



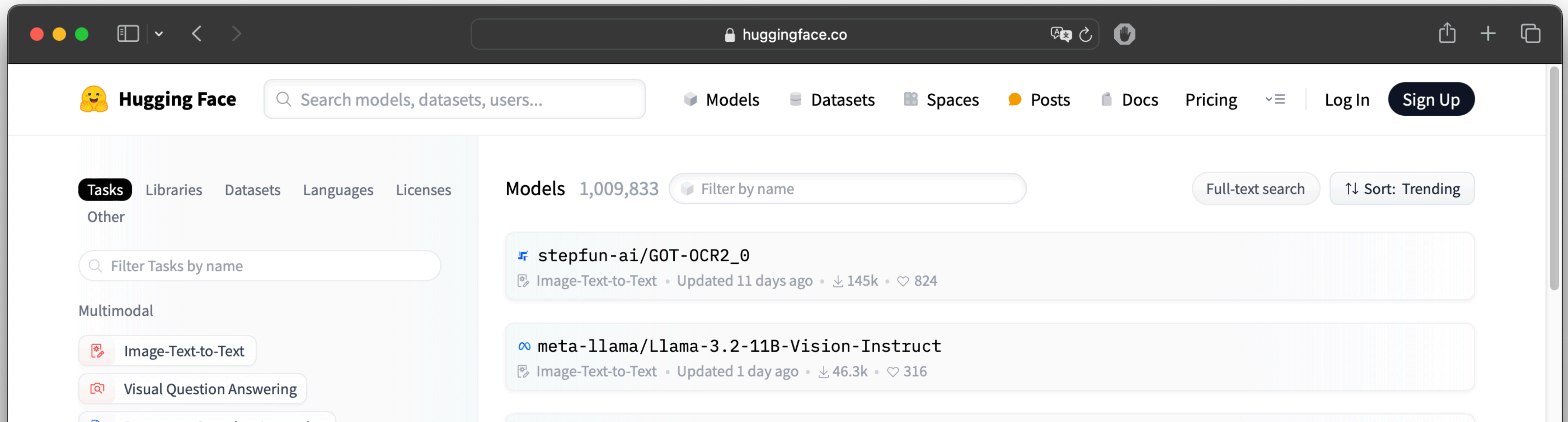
Optimizing PyTorch

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

Alvaro Moran - Hugging Face - 2024-10-03

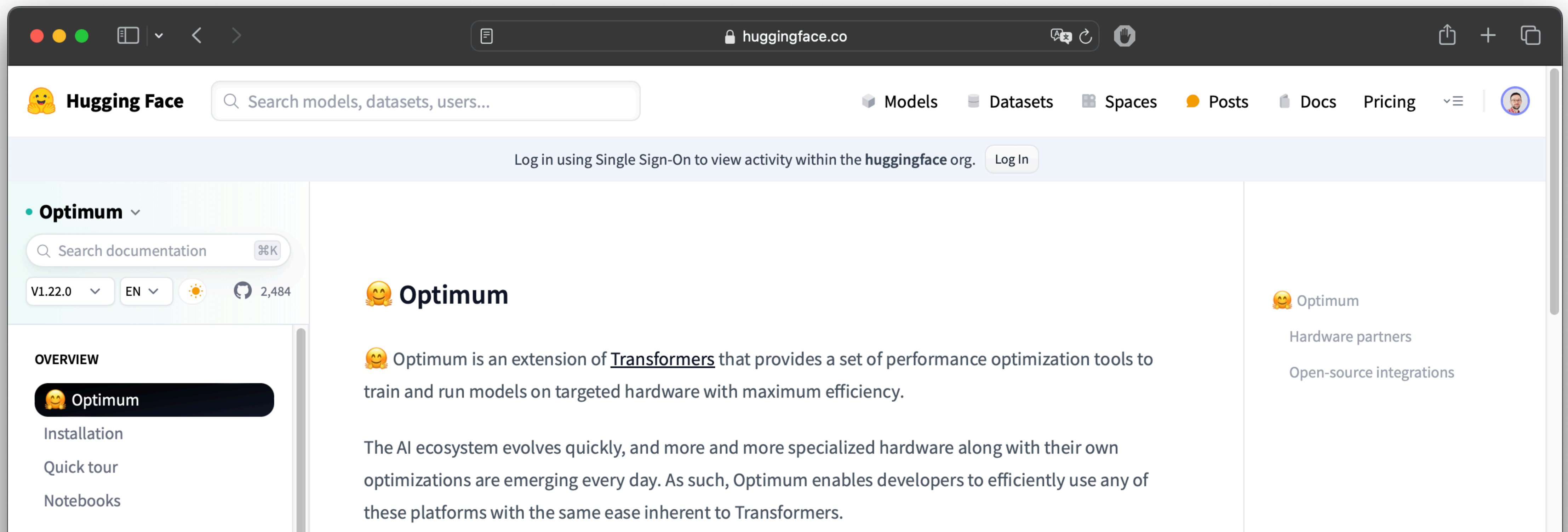
What is Hugging Face?

- 📅 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.



Who am I?

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



The screenshot shows the Hugging Face website interface. At the top, there is a navigation bar with the Hugging Face logo, a search bar, and links for Models, Datasets, Spaces, Posts, Docs, and Pricing. Below the navigation bar, there is a light blue banner with the text "Log in using Single Sign-On to view activity within the huggingface.org." and a "Log In" button. The main content area is divided into three columns. The left column is a sidebar with a search bar for documentation, version and language dropdowns (V1.22.0 and EN), and a list of links under the "OVERVIEW" section: Optimum (highlighted), Installation, Quick tour, and Notebooks. The middle column is the main content area, featuring the "Optimum" heading with a smiley face emoji, a paragraph describing it as an extension of Transformers for performance optimization, and a paragraph explaining its role in the AI ecosystem. The right column is a sidebar with the "Optimum" heading and two links: "Hardware partners" and "Open-source integrations".

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.
-  Compress data and information, avoid out-of-memory errors.
-  Tailor a model for constrained hardware environments.

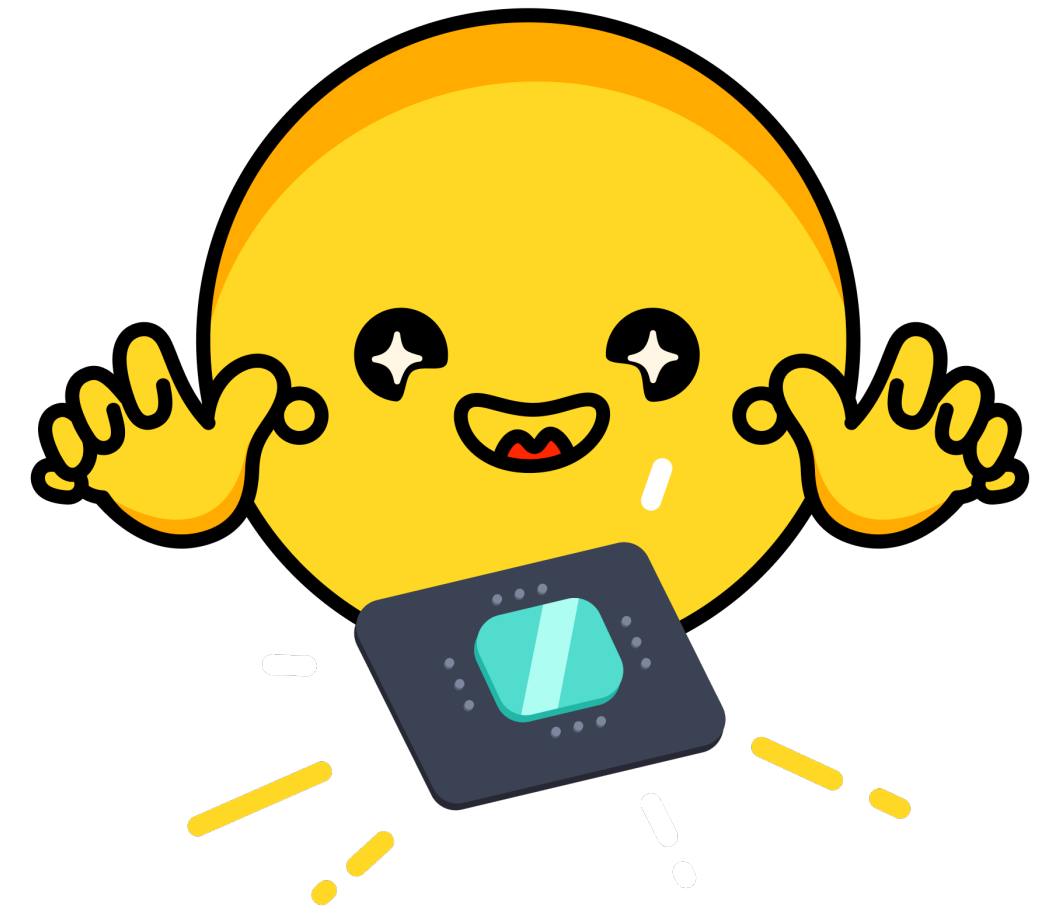
Use the Hardware

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
```

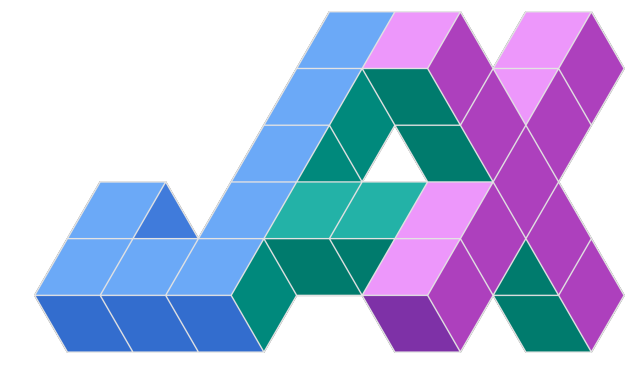
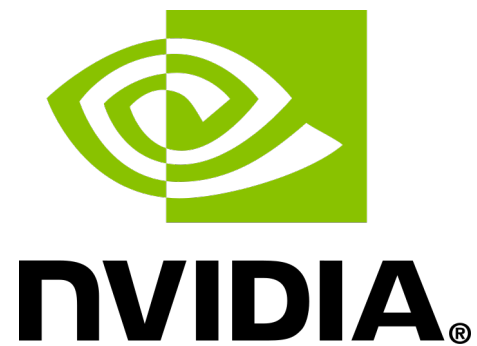
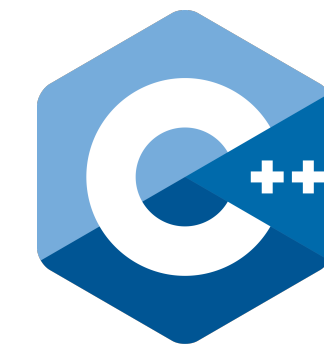
```
outputs = foo(x, y)
```

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**



Half and Mixed Precision, Quantization

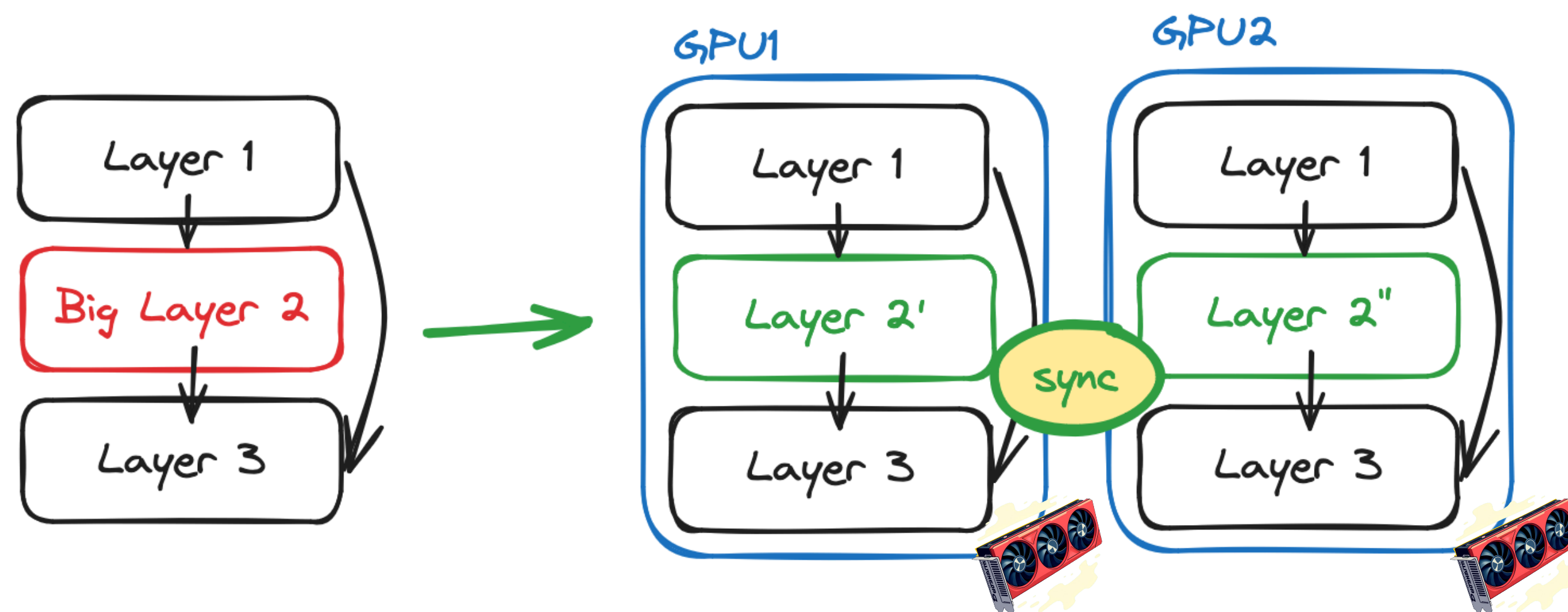
- Use `torch.float16`, `torch.bfloat16` if the hardware you use allows it. ➡ Better performance, lower memory, similar accuracy.
- Quantize your model: use `bitsandbytes`, `optimum-quant`, `marlin` or others. Mostly for matrix multiplication. ➡ Much lower memory, lower accuracy, but usually slower.



Distributed Inference and Training

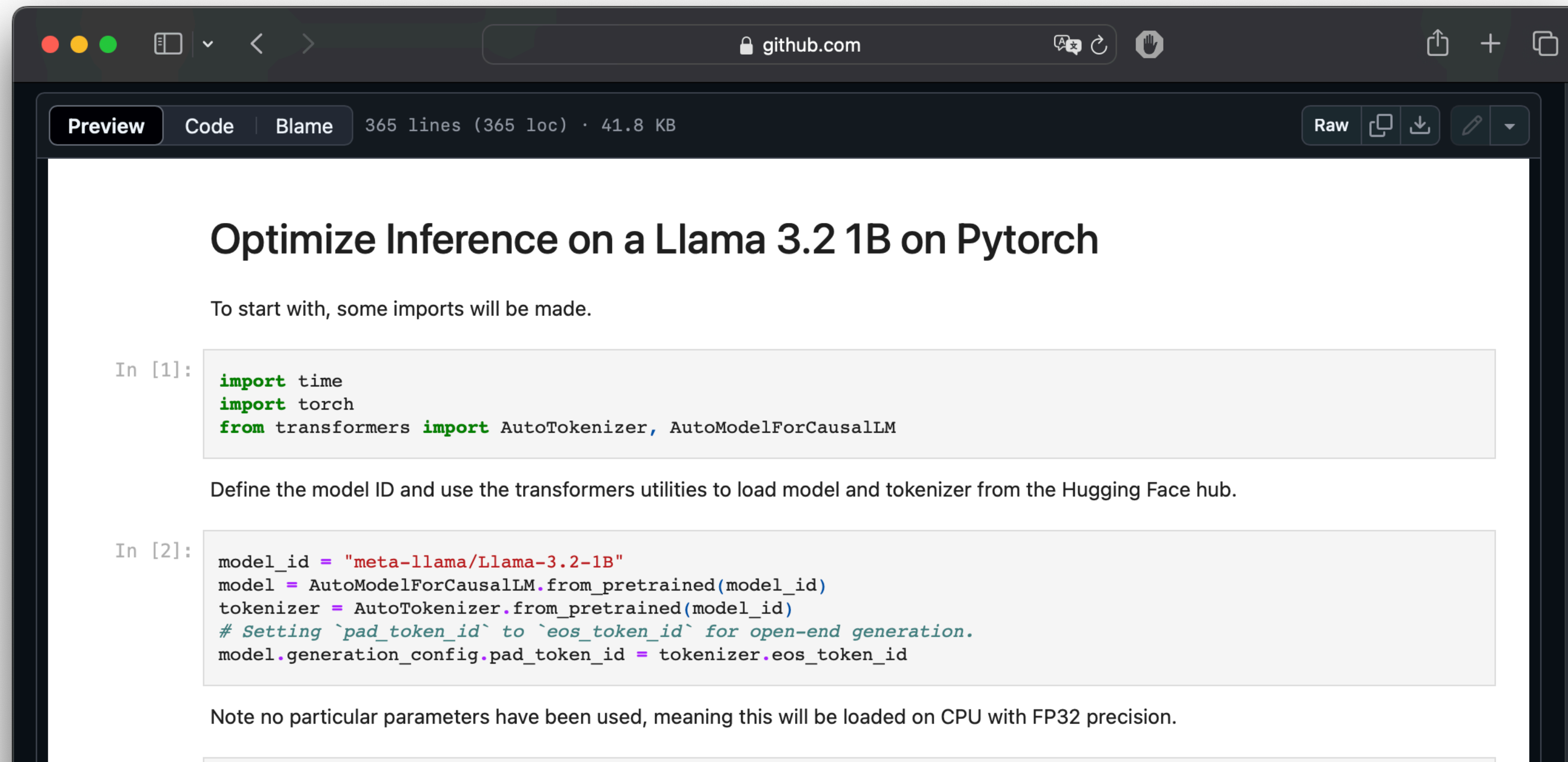
Model too big? Distribute!

- Inference → `torch.distributed`, difficult.
- Training, → Fully Sharded Data Parallel (FSDP), DeepZero.
- Use 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 GitHub repository page for a Jupyter notebook. The browser address bar shows 'github.com'. The repository page has tabs for 'Preview', 'Code', and 'Blame', with 'Preview' selected. The file name is 'Optimize Inference on a Llama 3.2 1B on Pytorch', and it shows '365 lines (365 loc) · 41.8 KB'. There are icons for 'Raw', 'Copy', 'Download', and 'Edit'. The notebook content is displayed in a light gray box with a dark background. It starts with a text block: 'To start with, some imports will be made.' followed by a code cell 'In [1]:' containing Python code for imports. Below that is another text block: 'Define the model ID and use the transformers utilities to load model and tokenizer from the Hugging Face hub.' followed by a code cell 'In [2]:' containing Python code for loading the model and tokenizer. At the bottom, there is a text block: 'Note no particular parameters have been used, meaning this will be loaded on CPU with FP32 precision.'

```
In [1]: import time
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM

Define the model ID and use the transformers utilities to load model and tokenizer from the Hugging Face hub.

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!



huggingface.co