测试了 Resnet18，Efficient-B0，ViT-B16, Transformer 四个模型中各个算子在 CIFAR-10 数据集上的时延和功耗情况。

一、使用的软件和工具

1. Python 脚本驱动的深度学习性能评估工具链

整个实验流程基于 Python 脚本构建，使用了以下模块/组件：

```
DNNProfile2/
├── main.py                          # 主程序入口
├── gpu_info.py                      # GPU 信息收集模块
├── gpu_utils.py                     # GPU 控制工具
├── power_monitor.py                 # 功耗监测模块
├── experiment_config.py             # 实验配置管理
├── run_experiment.py                # 实验执行
├── visualize_results.py             # 结果可视化
├── operator_visualization.py        # 算子可视化
└── models/                          # 模型定义
    ├── custom_models.py
    ├── efficientnet.py
    ├── vit.py
    └── transformer.py
```

二、实验流程说明

1. GPU 信息收集（gpu_info.py）

调用函数：

- get_gpu_info()：获取 GPU 硬件配置信息
- save_gpu_info(...)：保存 GPU 配置到 YAML 文件

功能：获取当前系统的 GPU 配置信息并保存为 `gpu_config.yml`。

工作原理：调用 nvidia-smi

```
# 通过nvidia-smi命令获取GPU基本信息
gpu_info_str = run_command("nvidia-smi --query-
gpu=gpu_name,driver_version,memory.total,power.limit --format=csv,noheader")

# 获取支持的核心频率范围
core_clocks_str = run_command("nvidia-smi --query-supported-clocks=graphics --
format=csv,noheader")

# 智能采样频率点（默认10个采样点）
sampled_core_clocks = [core_clocks[i] for i in indices]
```

2. 实验执行（run_experiment.py）

调用函数：run_experiment(...)

输入配置：

gpu_config.yml：保存 GPU 相关信息。

experiment_config.yml：保存实验模型结构、运行参数等。

输出文件：

results_时间戳.json：记录整体模型推理性能。

results_时间戳_operators.json：记录每个算子的性能数据。

2. 1 功耗监测机制

在 gpu_info.py 中，通过调用 nvidia-smi 工具来获取 GPU 的实时功耗数据，功耗信息会与其他 GPU 配置信息一起保存为 gpu_config.yml，便于后续实验使用。

使用 power_monitor.py 对实验时的功耗监测，实验前调用 start_monitoring，会启动一个进程，循环采样 GPU 功耗（通过 gpu_info.py），每次采样完毕 sleep 0.01s。实验结束调用 stop_monitoring，并且调用 get_stats 获得这段时间的平均、最小、最大能耗。

工作流程：

（1）. 启动监测: start_monitoring() 创建后台线程

（2）. 实时采样: 每 10ms 调用 nvidia-smi 获取功耗数据

（3）. 数据存储: 线程安全的功耗读数存储

（4）. 统计计算: 提供平均、最小、最大功耗统计

2.2 算子级性能分析

基于 PyTorch Profiler 捕获算子运行信息，然后打印里面的 Key，对应有记录时长，并通过 `平均功率 × 执行时间` 得到算子的执行能耗.

功耗如何参与算子分析:

1) PowerMonitor 对象通过后台线程实时记录 GPU 功耗。

2) 在最后一次迭代中，通过 power_monitor.get_stats()获取平均功耗。

3) 对于每个算子，PyTorch Profiler 可以得到执行时间，能耗用执行时间×平均功率得到:

energy_joules = (cuda_time_ms / 1000) * avg_power

2.3 GPU 频率控制（gpu_utils.py）

核心类: GPUController

利用 nvidia-smi -lgc 命令

```
def set_core_clock(self, frequency):
    # 设置GPU核心频率
    cmd = f"nvidia-smi -lgc {frequency} --id={self.gpu_id}"
    output, error, return_code = self.run_command(cmd)

    # 验证设置是否生效
    time.sleep(1)
    current = self.get_current_settings()
    return current["core_clock"]
```