PyTorch

TorchUnmix is a library which aims to provide automatic stain unmixing and augmentation for histopathology whole slide images. Separation of histochemical stains (unmixing) is performed by orthonormal transformation of the RGB pixel data from predefined light absorption coefficients called stain vectors [1]. Precomputed publicly available stain vector definitions are often used, but inter-laboratory variation due to the histology and/or image acquisition process is common, yielding suboptimal unmixing results. Classical stain vector estimation methods rely on abundant distribution of stains, making them less practical for sparser distributions as observed from immunohistochemical stains. Geis et al. proposed a method based on k-means clustering of pixel values in the hue-saturation-density color space to determine optimal stain vectors which has been used in this work [2]. While stain vectors may be used for quantification of individual stains, TorchUnmix also provides functionalities to perform stain augmentation. Stain augmentation is a method used during the training process of deep learning models to improve generalization by unmixing the image, stochastically modifying the individual stains, and then compositing the stains into the final augmented image [3]. To our knowledge, no other libraries fully implement the above methods in PyTorch, utilizing GPU-acceleration. Additionally, TorchUnmix has extended all calculations used to perform the automatic stain unmixing and augmentation to operate on batches of images, drastically accelerating execution performance speeds in comparison to other libraries.

Kai Fricke & Balaji Veeramani| Anyscale

Scalable Training and Inference With Ray AIR

Scaling machine learning is hard: Cloud platform solutions like SageMaker can limit flexibility, but a custom distributed framework is often too hard to implement. In effect, ML engineers struggle to scale their workloads from local prototyping to the cloud.

The Ray AI Runtime ("Ray AIR") is an integrated collection of machine learning libraries built around distributed computing framework Ray. It provides an easy to use interface for scalable data processing, training, tuning, batch prediction, and online serving. Adapting existing PyTorch training loops to Ray AIR's PyTorch integration needs as little as 10 lines of code changes. And scaling from local development to the cloud needs no code changes at all.

Jan Hückelheim| Argonne National Laboratory

AutoMAD: Mixed Mode Autodiff for PyTorch Models
Mixed Mode autodiff combines back-propagation and forward differentiation. Both modes have pros and cons: Back-propagation is efficient for scalar functions with many trainable parameters. Back-propagation uses memory for intermediate results, requires data flow reversal, scales poorly for many output variables. Forward differentiation is straightforward to implement, memory-efficient, and easy to vectorize/parallelize or port to new hardware. Forward mode scales poorly with large number of trainable parameters. AutoMAD makes it possible to combine both modes. Use forward differentiation for some layers, while using back-prop for others.

Daniel Haziza, Francisco Massa, Jeremy Reizenstein, Patrick Labatut, & Diana Liskovich| Meta AI

xFormers: Building Blocks for Efficient Transformers
We present xFormers, a toolbox to accelerate research on Transformers. It contains efficient components, like an exact memory-efficient multi-head attention that can accelerate trainings 2x while using a fraction of the memory. xFormers components are also customizable and can be combined together to build variations of Transformers. Our hope is to enable the next generation of research based on Transformers.

Max Balandat| Meta AI

linear_operator - Structured Linear Algebra in PyTorch
linear_operator (https://github.com/cornellius-gp/linear_operator) is a library for structured linear algebra built on PyTorch. It provides a LinearOperator class that represents a tensor that is never instantiated but is instead accessed through operations like matrix multiplication, solves, decompositions, and indexing. These objects use custom linear algebra operations that can exploit particular matrix structure (e.g. diagonal, block-diagonal, triangular, Kronecker, etc.) in computations in order to achieve substantial (many orders of magnitude) improvements in time and memory complexity. Moreover, many efficient linear algebra operations (e.g. solves, decompositions, indexing, etc.) can be automatically generated from the LinearOperator's matmul function. This makes it extremely easy to compose or implement custom LinearOperators.

The key aspect that makes linear_operator easy to use in PyTorch code is its integration with the `__torch_function__` interface - Common linear algebra operations (such as matrix multiplication, solve, SVD) are mapped to the respective torch functions (`__matmul__`, `torch.linalg.solve`, `torch.linalg.svd`), so that LinearOperator objects can be used as drop-in replacements for dense tensors even in existing code. LinearOperator operations themselves may return LinearOperator objects, automatically keeping track of algebraic structure after each computation. As a result, users never need to reason about what efficient linear algebra routines to use (so long as the input elements defined by the user encode known input structure).

Justin Zhao| Predibase

Declarative Machine Learning with Ludwig: End-to-end Machine Learning Pipelines Using Simple and Flexible Data-driven Configurations

Ludwig is a declarative machine learning framework that makes it easy to define and compare machine learning pipelines using a simple and flexible data-driven configuration system. The minimal configuration declares the input and output features with their respective data types. Users can specify additional parameters to preprocess, encode, and decode features, load from pre-trained models, compose the internal model architecture, set training parameters, or run hyperparameter optimization. Ludwig will build an end-to-end machine learning pipeline automatically, using whatever is explicitly specified in the configuration, while falling back to smart defaults for any parameters that are not. Scientists, engineers, and researchers use Ludwig to explore state-of-the-art model architectures, run hyperparameter search, and scale up to larger than available memory datasets and multi-node clusters, on a variety of problems using structured and unstructured features. Ludwig has 8.5K+ stars on Github and is built on top of PyTorch, Horovod, and Ray.

Christian Puhrsch | Meta AI

Generalized Shapes: Block Sparsity, MaskedTensor, NestedTensor

This poster presents an overview of available and ongoing developments related to sparse memory formats, masked computation, and support for collections of variably shaped data. In particular it contains a case study of block sparse memory formats, MaskedTensor, and NestedTensor.

Sang Keun Choe| Carnegie Mellon University

Betty: An Automatic Differentiation Library for Generalized Meta Learning

Betty is a simple, scalable and modular library for generalized meta-learning (GML) and multilevel optimization (MLO), built upon PyTorch, that allows a unified programming interface for a number of GML/MLO applications including few-shot learning, hyperparameter optimization, neural architecture search, data reweighting, and many more. The internal autodiff mechanism and the software design of Betty are developed by the novel interpretation of GML/MLO as a dataflow graph.

Samantha Andow | Meta AI

Functorch: Composable Function Transforms in Pytorch
Inspired by Google JAX, functorch is a library in Pytorch that offers composable vmap (vectorization) and autodiff transforms (grad, vjp, jvp). Since its first release alongside Pytorch 1.11, combining these transforms has helped users develop and explore new techniques that were previously tricky to write in Pytorch, like Neural Tangent Kernels and non-linear optimizations (see Theseus, also from PyTorch). This will go through some basic usages and highlight some research that leverages functorch.

Patrick Stiller, Jeyhun Rustamov, Friedrich Bethke, Maksim Zhdanov, Raj Sutarya, Mahnoor Tanveer, Karan Shah, Richard Pausch, Sunna Torge, Alexander Debus, Attila Cangi, Peter Steinbach, Michael Bussmann, & Nico Hoffmann | Helmholtz Zentrum Dresden-Rossendorf

Large-Scale Neural Solvers for Partial Differential Equations
Our open-source Neural Solvers framework provides data-free ML-based solvers for the study and analysis of phenomena in natural sciences built on top of Pytorch. We were the first to show that certain quantum systems modeled by the 2d Schr√∂dinger equation can be accurately solved while retaining strong scaling. We also developed a novel neural network architecture, GatedPINN [1], introducing adaptable domain decomposition into the training of Physics-informed Neural Networks based on the Mixture-of-Experts paradigm. Distributed large-scale training of our GatedPINN is facilitated by Horovod, resulting in excellent GPU utilization making Neural Solvers ready for the upcoming exascale era. Upcoming projects involve higher dimensional problems such as 3d laser systems and coupled models to study the Vlasov-Maxwell system. Further experiments on novel very scalable compute hardware paves the way for applications of high-fidelity Neural Solvers to real-world applications such as Inverse Scattering Problems.

Haoqi Fan | Meta AI

PyTorch Video: A Deep Learning Library for Video Understanding
PyTorchVideo is the deep learning library for video understanding research in PyTorch.

Jose Gallego-Posada (Juan Camilo Ramirez, co-author)| Mila, University of Montreal

Constrained Optimization in PyTorch With Cooper
Cooper (https://github.com/cooper-org/cooper) is a general-purpose, deep learning-first constrained optimization library in PyTorch. Cooper is (almost!) seamlessly integrated with PyTorch and preserves the usual loss ➡️ backward ➡️ step workflow. If you are already familiar with PyTorch, using Cooper will be a breeze!

This library aims to encourage and facilitate the study of constrained optimization problems in deep learning. Cooper focuses on non-convex constrained optimization problems for which the loss or constraints are not necessarily "nicely behaved" or "theoretically tractable". Moreover, Cooper has been designed to play nicely with mini-batched/stochastic estimates for the objective and constraint functions.

Cooper implements several popular constrained optimization protocols so you can focus on your project, while we handle the nitty-gritty behind the scenes.

Zhihan Fang | Meta AI

Model Preparation Federated Learning and Device Computation
Federated Learning with Differential Privacy has witnessed an increased adoption as one of the most promising ways to train machine learning models while preserving user privacy. Existing models in Meta around people attributes are mostly built on traditional centralized machine learning methods. Recently, due to the increasing concerns about user privacy internally and externally, Machine Learning teams at Meta are experiencing either signal loss or restriction on applying new features in models to further improve model performance. In this paper, we are introducing a generic framework we built for preparing and generating models for federated learning. The model preparation process is to utilize traditional machine learning to understand model structure and hyperparameters for the target problems including training, inference, evaluations. It also requires a simulation process to train the target model structure and understand the simulated environment on the server side to tune FL specific hyperparameters.
The model generation process is to generate device compatible models, which can be used directly on users' devices for federated learning. We applied the FL framework on our on-device models, and integrated with device signals to improve user experience and protect user privacy.

Wanchao Liang & Junjie Wang | Meta AI

Two Dimensional Parallelism Using Distributed Tensors
This talk will introduce 2-dimensional parallelism with PyTorch (Data Parallelism + Tensor Parallelism) using Distributed Tensor, a fundamental distributed primitive offered by PyTorch Distributed that empowers Tensor Parallelism. We have proven that using FSDP + Tensor Parallelism together could enable us to train large models like Transformer, and increase training performance. We offer end to end training techniques that enable you to train models in 2-D parallelism fashion, and checkpoint save/load models in a distributed manner.

Manu Joseph | Thoucentric

PyTorch Tabular: A Framework for Deep Learning with Tabular Data
In spite of showing unreasonable effectiveness in modalities like text and image, Deep Learning has always lagged Gradient Boosting in tabular data- both in popularity and performance. But recently there have been newer models created specifically for tabular

data, which is pushing the performance bar. Popularity is still a challenge, however, because there is no easy, ready-to-use library like Sci-Kit Learn for deep learning. PyTorch Tabular aims to change that by being an easy-to-use and flexible framework which makes using SOTA model architectures in tabular data as easy as Sci-Kit Learn.

Michael Gschwind, Christian Puhrsch, Driss Guessous, Rui Zhu, Daniel Haziza, & Francisco Massa | Meta AI

Better Transformer: Accelerating Transformer Inference in PyTorch
We introduce Better Transformer, the PyTorch project to accelerate Transformers for inference and training with out-of-the-box enablement by implementing the Better Transformer 'fastpath'. Fastpath accelerates many of the most commonly executed functions in Transformer models. Starting with PyTorch 1.13, the PyTorch Core API is implemented with accelerated operations to deliver up to 2x-4x speedups on many Transformer models, such as BERT and XLM-R. Accelerated operations are based on (1) operator and kernel fusion and (2) exploiting sparsity created by variable sequence-length NLP batches. In addition to improving MultiHeadAttention with fastpath, the model also includes sparsity support for MultiHeadAttention and TransformerEncoder modules to take advantage of variable sequence-length information with Nested Tensors for NLP models.

At present, we enable torchtext and Hugging Face domain libraries with Better Transformer, delivering significant speedups for text, image, and audio models. Starting with the next release, PyTorch core will include even faster fused kernels and training support. You can preview these features today with PyTorch Nightlies, the nightly preview builds of the upcoming PyTorch release.

Ke Wen, Pavel Belevich, & Anjali Sridhar | Meta AI

PiPPy: Automated Pipeline Parallelism for PyTorch
PiPPy is a library that provides automated pipeline parallelism for PyTorch models. With compiler techniques, PiPPy splits a model into pipeline stages without requiring model changes. PiPPy also provides a distributed runtime that distributes the split stages to multiple devices and hosts and orchestrates micro-batch execution in an overlapped fashion. We demonstrate application of PiPPy to Hugging Face models achieving 3x speedup on cloud platforms.

OPTIMIZATION

Keita Watanabe | Amazon Webservices

Practical Guide on PyTorch Inference Using AWS Inferentia
In this session we will go through step-by-step how to conduct the inference process of machine learning models using Inferentia. In addition, we compare the inference

performance with GPU and discuss the cost advantage. In the later part of the session, we will also cover model deployment on Kubernetes.

Mingfei Ma | Intel Corporation and kumo.ai

PyG Performance Optimization for CPU
Accelerating PyG CPU performance with faster sparse aggregation.
PyG is a library built upon PyTorch to easily write and train Graph Neural Networks, which heavily relies on the mechanism of Message Passing for information aggregation. We have optimized critical bottlenecks of Message Passing from PyTorch, including: 1. Scatter Reduce: maps to classic PyG use case when the EdgeIndex is stored in COO memory format. 2. SpMM Reduce: maps to the usage case when the EdgeIndex is stored in CSR memory format.

Jerry Zhang | Meta AI

Quantization in PyTorch 2.0 Export
Currently, PyTorch Architecture Optimization (torch.ao) offers two quantization flow tools: eager mode quantization (beta) and fx graph mode quantization (prototype). With PyTorch 2.0 coming up, we are going to redesign quantization on top of the PyTorch 2.0 export path, this talk will introduce our plans for supporting quantization in PyTorch 2.0 export path, its main advantages over the previous tools, and how modeling developers and backend developers will be interacting with this flow.

Naren Dasan, Dheeraj Peri, Bo Wang, Apurba Bose, George Stefanakis, & Nick Comly | NVIDIA

Wei Wei, Shirong Wu, Yinghai Lu | Meta

Torch-TensorRT: A Compiler for Accelerating PyTorch Inference Using TensorRT
Torch-TensorRT is an open-source compiler targeting NVIDIA GPUs for high-performance deep-learning inference in PyTorch. It combines the usability of PyTorch with the performance of TensorRT allowing for easy optimization of inference workloads on NVIDIA GPUs. Torch-TensorRT supports all classes of optimizations in TensorRT including reduced mixed precision down to INT8, through simple Python & C++ APIs designed to work directly from PyTorch. Torch-TensorRT outputs standard PyTorch modules as well as the TorchScript format to allow for a completely self-contained, portable, & static module with TensorRT engines embedded. We present recent improvements to Torch-TensorRT including the new FX frontend which allows developers to use a full Python workflow for optimizing models and extend Torch-TensorRT in Python, the unified Torch-TensorRT Runtime which enables hybrid FX + TorchScript workflows and discuss future work for the project.

Sanchit Jain | Intel Corporation

Accelerating Inference with PyTorch by Leveraging Graph Fusions With oneDNN Graph

The open-source oneDNN Graph library extends oneDNN with a flexible graph API to maximize the optimization opportunities for generating efficient code on AI hardware (currently x86-64 CPUs, but GPU support is on the way). It automatically identifies the graph partitions to be accelerated via fusion. Its fusion patterns entail fusing compute-intensive operations such as convolution, matmul and their neighbor operations for both inference and training use cases. Since PyTorch 1.12, oneDNN Graph has been supported as an experimental feature to speed up inference with Float32 datatype on x86-64 CPUs. Support for inference with oneDNN Graph using BFloat16 datatype exists in the PyTorch master branch, and hence also in nightly PyTorch releases. Intel Extension for PyTorch is an open-source library that builds on top of PyTorch, and can be thought of as a "staging-ground" for optimizations in PyTorch from Intel. It leverages oneDNN Graph for inference with int8 datatype. This poster presents reproducible results with PyTorch's TorchBench benchmarking suite to demonstrate the inference speedup achieved with PyTorch & oneDNN Graph using Float32, BFloat16 & int8 datatypes.

OTHER

Alban Desmaison | Meta AI

Back to Python: Extending PyTorch Without Touching C++
This poster presents the new extension points that the PyTorch team has designed to allow users to extend PyTorch from Python. We will cover an introduction to Tensor Subclassing, Modes and torch library. We will briefly describe each extension point and talk through examples such as memory profiling, logging used operators, quantization and custom sparse kernel all in less than 100 LOC. We will also introduce the new ways you can add new devices and author kernels without the need to modify PyTorch directly.

Brian Hirsh | Meta AI

Functionalization in PyTorch
Functionalization is a way to remove mutations from arbitrary PyTorch programs sent to downstream compilers. The PyTorch 2.0 stack is all about capturing graphs of PyTorch operations and sending them off to a compiler to get better performance.
PyTorch programs can mutate and alias state, making them unfriendly to compilers. Functionalization is a technique to take a program full of PyTorch operators, including mutable and aliasing operators, and remove all mutations from the program while preserving semantics.

Pankaj Takawale, Dagshayani Kamalaharan, Zbigniew Gasiorek, & Rahul Sharnagat | Walmart Labs

Walmart Search: Serving Models at a Scale on TorchServe

Walmart Search has embarked on the journey of adopting Deep Learning in the Search ecosystem for improving Search relevance in various parts. As our pilot use case, we wanted to serve the computationally intensive Bert Base model at runtime with an objective to achieve low latency and high throughput. We had JVM hosted web applications loading and serving multiple models. The experimental models were being loaded onto the same applications. These models are large in size and computation is expensive.
We were facing the following limitations with this approach: Refreshing model with the latest version or adding new experimental model would need application deployment. Increased memory pressure on a single application. Slow startup time due to loading multiple ML models during startup. Concurrency was not beneficial due to limited CPU (Metrics on concurrent model prediction vs sequential).

PRODUCTION

Joe Doliner & Jimmy Whitaker | Pachyderm, Inc.

TorchX: From Local Development to Kubernetes and Back

TorchX is incredibly useful for developing PyTorch applications quickly. But when it comes to deployment, nothing is easy. With docker development, Kubernetes, and customer schedulers, there's a lot to learn. In this talk, we'll discuss how organizations can deploy to production, why TorchX is a great system for this, and lessons we learned so you can avoid hitting them too.

Shauheen Zahirazami, Jack Cao, Blake Hechtman, Alex Wertheim | Google

Ronghang Hu| Meta AI

Training at Scale Using Fully Sharded Data Parallel (FSDP) with PyTorch/XLA

PyTorch/XLA enables PyTorch users to run their models on XLA devices including Google's Cloud TPUs. The latest improvements in PyTorch/XLA enables training PyTorch models using FSDP to train very large models. In this work we present benchmarks and Hardware Flops Utilization of training HuggingFace GPT-2 on Cloud TPU v4.

Rohan Varma & Andrew Gu | Meta AI

FSDP Production Readiness

This talk dives into recent advances in PyTorch Fully Sharded Data Parallel (FSDP) that have enabled better throughput, memory savings, and extensibility. These improvements have unblocked using FSDP for models of different modalities and for varying model and data sizes. We will share best practices to apply these features to specific use cases such as XLMR, FLAVA, ViT, DHEN, and GPT3-style models.

Erwin Huizenga & Nikita Namjoshi | Google

Orchestrating Pytorch Workflows With Kubeflow Pipelines and TorchX

TorchX is a universal job launcher for PyTorch applications that helps ML practitioners speed up iteration time and support end to end production. In this talk, we show you how to build and run TorchX components as a pipeline using the Kubeflow Pipeline (KFL) DSL. We go into detail on how to use KFP and TorchX to build components and how to use KFP DSL to orchestrate and run ML workflows.

Shauheen Zahirazami, James Rubin, Mehdi Amini, Thea Lamkin, Eugene Burmako, & Navid Khajouei | Google

A Community- led and OSS Ecosystem of ML Compiler and Infrastructure Projects

ML development is often stymied by incompatibilities between frameworks and hardware, forcing developers to compromise on technologies when building ML solutions. OpenXLA is a community-led and open-source ecosystem of ML compiler and infrastructure projects being co-developed by AI/ML leaders including Alibaba, Amazon Web Services, AMD, Arm, Apple, Google, Intel, Meta, NVIDIA, and more. It will address this challenge by letting ML developers build their models on leading frameworks and execute them with high performance across any hardware backend. This flexibility will let developers make the right choice for their project, rather than being locked into decisions by closed systems. Our community will start by collaboratively evolving the XLA compiler and StableHLO, a portable ML compute operation set that makes frameworks easier to deploy across different hardware options.

TOOLS

Mao Lin, Keren Zhou, & Penfei Su | UC Merced and OpenAI

Squeezing GPU Memory Usage in PyTorch

The limited GPU memory resources can often hinder the performance of GPU-accelerated applications. While PyTorch's Caching Allocator aims to minimize the number of expensive memory allocations and deallocations and maximize the efficient utilization of GPU memory resources, our study of common deep learning models revealed significant memory fragmentation problems. In some cases, up to 50% of GPU memory is wasted. To better understand the root causes of memory fragmentation, we developed a tool that visualizes GPU memory usage in two ways: the allocator view and the block view. The allocator view

presents memory usage with each allocation or deallocation event, and the block view shows the changes in specific memory blocks over time. Our analysis revealed the considerable potential to save GPU memory, which would relieve the bottleneck of limited resources. By employing strategies such as swapping, activation recomputation, and memory defragmentation, we were able to reduce GPU memory waste significantly.

Mohamed Masoud, Farfalla Hu, & Sergey Plis | Georgia State University Neuroneural Trends

"Brainchop": In Browser MRI Volumetric Segmentation and Rendering
In brainchop project, we bring high fidelity pre-trained deep learning models for volumetric analysis of structural magnetic resonance imaging (MRI) right to the browsers of scientists and clinicians with no requirement on their technical skills in setting up AI-solutions. All of this in an extensible open-source framework. Our tool is the first front-end MRI segmentation tool on the web that supports full brain volumetric processing in a single pass inside a browser. This property is powered by our lightweight and reliable deep learning model Meshnet that enables volumetric processing of the entire brain at once, which leads to increased accuracy with modest computational requirements. High-quality client-side processing solves the privacy problem, as the data does not need to leave the client. Moreover, browser-based implementation is able to take advantage of available hardware acceleration regardless of the brand or architecture.

GitHub: https://github.com/neuroneural/brainchop

Xu Zhao, Will Constable, David Berard, Taylor Robie, Eric Han, & Adnan Aziz | PyTorch Perf Infra Team - Meta

TorchBench: Quantifying PyTorch Performance During the Development Loop
Holding the line of performance is challenging for ML frameworks like PyTorch. The existing AI benchmarks like MLPerf are end-to-end, therefore require large volumes of datasets, at-scale GPU clusters, and long benchmarking time. We develop TorchBench, a novel AI benchmark suite which highlights with minimal data inputs, single GPU, and milliseconds-per-test latencies. TorchBench is now deployed as part of the PyTorch nightly release process, guarding performance/correctness regressions and testing experimental PyTorch features on SOTA machine learning models.

Gustaf Ahdritz, Sachin Kadyan, Will Gerecke, Luna Xia, Nazim Bouatta, Mohammed AlQuraishi | Weights & Biases

Democratizing AI for Biology With OpenFold
OpenFold, developed by Columbia University, is an open-source protein structure prediction model implemented with PyTorch. The goal of OpenFold is to verify that AlphaFold 2 — DeepMind's protein structure prediction model — can be reproduced from scratch and beyond that, make components of the system available to like-minded researchers and academics so they can build on top of it. During this research, Weights & Biases was used to accelerate OpenFold's reproduction of AlphaFold 2. The collaborative nature of W&B allowed

for insights to scale from a single researcher to the entire team and helped solve the reproducibility challenge in ML.