main

May 15, 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
23391        402  1.0
adilshamim8/cost-of-international-education                  Cost of
International Education                                 18950  2025-05-07
15:41:53.213000           4695         79  1.0
adilshamim8/social-media-addiction-vs-relationships          Students' Social
Media Addiction                           7851  2025-05-10 14:38:02.713000
```

```
2176         35  1.0
fatemehmohammadinia/heart-attack-dataset-tarik-a-rashid      Heart Attack
Dataset                                        16250  2025-04-30 21:58:22.740000
4573         80  1.0
ivankmk/thousand-ml-jobs-in-usa                              Machine Learning
Job Postings in the US                 1682058  2025-04-20 16:11:59.347000
4498         99  1.0
michaelmatta0/global-development-indicators-2000-2020        Global Development
Full Analysis (2000-2020)            1311638  2025-05-11 16:57:19.013000
791         26  1.0
mahdimashayekhi/fake-news-detection-dataset                  Fake News Detection
Dataset                              11735585  2025-04-27 14:52:10.607000
1969         26  1.0
aryan208/financial-transactions-dataset-for-fraud-detection  Financial
Transactions Dataset for Fraud Detection    290256858  2025-05-02
09:12:28.203000           1227         29  1.0
umeradnaan/daily-social-media-active-users                   Daily Social Media
Active Users                           126814  2025-05-05 02:11:50.873000
1700         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
2770         43  1.0
dnkumars/cryptocurrency-transaction-analytics-btc-and-eth    Cryptocurrency
Transaction Analytics: BTC & ETH         5167978  2025-05-11 15:16:52.107000
409         30  1.0
madhuraatmarambhagat/crop-recommendation-dataset             Crop Recommendation
Dataset                                65234  2025-05-08 17:02:09.397000
799         27  1.0
zahidmughal2343/global-cancer-patients-2015-2024
global_cancer_patients_2015_2024                          1261049  2025-04-14
00:05:23.367000           5249         65  1.0
razanaqvi14/real-and-fake-news                               Real & Fake News
42975911  2025-04-28 19:46:53.073000           1066         22  1.0
adilshamim8/greenhouse-plant-growth-metrics                  Greenhouse Plant
Growth                                  3041046  2025-04-19 07:33:57.787000
1802         28  1.0
wikimedia-foundation/wikipedia-structured-contents           Wikipedia
Structured Contents                        25121685657  2025-04-11
07:11:03.397000           2246        287  0.8125
glowstudygram/spotify-songs-and-artists-dataset              Spotify Songs and
Artists Dataset | Audio Features        68415  2025-04-27 12:38:36.850000
1873         31  0.8235294
nikolasgegenava/sneakers-classification                      Popular Sneakers
Classification                         17981294  2025-05-01 12:00:45.517000
1612         43  1.0
palvinder2006/ola-bike-ride-request                          Ola Bike Ride
Request                                   174975  2025-04-28 03:55:33.860000
1100         28  1.0
```

```
adilshamim8/predict-students-dropout-and-academic-success    Student Dropout &
Success Prediction Dataset             106181  2025-04-23 06:34:06.433000
2526         39  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

[7]:
```
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 DistilBertTokenizer,␣
 ↪DistilBertForSequenceClassification
```

[8]:
```
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": "distilbert-base-uncased"
}
```

[9]:
```
# ----------------------------
# 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 = outputs.loss
        loss.backward()
        optimizer.step()

        total_loss += loss.item()
        preds = outputs.logits.argmax(dim=1)
        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 = outputs.loss

            total_loss += loss.item()
            preds = outputs.logits.argmax(dim=1)
            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 = DistilBertTokenizer.from_pretrained(config["model_name"])
    train_df = pd.read_csv(os.path.join(args.data_dir, "train.csv"))
    if args.dry_run:
        train_df = train_df.sample(n=32, random_state=42)
    val_df = pd.read_csv(os.path.join(args.data_dir, "val.csv"))
    if args.dry_run:
        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)
    val_loader = DataLoader(JigsawDataset(val_df, tokenizer, config["max_len"]),
                            batch_size=config["batch_size"])

    model = DistilBertForSequenceClassification.
 ↪from_pretrained(config["model_name"])
    model.to(device)

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

    best_val_loss = float("inf")
    patience_counter = 0

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

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

        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(

tokenizer_config.json:   0%|          | 0.00/48.0 [00:00<?, ?B/s]

vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]

tokenizer.json:   0%|          | 0.00/466k [00:00<?, ?B/s]

config.json:   0%|          | 0.00/483 [00:00<?, ?B/s]

Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed.
Falling back to regular HTTP download. For better performance, install the
```

```
package with: `pip install huggingface_hub[hf_xet]` or `pip install hf_xet`
WARNING:huggingface_hub.file_download:Xet Storage is enabled for this repo, but
the 'hf_xet' package is not installed. Falling back to regular HTTP download.
For better performance, install the package with: `pip install
huggingface_hub[hf_xet]` or `pip install hf_xet`
```

```
model.safetensors:   0%|          | 0.00/268M [00:00<?, ?B/s]
```

```
Some weights of DistilBertForSequenceClassification were not initialized from
the model checkpoint at distilbert-base-uncased and are newly initialized:
['classifier.bias', 'classifier.weight', 'pre_classifier.bias',
'pre_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 112 batches


Partial Epoch Summary - Avg Loss: 0.2503, Avg Accuracy: 0.9166

Evaluating for 28 batches


Eval Summary - Avg Loss: 0.1531, Accuracy: 0.9481

Epoch 1: Train Loss=0.2503 Acc=0.9166 | Val Loss=0.1531 Acc=0.9481 | Time=53.61s
  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
```

[10]:
```
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
```

[11]:
```python
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)
        }
```

[12]:
```python
batch_size = 128
max_len = 128
model_name = "distilbert-base-uncased"
dataset_dir = os.getenv("DATA_DIR", "data/jigsaw/processed")
model_path = "models/model.pth"
```

[13]:
```python
val_df = pd.read_csv(os.path.join(dataset_dir, "val.csv"))

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

```
/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(
```

1.3.1 Measure inference performance of PyTorch model on CPU

[33]:
```
device = torch.device("cpu")
model = DistilBertForSequenceClassification.from_pretrained(model_name)
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 DistilBertForSequenceClassification were not initialized from
the model checkpoint at distilbert-base-uncased and are newly initialized:
['classifier.bias', 'classifier.weight', 'pre_classifier.bias',
'pre_classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

[33]:
```
==========================================================================
Layer (type:depth-idx)                                   Param #
==========================================================================
DistilBertForSequenceClassification                       --
 DistilBertModel: 1-1                                    --
      Embeddings: 2-1                                    --
           Embedding: 3-1                                23,440,896
           Embedding: 3-2                                393,216
           LayerNorm: 3-3                                1,536
           Dropout: 3-4                                  --
      Transformer: 2-2                                   --
           ModuleList: 3-5                               42,527,232
 Linear: 1-2                                              590,592
 Linear: 1-3                                              1,538
 Dropout: 1-4                                             --
==========================================================================
Total params: 66,955,010
Trainable params: 66,955,010
Non-trainable params: 0
==========================================================================
```

[34]:
```
model_size = os.path.getsize(model_path)
print(f"Model Size on Disk: {model_size/ (1e6) :.2f} MB")
```

```
Model Size on Disk: 267.85 MB
```

[35]:
```
def evaluate_test(model, loader, device, portion=0.01):
    model.eval()
    correct, total = 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 = outputs.loss

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

    return correct, total
```

[36]:
```
correct, total = evaluate_test(model, test_loader, device, portion=0.01)
accuracy = (correct / total) * 100
print(f"Accuracy: {accuracy:.2f}% ({correct}/{total} correct)")
```

```
Evaluating for 28 batches


Accuracy: 94.81% (3398/3584 correct)
```

Inference Latency

[38]:
```
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.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)
```

[39]:
```
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): 31.67 ms
Inference Latency (single sample, 95th percentile): 44.55 ms
Inference Latency (single sample, 99th percentile): 45.59 ms
Inference Throughput (single sample): 29.51 FPS
```

Batch throughput

[40]:
```
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.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)
```

[41]:
```
# 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: 52.82 FPS
```

Summary

[42]:
```
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: 267.85 MB
Accuracy: 94.81% (3398/3584 correct)
Inference Latency (single sample, median): 31.67 ms
Inference Latency (single sample, 95th percentile): 44.55 ms
Inference Latency (single sample, 99th percentile): 45.59 ms
Inference Throughput (single sample): 29.51 FPS
Batch Throughput: 52.82 FPS
```

Eager mode Summary Model Size on Disk: 267.85 MB
Accuracy: 94.81% (3398/3584 correct)
Inference Latency (single sample, median): 35.32 ms
Inference Latency (single sample, 95th percentile): 55.24 ms
Inference Latency (single sample, 99th percentile): 56.10 ms
Inference Throughput (single sample): 26.91 FPS
Batch Throughput: 40.85 FPS

Compiled Summary Model Size on Disk: 267.85 MB
Accuracy: 94.81% (3398/3584 correct)
Inference Latency (single sample, median): 31.67 ms
Inference Latency (single sample, 95th percentile): 44.55 ms
Inference Latency (single sample, 99th percentile): 45.59 ms
Inference Throughput (single sample): 29.51 FPS
Batch Throughput: 52.82 FPS

1.3.2 Measure inference performance of ONNX model on CPU¶

[43]:
```
!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
  Downloading onnxruntime-1.22.0-cp311-cp311-
```

```
manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl.metadata (4.5 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)
  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) (25.2.10)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-
packages (from onnxruntime) (24.2)
Requirement already satisfied: sympy in /usr/local/lib/python3.11/dist-packages
(from onnxruntime) (1.13.1)
Collecting humanfriendly>=9.1 (from coloredlogs->onnxruntime)
  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) (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
92.1 MB/s eta 0:00:00
Downloading
onnxruntime-1.22.0-cp311-cp311-manylinux_2_27_x86_64.manylinux_2_28_x86_64.whl
(16.4 MB)
                         16.4/16.4 MB
88.2 MB/s eta 0:00:00
Downloading coloredlogs-15.0.1-py2.py3-none-any.whl (46 kB)
                         46.0/46.0 kB
4.2 MB/s eta 0:00:00
Downloading humanfriendly-10.0-py2.py3-none-any.whl (86 kB)
                         86.8/86.8 kB
8.5 MB/s eta 0:00:00
Installing collected packages: onnx, humanfriendly, coloredlogs,
onnxruntime
Successfully installed coloredlogs-15.0.1 humanfriendly-10.0 onnx-1.18.0
onnxruntime-1.22.0
```

[1]:
```
import onnx
import onnxruntime as ort
```

[46]:
```
device = torch.device("cpu")
model = DistilBertForSequenceClassification.from_pretrained(model_name)
state_dict = torch.load(model_path, map_location=device)
model.load_state_dict(state_dict)
```

```
Some weights of DistilBertForSequenceClassification were not initialized from
```

```
the model checkpoint at distilbert-base-uncased and are newly initialized:
['classifier.bias', 'classifier.weight', 'pre_classifier.bias',
'pre_classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

[46]:
```
<All keys matched successfully>
```

[47]:
```
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)
```

[49]:
```
model_size = os.path.getsize(onnx_model_path)
print(f"Model Size on Disk: {model_size/ (1e6) :.2f} MB")
```

```
Model Size on Disk: 267.96 MB
```

Create inference session

[57]:
```
ort_session = ort.InferenceSession(onnx_model_path,␣
 ↪providers=['CPUExecutionProvider'])
ort_session.get_providers()
```

[57]:
```
['CPUExecutionProvider']
```

[58]:
```
correct = 0
total = 0

num_samples = int(0.0001 * len(test_loader.dataset))
samples_tested = 0

for batch in test_loader:
    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
```

[59]:
```
print(f"Accuracy: {accuracy:.2f}% ({correct}/{total} correct)")
```

```
Accuracy: 96.09% (123/128 correct)
```

Inference Latency

[60]:
```
# Prepare a single tokenized sample
model_input = tokenizer("This is a sample.", return_tensors="np",␣
 ↪max_length=max_len, padding="max_length", truncation=True)
single_input_ids = model_input["input_ids"]
```

```
single_attention_mask = model_input["attention_mask"]

# Setup ONNX Runtime session
ort_session = ort.InferenceSession("models/model.onnx")

# Warm-up
ort_session.run(None, {
    "input_ids": single_input_ids,
    "attention_mask": single_attention_mask
})

# Timing
latencies = []
for _ in range(100):
    start = time.time()
    ort_session.run(None, {
        "input_ids": single_input_ids,
        "attention_mask": single_attention_mask
    })
    latencies.append(time.time() - start)
```

[61]:
```
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): 40.11 ms
Inference Latency (single sample, 95th percentile): 40.53 ms
Inference Latency (single sample, 99th percentile): 41.30 ms
Inference Throughput (single sample): 24.92 FPS
```

Batch Throughput

[62]:
```
num_batches = 50

# Get a batch from the test data
batch = next(iter(test_loader))
input_ids = batch["input_ids"].numpy()
attention_mask = batch["attention_mask"].numpy()

# Warm-up
ort_session.run(None, {
    "input_ids": input_ids,
    "attention_mask": attention_mask
```

```
})

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

[63]:
```
batch_fps = (input_ids.shape[0] * num_batches) / np.sum(batch_times)
print(f"Batch Throughput: {batch_fps:.2f} FPS")
```

```
Batch Throughput: 41.60 FPS
```

Summary

[64]:
```
print(f"Accuracy: {accuracy:.2f}% ({correct}/{total} correct)")
print(f"Model Size on Disk: {model_size/ (1e6) :.2f} MB")
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")
```

```
Accuracy: 96.09% (123/128 correct)
Model Size on Disk: 267.96 MB
Inference Latency (single sample, median): 40.11 ms
Inference Latency (single sample, 95th percentile): 40.53 ms
Inference Latency (single sample, 99th percentile): 41.30 ms
Inference Throughput (single sample): 24.92 FPS
Batch Throughput: 41.60 FPS
```

1.3.3 Apply optimizations to ONNX model

[4]:
```
def benchmark_session(ort_session):
    print(f"Execution provider: {ort_session.get_providers()}")

    ## Benchmark accuracy (0.01% of test set)
    correct = 0
    total = 0
    num_samples = int(0.0001 * len(test_loader.dataset))
    samples_tested = 0
```

```
    for batch in test_loader:
        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 (0.01% sampled): {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")
```

Apply basic graph optimizations

[67]:
```
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"]
)
```

[68]:
```
onnx_model_path = "models/model_optimized.onnx"
```

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

```
Execution provider: ['CPUExecutionProvider']
Accuracy (0.01% sampled): 96.09% (123/128 correct)
Inference Latency (single sample, median): 24.31 ms
Inference Latency (single sample, 95th percentile): 25.66 ms
Inference Latency (single sample, 99th percentile): 32.61 ms
Inference Throughput (single sample): 40.47 FPS
Batch Throughput: 35.77 FPS
```

Dynamic quantization

[70]:
```
!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
2.9 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
31.8 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
108.7 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
```

[71]:
```python
import neural_compressor
from neural_compressor import quantization
```

[72]:
```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
)
```

```
2025-05-16 02:09:09 [INFO] Start auto tuning.
2025-05-16 02:09:09 [INFO] Quantize model without tuning!
2025-05-16 02:09:09 [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 02:09:09 [INFO] Adaptor has 5 recipes.
2025-05-16 02:09:09 [INFO] 0 recipes specified by user.
2025-05-16 02:09:09 [INFO] 3 recipes require future tuning.
2025-05-16 02:09:10 [INFO] *** Initialize auto tuning
2025-05-16 02:09:10 [INFO] {
2025-05-16 02:09:10 [INFO]     'PostTrainingQuantConfig': {
2025-05-16 02:09:10 [INFO]         'AccuracyCriterion': {
2025-05-16 02:09:10 [INFO]             'criterion': 'relative',
2025-05-16 02:09:10 [INFO]             'higher_is_better': True,
2025-05-16 02:09:10 [INFO]             'tolerable_loss': 0.01,
2025-05-16 02:09:10 [INFO]             'absolute': None,
2025-05-16 02:09:10 [INFO]             'keys': <bound method
AccuracyCriterion.keys of <neural_compressor.config.AccuracyCriterion object at
0x7b5d521c2290>>,
2025-05-16 02:09:10 [INFO]             'relative': 0.01
2025-05-16 02:09:10 [INFO]         },
2025-05-16 02:09:10 [INFO]         'approach': 'post_training_dynamic_quant',
2025-05-16 02:09:10 [INFO]         'backend': 'default',
2025-05-16 02:09:10 [INFO]         'calibration_sampling_size': [
2025-05-16 02:09:10 [INFO]             100
2025-05-16 02:09:10 [INFO]         ],
2025-05-16 02:09:10 [INFO]         'device': 'cpu',
2025-05-16 02:09:10 [INFO]         'domain': 'auto',
2025-05-16 02:09:10 [INFO]         'example_inputs': 'Not printed here due to
large size tensors…',
2025-05-16 02:09:10 [INFO]         'excluded_precisions': [
2025-05-16 02:09:10 [INFO]         ],
2025-05-16 02:09:10 [INFO]         'framework': 'onnxruntime',
2025-05-16 02:09:10 [INFO]         'inputs': [
2025-05-16 02:09:10 [INFO]         ],
2025-05-16 02:09:10 [INFO]         'model_name': '',
2025-05-16 02:09:10 [INFO]         'op_name_dict': None,
2025-05-16 02:09:10 [INFO]         'op_type_dict': None,
2025-05-16 02:09:10 [INFO]         'outputs': [
2025-05-16 02:09:10 [INFO]         ],
2025-05-16 02:09:10 [INFO]         'quant_format': 'default',
2025-05-16 02:09:10 [INFO]         'quant_level': 'auto',
2025-05-16 02:09:10 [INFO]         'recipes': {
2025-05-16 02:09:10 [INFO]             'smooth_quant': False,
2025-05-16 02:09:10 [INFO]             'smooth_quant_args': {
2025-05-16 02:09:10 [INFO]             },
2025-05-16 02:09:10 [INFO]             'layer_wise_quant': False,
2025-05-16 02:09:10 [INFO]             'layer_wise_quant_args': {
2025-05-16 02:09:10 [INFO]             },
2025-05-16 02:09:10 [INFO]             'fast_bias_correction': False,
```

```
2025-05-16 02:09:10 [INFO]             'weight_correction': False,
2025-05-16 02:09:10 [INFO]             'gemm_to_matmul': True,
2025-05-16 02:09:10 [INFO]             'graph_optimization_level': None,
2025-05-16 02:09:10 [INFO]             'first_conv_or_matmul_quantization':
True,
2025-05-16 02:09:10 [INFO]             'last_conv_or_matmul_quantization': True,
2025-05-16 02:09:10 [INFO]             'pre_post_process_quantization': True,
2025-05-16 02:09:10 [INFO]             'add_qdq_pair_to_weight': False,
2025-05-16 02:09:10 [INFO]             'optypes_to_exclude_output_quant': [
2025-05-16 02:09:10 [INFO]             ],
2025-05-16 02:09:10 [INFO]             'dedicated_qdq_pair': False,
2025-05-16 02:09:10 [INFO]             'rtn_args': {
2025-05-16 02:09:10 [INFO]             },
2025-05-16 02:09:10 [INFO]             'awq_args': {
2025-05-16 02:09:10 [INFO]             },
2025-05-16 02:09:10 [INFO]             'gptq_args': {
2025-05-16 02:09:10 [INFO]             },
2025-05-16 02:09:10 [INFO]             'teq_args': {
2025-05-16 02:09:10 [INFO]             },
2025-05-16 02:09:10 [INFO]             'autoround_args': {
2025-05-16 02:09:10 [INFO]             }
2025-05-16 02:09:10 [INFO]         },
2025-05-16 02:09:10 [INFO]         'reduce_range': None,
2025-05-16 02:09:10 [INFO]         'TuningCriterion': {
2025-05-16 02:09:10 [INFO]             'max_trials': 100,
2025-05-16 02:09:10 [INFO]             'objective': [
2025-05-16 02:09:10 [INFO]                 'performance'
2025-05-16 02:09:10 [INFO]             ],
2025-05-16 02:09:10 [INFO]             'strategy': 'basic',
2025-05-16 02:09:10 [INFO]             'strategy_kwargs': None,
2025-05-16 02:09:10 [INFO]             'timeout': 0
2025-05-16 02:09:10 [INFO]         },
2025-05-16 02:09:10 [INFO]         'use_bf16': True,
2025-05-16 02:09:10 [INFO]         'ni_workload_name': 'quantization'
2025-05-16 02:09:10 [INFO]     }
2025-05-16 02:09:10 [INFO] }
2025-05-16 02:09:10 [WARNING] [Strategy] Please install `mpi4py` correctly if
using distributed tuning; otherwise, ignore this warning.
2025-05-16 02:09:10 [WARNING] The model is automatically detected as an NLP
model. You can use 'domain' argument in 'PostTrainingQuantConfig' to overwrite
it
2025-05-16 02:09:10 [WARNING] Graph optimization level is automatically set to
ENABLE_EXTENDED. You can use 'recipe' argument in 'PostTrainingQuantConfig'to
overwrite it
2025-05-16 02:09:12 [INFO] Do not evaluate the baseline and quantize the model
with default configuration.
2025-05-16 02:09:12 [INFO] Quantize the model with default config.
2025-05-16 02:09:19 [INFO] |**********Mixed Precision Statistics*********|
```

```
2025-05-16 02:09:19 [INFO] +----------------------+-------+------+------+
2025-05-16 02:09:19 [INFO] |        Op Type       | Total | INT8 | FP32 |
2025-05-16 02:09:19 [INFO] +----------------------+-------+------+------+
2025-05-16 02:09:19 [INFO] |         MatMul       |   49  |  37  |  12  |
2025-05-16 02:09:19 [INFO] |         Gather       |   19  |  2   |  17  |
2025-05-16 02:09:19 [INFO] |    DequantizeLinear  |   2   |  2   |  0   |
2025-05-16 02:09:19 [INFO] | DynamicQuantizeLinear |   25  |  25  |  0   |
2025-05-16 02:09:19 [INFO] +----------------------+-------+------+------+
2025-05-16 02:09:19 [INFO] Pass quantize model elapsed time: 7455.17 ms
2025-05-16 02:09:19 [INFO] Save tuning history to
/content/nc_workspace/2025-05-16_02-09-04/./history.snapshot.
2025-05-16 02:09:20 [INFO] [Strategy] Found the model meets accuracy
requirements, ending the tuning process.
2025-05-16 02:09:20 [INFO] Specified timeout or max trials is reached! Found a
quantized model which meet accuracy goal. Exit.
2025-05-16 02:09:20 [INFO] Save deploy yaml to
/content/nc_workspace/2025-05-16_02-09-04/deploy.yaml
```

[73]:
```
# Save quantized model
quant_model_path = "models/model_quantized_dynamic.onnx"
q_model.save_model_to_file(quant_model_path)
```

[74]:
```
# 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: 69.24 MB
```

[75]:
```
ort_session = ort.InferenceSession(quant_model_path,␣
 ↪providers=["CPUExecutionProvider"])
benchmark_session(ort_session)
```

```
Execution provider: ['CPUExecutionProvider']
Accuracy (0.01% sampled): 93.75% (120/128 correct)
Inference Latency (single sample, median): 19.46 ms
Inference Latency (single sample, 95th percentile): 19.91 ms
Inference Latency (single sample, 99th percentile): 20.16 ms
Inference Throughput (single sample): 51.27 FPS
Batch Throughput: 61.72 FPS
```

1.3.4 Try a different execution providers

CPU

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

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

```
Accuracy (0.01% sampled): 96.09% (123/128 correct)
Inference Latency (single sample, median): 40.51 ms
Inference Latency (single sample, 95th percentile): 41.10 ms
Inference Latency (single sample, 99th percentile): 41.29 ms
Inference Throughput (single sample): 24.64 FPS
Batch Throughput: 25.55 FPS
```

CUDA

[14]:
```
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']
Accuracy (0.01% sampled): 96.09% (123/128 correct)
Inference Latency (single sample, median): 1.69 ms
Inference Latency (single sample, 95th percentile): 1.87 ms
Inference Latency (single sample, 99th percentile): 1.93 ms
Inference Throughput (single sample): 581.60 FPS
Batch Throughput: 5005.71 FPS
```

[14]:
```
'GPU'
```