

Optimizing PyTorch

Accelerating Training and Inference with Compilation, Custom Kernels, and Beyond

Alvaro Moran - Hugging Face - 2024-10-03

What is Hugging Face?

JUL
17

Founded in 2016

🏢 HQ'd in New York, offices in Paris and few other cities.

⭐ A Hub with over 1 000 000 models available.

👉 Focused on having a positive impact on the AI field.

The screenshot shows the Hugging Face website (huggingface.co) with a dark mode interface. The top navigation bar includes links for Models, Datasets, Spaces, Posts, Docs, Pricing, Log In, and Sign Up. On the left, there's a sidebar with sections for Tasks (selected), Libraries, Datasets, Languages, Licenses, Other, and Multimodal (with sub-options: Image-Text-to-Text and Visual Question Answering). The main content area features a search bar and a large heading 'Models 1,009,833'. Below this, two model cards are visible: 'stepfun-ai/GOT-OCR2_0' (Image-Text-to-Text, updated 11 days ago, 145k stars, 824 forks) and 'meta-llama/Llama-3.2-11B-Vision-Instruct' (Image-Text-to-Text, updated 1 day ago, 46.3k stars, 316 forks).

Who am I?

👋 ML Engineer, member of the *Optimization Team*.

The screenshot shows a web browser window for huggingface.co. The page displays the **Optimum** extension documentation. The top navigation bar includes links for Models, Datasets, Spaces, Posts, Docs, and Pricing. A search bar at the top left says "Search models, datasets, users...". On the left, a sidebar shows the "OVERVIEW" section for the Optimum extension, featuring a "Search documentation" bar, version V1.22.0, language EN, and a sun icon with 2,484 reviews. The main content area features a large heading "Optimum" with a smiling emoji. Below it, a paragraph explains that Optimum is an extension of [Transformers](#) for performance optimization. A quote from the text reads: "The AI ecosystem evolves quickly, and more and more specialized hardware along with their own optimizations are emerging every day. As such, Optimum enables developers to efficiently use any of these platforms with the same ease inherent to Transformers." To the right, there are sections for "Hardware partners" and "Open-source integrations", each with a small emoji icon.

Agenda

- Overview of PyTorch.
- Why optimization is useful.
- Key techniques: hardware usage, `torch.compile`, custom kernels, and mixed precision and distributed processing.



Pytorch Overview

- One of the most popular machine learning libraries.
- Accelerated tensor computing for CPUs, GPUs, etc.
- Deep neural library built on automatic differentiation system.
- Used in science to create and use models for complex tasks.



Why Do We Need Optimization

- 🏃 Faster inference and training.
- compressor Compress data and information, avoid out-of-memory errors.
- tailor Tailor a model for constrained hardware environments.

Use the Hardware

Pytorch can be accelerated on different hardware.

```
a = torch.tensor([1, 2, 3]).to("cuda")
```

```
model.cuda()
```

```
pipe = pipeline("image-segmentation",
                device="cuda",
                framework="pt")
```



torch.compile

It makes code faster when running several times on a given hardware.

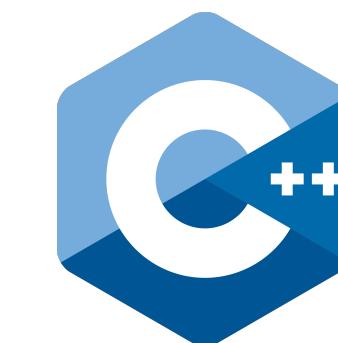
```
@torch.compile  
def foo(x, y):  
    a = torch.sin(x)  
    b = torch.cos(y)  
    return a + b  
  
outputs = foo(x, y)
```

NOTE: *torch.compile* can give massive speed-up, but it can be hard to use it on a whole model. Try using it on smaller code blocks.

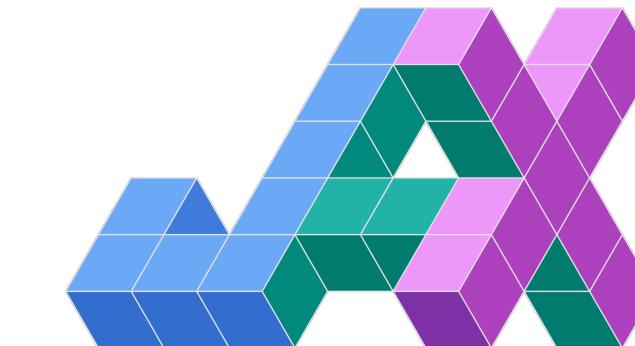
Custom Kernels

Some operations can be optimized to be even more efficient on a given hardware. To do that it's possible to use custom kernels:

- XLA and Pallas - **easy, only on GPU and TPU**
- Triton - somewhat hard - **GPU**
- CUDA (and C++ extensions) - **hard, GPU**



OpenXLA



Half and Mixed Precision, Quantization

- Use `torch.float16`, `torch.bfloat16` if the hardware you use allows it. ➔ Better performance, lower memory, similar accuracy.
- Quantize your model: use `bitsandbytes`, `optimum-quanto`, `marlin` or others. Mostly for matrix multiplication. ➔ Much lower memory, lower accuracy, but usually slower.

Distributed Inference and Training

If models are too big to fit into one GPU, it is possible to distribute processing across GPUs. To do that, it is possible to use different tools:

- Use `torch.distributed` to split processing across GPUs and synchronize (this can be difficult).
- When training, consider using Fully Sharded Data Parallel (FSDP) or similar techniques to *shard* the model across GPUs. See `torch.distributed.fsdp.FullyShardedDataParallel`
- Consider using HuggingFace's `accelerate` library to simplify these scenarios.

Practical Example

A [Jupyter notebook](#) is available to walk through some of the mentioned techniques, and it is possible to run it on a common laptop.

The screenshot shows a Jupyter notebook interface running on a GitHub page. The title of the notebook is "Optimize Inference on a Llama 3.2 1B on Pytorch". The first cell (In [1]) contains code to import time, torch, and AutoTokenizer, AutoModelForCausalLM from the transformers library. The second cell (In [2]) defines a model ID ("meta-llama/Llama-3.2-1B") and uses the transformers utilities to load the model and tokenizer from the Hugging Face hub. A note in cell [2] specifies setting `pad_token_id` to `eos_token_id` for open-end generation. The final note at the bottom states that no particular parameters have been used, meaning the model will be loaded on CPU with FP32 precision.

```
In [1]:  
import time  
import torch  
from transformers import AutoTokenizer, AutoModelForCausalLM  
  
In [2]:  
model_id = "meta-llama/Llama-3.2-1B"  
model = AutoModelForCausalLM.from_pretrained(model_id)  
tokenizer = AutoTokenizer.from_pretrained(model_id)  
# Setting `pad_token_id` to `eos_token_id` for open-end generation.  
model.generation_config.pad_token_id = tokenizer.eos_token_id  
  
Note no particular parameters have been used, meaning this will be loaded on CPU with FP32 precision.
```

Thank You!

