PyTorch Conference 2022 Tech Report

Won Jong Jeon

Software Lab

FUTUREWEI Technologies

Overview

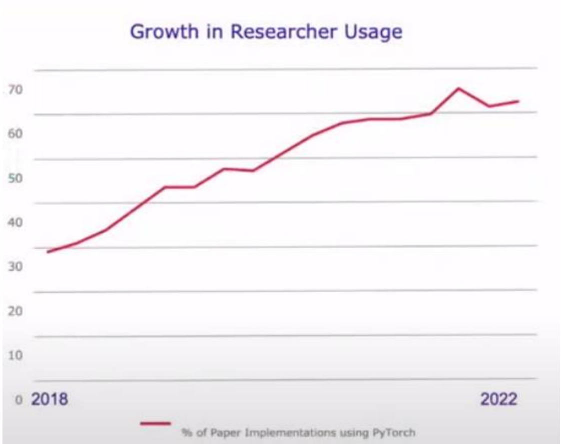
- When: December 2, 2022
- Where: New Orleans, LA, USA (virtual option available)
- Co-located with NeuralPS 2022
- Covering new software releases on PyTorch, use cases in academia and industry, as well as ML/DL development and production trends.
- Video stream is available at https://youtu.be/vbtGZL7IrAw.

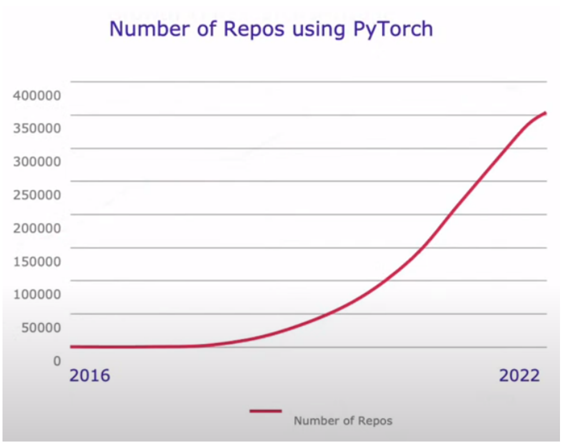
Schedule

- 8-9am: Registration/check-in
- 9-11:20am: Keynote & technical talks (by Meta AI)
- 11:30am-1pm: Lunch
- 1-3pm: Poster session & breakouts
- 3-4pm: Community/partner talks
- 4-5pm: Panel discussion

Introduction

- Rapid growth in research usage
- Rapid increase in # of repos using PyTorch
- Industrial usage
  - >44k LinkedIn professionals, >2,500 jobs, 50% increase in PyTorch professionals this year
- Top organizations contributing to PyTorch
  - Meta, Microsoft, Nvidia, Intel, Google, Quansight, AMD, AWS, IBM
- PyTorch Foundation
  - Under Linux Foundation (LF)
  - For technical autonomy, business governance



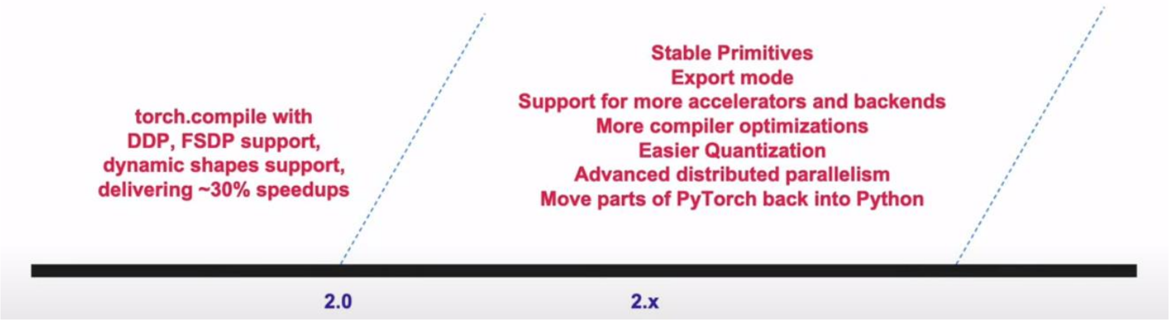


FUTUREWEI Technologies

Plan/roadmap of PyTorch 2.0 and beyond

- For ML scientists
  - 30%+ training speedups, lower memory usage with no changes to code or workflow

- For compiler/hardware engineer
  - Dramatically easier to write a PyTorch backend
- For large-scale DL projects
  - State of the art distributed capabilities
- For code contributor
  - Substantially more PyTorch is written in Python



FUTUREWEI Technologies

Plan/roadmap

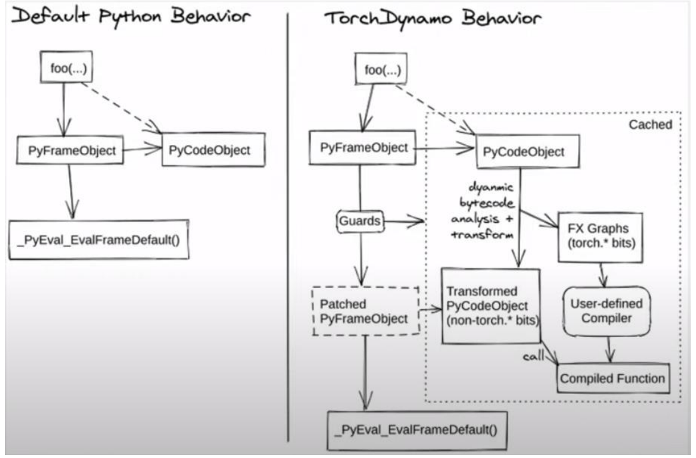
- 2.0 is fully backward compatible.
- Sets of benchmarks FP16 and FP32
  - timm (Python image models) https://github.com/rwightman/pytorch-image-models
  - TorchBench https://github.com/pytorch/benchmark
  - HuggingFace transformer benchmark https://huggingface.co/docs/transformers/benchmarks
- Dynamic shape and distributed support
- Robust to correctness and accuracy
  - AMP, FP16 / FP32
- Make PyTorch faster
  - Kernel fusion
  - Out-of-order execution
  - Automatic work placement (in multi-node multi-GPU environment)

Plan/road map

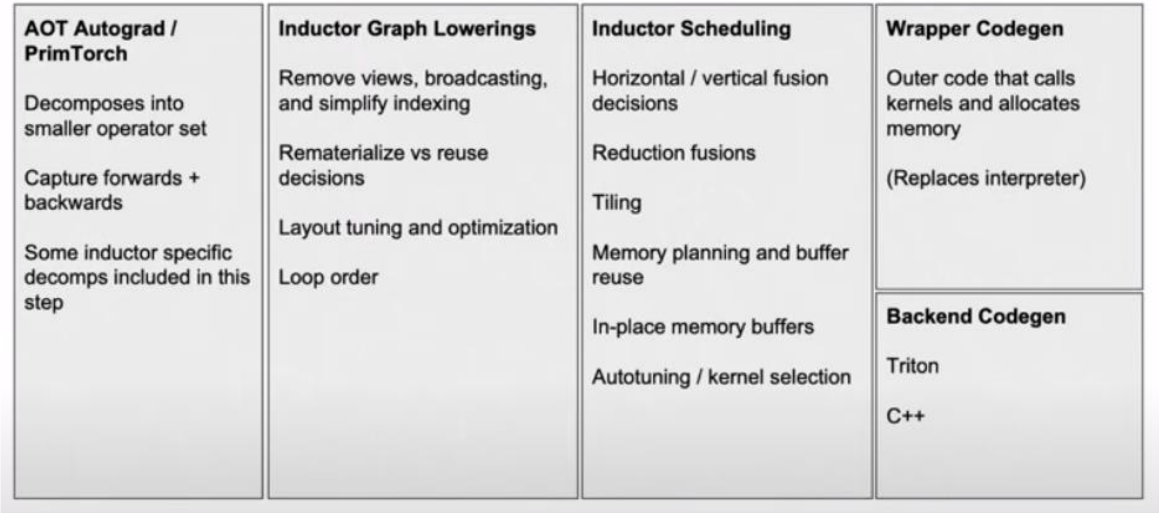
- TorchDynamo
  - To capture dynamic graphs without compromising user-experience?
  - TorchDynamo rewrite into blocks of graphs.
- AOT Autograd
  - https://pytorch.org/functorch/stable/notebooks/aot_autograd_optimizations.html
- Simplifying operator space
  - From 2000+ ops to ~250 primitive operators
- TorchInductor
  - Graph compilation, powered by OpenAI Triton (Python DSL for writing parallel code) https://triton-lang.org
  - Compiler written in Python
  - Support for CPU and GPU (Volta and Ampere)
  - Support for your own backend, nvFuser, TVM, XLA, AITemplate, TensorRT

TorchDynamo

- Graph capture fundamentally shift the efficiency of PyTorch.
- Features
  - Partial graph capture
    - Ability to skip unwanted parts of eager
  - Guarded graphs
    - Ability to check if captured graph is valid for execution
  - Just-in-time recapture
    - Recapture a graph if captured graph is invalid for execution
- TorchDynamo vs. TorchScript frontend
  - TorchDynamo does not require changing models
  - TorchDynamo reliably captures backward graphs (works for training)
- Status
  - Tested with 7k+ Github models, 20+ inference backends, 1+ training backends
  - 30+% geomean speedup (FP32, AMP, on A100 GPU)
- Current recommendations on graph capture
  - For training: all (but XLA or TPU) → Dynamo, Export to XLA or TPU → Lazy Tensor
  - For inference: embedded → TorchScript, non-embedded → TorchScript or torch.fx
  - For human-in-the-loop tool → torch.fx



TorchInductor



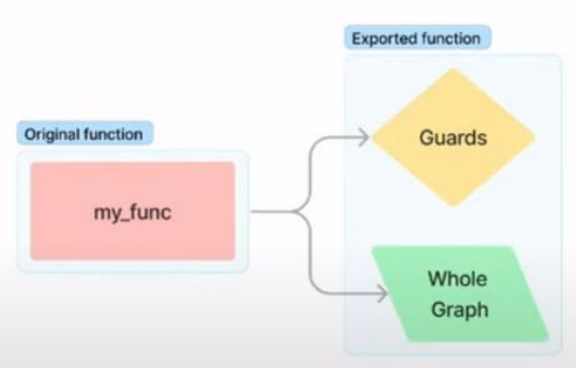
- A new compiler backend for PyTorch
- Principles
  - Similar abstractions to PyTorch eager
  - Written in Python, generates Triton and C++
  - Early focus on supporting a variety of operators, hardware, and optimization
- Technologies
  - Define-by-run loop-level IR: direct use of Python functions in IR definitions
  - Dynamic shape & strides: using SymPy symbolic math library (https://www.sympy.org) to reason about shapes, indexing, and managing guards.
  - Reuse state of the art languages: Triton for GPUs, C++/OpenMP for CPUs
- https://github.com/pytorch/pytorch/tree/master/torch/_dynamo

```python
def inner_fn(index: List[sympy.Expr]):
    i1, i0 = index
    tmp0 = ops.load("x", i1 + i0*size1)
    tmp1 = ops.load("x", 2*size1 + i0)
    return ops.add(tmp0, tmp1)

torchinductor.ir.Pointwise(
    device=torch.device("cuda"),
    dtype=torch.float32,
    inner_fn=inner_fn,
    ranges=[size0, size1],
)
```

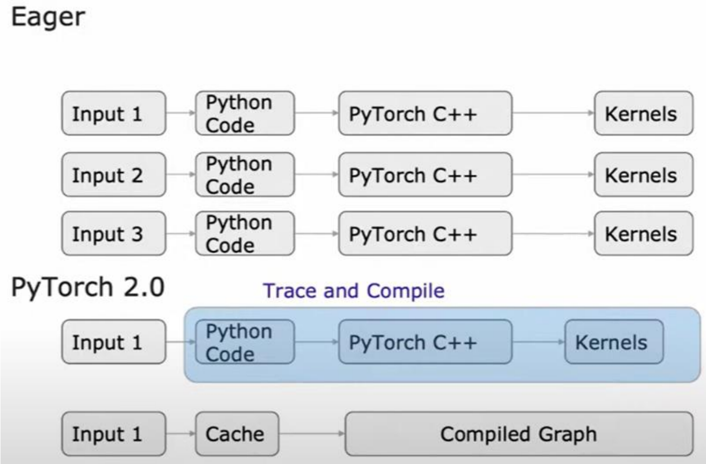
PyTorch 2.0 Export

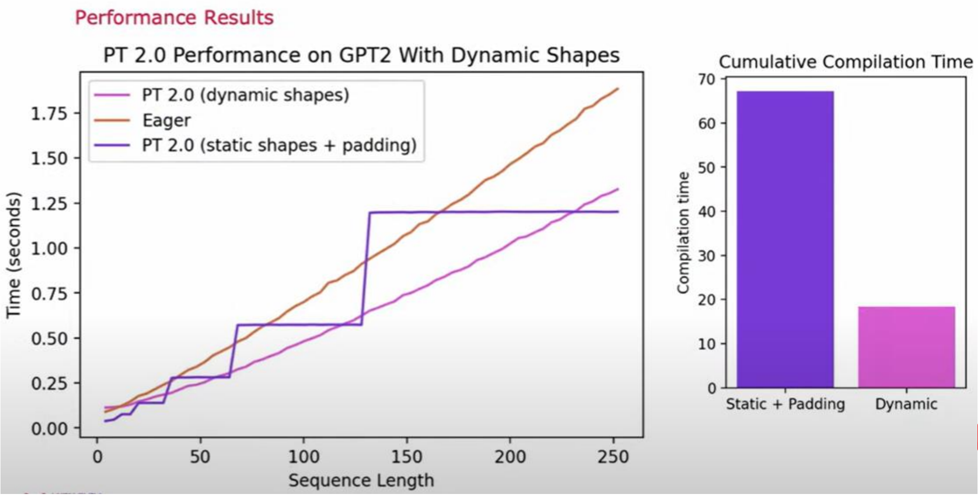
- Whole-graph export API
- Standardized IR and operator set
  - Primitive operator set (from 2000+ ops to ~250)
  - Consolidated compiler infrastructure: provide common infra for users to process graphs , vendor common passes in reusable form
- TorchDynamo export mode
  - torch._dynamo.export(my_func, input)
- Status
  - Significant functionality already in repo tree
  - Will be stabilized in 2.x series release



Dynamic shape support in PyTorch 2.0

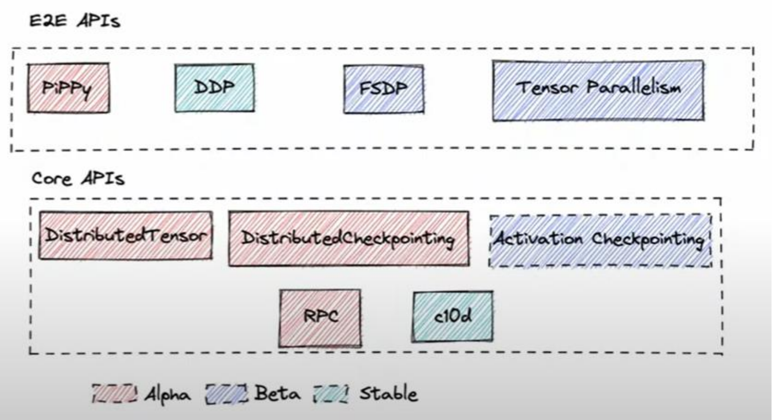
- Challenge
  - Python code, PyTorch C++, and kernels in eager mode can depend on shapes.
  - Cache in PyTorch 2.0 needs to know about shapes.
- Solution
  - Transition from old concrete static shapes to symbolic shapes
  - Leverages SymPy to build rich information about the shapes in the program
  - Allows TorchInductor to generate efficient code for different shapes without recompilation
  - Deep integration of symbolic shapes into PyTorch code components
- Performance enhancement in execution time and compilation time
- Additional benefits
  - Shape-checking of functions
  - Analyzing FLOPs of neural network symbolically

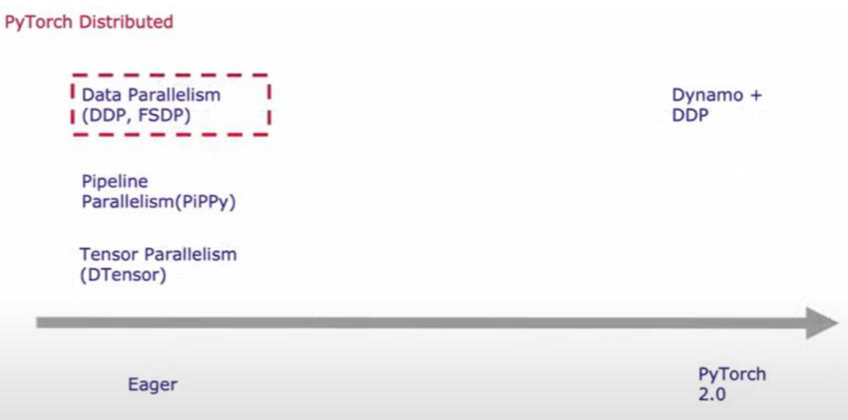




PyTorch Distributed

- DDP (Distributed Data Parallel) and FSDP (Fully Shared Data Parallel) for data parallel training
    - DDP working by replicating parameters
    - FSDP sharding parameters and using allgather to ensure replicated parameters on each device
    - Recent improvement including activation checkpointing, mixed precision, tackling OOM issues
    - Dynamo optimizes DDP wrapped modules
- PiPPy
    - A cross host pipeline parallelism API, enabling automatic spliiting of model using torch.fx and 2D parallelism with DDP
    - Recent improvement in deferred initialization, automated model split APIs (for user-guided splits)
- Tensor Parallelism Modules (Dtensor)
    - Dtensor is a subclass of tensor, providing a flexible annotation for both Shared and Replicated Tensors used in Tensor Parallelism.
    - Tensor Parallelism can be used together with FSDP in 2D parallel.

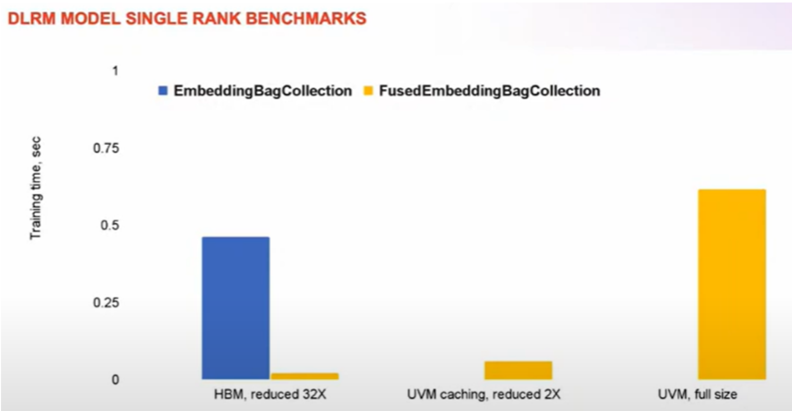


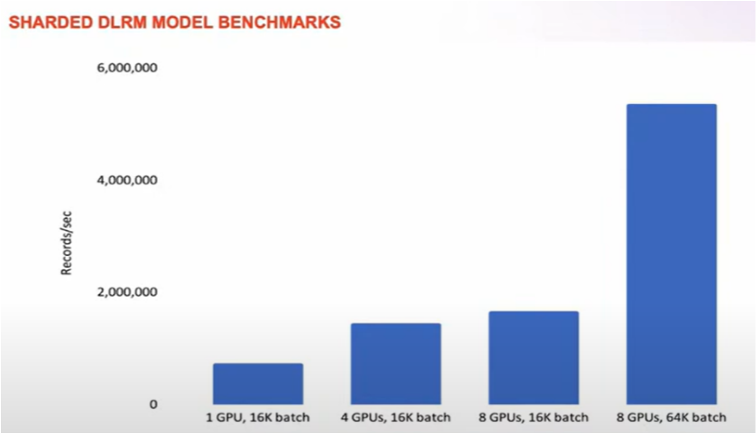


FUTUREWEI Technologies

TorchRec

- PyTorch domain library for large-scale recommender systems
  - Domain specific: TorchRec's custom PyTorch modules are built for recommender systems and optimized for distributed run-time environments.
  - Scalable: TorchRec leverages Model Parallelism and automatically adjusts as authors scale from 1 to N devices.
  - Performant: TorchRec's performance optimizations are born from research and built for production.
- Module-based model parallelism
  - TorchRec prepares PyTorch models for distributed training or inference.
- Additional features
  - Batched embeddings, fused optimizers, jagged tensors, hierarchical sharding, input batch pipelining, collectives quantization, embedded quantization, automated planning, HBM/DDR caching, …
- Performance enhancements
  - HBM (High Bandwidth Memory) and UVM (Unified Virtual Memory) optimizations
  - Sharded DLRM model over multiple GPUs

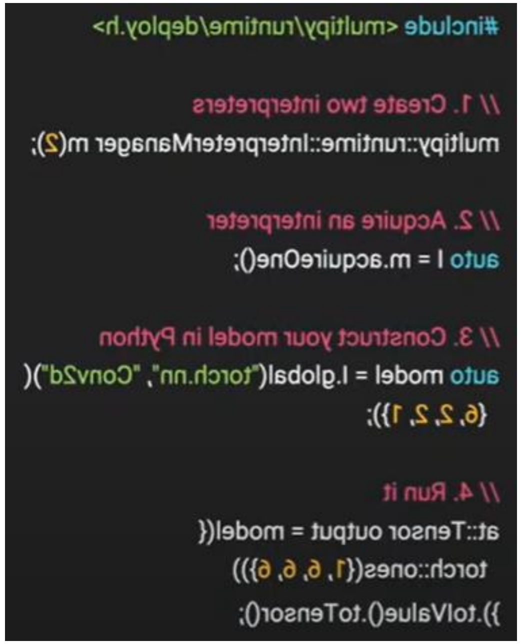


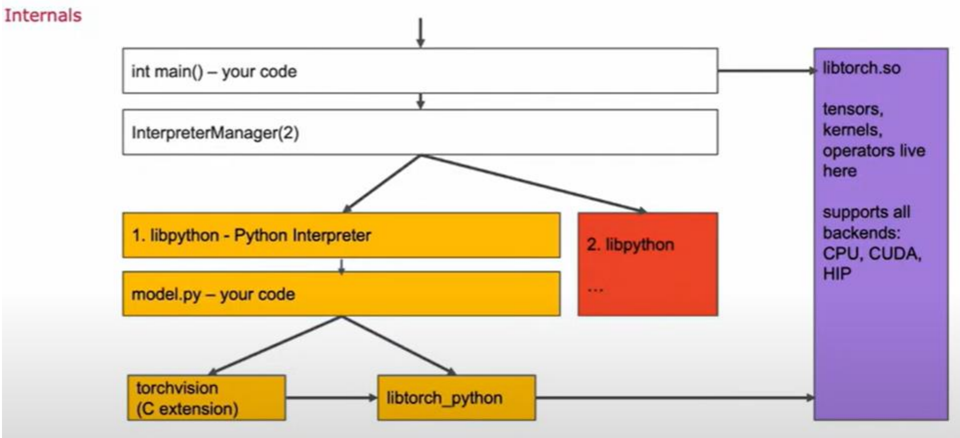


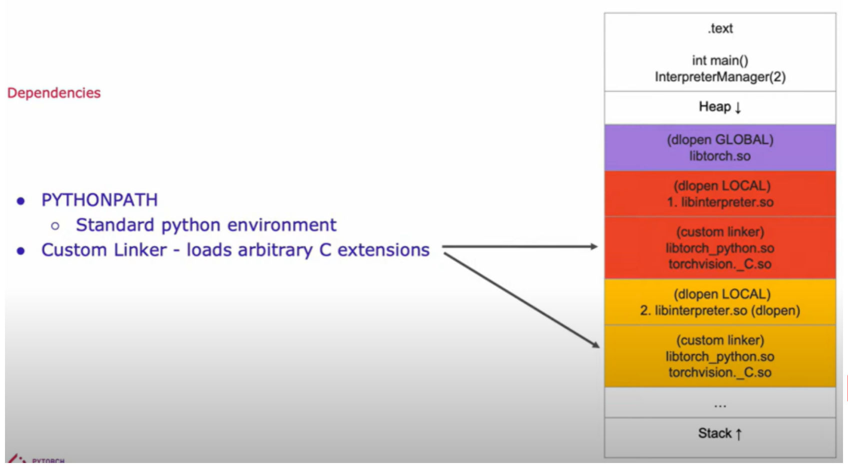
FUTUREWEI Technologies

torch::deploy (MultiPy)

- Running multiple Python interpreters inside a single process
- Running eager mode PyTorch models in production
- No GIL (Global Interpreter Lock)
- C++ library: wrapping Python object into C++
- Linux x86_64 (arm64 support as well)
- Currently in beta
- No modifications is needed, no tracing/scripting is needed
- Share the same model across multiple Python interpreters.
- Shared backend: libtorch/aten backend is the same, no extra copies of models
- No process boundaries: simplify production stack by eliminating data transfer between processes

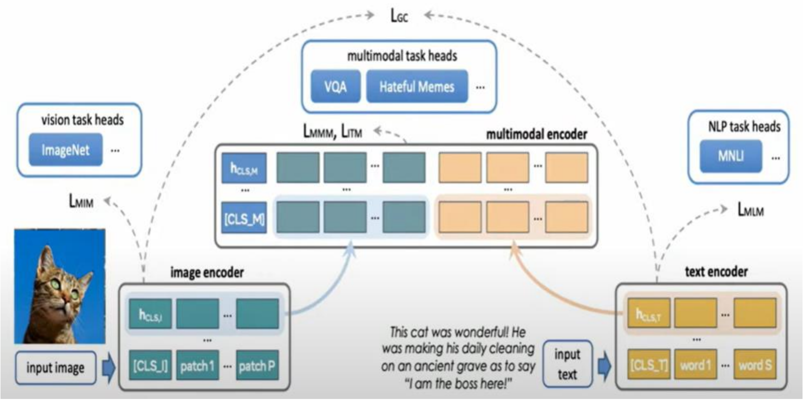






TorchMultimodal (beta)

- Tasks that can understand different types of input (e.g., textual, visual , audio)
  - Visual question answering, text to image / video retrieval, text to image generation
- Can use the understanding to generate outputs
  - Content understanding, integrity classifiers, self-driving cars
- Core principles
  - Modularity: modular low-level components can be used independently
  - Interoperability: enable to plug in components from other PyTorch libraries with minimal effort
  - Extensibility: task agnostic models can be extended with any task-specific layers
- Example offerings
  - Building blocks: vector quantized VAE, contrastive loss with temperature
  - End-to-end models: CLIP (Contrastive Language-Image Pretraining), MDETR (Modulated DETR)
  - Example scripts: Omnivore (single model for many visual modalities), FLAVA (Foundational Language And Vision Alignment Model)



FLAVA demo

FUTUREWEI Technologies

TorchRL

- Reinforcement learning and control library for PyTorch
  - Modular enough to easily swap between components
  - Syntax is familiar to RL practitioners
  - Aiming at PyTorch, but optionally support other libraries (e.g., gym/gymnasium, dm_control, habitat-lib, Jumanji, SMAC, dm_lab*, unity*, brax*)
- Efficiency
  - Efficient distributed replay buffer
  - Vectorized environments and transforms, fully operational on device
  - Vectorized advantage computation (10-100x faster)
- Modularity
  - Generic module class with few levels of abstraction (easy to hack)
  - Environments, modules, models, losses, etc. are supposed to be re-used across frameworks
  - MBRL/MFRL, off-/on-policy
  - Multi/single agent, offline RL, multi-task, distributed, meta-RL, MCTS, MBRL
- TensorDict
  - A new tensor container that allows to abstract away the irrelevant parts of each module.

On-device ML in PyTorch

- Challenges
  - More software platforms (OS, hardware)
  - More devices and hardware heterogeneity
  - Mixed compute unit execution (e.g., DSP+NPU)
  - More metrics to optimize for (e.g., power)
  - Increased model variety and complexity
  - Number and variety of experts involved
- Principles
  - Maintain PyTorch authoring semantics: clear programming model
  - Provide an extensible software stack: easy customization of the program
  - Supply out-of-the-box foundational components: portable and efficient runtime, compiler-based backends
  - Offer high developer productivity: developer SDK, documentations, examples, …
- Roadmap
  - 2019 PT1.3 experimental for Android/iOS
  - 2020 PT1.4 performance improvements
  - 2021 PT1.9 75% binary size reduction
  - Beta release coming late summer next year