# 胡一涛 - 本科科研助手思考题

## 联系方式

胡一涛
微信：16600641268
邮箱：yitao@tju.edu.cn
导师及项目简介：胡一涛-本科科研助手-20230918

# 思考题介绍

**题目要求**：请新建一个腾讯文档撰写编程练习需要提交的内容，也可撰写一段论文总结（不作要求），完成论文阅读和编程练习后**将文档链接微信发给胡老师**。

**截止时间：10 月 31 日**

## 题目 1：论文阅读

1. **阅读如下论文，学习编程练习的背景知识**

   **论文需要阅读的部分已经发送到微信群内，主要阅读黄色标注区域，其余内容不做要求**

   **论文题目**：Fast Distributed Inference Serving for Large Language Models

   **原文链接**：https://arxiv.org/abs/2305.05920

   重点关注以下问题：

   （1）这篇论文研究了一个什么问题？

   （2）这篇论文之前，别的论文是怎么解决这个问题的，这些论文有哪些不足？

   （3）这篇论文比前人结果好的核心的思想/创新（insight）是什么？

   （4）这篇论文的解决方案（solution）是什么？

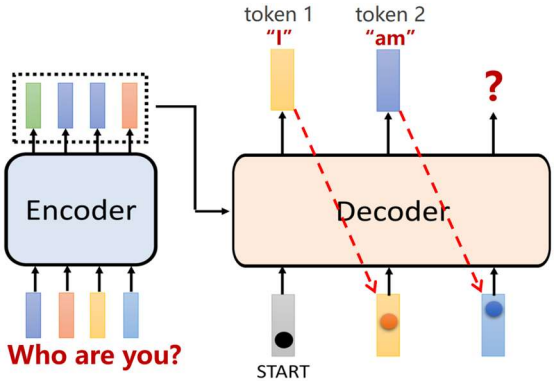
2. **背景介绍**

   大语言模型 LLM（Large Language Models）如 GPT（Generative Pretrained Transformer）和 LLaMA（Large Language Model Meta AI）等是一种拥有千万到千亿级别的参数量的深度学习自然语言处理模型，通常采用基于 Transformer 的模型结构。

   这些模型在训练和微调之后，就进入**模型推理（serving or inference）阶段**。以 GPT 为例，LLM 在推理阶段模型需要接收用户的输入请求（input 或 prompt；最小单位是 token，可以理解为一个单词），并将输入编码（encoding）为向量表示，然后传递给模型的神经网络层（主要是 Transformer）进行推理计算，得到输出结果（由多个 token 组成的 output）。

   LLM 推理通常有几个特征：（1）**输入输出长度不固定**，且输出长度无法/难以预测，即用户的问题长度没有固定限制，输出长度随问题而变化且难以预测；（2）**自回归（autoregressive）的推理模式**：如下图所示，与传统 DNN 模型的图像推理任务不同，LLM 的推理需要多次的迭代（iteration）才能得到完整的输出。每个 iteration 生成一个新的 token（可以简单地理解为一个单词），且每个 iteration 都需要根据上一个 iteration 的结果得到，因此称为“自回归模式(autoregressive pattern)”。LLM 的每个 iteration 的时间基本相同，但由

于输出长度不定，因此完成一句话的推理总时间未知。(3) **首个 iteration 时间长**：LLM 推理过程分为两个阶段：prefill 阶段和自回归阶段。prefill 阶段通过大量计算根据输入内容建立 KV Cache 并生成第一个 token。因此时间会比后续的自回归阶段生成 token 的时间更长；而后续自回归阶段生成每个 token 的时间都非常相近，在仿真实验时可以认为是相同的。在仿真时我们可以自行设置 每个请求将会输出的 token 数量，也就是需要经过多少个 iteration 才能完成请求。




一个简单的推理例子：输入“Who are you?”，第一个 iteration 根据输入编码向量推理得到“I”，第二个 iteration 根据输入和“I”得到“am”，……直到完成整句话的输出。

由于输入输出的不定长以及自回归特性，为了保证大量用户同时使用的推理效率与用户体验，这篇论文（FastServe）采用了 iteration-level 的调度策略 skip-join MLFQ，最小化平均作业完成时间（Job Completion Time），降低平均每个用户平均等待结果的时间。

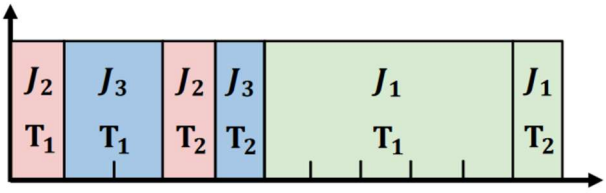
3. **可能有用的背景知识讲解课程**：https://www.bilibili.com/video/BV1v3411r78R

## 思考题 2：编程仿真

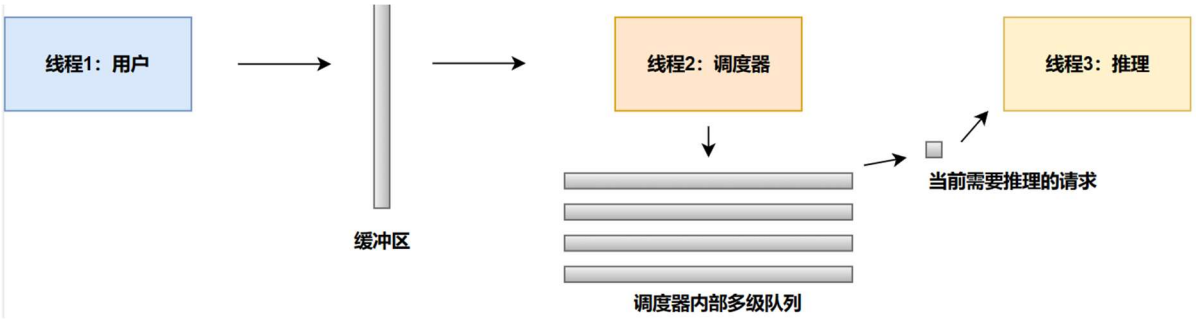
要求：完成编程后请将代码上传到 github 或 gitee 等平台，并将代码链接和运行结果、实验数据和分析写到论文总结的文档中。

参考论文 4.1 节中提出的 skip-join MLFQ 调度算法伪代码（论文第 6 页 Algorithm 1），编程**完成“skip-join MLFQ”调度策略仿真和实验**（即不进行真实的推理，用 sleep 一段时间模拟推理过程；编程语言可选用 C/C++/Python），并做实验观察调度器的性能。

此调度算法关注的指标是**平均任务完成时间 average JCT**，即所有请求从发送到收到结果的平均时间。如下图所示，J1,J2,J3 三个请求同时到达，J2 的 JCT 为 4，J3 的 JCT 为 5，J1 的 JCT 为 11，他们的平均 JCT 为(4+5+11)/3=6.67。




编写仿真需要使用多线程。参考经典的 Producer-Consumer 模型，我们共需要 3 个线程，简化的流程图如下图所示：




1. **线程 1**：用于**发送推理请求**的用户线程，需要设置发送分布，发送速率等参数。在没有发送完设置的请求数量之前，用户线程将持续以用户设置的发送速率发送请求，发送的请求可以暂时放入缓冲区内等待调度器线程调度，也可自行设置其他的方法。

2. **线程 2**：（请参考论文 Algorithm 1 实现）用于调度用户请求的**调度器线程**，需要参考论文中提出的 schedule 算法进行设计。多级队列包含三个参数，**Queue_num**, **Quantum** 和 **Quantum_rate**，Queue_num 代表多级队列一共有多少级，即包含了多少个队列；Quantum 需要设置为自回归阶段生成一个 token 的时间，而 Quantum_rate 则需要手动设置，它代表**队列之间能够推理自回归阶段 token 数量的倍数**。例如 Quantum=1, Quantum_rate=2 时，二级队列的 Quantum 即为 2，代表在自回归阶段可以一次生成两个 token，三级队列可以一次生成 4 个 token**。**
   a. 处理用户线程发送到缓冲区内的线程，根据请求提示词的长度决定将其分配到多级队列中的哪一级。**请求需要放置在 Quantum 大于 prefill 阶段推理时长的最高优先级的队列**。按照上文的例子，例如此刻到达的请求 prefill 阶段需要 3ms，则需要将其放在 Quantum 为 4 的第三级队列。
   b. 选中多级队列中第一个任务，将其放入推理线程中。
   c. 为简化模拟过程，不考虑任务公平性。（即不考虑伪代码中 Promote starved jobs 部分）

3. **线程 3**：用于模拟推理过程的**推理线程**，使用 time.sleep 或 chrono::high_resolution_clock 模拟处理时间。（注：为了**简化问题**，仿真时同一时刻只提交推理一个任务即可，当该任务的 quantum 用完之后再安排下一个任务执行。可以在线程 2 中使用“xxxqueue.get()”阻塞等待线程 3 中的任务用完 quantum 并返回后，继续进行调度；线程 3 同样使用“xxqueue.get()”阻塞等待线程 2 提交的任务）
   a. 放入推理线程的请求需要进行 n 次推理，也就是进行了 n 次迭代，生成了 n 个 token。设 x 为当前队列的级数，n 为 Quantum_rate 的 x-1 次方。如果该请求未达到它所需要的迭代次数，需要将其放入调度器的多级队列中。此时请求需要降级至下一级队列。

- **实验内容：**

1. **实验 1**：根据实现的算法，复现论文 Figure 6(c) skip-join MLFQ 调度示意图（即第一张示例图）的调度过程，验证复现算法的正确性。
2. **实验 2**：以附录 B 中的数据集作为输入，记录调度输出结果、每个任务的 JCT 以及总平均 JCT。

- **需要提交的内容**

1. 输出请求的执行过程（每推理一个 token 输出一行：任务 ID、生成 token 的编号）
2. 调度器完成所有任务后每个任务的 JCT 以及总的平均 JCT，能够以图像展示更佳

- **hint**
  a. Python 实现需要约 100-200 行代码
  b. C++实现需要约 400-600 行代码

## 附加题：

1. 分析本文所提出的“skip-join MLFQ”调度策略在不同输入输出特征下的性能：
   a. 尝试不同的数据发送速率，不同的发送分布下本文提出方法的性能。
   b. 改变多级队列中队列的数量，Quantum_rate，观察实验结果并分析该如何设置队列数量以及 Quantum_rate？
2. 尝试实现 FCFS(First Come First Serve, 先来先服务)的调度器，即对于到达调度器的请求，先到达的请求将会优先进行推理。并将 FCFS 的模拟调度结果与本文提出方法的模拟调度结果相比较。