import os, time, argparse, statistics, psutil, pandas as pd, numpy as np

import onnxruntime as ort

from transformers import AutoTokenizer

# ----------------- Helpers -----------------

def sizeof\_fmt(num, suffix="B"):

    for unit in ["", "K", "M", "G", "T"]:

        if abs(num) < 1024.0:

            return f"{num:3.1f}{unit}{suffix}"

        num /= 1024.0

    return f"{num:.1f}P{suffix}"

def estimate\_power(cpu\_util, max\_power=7.5):

    """

    Approximate power usage of Raspberry Pi (Watts).

    - cpu\_util: % CPU utilization (0–100)

    - max\_power: estimated peak power draw of Pi5 in Watts

    """

    return (cpu\_util / 100.0) \* max\_power

# ----------------- Main -----------------

if \_\_name\_\_ == "\_\_main\_\_":

    parser = argparse.ArgumentParser()

    parser.add\_argument("--file", type=str, required=True, help="Input text file (one sentence per line)")

    parser.add\_argument("--runs", type=int, default=100, help="Number of inference runs per sentence")

    parser.add\_argument("--model", type=str, default="models/distilbert-sst2-onnx/model.onnx", help="ONNX model path")

    parser.add\_argument("--tokenizer", type=str, default="distilbert-base-uncased-finetuned-sst-2-english", help="Tokenizer name")

    parser.add\_argument("--max-power", type=float, default=7.5, help="Peak system power in Watts (adjustable)")

    args = parser.parse\_args()

    # Load sentences

    with open(args.file, "r", encoding="utf-8") as f:

        sentences = [line.strip() for line in f if line.strip()]

    # Load tokenizer

    t0 = time.time()

    tokenizer = AutoTokenizer.from\_pretrained(args.tokenizer)

    t1 = time.time()

    tok\_load\_time = (t1 - t0) \* 1000

    # Load ONNX model

    t0 = time.time()

    session = ort.InferenceSession(args.model)

    t1 = time.time()

    model\_load\_time = (t1 - t0) \* 1000

    # Run benchmark

    latencies, power\_readings, energy\_per\_inference = [], [], []

    labels = ["negative", "positive"]

    rss\_before = psutil.Process(os.getpid()).memory\_info().rss

    start\_time = time.time()

    for \_ in range(args.runs):

        for text in sentences:

            inputs = tokenizer(text, return\_tensors="np")

            # Measure start

            t0 = time.time()

            outputs = session.run(None, dict(inputs))

            t1 = time.time()

            # Latency (ms)

            latency = (t1 - t0) \* 1000

            latencies.append(latency)

            # Power estimation (W)

            cpu\_util = psutil.cpu\_percent(interval=None)

            est\_power = estimate\_power(cpu\_util, args.max\_power)

            power\_readings.append(est\_power)

            # Energy estimation (J) = Power × Time

            energy = est\_power \* (latency / 1000.0)

            energy\_per\_inference.append(energy)

    total\_time = time.time() - start\_time

    rss\_after = psutil.Process(os.getpid()).memory\_info().rss

    # Stats

    mean\_lat = statistics.mean(latencies)

    med\_lat = statistics.median(latencies)

    p90 = np.percentile(latencies, 90)

    p95 = np.percentile(latencies, 95)

    p99 = np.percentile(latencies, 99)

    throughput = len(latencies) / total\_time

    avg\_cpu = psutil.cpu\_percent(interval=None)

    avg\_power = statistics.mean(power\_readings)

    total\_energy = sum(energy\_per\_inference)

    avg\_energy = statistics.mean(energy\_per\_inference)

    # Save CSV

    df = pd.DataFrame({"latency\_ms": latencies, "power\_W": power\_readings, "energy\_J": energy\_per\_inference})

    csv\_path = os.path.join(os.getcwd(), "benchmark\_results.csv")

    df.to\_csv(csv\_path, index=False)

    # Report

    print("\n===== SYSTEM BENCHMARK =====")

    print(f"Model:            {args.model}  ({sizeof\_fmt(os.path.getsize(args.model))})")

    print(f"Tokenizer:        {args.tokenizer}  (load {tok\_load\_time:.1f} ms)")

    print(f"ONNX cold-start:  {model\_load\_time:.1f} ms")

    print(f"Runs:             {len(latencies)} inferences over {total\_time:.2f}s")

    print(f"Throughput:       {throughput:.2f} sentences/sec")

    print(f"Latency (ms):     mean={mean\_lat:.2f}  median={med\_lat:.2f}  "

          f"p90={p90:.2f}  p95={p95:.2f}  p99={p99:.2f}")

    print(f"Memory RSS:       before={sizeof\_fmt(rss\_before)}  after={sizeof\_fmt(rss\_after)}")

    print(f"CPU util (avg):   {avg\_cpu:.1f}%")

    print(f"Power (avg est.): {avg\_power:.2f} W")

    print(f"Total Energy:     {total\_energy:.4f} J")

    print(f"Energy/inference: {avg\_energy:.4f} J")

    print(f"CSV log:          {csv\_path}")

    print("================================\n")