ZXHPC2025 Writeup-Caffeine

1. a-plus-b

1.1 题目解法描述

复制粘贴示例程序

1.2 优化过程记录

复制粘贴示例程序

1.3 证明材料

复制粘贴示例程序

1.4 源代码与相关文件

复制粘贴示例程序

---

2. cos-sim

2.1 题目解法描述

这道题的核心在于计算 N 个高维向量之间的余弦相似度，并快速找出每个向量除自身外最相似的四个向量。直接两两计算复杂度为 O(N^2D)，效率低下，因此关键在于使用 **FAISS** 库的 `IndexFlatIP` 索引。首先对所有向量进行 **L2 归一化**，使得余弦相似度可以直接用内积计算，然后将所有向量加入 `IndexFlatIP` 索引中，利用其高效的向量内积搜索快速获取每个向量的 top-5（包含自身），最后去掉自身得到 top-4 的相似度。FAISS 的向量索引加速和批量搜索能力，大幅减少了计算时间，使得在 N=20000、D=4096 的高维数据下仍能高效完成计算，同时保证了结果的精度（误差小于 1e-6）。

2.2 优化过程记录

环境配置：

代码块

```
# 激活 Anaconda 的基础环境（初始化 conda 命令环境变量）
source /opt/app/anaconda3/bin/activate

```

```
# 创建一个名为 zhangqihan 的新虚拟环境，并指定 Python 版本为 3.9
conda create -n zhangqihan python=3.9

# 安装 FAISS C++ 库
conda install -c conda-forge faiss-cpu
```

程序构建：

cos-opt.cpp

```cpp
#include <bits/stdc++.h>        // GNU 扩展头文件，包含了大部分常见的 C++ 标准库
#include <faiss/IndexFlat.h>    // FAISS 库中基于平面向量的索引类（IndexFlat）
#include <faiss/utils/Heap.h>   // FAISS 工具函数（堆相关），虽然这里没直接用到

using namespace std;

int main() {
    ios::sync_with_stdio(false);  // 关闭 iostream 与 C 标准 IO 的同步，提高输入输出效率
    cin.tie(nullptr);             // 解除 cin 与 cout 的绑定，加速输入输出

    // === 读取输入数据维度 ===
    uint32_t N, D;
    fread(&N, sizeof(uint32_t), 1, stdin);  // 从标准输入读取向量个数 N
    fread(&D, sizeof(uint32_t), 1, stdin);  // 从标准输入读取向量维度 D

    // === 读取向量数据 ===
    vector<float> data(N * D);              // 存储 N 个 D 维向量的连续内存
    fread(data.data(), sizeof(float), N * D, stdin);  // 从标准输入读取所有向量

    // === 归一化向量（L2 范数归一化） ===
    for (size_t i = 0; i < N; i++) {
        float norm = 0;
        float* v = data.data() + i * D;     // 指向第 i 个向量的起始位置
        for (size_t j = 0; j < D; j++)
            norm += v[j] * v[j];            // 计算平方和
        norm = 1.0f / (sqrtf(norm) + 1e-12f); // 取倒数并避免除零
        for (size_t j = 0; j < D; j++)
            v[j] *= norm;                   // 向量各分量归一化
    }

    // === 构建 FAISS 内积索引 ===
    faiss::IndexFlatIP index(D);    // 建立基于内积的平面索引，维度为 D
    index.add(N, data.data());      // 将所有向量加入索引

    // === 搜索每个向量的 top-5 相似向量（包含自己） ===
```

```
    vector<float> sims(N * 5);      // 存储相似度（内积值）
    vector<faiss::idx_t> idxs(N * 5); // 存储对应的索引 ID
    index.search(N, data.data(), 5, sims.data(), idxs.data());
    // 对 N 个查询向量（即自身），检索最相似的 5 个向量

    // === 去掉“自己”，只保留 top-4 结果 ===
    vector<float> result(N * 4);
    for (size_t i = 0; i < N; i++) {
        // sims[i*5 + 0] 是自己与自己的相似度（通常为 1），需要跳过
        memcpy(result.data() + i * 4, sims.data() + i * 5 + 1, 4 * sizeof(float));
        // 拷贝除自身外的前 4 个相似度值
    }

    // === 输出结果 ===
    // 将最终的相似度矩阵（N x 4）写到标准输出，格式为 float32 小端序
    fwrite(result.data(), sizeof(float), result.size(), stdout);

    return 0;
}
```

任务提交（可直接复现）：

Bash

```
# === 申请并登录计算节点 ===
salloc -N 1 -n 1 -p c003t -c 40 --mem=128G   # 向作业调度系统申请：
                                             # -N 1    ：1 个节点
                                             # -n 1    ：1 个任务
                                             # -p c003t：使用分区 c003t
                                             # -c 40   ：申请 40 个 CPU 核
                                             # --mem=128G ：申请 128GB 内存
ssh cpuXX                                    # 登录分配到的计算节点，XX 是节点编号

# === 激活 conda 环境（包含 faiss 库） ===
source /opt/app/anaconda3/bin/activate zhangqihan   # 激活名为 zhangqihan 的环境
                                                    # 其中 Python 版本为 3.9.23

# === 切换到源码目录 ===
cd /home/u2023211111/share/zqh/cos-sim/    # 进入项目源码路径

# === 使用自带 gcc 10.1.0 进行编译优化 ===
export CC=/opt/app/gcc/10.1.0/bin/gcc      # 指定 C 编译器
export CXX=/opt/app/gcc/10.1.0/bin/g++     # 指定 C++ 编译器
export PATH=/opt/app/gcc/10.1.0/bin:$PATH # 确保调用的是这个版本的 gcc/g++

```

```bash
# === 编译程序 ===
g++ -O3 -march=native -funroll-loops cos-opt.cpp -o cos-opt-final \
-I$CONDA_PREFIX/include \    # 包含路径：使用 conda 环境里的头文件（如 faiss）
-L$CONDA_PREFIX/lib \        # 链接路径：使用 conda 环境里的库文件
-lfaiss -lblas -fopenmp      # 链接 faiss、BLAS、OpenMP 库

# === 配置运行时动态库路径 ===
export LD_LIBRARY_PATH=$CONDA_PREFIX/lib:$LD_LIBRARY_PATH
# 确保运行时可以找到 conda 环境下的 libfaiss.so、libblas.so 等动态库

# === 运行打分脚本 ===
zxscorer "https://hpci.chouhsing.org/problems/cos-sim/" \
--token="b311594f-8e23-5551-88a2-0ca70c22584f" -- ./cos-opt-final
# zxscorer：比赛平台提供的打分工具
# --token：提交身份认证 token
# ./cos-opt-final：刚编译出的可执行程序

# 最后多交几发凹出满分

```

2.3 证明材料

JobId=20389618 JobName=interactive

运行日志

```
Problem: cos-sim
Score: 100
User: u2023211111
Command: ./cos-opt
Job ID: 20389618
Job Name: N/A
Job Info: JobId=20389618 JobName=interactive
   UserId=u2023211111(505604) GroupId=u2023211111(505604) MCS_label=N/A
   Priority=100 Nice=0 Account=u2023211111 QOS=normal
   JobState=RUNNING Reason=None Dependency=(null)
   Requeue=1 Restarts=0 BatchFlag=0 Reboot=0 ExitCode=0:0
   RunTime=00:30:31 TimeLimit=7-00:00:00 TimeMin=N/A
```

```
   SubmitTime=2025-08-24T04:10:06 EligibleTime=2025-08-24T04:10:06
   AccrueTime=Unknown
   StartTime=2025-08-24T04:10:06 EndTime=2025-08-31T04:10:06 Deadline=N/A
   SuspendTime=None SecsPreSuspend=0 LastSchedEval=2025-08-24T04:10:06
Scheduler=Main
   Partition=c003t AllocNode:Sid=workstation:13844
   ReqNodeList=(null) ExcNodeList=(null)
   NodeList=cpu7
   BatchHost=cpu7
   NumNodes=1 NumCPUs=40 NumTasks=1 CPUs/Task=40 ReqB:S:C:T=0:0:*:*
   TRES=cpu=40,mem=128G,node=1,billing=40
   Socks/Node=* NtasksPerN:B:S:C=0:0:*:* CoreSpec=*
   MinCPUsNode=40 MinMemoryNode=128G MinTmpDiskNode=0
   Features=(null) DelayBoot=00:00:00
   OverSubscribe=OK Contiguous=0 Licenses=(null) Network=(null)
   Command=(null)
   WorkDir=/home/u2023211111
   Power=




Run log:
Test case: Test 1
Duration: 95.080ms
Diff result: Maximum difference: 1.08033e-07
Maximum relative difference: 2.50127e-06
Diff stderr:
Tip:
Score: 25
Passed

Test case: Test 2
Duration: 245.621ms
Diff result: Maximum difference: 1.60187e-07
Maximum relative difference: 3.04372e-06
Diff stderr:
Tip:
Score: 25
Passed

Test case: Test 3
Duration: 576.979ms
Diff result: Maximum difference: 1.71363e-07
Maximum relative difference: 2.53563e-06
Diff stderr:
Tip:
```

```
Score: 25
Passed

Test case: Test 4
Duration: 1.573s
Diff result: Maximum difference: 1.63913e-07
Maximum relative difference: 2.77622e-06
Diff stderr:
Tip:
Score: 25
Passed
```

2.4 源代码与相关文件

源码路径：/home/u2023211111/share/zqh/cos-sim/cos-opt.cpp

相关文件见附件

---

3. llm-challenge

3.1 题目解法描述

3.1.1 为什么选择 llama.cpp

选择 **llama.cpp** 而不选择其他框架，是因为本次竞赛要求在纯 CPU 的 Linux x86 环境下高效完成大规模 LLM 推理，而 llama.cpp 原生支持 CPU 推理、多核并行和向量化指令，同时兼容量化模型（如 Q8_0），能够显著降低内存占用和提升推理速度。相比 PyTorch、TensorFlow 等框架依赖 GPU 或复杂环境，llama.cpp 部署轻量、启动快速，并提供连续批处理和上下文缓存机制，能够高效处理并发请求，保证在有限时间内完成 100 道题的推理任务，兼顾速度、正确率和资源利用率，符合竞赛的硬件和时间约束。

3.1.2 为什么选择 CS 架构

选择 **CS（Client-Server）架构** 的原因在于，它可以将模型加载和推理任务集中在服务器端常驻运行，充分利用 HPC 节点的多核 CPU 和大内存资源，同时客户端只负责题目解析、并发请求和答案提取，从而实现高吞吐和低延迟。相比本地逐题推理或一次性批量脚本，CS 架构可以支持多线程并发、连续批处理和上下文缓存，避免重复加载模型，提高资源利用率和稳定性，非常适合本次 100 道题 LLM 挑战的时间敏感和高并发要求。

3.1.3 为什么选择 Qwen3-4B-Q8_0

选择 **Qwen3-4B-Q8_0** 而非其他模型或量化参数，是因为它在 **模型规模、推理速度和内存占用**之间实现了最佳平衡：4B 参数保证了较高的理解和推理能力，而 Q8_0 量化显著降低了内存需求和计算开

销，使其可以在 40 核 CPU 节点上高速运行。相比更大模型（如 8B 或 14B）会占用过多内存、延长推理时间，或更低量化精度（如 Q4）可能损失准确率，Qwen3-4B-Q8_0 能够在保证正确率的同时实现低延迟、高吞吐，非常适合本次竞赛对 100 道题快速批量推理的需求。

测试过的模型目录：（部分为 Safetensors 格式，因纯 Transformers 推理速度过慢而被舍弃）

```
[u2023211111@cpu13 LLaMA]$ ls
Jan-v1-4B-Q8_0.gguf  llama-2-7b-chat.Q8_0.gguf
[u2023211111@cpu13 LLaMA]$ ls ../Qwen
DeepSeek-R1-0528-Qwen3-8B-Q8_0.gguf  Qwen3-1.7B-Q8_0.gguf     Qwen3-4B-Q6_K.gguf
Qwen2.5-3B                           Qwen3-4B-FP8            Qwen3-4B-Q8_0.gguf
Qwen3-0.6B                           Qwen3-4B-Instruct-2507  Qwen3-4B-Thinking-2507
Qwen3-14B-Q4_K_M.gguf                Qwen3-4B-Q4_K_M.gguf    Qwen3-8B-Q4_K_M.gguf
Qwen3-1.7B                           Qwen3-4B-Q5_K_M.gguf    Qwen3-8B-Q8_0.gguf
```

3.1.4 为什么选择这些采样参数

选择这样的采样参数，是为了在保证速度和稳定性的同时兼顾一定的随机性以提高正确率：
`max_tokens=1` 限制每次生成只输出一个字符（A/B/C/D），减少推理开销；
`temperature=1.0` 允许模型在不确定时有适度探索，避免总是贪心选择固定答案；`top_p=0.6` 和 `top_k=4` 限制采样范围（A/B/C/D），使输出更加集中在高概率选项上，减少错误答案概率；
`min_p=0` 与 `presence_penalty=0` 保持生成不受额外约束干扰，从而快速、稳定地生成每道题的最可能答案，满足 100 道题批量推理的效率需求。

最终 Diff stderr: Correct: 76/100

3.1.5 为什么选择这些 llama-server 启动命令

并行与线程相关

1. `--threads 40`

- 指定用于推理的 40 个线程。
- 每个线程独立执行模型计算，充分利用多核 CPU 提升并行能力，适合 HPC 节点 40 核 CPU 环境。

2. `--threads-batch 40`

- 指定在批量和提示处理时使用 40 个线程。
- 与 `--threads` 配合，实现线程内批处理，提高矩阵运算吞吐量，降低线程切换开销。

3. `--parallel 28`

- （神秘数字，很重要）并行请求数限制为 28，用于控制服务器负载和**一次处理的上下文长度**。
- 保证不会因为并发过多导致线程竞争或**上下文长度过短，同时影响推理速度和准确率**。

内存与 KV 优化

4. `--flash-attn`
- 启用 Flash Attention。
- 通过优化注意力计算，降低内存访问和运算延迟，加快 Transformer 模型推理速度。

5. `--kv-unified`
- 使用 统一 KV 缓存 存储所有序列的键值对。
- 避免多缓冲区管理开销，提升内存利用率。

6. `--no-kv-offload`
- 禁止 KV 缓存卸载到磁盘或其他存储。
- 保证所有计算在内存中完成，减少 IO 延迟。

7. `--defrag-thold 0.05`
- 设置内存碎片整理阈值为 5%。
- 防止 KV 缓存过度碎片化，提高内存连续性和访问效率。

8. `--mlock`
- 锁定内存，防止被操作系统换出。
- 确保推理过程中内存访问延迟最小化。

9. `--no-mmap`
- 禁止使用 mmap 直接映射模型文件到虚拟内存。
- 避免 mmap 可能带来的页表开销和 NUMA 不均衡。

CPU 核心绑定与 NUMA

10. `--cpu-range 0-39`
- 将推理线程绑定到 CPU 核 0~39。
- 避免线程在不同核心迁移导致缓存失效，提高 CPU cache 利用率和推理性能。

11. `--numa distribute`
- 在多 NUMA 节点系统中分布线程，使内存访问更均衡。
- 对于 HPC 节点大内存、多核配置，避免单节点内存过载，提高整体吞吐。

请求与批处理策略

12. `--cont-batching`

- 启用连续批处理（Continuous Batching）。
- 优点：多个短请求可被合并为一个批次发送到模型计算，减少上下文切换和线程等待，提高吞吐。

13. `--context-shift`

- 上下文位置滑动机制，允许连续请求共享上下文缓存。
- 支持连续生成和长上下文处理，减少重复上下文计算，提高推理速度和效率。

14. `--port 11411`

- 监听 HTTP 接口端口，用于客户端访问。
- 保证客户端可以通过 `http://127.0.0.1:11411/v1/completions` 发起请求。

15. `--prio 3 --prio-batch 3`

- 将服务器进程和批处理线程优先级设置为 实时级（realtime）。
- 保证高响应性。

最终 Duration: 36.346s

3.1.6 为什么这样写 prompt

prompt_final：

prompt_final:

```
# - 终极精简版本，极大减少上下文长度占用，提升在 max_new_tokens=1 时的执行效率
# - 每个问题填充到 {q}，要求模型输出单个选项（A/B/C/D）
# - 添加回退机制：If unsure, choose B，防止模型输出 “I'm sorry, ...”，同时保证收益最大化
PROMPT_TEMPLATE = """{q}.If unsure, choose B.Answer:"""
```

prompt_v1：

prompt_v1:

```
# - 初始版本，内容冗长，占用大量上下文长度
# - 提供了详细指令（阅读问题和选项、只输出单个字母、不要额外说明、独立处理问题）
# - 对 max_new_tokens=1 情况帮助不大，存在推理速度和上下文长度占用的效率浪费
PROMPT_TEMPLATE = """You are an AI assistant that answers multiple-choice questions.
Instructions:
1. Read the question and all answer options carefully.
2. Only output the correct answer as a single uppercase letter: A, B, C, or D.
```

```
3. Do NOT output any explanations, reasoning steps, or extra text.
4. Make sure the output is exactly one of A, B, C, or D.
5. Each question should be treated independently; do not copy answers from other questions.

Question:
{q}
Answer:"""
```

prompt_v2：

prompt_v2：

```
# - 精简版本，只保留必要提示
# - 大幅减少上下文长度，提高在 max_new_tokens=1 情况下的处理效率
# - 仍能保证模型输出单个选项 A/B/C/D
PROMPT_TEMPLATE = """You are an AI assistant that answers multiple-choice questions.
Only output a single uppercase letter: A, B, C, or D.

Question:
{q}
Answer:"""
```

prompt_v3（A/B/C/D）：

prompt_v3（A/B/C/D）：

```
# - 进一步精简，占用上下文长度最小化
# - 每次测试提供不同回退选项（A/B/C/D），防止模型输出 “I'm sorry, ...”
# - 通过反复测试不同回退选项，找到最佳收益的最终 prompt_final
PROMPT_TEMPLATE = """{q}.If unsure, choose A/B/C/D.Answer:"""
```

3.2 优化过程记录

环境配置：

代码块

```
# 切换到工作目录
cd /home/u2023211111/share/zqh

# 克隆 Qwen3-4B-GGUF 模型仓库到本地
```

```
git clone https://hf-mirror.com/Qwen/Qwen3-4B-GGUF # 通过 git 或其他手段下载 Qwen3-4B-Q8_0.gguf 模型


# 安装 uv（一个轻量级的版本管理工具，类似 pyenv/virtualenv）
curl -LsSf https://astral.sh/uv/install.sh | sh

# 检查 uv 是否安装成功以及版本
uv --version

# 使用 uv 安装指定版本的 Python
uv python install 3.11.13

# 设置 uv 当前使用的 Python 版本
uv python use 3.11.13

# 使用 uv 创建一个 Python 虚拟环境，名字为 zhangqihan
uv venv create zhangqihan

# 激活虚拟环境，确保 uv 管理的 Python 版本是 3.11.13
source /home/u2023211111/share/zqh/zhangqihan/bin/activate # uv python = 3.11.13


# 切换到 llama 目录准备下载 llama.cpp
cd /home/u2023211111/share/zqh/llama

# 克隆 llama.cpp 仓库到本地
git clone https://github.com/ggml-org/llama.cpp.git # 下载 llama.cpp 库，用于加载 GGUF 模型

# 进入构建目录
cd llama.cpp/build

# 安装 cmake（构建 C++ 项目必备工具）
uv pip install cmake

# 配置 llama.cpp 构建选项，关闭 curl 支持（避免依赖）
cmake .. -DLLAMA_CURL=OFF

# 编译 llama.cpp，-j 表示使用多线程加速编译
make -j

# 在 HPC 环境上申请节点，使用 llama-cli 测试模型
salloc -N 1 -n 1 -p c003t -c 40 --mem=160G
ssh cpuXX # XX换为实际申请到的cpu节点序号

```

```
# 使用 llama-cli 测试模型，-m 指定模型文件路径
# -p 提供 prompt 输入，进行推理
/home/u2023211111/share/zqh/llama/llama.cpp/build/bin/llama-cli \
-m /home/u2023211111/share/zqh/Qwen/Qwen3-4B-Q8_0.gguf -p "Hello, how are you?"

# user
# Hello, how are you?
# assistant
# <think>
# Okay, the user greeted me with "Hello, how are you?"
# I need to respond appropriately. First, I should acknowledge their greeting.
# Since I'm an AI, I don't have feelings, but I can say I'm here to help.
# I should keep it friendly and open-ended to encourage them to ask questions.
# Maybe add an emoji to keep it light. Let me check if that's natural.
# Yeah, that should work.
# </think>
# Hello! I'm just a language model, so I don't have feelings,
# but I'm here and ready to help! How can I assist you today? 😊
```

程序构建：

qwen_llama_client.py

```python
import sys
import json
import urllib.request
import urllib.error
from concurrent.futures import ThreadPoolExecutor, as_completed

# 本地 LLaMA 模型服务的 URL（遵循 OpenAI 风格 API）
LLAMA_SERVER_URL = "http://127.0.0.1:11411/v1/completions"

# 日志文件路径，用于记录每个问题的请求与响应
LOG_FILE = "llama_batch_log.jsonl"

# 最大并发线程数（同时处理的问题数量）
MAX_WORKERS = 28

# 提示模板，每个问题会填充到 {q}，并要求模型回答选项（A/B/C/D）
# 默认答案回退到 B，防止模型输出 "I'm sorry, ..."
PROMPT_TEMPLATE = """{q}.If unsure, choose B.Answer:"""


def read_questions():
    """
    从标准输入读取所有问题，并按空行分隔。
```

```
    返回问题列表。
    """
    content = sys.stdin.read()  # 一次性读入所有输入
    # 去掉首尾空白，按空行切分问题，去除空白问题
    questions = [q.strip() for q in content.split("\n\n") if q.strip()]
    return questions


def build_payload(question):
    """
    根据单个问题构建 HTTP 请求的 payload（JSON 数据）。
    包括 prompt、生成参数（max_tokens、temperature、top_p 等）。
    """
    prompt = PROMPT_TEMPLATE.format(q=question)  # 格式化提示
    payload = json.dumps({
        "prompt": prompt,
        "max_tokens": 1,       # 只生成一个 token（只需输出选项 A/B/C/D）
        "temperature": 1.0,    # 温度参数，越大越随机，0=贪心
        "top_p": 0.6,          # nucleus sampling，采样概率阈值
        "top_k": 4,            # 限制候选 token 数量
        "min_p": 0,            # 不做下限限制
        "presence_penalty": 0 # 惩罚项（此情境允许重复）
    }).encode("utf-8")        # 转为字节串以便 HTTP 发送
    return payload


def fetch_answer(idx, question):
    """
    发送请求到 LLaMA 服务，获取某个问题的答案。
    - idx: 问题编号
    - question: 问题文本
    返回 (idx, 答案字母)。
    """
    payload = build_payload(question)
    # 构造 HTTP POST 请求
    req = urllib.request.Request(
        LLAMA_SERVER_URL,
        data=payload,
        headers={"Content-Type": "application/json"}
    )
    try:
        # 发送请求并等待响应，超时时间 60s
        with urllib.request.urlopen(req, timeout=60) as resp:
            data = json.load(resp)  # 解析 JSON 响应

            # 写入日志（包含 index、问题和完整响应）
            log_entry = {"index": idx, "question": question, "response": data}
```

```
            with open(LOG_FILE, "a", encoding="utf-8") as f:
                f.write(json.dumps(log_entry, ensure_ascii=False) + "\n")

            # 从响应中提取模型输出的文本
            text = data.get("choices", [{}])[0].get("text", "")

            # 默认答案设为 "B"
            ans = "B"
            # 遍历输出的字符，找第一个 A/B/C/D
            for c in text.upper():
                if c in "ABCD":
                    ans = c
                    break
            return idx, ans

    except Exception as e:
        # 出现错误时记录日志，并返回默认答案 "B"
        log_entry = {"index": idx, "question": question, "error": str(e)}
        with open(LOG_FILE, "a", encoding="utf-8") as f:
            f.write(json.dumps(log_entry, ensure_ascii=False) + "\n")
        return idx, "B"


def main():
    """
    主函数：
    - 读取问题
    - 并发调用 fetch_answer 获取答案
    - 按问题顺序输出结果
    """
    questions = read_questions()
    results = [None] * len(questions)  # 用于存放答案，保持顺序

    # 使用线程池并发请求
    with ThreadPoolExecutor(max_workers=MAX_WORKERS) as executor:
        # 提交任务（每个问题一个任务），返回 future → idx 的映射
        futures = {executor.submit(fetch_answer, idx, q): idx for idx, q in
enumerate(questions)}
        # 按任务完成顺序取结果
        for future in as_completed(futures):
            idx, ans = future.result()
            results[idx] = ans  # 按原始顺序保存

    # 逐行输出所有答案
    for ans in results:
        print(ans)


```

```
117
118 # 脚本入口
119 if __name__ == "__main__":
120     main()
```

任务提交（可直接复现）：

Server:

Bash

```
1  # 在 HPC 环境上申请节点
2  salloc -N 1 -n 1 -p c003t -c 40 --mem=160G
3  ssh cpuXX # XX换为实际申请到的cpu节点序号
4
5  # 激活 uv Python 虚拟环境
6  source /home/u2023211111/share/zqh/zhangqihan/bin/activate # uv python =
   3.11.13
7
8  # 启动 llama.cpp 的 LLM 服务端
9  /home/u2023211111/share/zqh/llama/llama.cpp/build/bin/llama-server \
10 --model /home/u2023211111/share/zqh/Qwen/Qwen3-4B-Q8_0.gguf \
11 --threads 40 --threads-batch 40 --flash-attn --kv-unified --no-kv-offload \
12 --defrag-thold 0.05 --mlock --no-mmap --cpu-range 0-39 --parallel 28 \
13 --cont-batching --numa distribute --port 11411 --prio 3 --prio-batch 3 \
14 --context-shift
15
16 # main: server is listening on http://127.0.0.1:11411 - starting the main loop
17 # srv  update_slots: all slots are idle
```

Client:

Bash

```
1  # SSH 登录到与 Server 相同的节点
2  ssh cpuXX # XX换为Server实际申请到的cpu节点序号
3
4  # 激活 uv Python 虚拟环境（与 Server 端一致）
5  source /home/u2023211111/share/zqh/zhangqihan/bin/activate # uv python =
   3.11.13
6
7  # 切换到客户端目录
8  cd /home/u2023211111/share/zqh/infer
9
10 # 启动 zxscorer 并运行 LLM 客户端脚本
```

```
srun zxscorer "https://hpci.chouhsing.org/problems/llm-challenge/" \
--token="b311594f-8e23-5551-88a2-0ca70c22584f" -- python3 qwen_llama_client.py

# 最后使用同目录下 ./run_loop.sh 脚本多交几百发凹出满分（换算后）
```

3.3 证明材料

JobId=20390147 JobName=interactive

运行日志

```
Problem: llm-challenge
Score: 77.53758141895084
User: u2023211111
Command: python3 qwen_llama_client.py
Job ID: 20390147
Job Name: zxscorer
Job Info: JobId=20390147 JobName=interactive
   UserId=u2023211111(505604) GroupId=u2023211111(505604) MCS_label=N/A
   Priority=100 Nice=0 Account=u2023211111 QOS=normal
   JobState=RUNNING Reason=None Dependency=(null)
   Requeue=1 Restarts=0 BatchFlag=0 Reboot=0 ExitCode=0:0
   RunTime=02:48:09 TimeLimit=7-00:00:00 TimeMin=N/A
   SubmitTime=2025-08-24T10:55:59 EligibleTime=2025-08-24T10:55:59
   AccrueTime=Unknown
   StartTime=2025-08-24T10:55:59 EndTime=2025-08-31T10:55:59 Deadline=N/A
   SuspendTime=None SecsPreSuspend=0 LastSchedEval=2025-08-24T10:55:59
Scheduler=Main
   Partition=c003t AllocNode:Sid=workstation:11919
   ReqNodeList=(null) ExcNodeList=(null)
   NodeList=cpu8
   BatchHost=cpu8
   NumNodes=1 NumCPUs=40 NumTasks=1 CPUs/Task=40 ReqB:S:C:T=0:0:*:*
   TRES=cpu=40,mem=160G,node=1,billing=40
   Socks/Node=* NtasksPerN:B:S:C=0:0:*:* CoreSpec=*
   MinCPUsNode=40 MinMemoryNode=160G MinTmpDiskNode=0
```

```
   Features=(null) DelayBoot=00:00:00
   OverSubscribe=OK Contiguous=0 Licenses=(null) Network=(null)
   Command=(null)
   WorkDir=/home/u2023211111
   Power=



Run log:
Test case: Test 1
Duration: 36.346s
Diff result: 76
Diff stderr: Correct: 76/100
Tip:
Score: 77.53758141895084
Passed
```

3.4 源代码与相关文件

llama.cpp路径：/home/u2023211111/share/zqh/llama/llama.cpp/build/bin/llama-server

llama.cpp仓库地址：https://github.com/ggml-org/llama.cpp

Qwen3-4B-Q8_0路径：/home/u2023211111/share/zqh/Qwen/Qwen3-4B-Q8_0.gguf

Qwen3-4B-Q8_0仓库地址：https://huggingface.co/Qwen/Qwen3-4B-GGUF

Client代码路径：/home/u2023211111/share/zqh/infer/qwen_llama_client.py

相关文件见附件

---

4. md5-new

4.1 题目解法描述

没写喵

4.2 优化过程记录

没写喵

4.3 证明材料

没写喵

4.4 源代码与相关文件

没写喵

---

5. traffic-detector

5.1 题目解法描述

本题通过对TCP和DNS流量的规则化分析，实现恶意行为检测：使用五元组和哈希表高效管理TCP流状态，记录SYN包与后续流量判断端口扫描；通过域名前缀长度累加识别DNS隧道；结合 `mmap` 快速读入数据、线程局部统计、多线程并行处理和字典序排序，实现了在HPC环境下对大规模流量文件的高性能统计与输出。

5.2 优化过程记录

代码块

```
salloc -N 1 -n 1 -p c003t -c 40 --mem=128G
ssh cpuXX # XX换为实际申请到的cpu节点序号

source ~/share/zqh/zhangqihan/bin/activate # uv python = 3.11.13

cd share/zqh/traffic-detector/

g++ -O3 -march=native -flto -pipe -pthread traffic.cpp -o traffic

zxscorer "https://hpci.chouhsing.org/problems/traffic-detector/" \
--token="b311594f-8e23-5551-88a2-0ca70c22584f" -- ./traffic
```

5.3 证明材料

JobId=20390493 JobName=interactive

代码块

```
Problem: traffic-detector
Score: 90.30143156070073
User: u2023211111
Command: ./traffic
Job ID: 20390493
Job Name: N/A
Job Info: JobId=20390493 JobName=interactive
   UserId=u2023211111(505604) GroupId=u2023211111(505604) MCS_label=N/A
   Priority=100 Nice=0 Account=u2023211111 QOS=normal
   JobState=RUNNING Reason=None Dependency=(null)
   Requeue=1 Restarts=0 BatchFlag=0 Reboot=0 ExitCode=0:0
   RunTime=00:13:40 TimeLimit=7-00:00:00 TimeMin=N/A
   SubmitTime=2025-08-24T11:41:09 EligibleTime=2025-08-24T11:41:09
   AccrueTime=Unknown
   StartTime=2025-08-24T11:41:09 EndTime=2025-08-31T11:41:09 Deadline=N/A
   SuspendTime=None SecsPreSuspend=0 LastSchedEval=2025-08-24T11:41:09
Scheduler=Main
   Partition=c003t AllocNode:Sid=workstation:28291
   ReqNodeList=(null) ExcNodeList=(null)
   NodeList=cpu6
   BatchHost=cpu6
   NumNodes=1 NumCPUs=40 NumTasks=1 CPUs/Task=40 ReqB:S:C:T=0:0:*:*
   TRES=cpu=40,mem=128G,node=1,billing=40
   Socks/Node=* NtasksPerN:B:S:C=0:0:*:* CoreSpec=*
   MinCPUsNode=40 MinMemoryNode=128G MinTmpDiskNode=0
   Features=(null) DelayBoot=00:00:00
   OverSubscribe=OK Contiguous=0 Licenses=(null) Network=(null)
   Command=(null)
   WorkDir=/home/share/zqh/infer
   Power=



Run log:
Test case: Test Case
Duration: 24.535s
Diff result: Files are identical
Diff stderr: None
Tip:
Score: 90.30143156070073
Passed
```

5.4 源代码与相关文件

源码路径：/home/u2023211111/share/zqh/traffic-detector/traffic.cpp

相关文件见附件