model_optimizations

May 17, 2025

1 Inference Optimizations

Note: used google collab for GPU because of time constraint.
In real pipeline this would be done after training on GPU instance.

1.1 Pre-process Data

[1]:
```
# Upload kaggle.json
from google.colab import files
files.upload()

# Move to the correct path
!mkdir -p /content/.kaggle
!cp kaggle.json /content/.kaggle/
!chmod 600 /content/.kaggle/kaggle.json

# Set the environment variable so the API knows where to look
import os
os.environ['KAGGLE_CONFIG_DIR'] = "/content/.kaggle"

# Test
!kaggle datasets list
```

```
<IPython.core.display.HTML object>

Saving kaggle.json to kaggle.json
ref                                                            title
size  lastUpdated                 downloadCount  voteCount  usabilityRating
-------------------------------------------------------------
-------------------------------------------------  -----------
-------------------------  -------------  ---------  ---------------
jayaantanaath/student-habits-vs-academic-performance         Student Habits vs
Academic Performance                     19512  2025-04-12 10:49:08.663000
24300        415  1.0
adilshamim8/cost-of-international-education                  Cost of
International Education                                18950  2025-05-07
15:41:53.213000           5202         87  1.0
adilshamim8/social-media-addiction-vs-relationships          Students' Social
Media Addiction                          7851  2025-05-10 14:38:02.713000
```

2519 39 1.0
ivankmk/thousand-ml-jobs-in-usa Machine Learning
Job Postings in the US 1682058 2025-04-20 16:11:59.347000
4919 108 1.0
fatemehmohammadinia/heart-attack-dataset-tarik-a-rashid Heart Attack
Dataset 16250 2025-04-30 21:58:22.740000
4817 80 1.0
mahdimashayekhi/fake-news-detection-dataset Fake News Detection
Dataset 11735585 2025-04-27 14:52:10.607000
2075 28 1.0
michaelmatta0/global-development-indicators-2000-2020 Global Development
Full Analysis (2000-2020) 1311638 2025-05-11 16:57:19.013000
842 26 1.0
madhuraatmarambhagat/crop-recommendation-dataset Crop Recommendation
Dataset 65234 2025-05-08 17:02:09.397000
918 29 1.0
aryan208/financial-transactions-dataset-for-fraud-detection Financial
Transactions Dataset for Fraud Detection 290256858 2025-05-02
09:12:28.203000 1296 30 1.0
umeradnaan/daily-social-media-active-users Daily Social Media
Active Users 126814 2025-05-05 02:11:50.873000
1747 23 1.0
khushikyad001/impact-of-screen-time-on-mental-health Impact of Screen
Time on Mental Health 64873 2025-04-20 18:01:47.570000
2854 43 1.0
zahidmughal2343/global-cancer-patients-2015-2024
global_cancer_patients_2015_2024 1261049 2025-04-14
00:05:23.367000 5348 65 1.0
razanaqvi14/real-and-fake-news Real & Fake News
42975911 2025-04-28 19:46:53.073000 1133 24 1.0
wikimedia-foundation/wikipedia-structured-contents Wikipedia
Structured Contents 25121685657 2025-04-11
07:11:03.397000 2402 298 0.8125
adilshamim8/greenhouse-plant-growth-metrics Greenhouse Plant
Growth 3041046 2025-04-19 07:33:57.787000
1833 28 1.0
stephennanga/malawi-datasets Malawi National
Football Team Matches 1306 2025-04-07 23:07:51.297000
1404 43 1.0
nikolasgegenava/sneakers-classification Popular Sneakers
Classification 17981294 2025-05-01 12:00:45.517000
1677 46 1.0
dnkumars/cryptocurrency-transaction-analytics-btc-and-eth Cryptocurrency
Transaction Analytics: BTC & ETH 5167978 2025-05-11 15:16:52.107000
441 38 1.0
palvinder2006/ola-bike-ride-request Ola Bike Ride
Request 174975 2025-04-28 03:55:33.860000
1123 28 1.0

```
brendanartley/openfwi-preprocessed-72x72                    OpenFWI
Preprocessed 72x72                          21254845946  2025-05-13
22:11:02.327000            501          18  1.0
```

[2]:
```python
import argparse
import os
import pandas as pd
from sklearn.model_selection import train_test_split
import zipfile

# os.environ["KAGGLE_CONFIG_DIR"] = os.path.abspath(".kaggle") # Use local .
↪kaggle directory
from kaggle.api.kaggle_api_extended import KaggleApi
```

[3]:
```python
def download_jigsaw(kaggle_dir):
    os.makedirs(kaggle_dir, exist_ok=True)

    api = KaggleApi()
    api.authenticate()

    # Download competition data
    api.competition_download_files(
        "jigsaw-unintended-bias-in-toxicity-classification",
        path=kaggle_dir
    )

    # Unzip
    zip_path = os.path.join(kaggle_dir,␣
↪"jigsaw-unintended-bias-in-toxicity-classification.zip")
    with zipfile.ZipFile(zip_path, "r") as zip_ref:
        zip_ref.extractall(kaggle_dir)

    print("Downloaded and extracted Jigsaw dataset.")


def preprocess(kaggle_dir, output_dir, split_ratio=0.2):
    # Create output directory if missing
    os.makedirs(output_dir, exist_ok=True)

    # Ensure input file exists, create parent dir if needed (just in case)
    os.makedirs(kaggle_dir, exist_ok=True)
    input_path = os.path.join(kaggle_dir, "train.csv")
    if not os.path.exists(input_path):
        raise FileNotFoundError(f"train.csv not found in {kaggle_dir}")

    df = pd.read_csv(input_path).dropna(subset=["comment_text"])
```

```
    # Keep only the needed columns
    df = df[["comment_text", "target"]]

    # Binarize target (optional: uncomment if needed)
    # df["target"] = (df["target"] >= 0.5).astype(int)

    # Split
    train_df, val_df = train_test_split(df, test_size=split_ratio,␣
 ↪random_state=42)

    os.makedirs(output_dir, exist_ok=True)
    train_df.to_csv(os.path.join(output_dir, "train.csv"), index=False)
    val_df.to_csv(os.path.join(output_dir, "val.csv"), index=False)

    print(f"Saved {len(train_df)} training and {len(val_df)} validation samples␣
 ↪to {output_dir}")
```

[4]:
```
kaggle_dir = "data/jigsaw/raw/"
output_dir = "data/jigsaw/processed/"
val_split = 0.2

download_jigsaw(kaggle_dir)
preprocess(kaggle_dir, output_dir, val_split)
```

```
Downloaded and extracted Jigsaw dataset.
Saved 1443896 training and 360975 validation samples to data/jigsaw/processed/
```

1.2 Training

[5]:
```
import os
import time
import argparse
import torch
import torch.nn as nn
import torch.optim as optim
import pandas as pd
from torch.utils.data import DataLoader, Dataset
from transformers import AutoTokenizer, AutoModelForSequenceClassification
```

[6]:
```
config = {
    "initial_epochs": 2,
    "total_epochs": 1,
    "patience": 2,
    "batch_size": 128,
    "lr": 2e-5,
    "fine_tune_lr": 1e-5,
    "max_len": 128,
```

```
    "dropout_probability": 0.3,
    "model_name": "google/bert_uncased_L-2_H-128_A-2"
}
```

[7]:
```
# ----------------------------
# Dataset
# ----------------------------
class JigsawDataset(Dataset):
    def __init__(self, df, tokenizer, max_len):
        self.texts = df["comment_text"].tolist()
        self.labels = (df["target"] >= 0.5).astype(int).tolist()
        self.tokenizer = tokenizer
        self.max_len = max_len

    def __len__(self):
        return len(self.texts)

    def __getitem__(self, idx):
        inputs = self.tokenizer(
            self.texts[idx],
            truncation=True,
            padding="max_length",
            max_length=self.max_len,
            return_tensors="pt"
        )
        return {
            "input_ids": inputs["input_ids"].squeeze(0),
            "attention_mask": inputs["attention_mask"].squeeze(0),
            "labels": torch.tensor(self.labels[idx], dtype=torch.long)
        }
```

[8]:
```
# ----------------------------
# Training + Evaluation Functions
# ----------------------------
from tqdm import tqdm

def train_epoch(model, loader, criterion, optimizer, device, portion=0.01):
    model.train()
    total_loss, correct, total = 0, 0, 0

    num_batches = int(portion * len(loader)) # Doing part of the training␣
 ↪because my part is inference and monitoring
    print(f"Training for {num_batches} batches")

    for i, batch in enumerate(tqdm(loader, desc="Training", leave=False)):
        if i >= num_batches:
            break
```

```
        optimizer.zero_grad()
        batch = {k: v.to(device) for k, v in batch.items()}
        outputs = model(**batch)
        loss = criterion(outputs.logits.view(-1), batch["labels"].float())
        loss.backward()
        optimizer.step()

        total_loss += loss.item()
        probs = torch.sigmoid(outputs.logits.view(-1))
        preds = (probs > 0.5).long()
        correct += (preds == batch["labels"]).sum().item()
        total += batch["labels"].size(0)

    avg_loss = total_loss / num_batches
    avg_acc = correct / total
    print(f"Partial Epoch Summary - Avg Loss: {avg_loss:.4f}, Avg Accuracy:␣
↪{avg_acc:.4f}\n")

    return avg_loss, avg_acc

def evaluate(model, loader, criterion, device, portion=0.01):
    model.eval()
    total_loss, correct, total = 0, 0, 0

    num_batches = int(portion * len(loader))
    print(f"Evaluating for {num_batches} batches")

    with torch.no_grad():
        for i, batch in enumerate(tqdm(loader, desc="Evaluating", leave=False)):
            if i >= num_batches:
                break

            batch = {k: v.to(device) for k, v in batch.items()}
            outputs = model(**batch)
            loss = criterion(outputs.logits.view(-1), batch["labels"].float())
            total_loss += loss.item()

            probs = torch.sigmoid(outputs.logits.view(-1))
            preds = (probs > 0.5).long()
            correct += (preds == batch["labels"]).sum().item()
            total += batch["labels"].size(0)

    avg_loss = total_loss / num_batches
    avg_acc = correct / total
    print(f"Eval Summary - Avg Loss: {avg_loss:.4f}, Accuracy: {avg_acc:.4f}\n")
```

```
    return avg_loss, avg_acc
```

[9]:
```
# ---------------------------
# Main Training Pipeline
# ---------------------------
def main(args):
    # made to run in command line originally
    # parser = argparse.ArgumentParser()
    # parser.add_argument("--data-dir", type=str, required=True,␣
 ↪help="Directory with train.csv and val.csv")
    # parser.add_argument("--save-path", type=str, required=True, help="Path to␣
 ↪save the trained model")
    # parser.add_argument("--dry-run", action="store_true", help="Run a quick␣
 ↪test on a small sample")
    # args = parser.parse_args()

    os.makedirs(os.path.dirname(args.save_path), exist_ok=True)
    device = torch.device("cuda" if torch.cuda.is_available() else ("mps" if␣
 ↪torch.backends.mps.is_available() else "cpu"))
    print(f"Using device: {device}")

    tokenizer = AutoTokenizer.from_pretrained(config["model_name"])
    train_df = pd.read_csv(os.path.join(args.data_dir, "train.csv"))
    val_df = pd.read_csv(os.path.join(args.data_dir, "val.csv"))

    if args.dry_run:
        train_df = train_df.sample(n=32, random_state=42)
        val_df = val_df.sample(n=32, random_state=42)

    train_loader = DataLoader(JigsawDataset(train_df, tokenizer,␣
 ↪config["max_len"]), batch_size=config["batch_size"], shuffle=True,␣
 ↪num_workers=4, pin_memory=True)
    val_loader = DataLoader(JigsawDataset(val_df, tokenizer,␣
 ↪config["max_len"]), batch_size=config["batch_size"], num_workers=8,␣
 ↪pin_memory=True)


    model = AutoModelForSequenceClassification.from_pretrained(
      config["model_name"],
      num_labels=1  # binary classification
    )
    model.to(device)

    criterion = nn.BCEWithLogitsLoss()
    optimizer = optim.Adam(model.parameters(), lr=config["lr"])

    best_val_loss = float("inf")
```

```
    patience_counter = 0
    batch_portion = 1.0

    for epoch in range(config["total_epochs"]):
        start = time.time()

        train_loss, train_acc = train_epoch(model, train_loader, criterion,␣
↪optimizer, device, portion=batch_portion)
        val_loss, val_acc = evaluate(model, val_loader, criterion, device,␣
↪portion=batch_portion)

        print(f"Epoch {epoch+1}: Train Loss={train_loss:.4f} Acc={train_acc:.
↪4f} | Val Loss={val_loss:.4f} Acc={val_acc:.4f} | Time={time.time() - start:.
↪2f}s")

        if val_loss < best_val_loss:
            best_val_loss = val_loss
            torch.save(model.state_dict(), args.save_path)
            patience_counter = 0
            print("  Validation loss improved. Model saved.")
        else:
            patience_counter += 1
            print(f"  No improvement. Patience: {patience_counter}")
            if patience_counter >= config["patience"]:
                print("  Early stopping.")
                break
```

[10]:
```
# simulate arguments
class args:
  data_dir = "data/jigsaw/processed/"
  save_path = "models/model.pth"
  dry_run = False

main(args)
```

```
Using device: cuda

/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94:
UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab
(https://huggingface.co/settings/tokens), set it as secret in your Google Colab
and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access
public models or datasets.
  warnings.warn(
Some weights of BertForSequenceClassification were not initialized from the
```

```
model checkpoint at google/bert_uncased_L-2_H-128_A-2 and are newly initialized:
['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.

Training for 11281 batches


Partial Epoch Summary - Avg Loss: 0.1735, Avg Accuracy: 0.9397

Evaluating for 2821 batches


Eval Summary - Avg Loss: 0.1372, Accuracy: 0.9476

Epoch 1: Train Loss=0.1735 Acc=0.9397 | Val Loss=0.1372 Acc=0.9476 |
Time=217.46s
  Validation loss improved. Model saved.
```

1.3 Inference Optimization

[11]:
```
!pip install torchinfo
```

```
Collecting torchinfo
  Downloading torchinfo-1.8.0-py3-none-any.whl.metadata (21 kB)
Downloading torchinfo-1.8.0-py3-none-any.whl (23 kB)
Installing collected packages: torchinfo
Successfully installed torchinfo-1.8.0
```

[12]:
```
import os
import torch
from torch.utils.data import DataLoader, Dataset
from transformers import DistilBertTokenizer,␣
 ↪DistilBertForSequenceClassification
from torchinfo import summary
import time
import numpy as np
import pandas as pd
```

[13]:
```
class JigsawDataset(Dataset):
    def __init__(self, df, tokenizer, max_len):
        self.texts = df["comment_text"].tolist()
        self.labels = (df["target"] >= 0.5).astype(int).tolist()
        self.tokenizer = tokenizer
        self.max_len = max_len

    def __len__(self):
        return len(self.texts)
```

```
    def __getitem__(self, idx):
        inputs = self.tokenizer(
            self.texts[idx],
            truncation=True,
            padding="max_length",
            max_length=self.max_len,
            return_tensors="pt"
        )
        return {
            "input_ids": inputs["input_ids"].squeeze(0),
            "attention_mask": inputs["attention_mask"].squeeze(0),
            "labels": torch.tensor(self.labels[idx], dtype=torch.long)
        }
```

[21]:
```
batch_size = 128
max_len = 128
model_name = "google/bert_uncased_L-2_H-128_A-2"
dataset_dir = os.getenv("DATA_DIR", "data/jigsaw/processed")
model_path = "models/model.pth"
```

[41]:
```
val_df = pd.read_csv(os.path.join(dataset_dir, "val.csv"))

tokenizer = AutoTokenizer.from_pretrained(model_name)
test_loader = DataLoader(JigsawDataset(val_df, tokenizer, max_len),␣
 ↪batch_size=batch_size, shuffle=False, num_workers=8)
```

1.3.1 Measure inference performance of PyTorch model on CPU

[52]:
```
device = torch.device("cpu")
model = AutoModelForSequenceClassification.from_pretrained(model_name,␣
 ↪num_labels=1)
state_dict = torch.load(model_path, map_location=device)
model.load_state_dict(state_dict)
model.compile() # Test Compile mode
model.eval()
summary(model)
```

```
Some weights of BertForSequenceClassification were not initialized from the
model checkpoint at google/bert_uncased_L-2_H-128_A-2 and are newly initialized:
['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

[52]:
```
==========================================================================
=====
Layer (type:depth-idx)                                   Param #
```

```
==========================================================================
=====
BertForSequenceClassification                                --
 BertModel: 1-1                                              --
      BertEmbeddings: 2-1                                    --
           Embedding: 3-1                                    3,906,816
           Embedding: 3-2                                    65,536
           Embedding: 3-3                                    256
           LayerNorm: 3-4                                    256
           Dropout: 3-5                                      --
      BertEncoder: 2-2                                       --
           ModuleList: 3-6                                   396,544
      BertPooler: 2-3                                        --
           Linear: 3-7                                       16,512
           Tanh: 3-8                                         --
 Dropout: 1-2                                                --
 Linear: 1-3                                                 129
==========================================================================
=====
Total params: 4,386,049
Trainable params: 4,386,049
Non-trainable params: 0
==========================================================================
=====
```

[53]:
```python
model_size = os.path.getsize(model_path)
print(f"Model Size on Disk: {model_size/ (1e6) :.2f} MB")
```

```
Model Size on Disk: 17.56 MB
```

[54]:
```python
def evaluate_test(model, loader, device=None, portion=0.01):
    if device is None:
        device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    model.to(device) # test accuracy of gpu if no device specified
    model.eval()
    correct, total = 0, 0

    num_batches = int(portion * len(loader))
    print(f"Evaluating for {num_batches} batches on device: {device}")

    with torch.no_grad():
        for i, batch in enumerate(tqdm(loader, desc="Evaluating", leave=False)):
            if i >= num_batches:
                break

            batch = {k: v.to(device) for k, v in batch.items()}
            outputs = model(**batch)
```

```
            preds = outputs.logits.argmax(dim=1)
            correct += (preds == batch["labels"]).sum().item()
            total += batch["labels"].size(0)

    return correct, total
```

[55]:
```
correct, total = evaluate_test(model, test_loader, portion=1.0)
accuracy = (correct / total) * 100
print(f"Accuracy: {accuracy:.2f}% ({correct}/{total} correct)")
```

```
Evaluating for 2821 batches on device: cuda

Evaluating:   0%|          | 0/2821 [00:00<?,
?it/s]/usr/local/lib/python3.11/dist-packages/torch/_inductor/compile_fx.py:194:
UserWarning: TensorFloat32 tensor cores for float32 matrix multiplication
available but not enabled. Consider setting
`torch.set_float32_matmul_precision('high')` for better performance.
  warnings.warn(

Accuracy: 92.02% (332165/360975 correct)
```

Inference Latency

[56]:
```
num_trials = 100

# 1) get one batch as a dict
batch = next(iter(test_loader))
# 2) extract the first example and move to device
input_ids      = batch["input_ids"][0].unsqueeze(0).to(device)
attention_mask = batch["attention_mask"][0].unsqueeze(0).to(device)

model.to(device)
model.eval()

# 3) warm-up
with torch.no_grad():
    _ = model(input_ids=input_ids, attention_mask=attention_mask)

# 4) timed runs
latencies = []
for _ in range(num_trials):
    start = time.perf_counter()
    with torch.no_grad():
        _ = model(input_ids=input_ids, attention_mask=attention_mask)
    latencies.append(time.perf_counter() - start)
```

[57]:
```
print(f"Inference Latency (single sample, median): {np.percentile(latencies,␣
↪50) * 1000:.2f} ms")
```

```
print(f"Inference Latency (single sample, 95th percentile): {np.
 ↪percentile(latencies, 95) * 1000:.2f} ms")
print(f"Inference Latency (single sample, 99th percentile): {np.
 ↪percentile(latencies, 99) * 1000:.2f} ms")
print(f"Inference Throughput (single sample): {num_trials/np.sum(latencies):.
 ↪2f} FPS")
```

```
Inference Latency (single sample, median): 1.82 ms
Inference Latency (single sample, 95th percentile): 2.02 ms
Inference Latency (single sample, 99th percentile): 2.28 ms
Inference Throughput (single sample): 540.00 FPS
```

Batch throughput

[58]:
```
num_batches = 10  # Number of trials

# 1) Grab one batch (a dict) and move to device, dropping labels
batch = next(iter(test_loader))
batch = {k: v.to(device) for k, v in batch.items() if k != "labels"}

model.to(device)
model.eval()

# 2) Warm-up
with torch.no_grad():
    model(**batch)

# 3) Timed runs
batch_times = []
for _ in range(num_batches):
    start = time.perf_counter()
    with torch.no_grad():
        model(**batch)
    batch_times.append(time.perf_counter() - start)
```

[59]:
```
# assume `batch` is the dict you moved to device and `batch_times` is your list␣
 ↪of durations
batch_size    = batch["input_ids"].shape[0]
total_samples = batch_size * num_batches
batch_fps     = total_samples / np.sum(batch_times)

print(f"Batch Throughput: {batch_fps:.2f} FPS")
```

```
Batch Throughput: 2170.57 FPS
```

Summary

[60]:
```
print(f"Model Size on Disk: {model_size/ (1e6) :.2f} MB")
print(f"Accuracy: {accuracy:.2f}% ({correct}/{total} correct)")
```

```
print(f"Inference Latency (single sample, median): {np.percentile(latencies,␣
↪50) * 1000:.2f} ms")
print(f"Inference Latency (single sample, 95th percentile): {np.
↪percentile(latencies, 95) * 1000:.2f} ms")
print(f"Inference Latency (single sample, 99th percentile): {np.
↪percentile(latencies, 99) * 1000:.2f} ms")
print(f"Inference Throughput (single sample): {num_trials/np.sum(latencies):.
↪2f} FPS")
print(f"Batch Throughput: {batch_fps:.2f} FPS")
```

```
Model Size on Disk: 17.56 MB
Accuracy: 92.02% (332165/360975 correct)
Inference Latency (single sample, median): 1.82 ms
Inference Latency (single sample, 95th percentile): 2.02 ms
Inference Latency (single sample, 99th percentile): 2.28 ms
Inference Throughput (single sample): 540.00 FPS
Batch Throughput: 2170.57 FPS
```

Eager mode Summary Model Size on Disk: 17.56 MB
Accuracy: 92.02% (332165/360975 correct)
Inference Latency (single sample, median): 3.61 ms
Inference Latency (single sample, 95th percentile): 4.03 ms
Inference Latency (single sample, 99th percentile): 4.76 ms
Inference Throughput (single sample): 270.66 FPS
Batch Throughput: 1358.45 FPS

Compiled Summary Model Size on Disk: 17.56 MB
Accuracy: 92.02% (332165/360975 correct)
Inference Latency (single sample, median): 1.82 ms
Inference Latency (single sample, 95th percentile): 2.02 ms
Inference Latency (single sample, 99th percentile): 2.28 ms
Inference Throughput (single sample): 540.00 FPS
Batch Throughput: 2170.57 FPS

1.3.2 Measure inference performance of ONNX model on CPU¶

[61]:
```
!pip install onnx onnxruntime-gpu
```

```
Collecting onnx
  Downloading
onnx-1.18.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
(6.9 kB)
Collecting onnxruntime-gpu
  Downloading onnxruntime_gpu-1.22.0-cp311-cp311-
manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl.metadata (4.9 kB)
Requirement already satisfied: numpy>=1.22 in /usr/local/lib/python3.11/dist-
```

```
packages (from onnx) (2.0.2)
Requirement already satisfied: protobuf>=4.25.1 in
/usr/local/lib/python3.11/dist-packages (from onnx) (5.29.4)
Requirement already satisfied: typing_extensions>=4.7.1 in
/usr/local/lib/python3.11/dist-packages (from onnx) (4.13.2)
Collecting coloredlogs (from onnxruntime-gpu)
  Downloading coloredlogs-15.0.1-py2.py3-none-any.whl.metadata (12 kB)
Requirement already satisfied: flatbuffers in /usr/local/lib/python3.11/dist-
packages (from onnxruntime-gpu) (25.2.10)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-
packages (from onnxruntime-gpu) (24.2)
Requirement already satisfied: sympy in /usr/local/lib/python3.11/dist-packages
(from onnxruntime-gpu) (1.13.1)
Collecting humanfriendly>=9.1 (from coloredlogs->onnxruntime-gpu)
  Downloading humanfriendly-10.0-py2.py3-none-any.whl.metadata (9.2 kB)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
/usr/local/lib/python3.11/dist-packages (from sympy->onnxruntime-gpu) (1.3.0)
Downloading
onnx-1.18.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (17.6 MB)
                         17.6/17.6 MB
66.4 MB/s eta 0:00:00
Downloading onnxruntime_gpu-1.22.0-cp311-cp311-
manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl (283.2 MB)
                         283.2/283.2 MB
6.4 MB/s eta 0:00:00
Downloading coloredlogs-15.0.1-py2.py3-none-any.whl (46 kB)
                         46.0/46.0 kB
4.0 MB/s eta 0:00:00
Downloading humanfriendly-10.0-py2.py3-none-any.whl (86 kB)
                         86.8/86.8 kB
9.3 MB/s eta 0:00:00
Installing collected packages: onnx, humanfriendly, coloredlogs,
onnxruntime-gpu
Successfully installed coloredlogs-15.0.1 humanfriendly-10.0 onnx-1.18.0
onnxruntime-gpu-1.22.0
```

[62]:
```
import onnx
import onnxruntime as ort
```

[71]:
```
device = torch.device("cpu")
model = AutoModelForSequenceClassification.from_pretrained(model_name,␣
 ↪num_labels=1)
state_dict = torch.load(model_path, map_location=device)
model.load_state_dict(state_dict)
```

```
Some weights of BertForSequenceClassification were not initialized from the
model checkpoint at google/bert_uncased_L-2_H-128_A-2 and are newly initialized:
['classifier.bias', 'classifier.weight']
```

```
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

[71]:
```
<All keys matched successfully>
```

[72]:
```
onnx_model_path = "models/model.onnx"

# dummy input - used to clarify the input shape
batch_size = 1
seq_len    = max_len
dummy_input_ids = torch.randint(
    low=0,
    high=tokenizer.vocab_size,
    size=(batch_size, seq_len),
    dtype=torch.long,
    device=model.device
)
dummy_attention_mask = torch.ones(
    (batch_size, seq_len),
    dtype=torch.long,
    device=model.device
)

# export
torch.onnx.export(
    model,
    (dummy_input_ids, dummy_attention_mask),
    onnx_model_path,
    export_params=True,
    opset_version=14,
    do_constant_folding=True,
    input_names=["input_ids", "attention_mask"],
    output_names=["logits"],
    dynamic_axes={
        "input_ids":       {0: "batch_size", 1: "seq_len"},
        "attention_mask":  {0: "batch_size", 1: "seq_len"},
        "logits":          {0: "batch_size"}
    }
)

# sanity check
onnx_model = onnx.load(onnx_model_path)
onnx.checker.check_model(onnx_model)
```

[73]:
```
model_size = os.path.getsize(onnx_model_path)
print(f"Model Size on Disk: {model_size/ (1e6) :.2f} MB")
```

```
Model Size on Disk: 17.61 MB
```

1.3.3 Apply optimizations to ONNX model

[76]:
```
def benchmark_session(ort_session):
    print(f"Execution provider: {ort_session.get_providers()}")

    correct = 0
    total = 0
    num_samples = len(test_loader.dataset)
    samples_tested = 0

    for batch in tqdm(test_loader, desc="ONNX Inference"):
        if samples_tested >= num_samples:
            break

        input_ids = batch["input_ids"].numpy()
        attention_mask = batch["attention_mask"].numpy()
        labels = batch["labels"].numpy()

        outputs = ort_session.run(None, {
            "input_ids": input_ids,
            "attention_mask": attention_mask
        })[0]

        predicted = np.argmax(outputs, axis=1)
        batch_size = labels.shape[0]
        correct += (predicted == labels).sum()
        total += batch_size
        samples_tested += batch_size

    accuracy = (correct / total) * 100
    print(f"Accuracy: {accuracy:.2f}% ({correct}/{total} correct)")

    ## Benchmark inference latency for single sample
    num_trials = 100
    single_batch = next(iter(test_loader))
    input_ids = single_batch["input_ids"][:1].numpy()
    attention_mask = single_batch["attention_mask"][:1].numpy()

    ort_session.run(None, {
        "input_ids": input_ids,
        "attention_mask": attention_mask
    })

    latencies = []
    for _ in range(num_trials):
        start = time.time()
        ort_session.run(None, {
```

```
            "input_ids": input_ids,
            "attention_mask": attention_mask
        })
        latencies.append(time.time() - start)

    print(f"Inference Latency (single sample, median): {np.
↪percentile(latencies, 50) * 1000:.2f} ms")
    print(f"Inference Latency (single sample, 95th percentile): {np.
↪percentile(latencies, 95) * 1000:.2f} ms")
    print(f"Inference Latency (single sample, 99th percentile): {np.
↪percentile(latencies, 99) * 1000:.2f} ms")
    print(f"Inference Throughput (single sample): {num_trials / np.
↪sum(latencies):.2f} FPS")

    ## Benchmark batch throughput
    num_batches = 50
    input_ids = single_batch["input_ids"].numpy()
    attention_mask = single_batch["attention_mask"].numpy()

    ort_session.run(None, {
        "input_ids": input_ids,
        "attention_mask": attention_mask
    })

    batch_times = []
    for _ in range(num_batches):
        start = time.time()
        ort_session.run(None, {
            "input_ids": input_ids,
            "attention_mask": attention_mask
        })
        batch_times.append(time.time() - start)

    batch_fps = (input_ids.shape[0] * num_batches) / np.sum(batch_times)
    print(f"Batch Throughput: {batch_fps:.2f} FPS")
```

Base ONNX

[78]:
```
onnx_model_path = "models/model.onnx"
ort_session = ort.InferenceSession(onnx_model_path,␣
↪providers=["CPUExecutionProvider"])
benchmark_session(ort_session)
```

```
Execution provider: ['CPUExecutionProvider']

ONNX Inference: 100%|      | 2821/2821 [04:07<00:00, 11.40it/s]

Accuracy: 92.02% (332165/360975 correct)
```

```
Inference Latency (single sample, median): 1.16 ms
Inference Latency (single sample, 95th percentile): 1.59 ms
Inference Latency (single sample, 99th percentile): 1.70 ms
Inference Throughput (single sample): 779.74 FPS
Batch Throughput: 1980.38 FPS
```

Apply basic graph optimizations

[79]:
```
onnx_model_path = "models/model.onnx"
optimized_model_path = "models/model_optimized.onnx"

session_options = ort.SessionOptions()
session_options.graph_optimization_level = ort.GraphOptimizationLevel.
 ↪ORT_ENABLE_EXTENDED
session_options.optimized_model_filepath = optimized_model_path

ort_session = ort.InferenceSession(
    onnx_model_path,
    sess_options=session_options,
    providers=["CPUExecutionProvider"]
)
```

[80]:
```
onnx_model_path = "models/model_optimized.onnx"
ort_session = ort.InferenceSession(onnx_model_path,␣
 ↪providers=["CPUExecutionProvider"])
benchmark_session(ort_session)
```

```
Execution provider: ['CPUExecutionProvider']

ONNX Inference: 100%|      | 2821/2821 [04:07<00:00, 11.40it/s]

Accuracy: 92.02% (332165/360975 correct)


Inference Latency (single sample, median): 1.47 ms
Inference Latency (single sample, 95th percentile): 1.54 ms
Inference Latency (single sample, 99th percentile): 1.69 ms
Inference Throughput (single sample): 680.18 FPS
Batch Throughput: 1466.31 FPS
```

Dynamic quantization

[81]:
```
!pip install neural-compressor
```

```
Collecting neural-compressor
  Downloading neural_compressor-3.3.1-py3-none-any.whl.metadata (15 kB)
Collecting deprecated>=1.2.13 (from neural-compressor)
  Downloading Deprecated-1.2.18-py2.py3-none-any.whl.metadata (5.7 kB)
Collecting numpy<2.0 (from neural-compressor)
  Downloading
numpy-1.26.4-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
```

```
(61 kB)
                                    61.0/61.0 kB
2.4 MB/s eta 0:00:00
Requirement already satisfied: opencv-python-headless in
/usr/local/lib/python3.11/dist-packages (from neural-compressor) (4.11.0.86)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages
(from neural-compressor) (2.2.2)
Requirement already satisfied: Pillow in /usr/local/lib/python3.11/dist-packages
(from neural-compressor) (11.2.1)
Requirement already satisfied: prettytable in /usr/local/lib/python3.11/dist-
packages (from neural-compressor) (3.16.0)
Requirement already satisfied: psutil in /usr/local/lib/python3.11/dist-packages
(from neural-compressor) (5.9.5)
Requirement already satisfied: py-cpuinfo in /usr/local/lib/python3.11/dist-
packages (from neural-compressor) (9.0.0)
Requirement already satisfied: pycocotools in /usr/local/lib/python3.11/dist-
packages (from neural-compressor) (2.0.8)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.11/dist-packages
(from neural-compressor) (6.0.2)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-
packages (from neural-compressor) (2.32.3)
Collecting schema (from neural-compressor)
  Downloading schema-0.7.7-py2.py3-none-any.whl.metadata (34 kB)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-
packages (from neural-compressor) (1.6.1)
Requirement already satisfied: wrapt<2,>=1.10 in /usr/local/lib/python3.11/dist-
packages (from deprecated>=1.2.13->neural-compressor) (1.17.2)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.11/dist-packages (from pandas->neural-compressor)
(2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-
packages (from pandas->neural-compressor) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-
packages (from pandas->neural-compressor) (2025.2)
Requirement already satisfied: wcwidth in /usr/local/lib/python3.11/dist-
packages (from prettytable->neural-compressor) (0.2.13)
Requirement already satisfied: matplotlib>=2.1.0 in
/usr/local/lib/python3.11/dist-packages (from pycocotools->neural-compressor)
(3.10.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.11/dist-packages (from requests->neural-compressor)
(3.4.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-
packages (from requests->neural-compressor) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.11/dist-packages (from requests->neural-compressor)
(2.4.0)
Requirement already satisfied: certifi>=2017.4.17 in
```

```
/usr/local/lib/python3.11/dist-packages (from requests->neural-compressor)
(2025.4.26)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-
packages (from scikit-learn->neural-compressor) (1.15.3)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-
packages (from scikit-learn->neural-compressor) (1.5.0)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.11/dist-packages (from scikit-learn->neural-compressor)
(3.6.0)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.11/dist-packages (from
matplotlib>=2.1.0->pycocotools->neural-compressor) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-
packages (from matplotlib>=2.1.0->pycocotools->neural-compressor) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.11/dist-packages (from
matplotlib>=2.1.0->pycocotools->neural-compressor) (4.58.0)
Requirement already satisfied: kiwisolver>=1.3.1 in
/usr/local/lib/python3.11/dist-packages (from
matplotlib>=2.1.0->pycocotools->neural-compressor) (1.4.8)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.11/dist-packages (from
matplotlib>=2.1.0->pycocotools->neural-compressor) (24.2)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.11/dist-packages (from
matplotlib>=2.1.0->pycocotools->neural-compressor) (3.2.3)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-
packages (from python-dateutil>=2.8.2->pandas->neural-compressor) (1.17.0)
Downloading neural_compressor-3.3.1-py3-none-any.whl (1.8 MB)
                         1.8/1.8 MB
30.5 MB/s eta 0:00:00
Downloading Deprecated-1.2.18-py2.py3-none-any.whl (10.0 kB)
Downloading
numpy-1.26.4-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (18.3
MB)
                         18.3/18.3 MB
112.9 MB/s eta 0:00:00
Downloading schema-0.7.7-py2.py3-none-any.whl (18 kB)
Installing collected packages: schema, numpy, deprecated, neural-compressor
  Attempting uninstall: numpy
    Found existing installation: numpy 2.0.2
    Uninstalling numpy-2.0.2:
      Successfully uninstalled numpy-2.0.2
```

```
ERROR: pip's dependency resolver does not currently take into account all
the packages that are installed. This behaviour is the source of the following
dependency conflicts.
thinc 8.3.6 requires numpy<3.0.0,>=2.0.0, but you have numpy 1.26.4 which is
incompatible.
Successfully installed deprecated-1.2.18 neural-compressor-3.3.1
numpy-1.26.4 schema-0.7.7
```

[82]:
```python
import neural_compressor
from neural_compressor import quantization
```

[87]:
```python
# Load ONNX model
model_path = "models/model.onnx"
fp32_model = neural_compressor.model.onnx_model.ONNXModel(model_path)

# Configure dynamic quantization
config_ptq = neural_compressor.PostTrainingQuantConfig(
    approach="dynamic"
)

# Quantize
q_model = quantization.fit(
    model=fp32_model,
    conf=config_ptq
)

# Save quantized model
quant_model_path = "models/model_quantized_dynamic.onnx"
q_model.save_model_to_file(quant_model_path)
```

```
2025-05-16 22:47:35 [INFO] Start auto tuning.
2025-05-16 22:47:35 [INFO] Quantize model without tuning!
2025-05-16 22:47:35 [INFO] Quantize the model with default configuration without
evaluating the model.                To perform the tuning process, please
either provide an eval_func or provide an                    eval_dataloader an
eval_metric.
2025-05-16 22:47:35 [INFO] Adaptor has 5 recipes.
2025-05-16 22:47:35 [INFO] 0 recipes specified by user.
2025-05-16 22:47:35 [INFO] 3 recipes require future tuning.
2025-05-16 22:47:35 [INFO] *** Initialize auto tuning
2025-05-16 22:47:35 [INFO] {
2025-05-16 22:47:35 [INFO]     'PostTrainingQuantConfig': {
2025-05-16 22:47:35 [INFO]         'AccuracyCriterion': {
2025-05-16 22:47:35 [INFO]             'criterion': 'relative',
2025-05-16 22:47:35 [INFO]             'higher_is_better': True,
2025-05-16 22:47:35 [INFO]             'tolerable_loss': 0.01,
```

```
2025-05-16 22:47:35 [INFO]             'absolute': None,
2025-05-16 22:47:35 [INFO]             'keys': <bound method
AccuracyCriterion.keys of <neural_compressor.config.AccuracyCriterion object at
0x798d5433d450>>,
2025-05-16 22:47:35 [INFO]             'relative': 0.01
2025-05-16 22:47:35 [INFO]         },
2025-05-16 22:47:35 [INFO]         'approach': 'post_training_dynamic_quant',
2025-05-16 22:47:35 [INFO]         'backend': 'default',
2025-05-16 22:47:35 [INFO]         'calibration_sampling_size': [
2025-05-16 22:47:35 [INFO]             100
2025-05-16 22:47:35 [INFO]         ],
2025-05-16 22:47:35 [INFO]         'device': 'cpu',
2025-05-16 22:47:35 [INFO]         'domain': 'auto',
2025-05-16 22:47:35 [INFO]         'example_inputs': 'Not printed here due to
large size tensors…',
2025-05-16 22:47:35 [INFO]         'excluded_precisions': [
2025-05-16 22:47:35 [INFO]         ],
2025-05-16 22:47:35 [INFO]         'framework': 'onnxruntime',
2025-05-16 22:47:35 [INFO]         'inputs': [
2025-05-16 22:47:35 [INFO]         ],
2025-05-16 22:47:35 [INFO]         'model_name': '',
2025-05-16 22:47:35 [INFO]         'op_name_dict': None,
2025-05-16 22:47:35 [INFO]         'op_type_dict': None,
2025-05-16 22:47:35 [INFO]         'outputs': [
2025-05-16 22:47:35 [INFO]         ],
2025-05-16 22:47:35 [INFO]         'quant_format': 'default',
2025-05-16 22:47:35 [INFO]         'quant_level': 'auto',
2025-05-16 22:47:35 [INFO]         'recipes': {
2025-05-16 22:47:35 [INFO]             'smooth_quant': False,
2025-05-16 22:47:35 [INFO]             'smooth_quant_args': {
2025-05-16 22:47:35 [INFO]             },
2025-05-16 22:47:35 [INFO]             'layer_wise_quant': False,
2025-05-16 22:47:35 [INFO]             'layer_wise_quant_args': {
2025-05-16 22:47:35 [INFO]             },
2025-05-16 22:47:35 [INFO]             'fast_bias_correction': False,
2025-05-16 22:47:35 [INFO]             'weight_correction': False,
2025-05-16 22:47:35 [INFO]             'gemm_to_matmul': True,
2025-05-16 22:47:35 [INFO]             'graph_optimization_level': None,
2025-05-16 22:47:35 [INFO]             'first_conv_or_matmul_quantization':
True,
2025-05-16 22:47:35 [INFO]             'last_conv_or_matmul_quantization': True,
2025-05-16 22:47:35 [INFO]             'pre_post_process_quantization': True,
2025-05-16 22:47:35 [INFO]             'add_qdq_pair_to_weight': False,
2025-05-16 22:47:35 [INFO]             'optypes_to_exclude_output_quant': [
2025-05-16 22:47:35 [INFO]             ],
2025-05-16 22:47:35 [INFO]             'dedicated_qdq_pair': False,
2025-05-16 22:47:35 [INFO]             'rtn_args': {
2025-05-16 22:47:35 [INFO]             },
```

```
2025-05-16 22:47:35 [INFO]             'awq_args': {
2025-05-16 22:47:35 [INFO]             },
2025-05-16 22:47:35 [INFO]             'gptq_args': {
2025-05-16 22:47:35 [INFO]             },
2025-05-16 22:47:35 [INFO]             'teq_args': {
2025-05-16 22:47:35 [INFO]             },
2025-05-16 22:47:35 [INFO]             'autoround_args': {
2025-05-16 22:47:35 [INFO]             }
2025-05-16 22:47:35 [INFO]         },
2025-05-16 22:47:35 [INFO]         'reduce_range': None,
2025-05-16 22:47:35 [INFO]         'TuningCriterion': {
2025-05-16 22:47:35 [INFO]             'max_trials': 100,
2025-05-16 22:47:35 [INFO]             'objective': [
2025-05-16 22:47:35 [INFO]                 'performance'
2025-05-16 22:47:35 [INFO]             ],
2025-05-16 22:47:35 [INFO]             'strategy': 'basic',
2025-05-16 22:47:35 [INFO]             'strategy_kwargs': None,
2025-05-16 22:47:35 [INFO]             'timeout': 0
2025-05-16 22:47:35 [INFO]         },
2025-05-16 22:47:35 [INFO]         'use_bf16': True,
2025-05-16 22:47:35 [INFO]         'ni_workload_name': 'quantization'
2025-05-16 22:47:35 [INFO]     }
2025-05-16 22:47:35 [INFO] }
2025-05-16 22:47:35 [WARNING] [Strategy] Please install `mpi4py` correctly if
using distributed tuning; otherwise, ignore this warning.
2025-05-16 22:47:35 [WARNING] The model is automatically detected as an NLP
model. You can use 'domain' argument in 'PostTrainingQuantConfig' to overwrite
it
2025-05-16 22:47:35 [WARNING] Graph optimization level is automatically set to
ENABLE_EXTENDED. You can use 'recipe' argument in 'PostTrainingQuantConfig'to
overwrite it
2025-05-16 22:47:35 [INFO] Do not evaluate the baseline and quantize the model
with default configuration.
2025-05-16 22:47:35 [INFO] Quantize the model with default config.
2025-05-16 22:47:36 [INFO] |**********Mixed Precision Statistics*********|
2025-05-16 22:47:36 [INFO] +----------------------+-------+------+------+
2025-05-16 22:47:36 [INFO] |        Op Type       | Total | INT8 | FP32 |
2025-05-16 22:47:36 [INFO] +----------------------+-------+------+------+
2025-05-16 22:47:36 [INFO] |         MatMul       |   17  |  13  |  4   |
2025-05-16 22:47:36 [INFO] |         Gather       |   14  |  3   |  11  |
2025-05-16 22:47:36 [INFO] |    DequantizeLinear  |   3   |  3   |  0   |
2025-05-16 22:47:36 [INFO] | DynamicQuantizeLinear |   9   |  9   |  0   |
2025-05-16 22:47:36 [INFO] +----------------------+-------+------+------+
2025-05-16 22:47:36 [INFO] Pass quantize model elapsed time: 576.55 ms
2025-05-16 22:47:36 [INFO] Save tuning history to
/content/nc_workspace/2025-05-16_22-45-51/./history.snapshot.
2025-05-16 22:47:36 [INFO] [Strategy] Found the model meets accuracy
requirements, ending the tuning process.
```

```
2025-05-16 22:47:36 [INFO] Specified timeout or max trials is reached! Found a
quantized model which meet accuracy goal. Exit.
2025-05-16 22:47:36 [INFO] Save deploy yaml to
/content/nc_workspace/2025-05-16_22-45-51/deploy.yaml
```

[88]:
```
# Print model size
model_size = os.path.getsize(quant_model_path)
print(f"Model Size on Disk: {model_size / 1e6:.2f} MB")
```

```
Model Size on Disk: 4.51 MB
```

[89]:
```
ort_session = ort.InferenceSession(quant_model_path,␣
 ↪providers=["CPUExecutionProvider"])
benchmark_session(ort_session)
```

```
Execution provider: ['CPUExecutionProvider']

ONNX Inference: 100%|      | 2821/2821 [03:34<00:00, 13.18it/s]

Accuracy: 92.02% (332165/360975 correct)


Inference Latency (single sample, median): 1.52 ms
Inference Latency (single sample, 95th percentile): 1.62 ms
Inference Latency (single sample, 99th percentile): 1.67 ms
Inference Throughput (single sample): 653.39 FPS
Batch Throughput: 1767.16 FPS
```

1.3.4 CUDA execution provider

[90]:
```
onnx_model_path = "models/model.onnx"
ort_session = ort.InferenceSession(onnx_model_path,␣
 ↪providers=["CUDAExecutionProvider"])
benchmark_session(ort_session)
ort.get_device()
```

```
Execution provider: ['CUDAExecutionProvider', 'CPUExecutionProvider']

ONNX Inference: 100%|      | 2821/2821 [00:36<00:00, 78.27it/s]

Accuracy: 92.02% (332165/360975 correct)


Inference Latency (single sample, median): 0.72 ms
Inference Latency (single sample, 95th percentile): 0.79 ms
Inference Latency (single sample, 99th percentile): 0.87 ms
Inference Throughput (single sample): 1361.41 FPS
Batch Throughput: 78767.20 FPS
```

[90]:
```
'GPU'
```