Fast On-device LLM Inference with NPUs

Daliang Xu[♦], Hao Zhang[◇], Liming Yang[♦], Ruiqi Liu[♦], Gang Huang[♦], Mengwei Xu[◇*], Xuanzhe Liu[♦]
[♦]Peking University, [◇]Beijing University of Posts and Telecommunications
Code and demo: https://github.com/UbiquitousLearning/mllm

ASPLOS'25

汇报人：姚昌硕

2024年11月15日

作者团队

- 徐梦炜 北京邮电大学
- 研究领域:
  - **资源高效的AI系统**：智能手机、可穿戴设备和物联网设备等资源受限平台上的推理和训练 （尤其是LLM）
  - 卫星平台与计算
- xumengwei.github.io

作者团队

- 刘譞哲 北京大学
- 研究领域:
  - **移动计算系统的性能测量与优化**
  - Web APIs 和 mashups
  - 用户行为/交互分析
- www.liuxuanzhe.com

研究背景

- 研究背景
  - 移动端本地LLM推理
  - 存在的问题
  - LLM推理两个阶段
- 问题分析与发现
- 系统结构与原理
- 实验
- 总结

- LLM的广泛应用
  - 语音助手 文本生成 … …
- 移动端本地LLM推理的价值
  - 低时延，实时响应
  - 隐私保护
  - 离线响应
  - 能源效率
  - 个性化定制

Apple Intelligence

Android AICore

Google LLC

研究背景

- 研究背景
  - 移动端本地LLM推理
  - 存在的问题
  - LLM推理两个阶段
- 问题分析与发现
- 系统结构与原理
- 实验
- 总结

- 存在的问题：高推理时延
- 原因：
  - 移动端CPU/GPU性能有限
  - 移动端LLM任务通常需要长prompt
    - UI 自动化任务: 600-800 tokens for each step
    - 模仿用户语气回复电子邮件：1500 tokens
  - LLM模型支持长的上下文窗口
    - Qwen2-1.5B：32K tokens

研究背景

- 研究背景
  - 移动端本地LLM推理
  - 存在的问题
  - LLM推理两个阶段
- 问题分析与发现
- 系统结构与原理
- 实验
- 总结

- 移动NPU的普及及架构优势
  - 现代移动端SoC普遍集成了NPU
- 优势：整数运算/并行计算
  - 计算大规模整数矩阵乘法，速度和能效均优于CPU/GPU
- 局限性：浮点数运算，NPU SDK闭源

研究背景

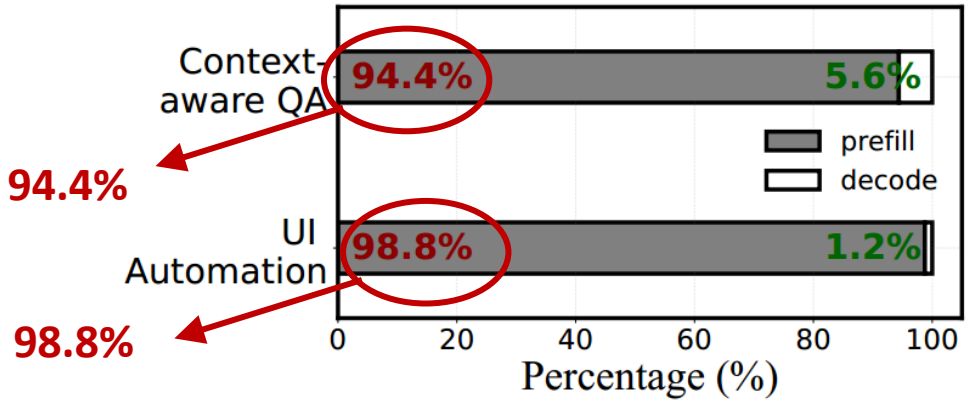
- 研究背景
  - 移动端本地LLM推理
  - 存在的问题
  - LLM推理两个阶段
- 问题分析与发现
- 系统结构与原理
- 实验
- 总结

- Prefill: 从prompt输入到第一个token输出的过程
  - 计算密集型
  - 只会一次性写入KV Cache，不会读取
  - Prompt一次性输入，并行度高
- Decode: 逐步生成输出token直至结尾的过程
  - 访存密集型
  - 频繁读写KV Cache
  - Decoder通过自回归方式生成输出，并行度低

研究背景

- 研究背景
  - 移动端本地LLM推理
  - 存在的问题
  - LLM推理两个阶段
- 问题分析与发现
- 系统结构与原理
- 实验
- 总结

- 在移动端，Prefill耗时很长，主导端到端时延



- NPU适合高并行度的计算
- 现有工作主要集中在Decode阶段，Prefill阶段较少
  - 激活稀疏性 activation sparsity
  - 推测解码 speculative decoding

- 作者使用移动端NPU对LLM推理中的Prefill阶段进行加速

问题分析与发现

- 研究背景
- 问题分析与发现
  - 可变长的Prompt
  - 离群激活值
  - 浮点数运算
- 系统结构与原理
- 实验
- 总结

- 现有DNN引擎不支持移动NPU上的LLM加速
- 现有的NPU设计和LLM推理流程之间存在巨大的鸿沟

问题分析与发现

- 研究背景
- 问题分析与发现
  - 可变长的Prompt
  - 离群激活值
  - 浮点数运算
- 系统结构与原理
- 实验
- 总结

- LLM需求：
  - 可变长的输入prompt → 动态构建和优化计算图
  - 每次推理需重建计算图，难以复用相同的图结构
- NPU特点：
  - 每次变更计算图，NPU需要花费时间进行构建和优化
- 后果：
  - 频繁重建和优化图，导致性能瓶颈，耗时较长
  - 若不做优化，NPU相比CPU反而更慢

问题分析与发现

- 研究背景
- 问题分析与发现
  - 可变长的Prompt
  - 离群激活值
  - 浮点数运算
- 系统结构与原理
- 实验
- 总结

- LLM需求：
  - 激活值存在较大波动，导致对Per-Tensor量化非常困难
  - 细粒度的量化方法（如Per-Group）可保证精度
- NPU特点：
  - Per-Group量化需拆分矩阵计算，并需要浮点加法聚合结果
- 后果：
  - 细粒度的量化方法使得NPU无法充分发挥效率优势，反而增加了推理的开销
  - 精度 / 性能难以平衡

问题分析与发现

- 研究背景
- 问题分析与发现
  - 可变长的Prompt
  - 离群激活值
  - 浮点数运算
- 系统结构与原理
- 实验
- 总结

- LLM需求：
  - 部分层依赖浮点数计算（Attention，LayerNorm）
  - 这些浮点操作对于保持模型精度至关重要
- NPU特点：
  - 浮点数的计算效率低
- 后果：
  - 量化 → 模型精度下降
  - 不量化 → NPU无法高效计算，性能下降
  - 精度 / 性能难以平衡

问题分析与发现

- 研究背景
- 问题分析与发现
  - 可变长的Prompt
  - 离群激活值
  - 浮点数运算
- 系统结构与原理
- 实验
- 总结

- LLM推理需求与移动端NPU特性存在矛盾

LLM需求	NPU特点	造成的问题
处理可变长度的prompt	通常只支持静态 计算图结构	重建计算图：代价大 填充到统一长度：浪费计算资源
激活值分布存在异常大/小的值 故常使用Per-Group量化	对Per-Group量化 支持不佳	需要每个Group分别计算 然后规约得到结果
部分计算依赖浮点数，无法量化 (Attention / LayerNorm)	提供整数计算加速 浮点计算能力较弱	需要调度到CPU/GPU 增加调度开销

mllm-NPU 系统概述

- 研究背景
- 问题分析与发现
- 系统结构与原理
  - 系统概述
  - ① 计算图切分
  - ② 离群值处理
  - ③ 乱序调度
- 实验
- 总结

- 针对前面的问题，提出三个解决方案
  - 处理可变长度的prompt → ① Chunk-sharing graph execution
  - 激活值存在离群值(outlier) → ② Shadow Outlier Execution
  - 部分计算需要调度到CPU → ③ Out-of-order subgraph execution

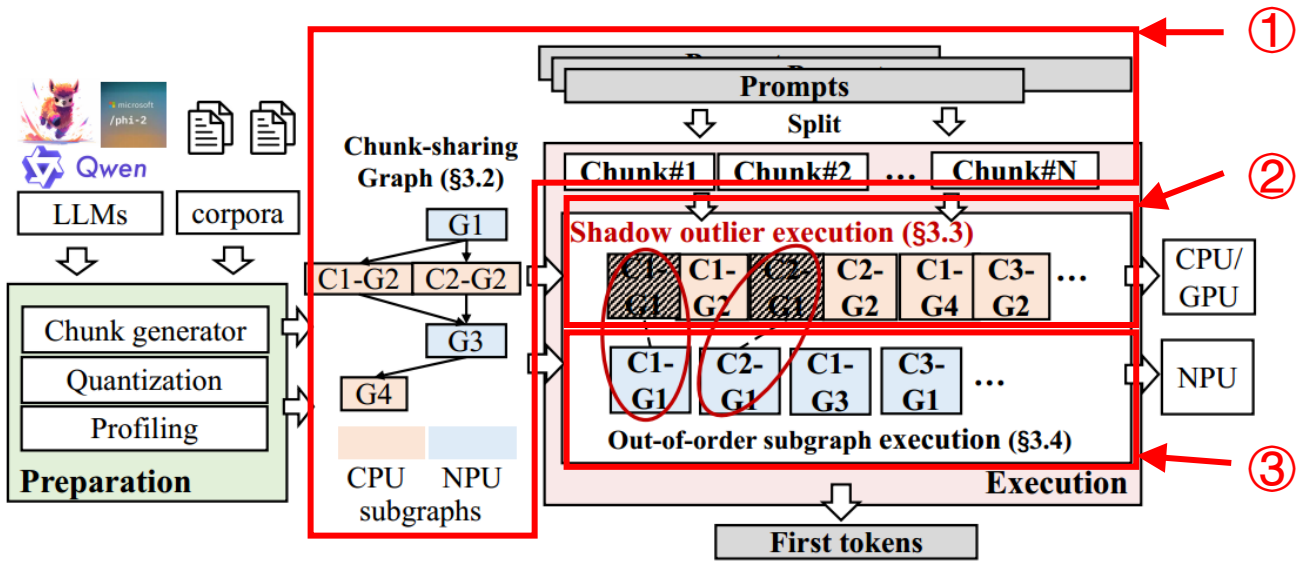


Figure 6. The workflow of mllm-NPU.

① Chunk-sharing graph execution

- 研究背景
- 问题分析与发现
- 系统结构与原理
  - 系统概述
  - ① 计算图切分
  - ② 离群值处理
  - ③ 乱序调度
- 实验
- 总结

- ① Chunk-sharing graph execution (计算图切分)

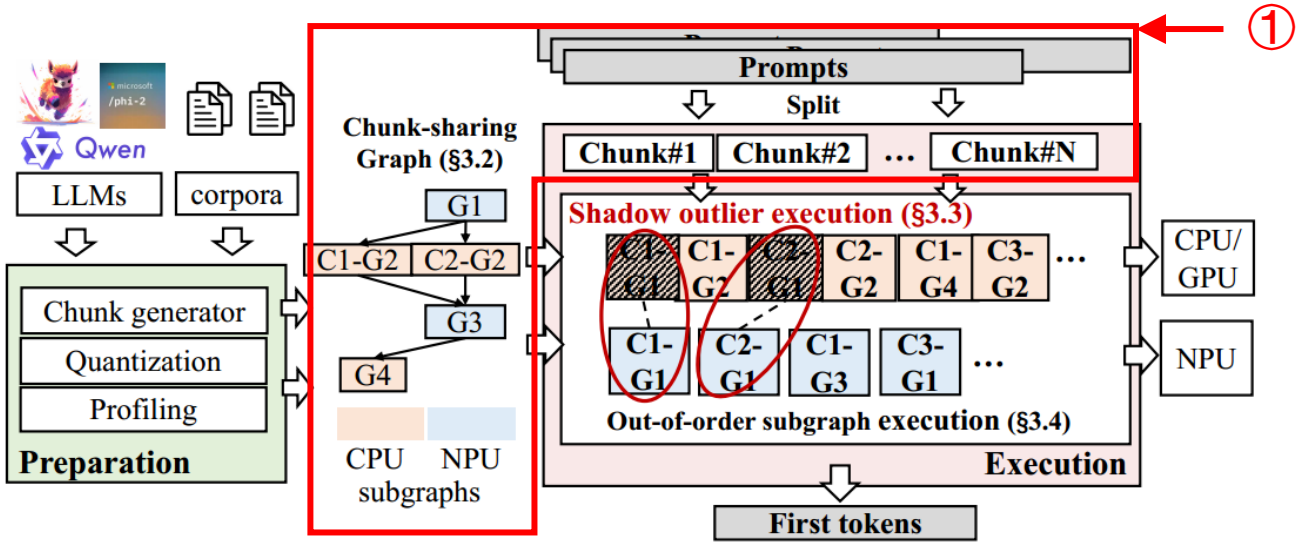
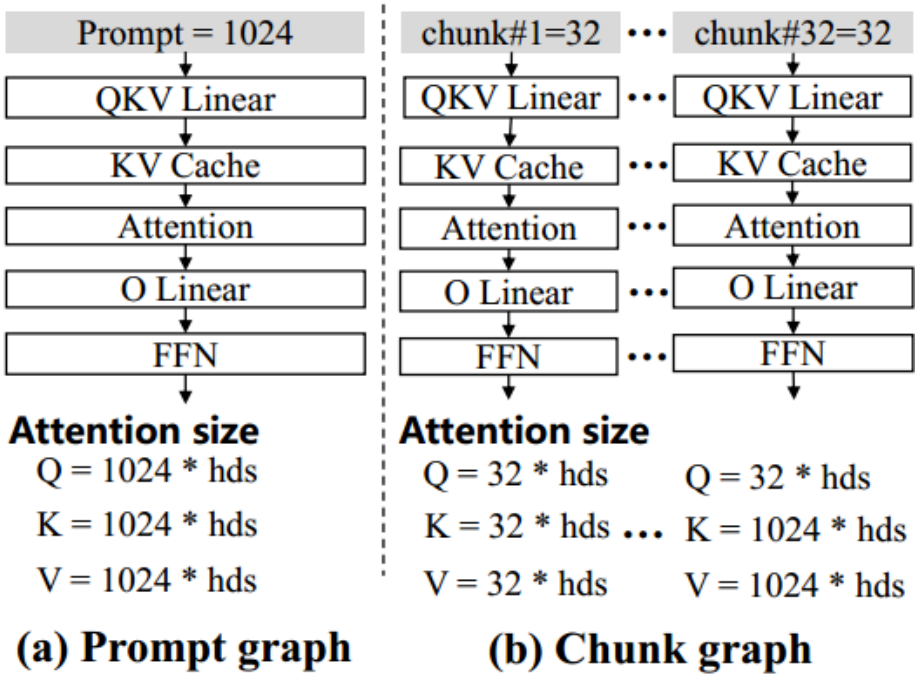


Figure 6. The workflow of `mllm-NPU`.

① Chunk-sharing graph execution

- 研究背景
- 问题分析与发现
- 系统结构与原理
  - 系统概述
  - ① 计算图切分
  - ② 离群值处理
  - ③ 乱序调度
- 实验
- 总结

- ① Chunk-sharing graph execution (计算图切分)
- 关键思路：把prompt划分为固定尺寸的chunk



① Chunk-sharing graph execution

- 研究背景
- 问题分析与发现
- 系统结构与原理
  - 系统概述
  - ① 计算图切分
  - ② 离群值处理
  - ③ 乱序调度
- 实验
- 总结

- 静态算子(绿) 只与chunk长度有关 可以共享
- 动态算子(红) 与chunk位置有关 不可以共享
- 这种方法显著降低了内存开销并增强了可扩展性

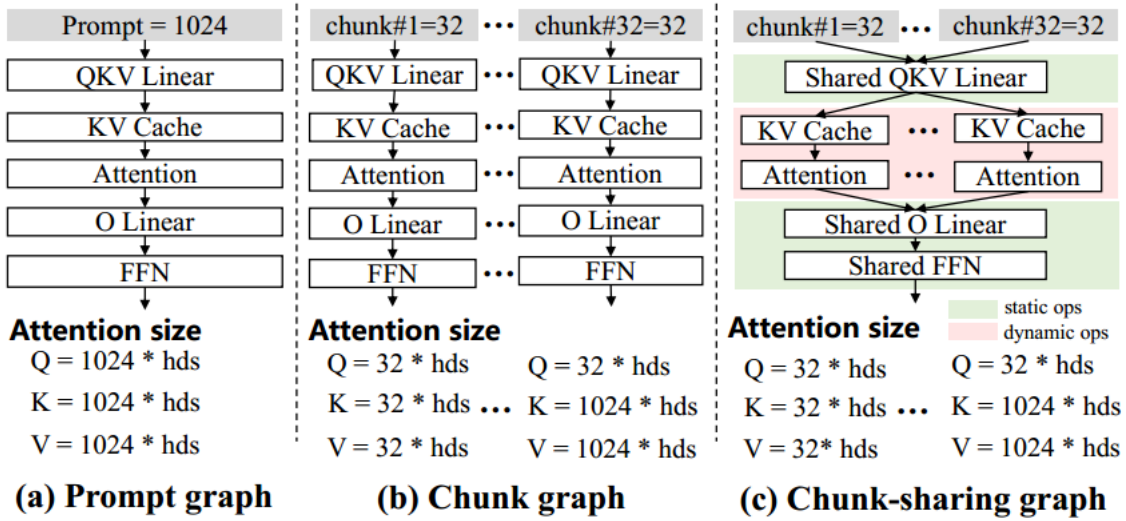


Figure 7. The illustration of prompt graph, chunk graph and chunk-sharing graph. The chunk length is 32.

② Shadow Outlier Execution

- 研究背景
- 问题分析与发现
- 系统结构与原理
  - 系统概述
  - ① 计算图切分
  - ② 离群值处理
  - ③ 乱序调度
- 实验
- 总结

- ② Shadow Outlier Execution (离群值处理)

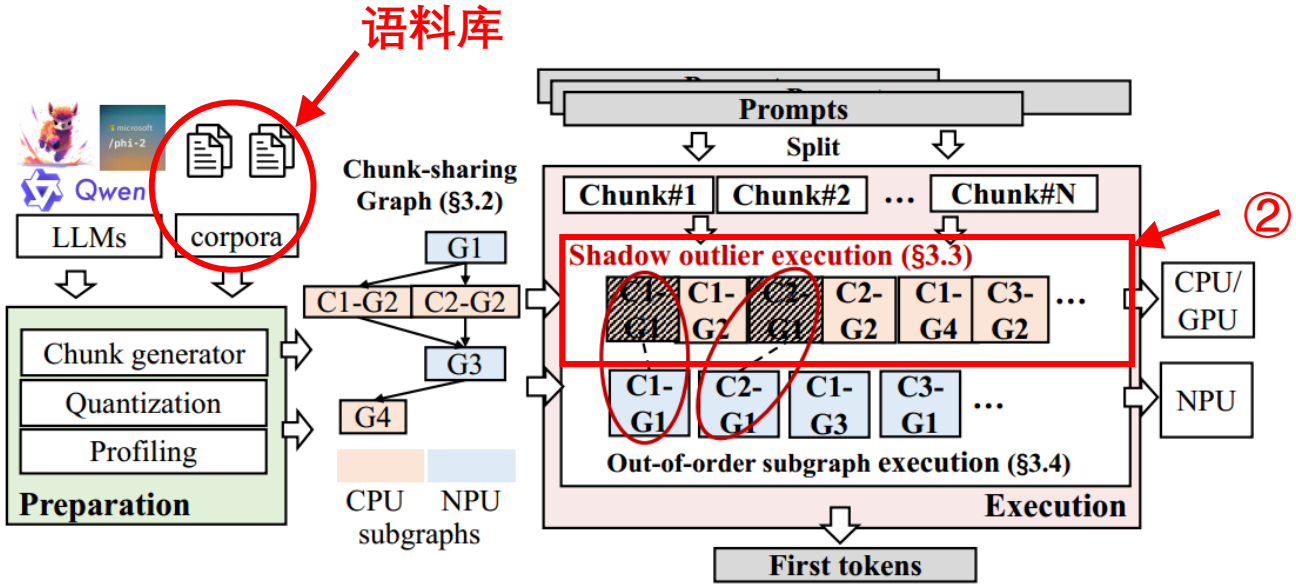
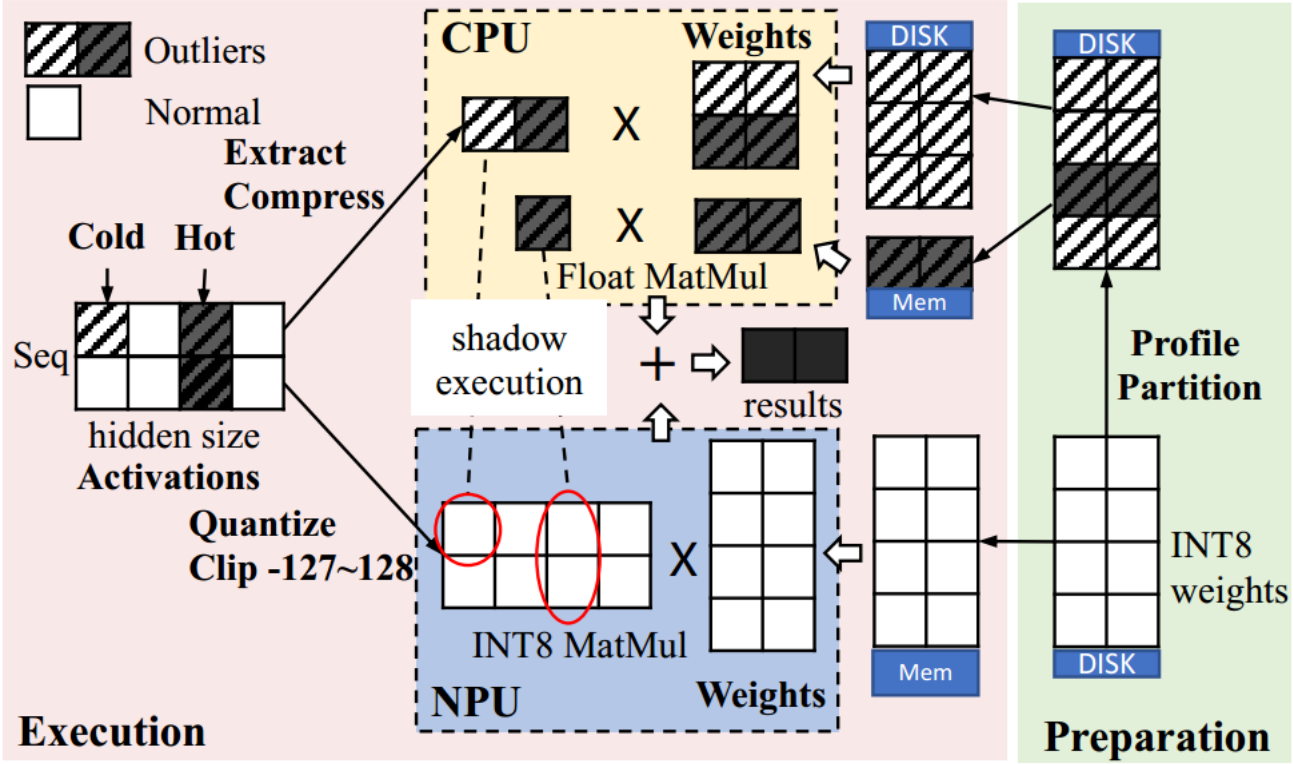


Figure 6. The workflow of `mllm-NPU`.

② Shadow Outlier Execution

- 研究背景
- 问题分析与发现
- 系统结构与原理
  - 系统概述
  - ① 计算图切分
  - ② 离群值处理
  - ③ 乱序调度
- 实验
- 总结

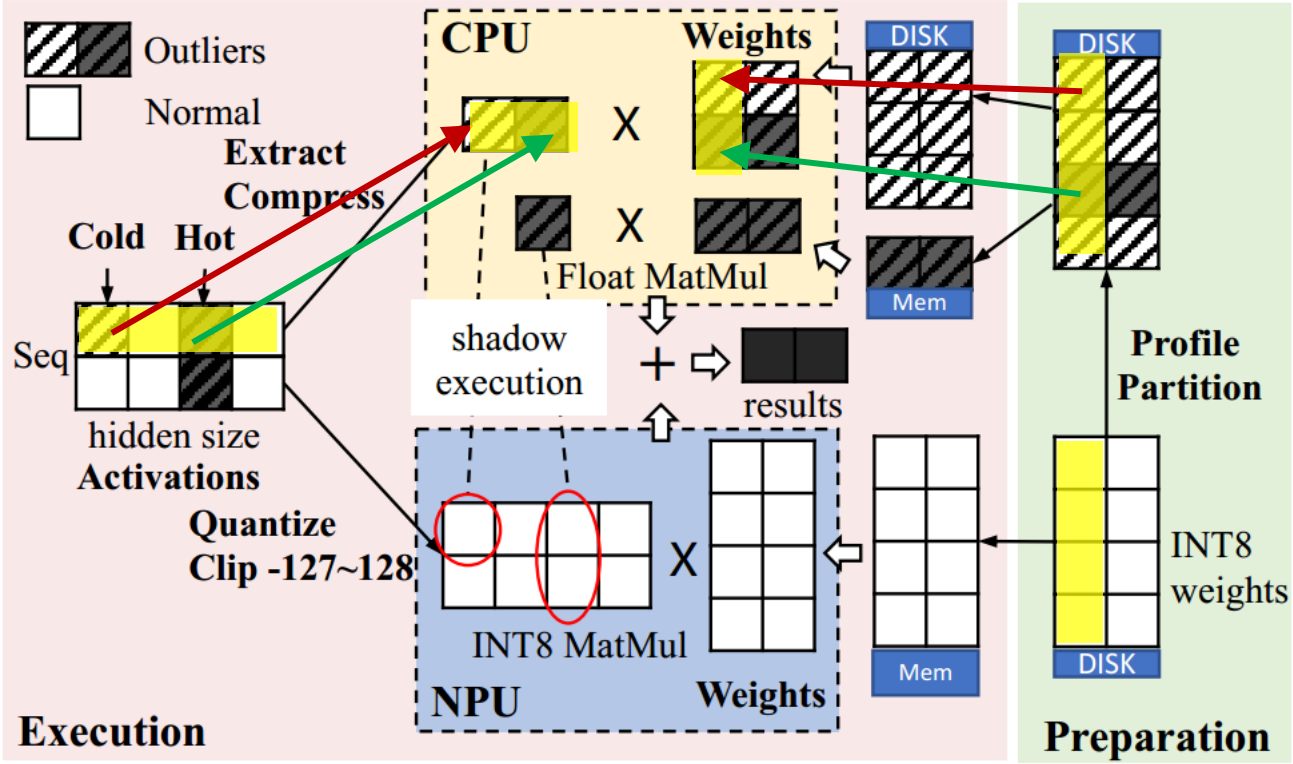
- ② Shadow Outlier Execution (离群值处理)
- 关键思路：将关键的outlier调度到CPU上以浮点数计算



② Shadow Outlier Execution

- 研究背景
- 问题分析与发现
- 系统结构与原理
  - 系统概述
  - ① 计算图切分
  - ② 离群值处理
  - ③ 乱序调度
- 实验
- 总结

- ② Shadow Outlier Execution (离群值处理)
- 关键思路：将关键的outlier调度到CPU上以浮点数计算



② Shadow Outlier Execution

- 研究背景
- 问题分析与发现
- 系统结构与原理
  - 系统概述
  - ① 计算图切分
  - ② 离群值处理
  - ③ 乱序调度
- 实验
- 总结

- outlier非常稀疏（约5-15个通道，仅占总通道的0.1%-0.3%）
- → CPU上计算时间小于NPU上执行时间

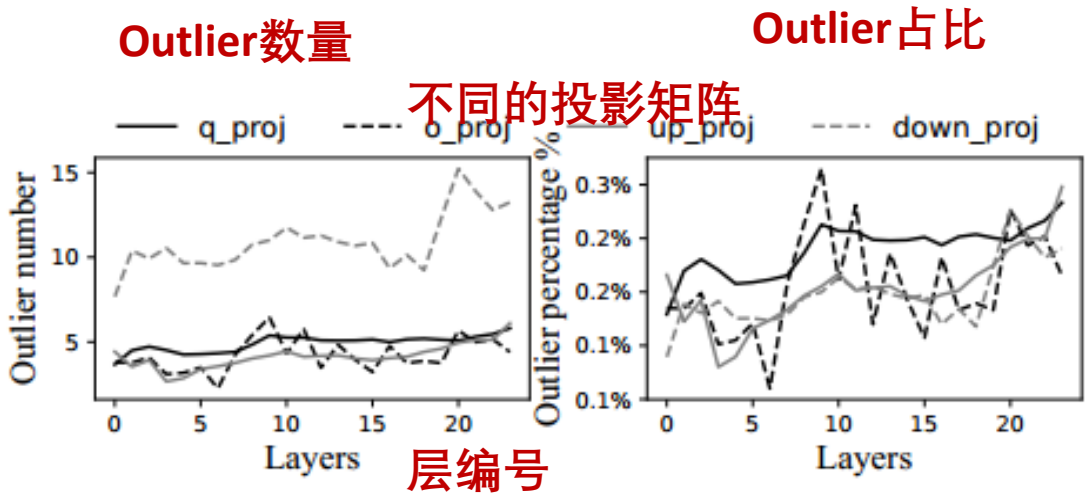
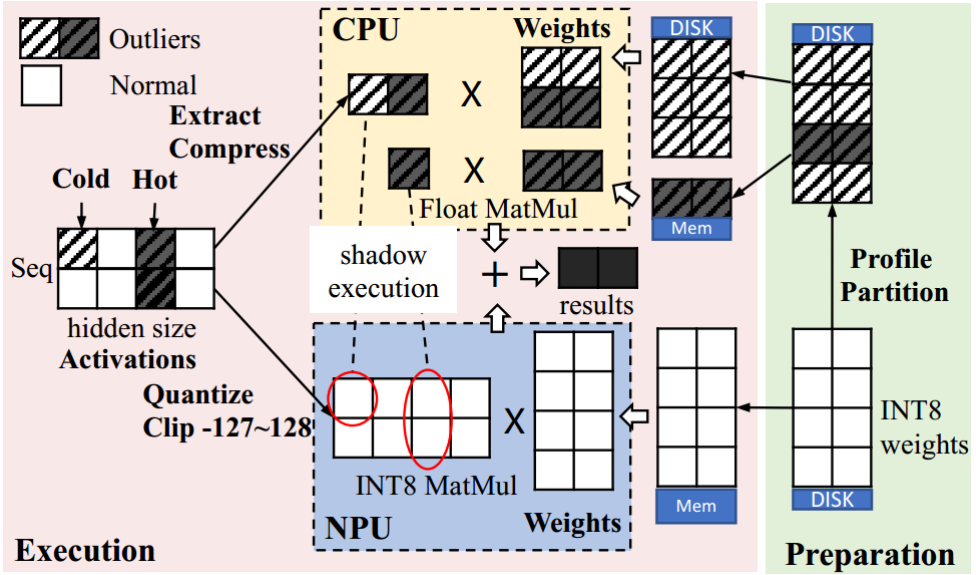


Figure 10. The average number and percentage of outlier channels per layer on Qwen1.5-1.8B model using the wikitext dataset under 2048 inference. **Less than 0.3% channels have outliers during one inference.**

② Shadow Outlier Execution

- 研究背景
- 问题分析与发现
- 系统结构与原理
  - 系统概述
  - ① 计算图切分
  - ② 离群值处理
  - ③ 乱序调度
- 实验
- 总结

- 改进的地方
  - ⅰ 两倍内存占用
  - ⅱ NPU-CPU同步时间开销
- 针对ⅰ，按照使用频率，对权重进行区分冷热
- 针对ⅱ，剪枝，尽量减少需要传输的张量的尺寸/数量



② Shadow Outlier Execution

- 研究背景
- 问题分析与发现
- 系统结构与原理
  - 系统概述
  - ① 计算图切分
  - ② 离群值处理
  - ③ 乱序调度
- 实验
- 总结

- 针对 i （两倍内存占用）
- 发现：大多数outlier往往出现在一小部分通道
- 不到3%的通道包括了超过80% outlier
- → CPU访问的内存中只需保留常用的那些通道的权重，其余权重在需要时从磁盘读取

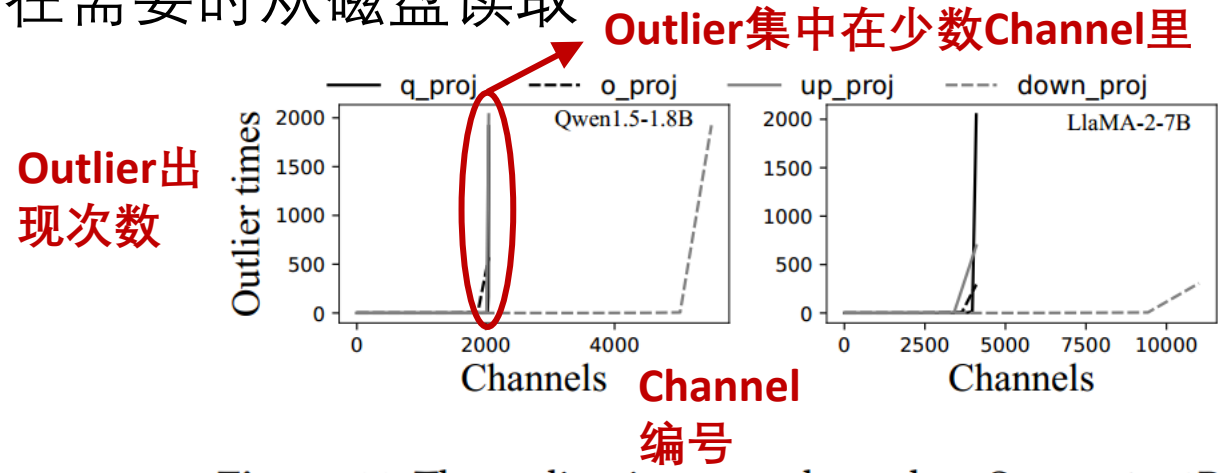


Figure 11. The outlier times per channel on Qwen1.5-1.8B model using the wikitext dataset under 2048 inference. **Less than 3% channels contribute over 80% outliers.**

② Shadow Outlier Execution

- 研究背景
- 问题分析与发现
- 系统结构与原理
  - 系统概述
  - ① 计算图切分
  - ② 离群值处理
  - ③ 乱序调度
- 实验
- 总结

- 针对 i （两倍内存占用）

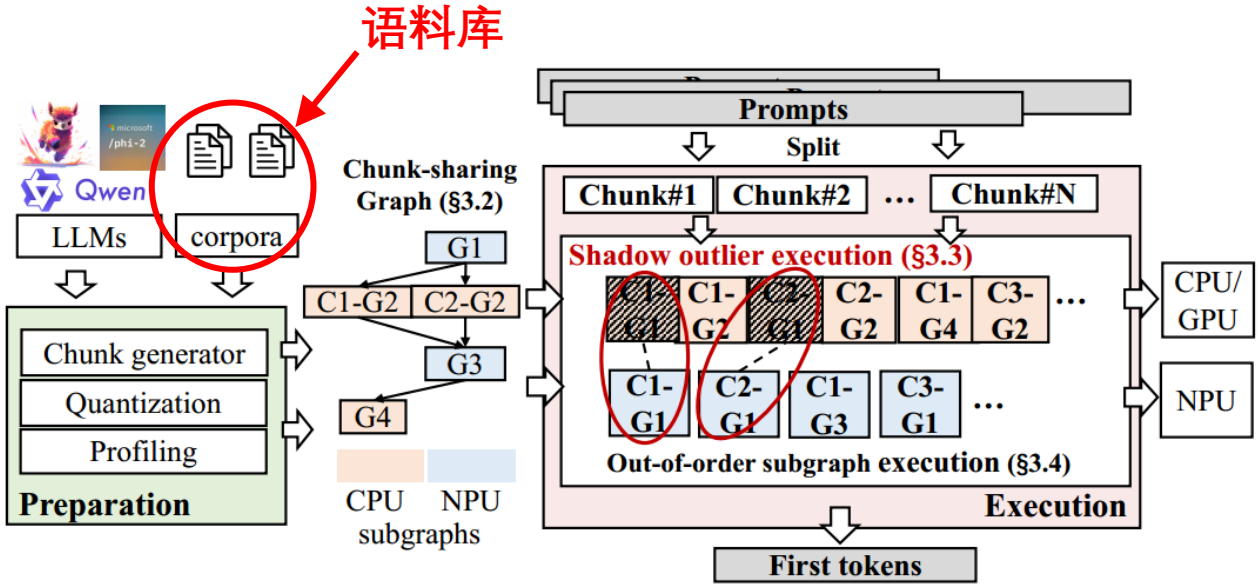
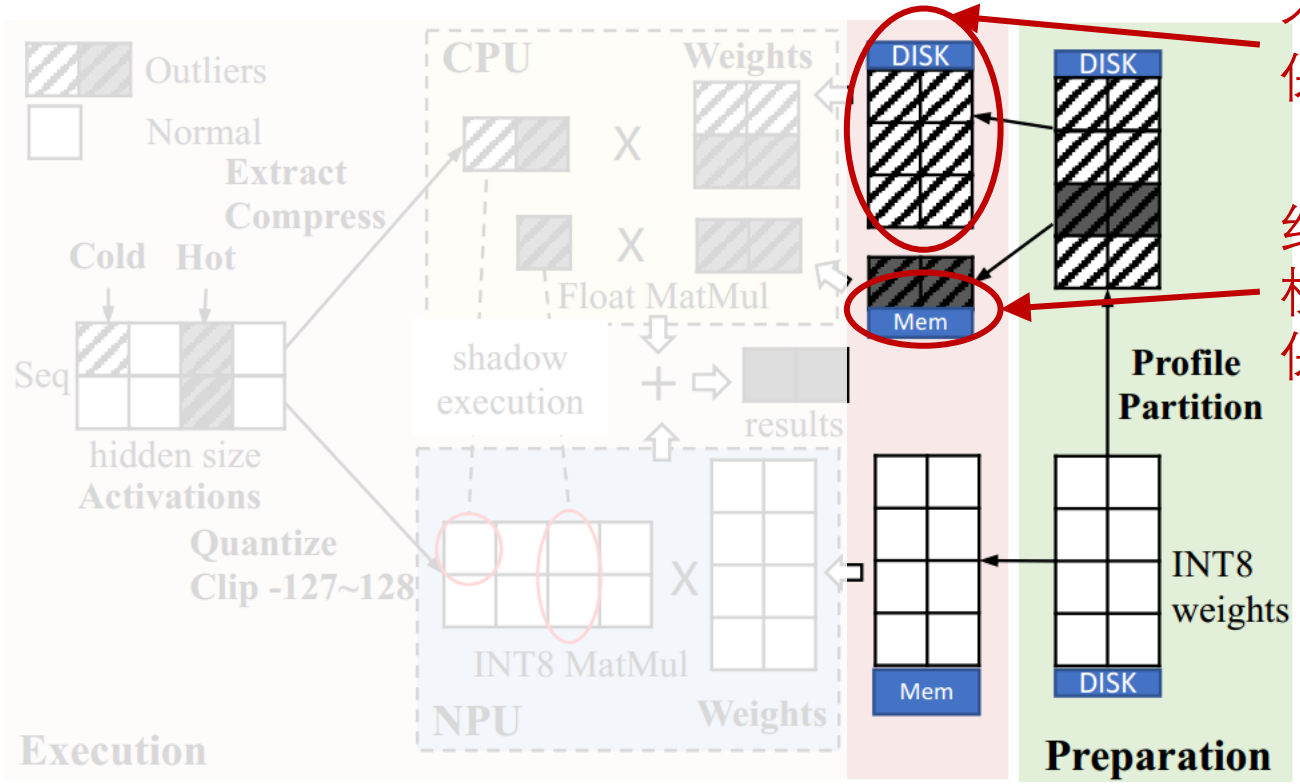


Figure 6. The workflow of `mllm-NPU`.

② Shadow Outlier Execution

- 研究背景
- 问题分析与发现
- 系统结构与原理
  - 系统概述
  - ① 计算图切分
  - ② 离群值处理
  - ③ 乱序调度
- 实验
- 总结

- 针对 i （两倍内存占用） 使用语料库，统计经常出现outlier的位置

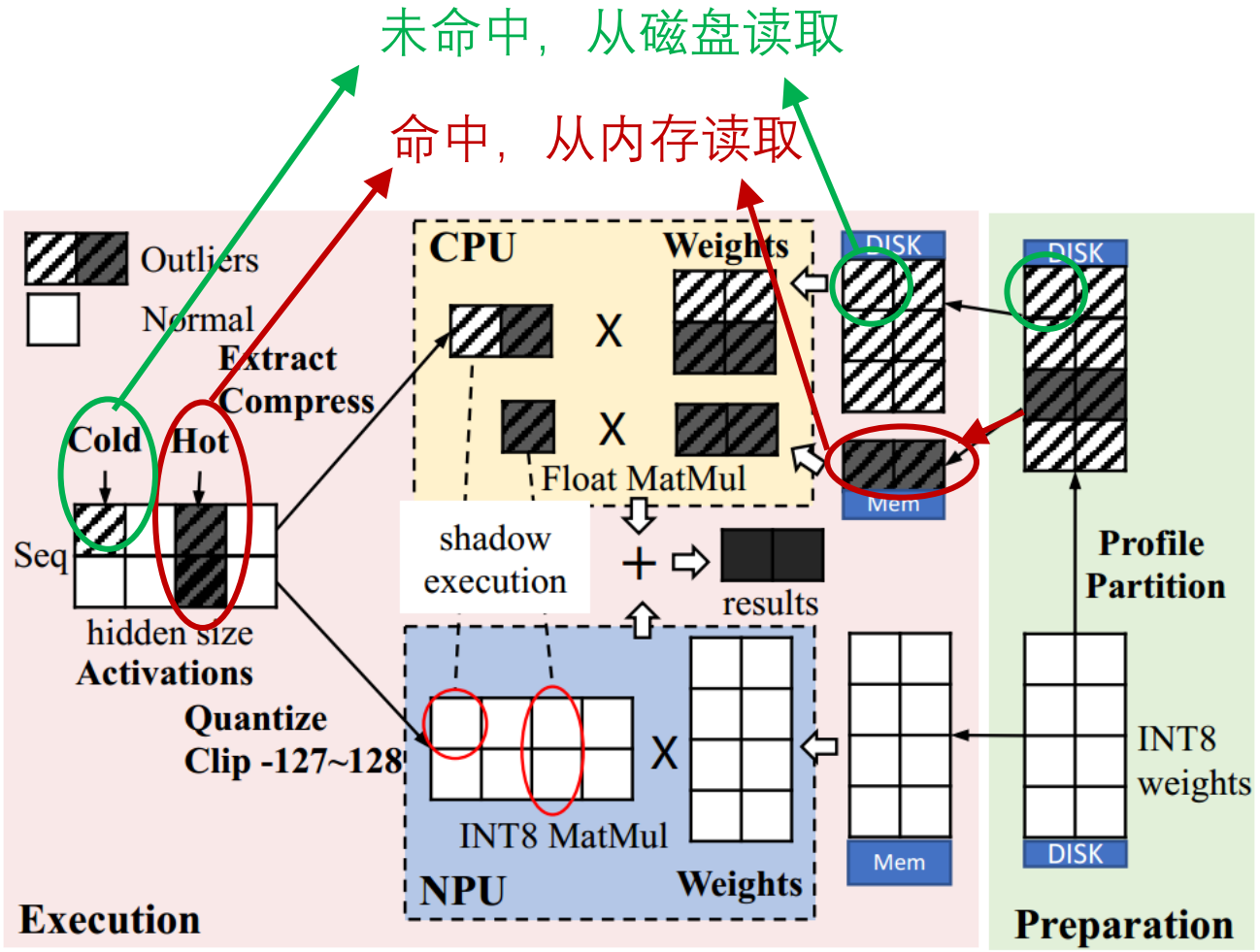


不常用的权重(Cold)
保存到磁盘

经常参与outlier计算的
权重(Hot)
保存到内存

② Shadow Outlier Execution

- 研究背景
- 问题分析与发现
- 系统结构与原理
  - 系统概述
  - ① 计算图切分
  - ② 离群值处理
  - ③ 乱序调度
- 实验
- 总结



② Shadow Outlier Execution

- 研究背景
- 问题分析与发现
- 系统结构与原理
  - 系统概述
  - ① 计算图切分
  - ② 离群值处理
  - ③ 乱序调度
- 实验
- 总结

- 针对 ii （NPU-CPU同步时间开销）
- 发现：大多数outlier可以在不影响准确性的情况下剪枝
- 定义：outlier重要性 = 张量中最大值/量化缩放系数(scale)
- 比值越大，表示激活值分布越分散，量化误差越显著
- 只针对重要性高的outlier进行调度

- 使用语料库，观察中间结果，修剪不重要的outlier
- 离线剖析来修剪前85%最不重要层的异常值，从而减小了CPU-NPU同步花费。

③ Out-of-order subgraph execution

- 研究背景
- 问题分析与发现
- 系统结构与原理
  - 系统概述
  - ① 计算图切分
  - ② 离群值处理
  - ③ 乱序调度
- 实验
- 总结

- ③ Out-of-order subgraph execution (乱序调度)

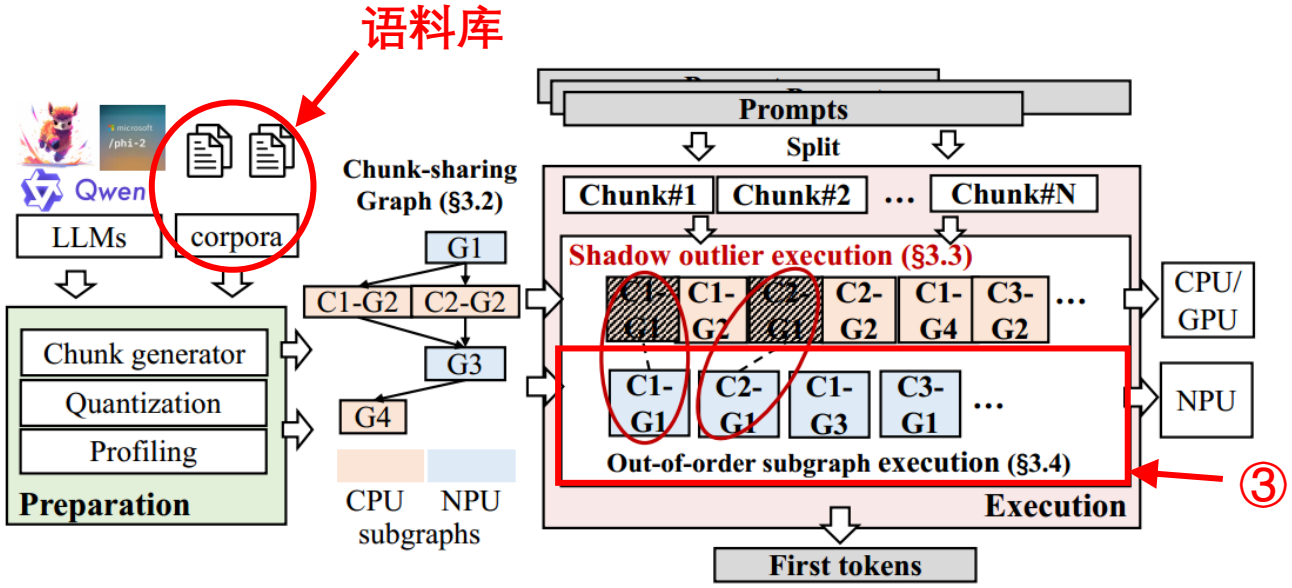
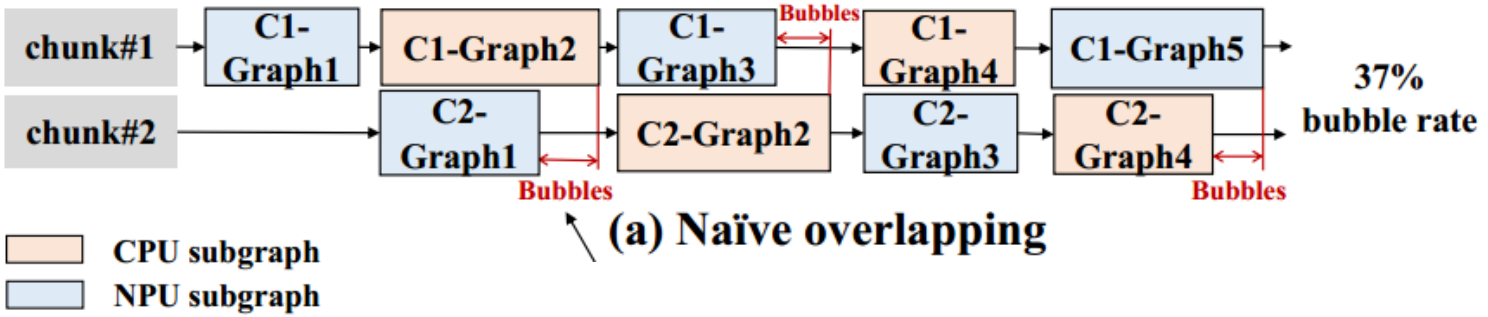


Figure 6. The workflow of `mllm-NPU`.

③ Out-of-order subgraph execution

- 研究背景
- 问题分析与发现
- 系统结构与原理
  - 系统概述
  - ① 计算图切分
  - ② 离群值处理
  - ③ 乱序调度
- 实验
- 总结

- ③ Out-of-order subgraph execution
- LLM量化算法不能完全消除浮点运算
- LayerNorm, Attention， 以及①中提取的部分需要放在CPU/GPU上计算
- 然而，简单地按次序重叠调度是低效的，导致很大的执行空泡（37%）



③ Out-of-order subgraph execution

- 研究背景
- 问题分析与发现
- 系统结构与原理
  - 系统概述
  - ① 计算图切分
  - ② 离群值处理
  - ③ 乱序调度
- 实验
- 总结

- 目标：找到一种调度顺序，使得总时间最短
- 约束：
  - 同时只能执行一个计算子图（移动端并行性/抢占能力较弱）
  - 计算顺序的前后依赖关系
- NP-Hard (可规约为旅行商问题)
  - → 在线启发式算法

③ Out-of-order subgraph execution

- 研究背景
- 问题分析与发现
- 系统结构与原理
  - 系统概述
  - ① 计算图切分
  - ② 离群值处理
  - ③ 乱序调度
- 实验
- 总结

- 准备阶段:
  - 使用语料库，记录各个计算图子图的执行时间、记录依赖关系
- 定义子图g的贡献度C (S为g执行完毕后, 可执行的子图集合):

$$C = \begin{cases} \sum T_i, \forall i \in S \text{ if } g \text{ is on the CPU/GPU} \\ -\sum T_i, \forall i \in S \text{ if } g \text{ is on the NPU} \end{cases}$$

子图C执行完后，可以执行的所有子图的时间总和

③ Out-of-order subgraph execution

- 研究背景
- 问题分析与发现
- 系统结构与原理
  - 系统概述
  - ① 计算图切分
  - ② 离群值处理
  - ③ 乱序调度
- 实验
- 总结

- 在线调度阶段:
  - Step1 计算所有当前可执行的子图的贡献度C
  - Step2 选取C最大的子图执行
  - Step3 执行完毕后，可执行子图增加
  - Step4 回到Step1 直到所有子图执行完毕

系统结构与原理

- 研究背景
- 问题分析与发现
- 系统结构与原理
  - 系统概述
  - ① 计算图切分
  - ② 离群值处理
  - ③ 乱序调度
- 实验
- 总结

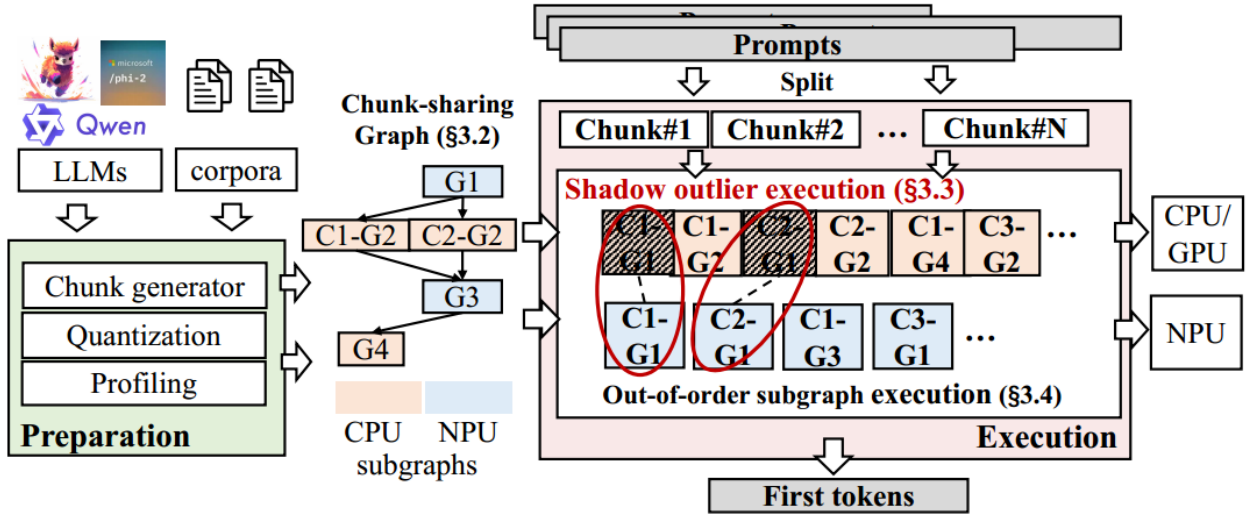


Figure 6. The workflow of `mllm-NPU`.

- 处理可变长度的prompt → ① Chunk-sharing graph execution (计算图切分)
- 激活值存在离群值(outlier) → ② Shadow Outlier Execution (离群值处理)
- 部分计算需要调度到CPU → ③ Out-of-order subgraph execution (乱序调度)

实验设置

- 研究背景
- 问题分析与发现
- 系统结构与原理
- 实验
  - 实验设置
  - 预填充性能
  - 端到端性能
  - 推理准确度
  - 内存消耗
  - 消融实验
- 总结

- 硬件
  - 小米14 Snapdragon 8 Gen 3（16GB内存）Android OS 13
  - 红米K60Pro Snapdragon 8 Gen 2（16GB内存）Android OS 13
- 模型
  - Qwen1.5-1.8B （阿里）
  - Phi2-2.7B （Microsoft）
  - Mistral-7B （Mistral AI）
  - Gemma-2B （Google）
  - LLaMA2-Chat-7B （Meta）

实验设置

- 研究背景
- 问题分析与发现
- 系统结构与原理
- 实验
  - 实验设置
  - 预填充性能
  - 端到端性能
  - 推理准确度
  - 内存消耗
  - 消融实验
- 总结

- 数据集
  - LAMBADA 长文本生成中的上下文理解能力 （2016）
  - HellaSwag 多选题，对情境和故事流的推理能力 （2019）
  - WinoGrande 选择题，常识推理 （2021）
  - Open-Book 常识性和科学性知识的开放域的问答 （EMNLP 2018）
  - MMLU 覆盖广泛领域的多任务理解 （ICLR 2021）

实验设置

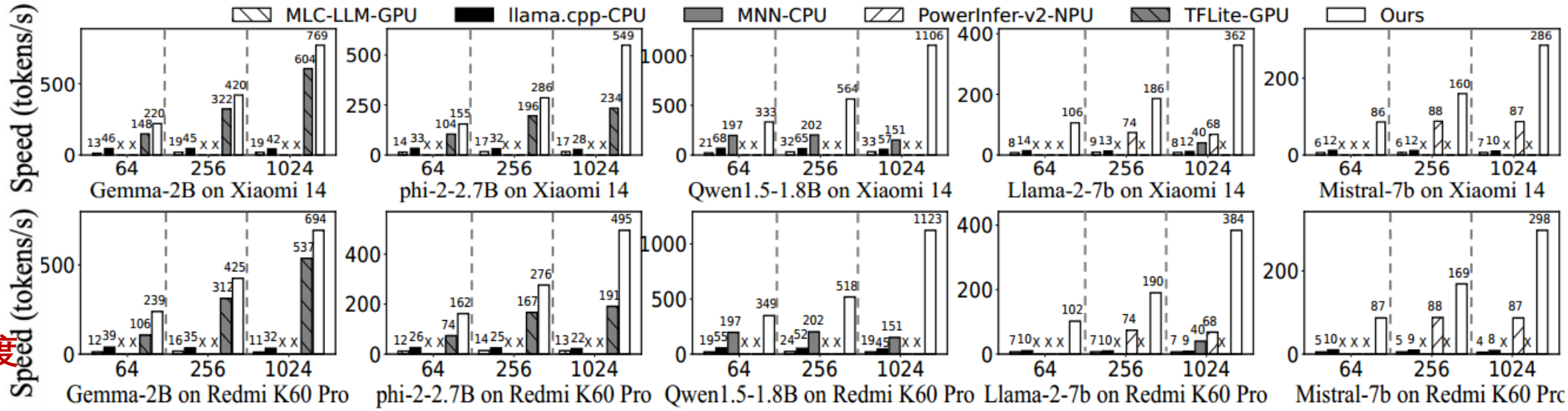
- 研究背景
- 问题分析与发现
- 系统结构与原理
- 实验
  - 实验设置
  - 预填充性能
  - 端到端性能
  - 推理准确度
  - 内存消耗
  - 消融实验
- 总结

- Baseline
  - 广泛使用的移动端LLM推理引擎：
    - TFLite（Google）　　MNN（阿里）
    - llama.cpp （开源社区）
  - 近期新提出的LLM推理方案
    - MLC-LLM（陈天奇, 2023）
    - PowerInfer-v2（上交IPADS, 2024.6）　（使用NPU）
- 实验项目：
  - 预填充性能、端到端性能、推理准确度、内存消耗、消融实验

实验

- 研究背景
- 问题分析与发现
- 系统结构与原理
- 实验
  - 实验设置
  - 预填充性能
  - 端到端性能
  - 推理准确度
  - 内存消耗
  - 消融实验
- 总结

- 预填充性能（速度）
  - 所有情形都有很好的性能表现
  - Prompt越长，性能优势越明显



预填充速度
token/s

Figure 14. Prefill speed under different prompt lengths on different devices (datasets: Longbench-2wiki-Multi-doc QA).

实验

- 研究背景
- 问题分析与发现
- 系统结构与原理
- 实验
  - 实验设置
  - 预填充性能
  - 端到端性能
  - 推理准确度
  - 内存消耗
  - 消融实验
- 总结

- 预填充性能（能耗）
  - NPU相比CPU/GPU具备能耗优势

能耗 J

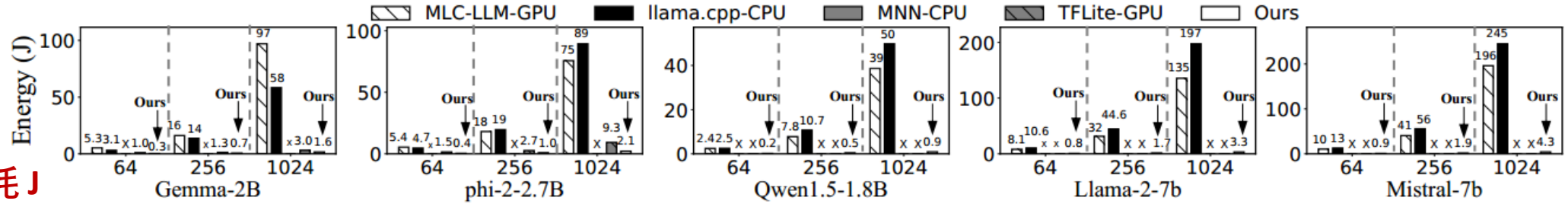


Figure 15. Energy consumption under different prompt lengths on Redmi K60 Pro (datasets: Longbench-2wiki-Multi-doc QA).

实验

- 研究背景
- 问题分析与发现
- 系统结构与原理
- 实验
  - 实验设置
  - 预填充性能
  - 端到端性能
  - 推理准确度
  - 内存消耗
  - 消融实验
- 总结

- 端到端性能（推理时延）
  - DroidTask 数据集的 UI 自动化任务
  - LongBench 数据集的上下文感知自动电子邮件回复

Table 4. End-to-end latency comparison across different frameworks using real mobile applications execution on Xiaomi 14.

LLM	Datasets	MLC	LCPP	MNN	PI	TFLite	Ours	Speedup	Datasets	MLC	LCPP	MNN	PI	TFLite	Ours	Speedup
Qwen1.5-1.8B	Longbench: 2wiki -Multi-doc QA (prompt length: 1500 tokens)	45.6	26.7	10.6	-	-	1.7	**6.2-26.8×**	Longbench: TriviaQA (prompt length: 1500 tokens)	46.0	27.0	11.2	-	-	2.0	**5.6-23.0×**
Gemma-2B		78.4	34.6	-	-	2.6	1.9	**1.4-41.3×**		81.8	36.2	-	-	2.8	2.2	**1.3-37.2×**
Phi-2-2.7B		87.0	53.3	13.0	-	6.3	3.1	**2.0-28.1×**		91.4	56.3	14.7	-	6.8	3.6	**1.9-25.4×**
LlaMA-2-7B		184.7	146.0	22.4	19.8	-	5.3	**3.7-34.8×**		197.3	156.2	23.8	21.8	-	6.2	**3.5-31.8×**
Mistral-7b		254.2	200.2		20.0	-	5.5	**3.6-46.2×**		266.2	210.0	-	21.5	-	6.4	**3.4-41.6×**
Geo-mean (speedup)		**34.7×**	**21.8×**	**4.8×**	**3.7×**	**1.7×**	-			**31.0×**	**19.6×**	**4.4×**	**3.4×**	**1.6×**	-	
LLM	**Datasets**	**MLC**	**LCPP**	**MNN**	**PI**	**TFLite**	**Ours**	**Speedup**	**Datasets**	**MLC**	**LCPP**	**MNN**	**PI**	**TFLite**	**Ours**	**Speedup**
Qwen1.5-1.8B	DroidTask: clock (prompt length: 800 tokens))	21.0	10.4	3.9	-	-	1.4	**2.8-15.0×**	DroidTask: applauncher (prompt length: 600 tokens)	16.2	8.1	3.1	-	-	1.1	**2.8-14.7×**
Gemma-2B		39.4	16.5	-	-	2.5	1.2	**2.1-32.8×**		29.4	12.3	-	-	1.9	0.9	**2.1-32.7×**
Phi-2-2.7B		46.6	25.0	7.4	-	4.2	3.1	**1.4-15.0×**		35.4	19.0	5.9	-	3.2	2.4	**1.3-14.8×**
LlaMA-2-7B		87.7	60.4	10.6	11.1	-	4.8	**2.2-18.3×**		63.7	43.9	7.7	8.2	-	3.6	**2.1-17.7×**
Mistral-7b		122.3	68.6	-	12.0	-	4.9	**2.4-25.0×**		90.1	50.6	-	8.9	-	3.8	**2.3-23.7×**
Geo-mean (speedup)		**20.2×**	**10.8×**	**2.4×**	**2.4×**	**1.7×**	-			**19.7×**	**10.5×**	**2.5×**	**2.3×**	**1.7×**	-	

**LCPP* and *PI* in the first row represent llama.cpp and PowerInfer-V2, respectively.*

实验

- 研究背景
- 问题分析与发现
- 系统结构与原理
- 实验
  - 实验设置
  - 预填充性能
  - 端到端性能
  - 推理准确度
  - 内存消耗
  - 消融实验
- 总结

- 所有数据集/子任务上推理时延最低
- 相较于TFLite-GPU加速效果较弱
  - mllm-NPU在decode阶段使用CPU, TFLite-GPU全程使用GPU
- DroidTask数据集prompt更短， mllm-NPU提升略小

Table 4. End-to-end latency comparison across different frameworks using real mobile applications execution on Xiaomi 14.

LLM	Datasets	MLC	LCPP	MNN	PI	TFLite	Ours	Speedup	Datasets	MLC	LCPP	MNN	PI	TFLite	Ours	Speedup
Qwen1.5-1.8B	Longbench: 2wiki -Multi-doc QA (prompt length: 1500 tokens)	45.6	26.7	10.6	-	-	1.7	**6.2-26.8×**	Longbench: TriviaQA (prompt length: 1500 tokens)	46.0	27.0	11.2	-	-	2.0	**5.6-23.0×**
Gemma-2B		78.4	34.6	-	-	2.6	1.9	**1.4-41.3×**		81.8	36.2	-	-	2.8	2.2	**1.3-37.2×**
Phi-2-2.7B		87.0	53.3	13.0	-	6.3	3.1	**2.0-28.1×**		91.4	56.3	14.7	-	6.8	3.6	**1.9-25.4×**
LlaMA-2-7B		184.7	146.0	22.4	19.8	-	5.3	**3.7-34.8×**		197.3	156.2	23.8	21.8	-	6.2	**3.5-31.8×**
Mistral-7b		254.2	200.2		20.0	-	5.5	**3.6-46.2×**		266.2	210.0	-	21.5	-	6.4	**3.4-41.6×**
Geo-mean (speedup)		**34.7×**	**21.8×**	**4.8×**	**3.7×**	**1.7×**	-			**31.0×**	**19.6×**	**4.4×**	**3.4×**	**1.6×**	-	
LLM	**Datasets**	**MLC**	**LCPP**	**MNN**	**PI**	**TFLite**	**Ours**	**Speedup**	**Datasets**	**MLC**	**LCPP**	**MNN**	**PI**	**TFLite**	**Ours**	**Speedup**
Qwen1.5-1.8B	DroidTask: clock (prompt length: 800 tokens))	21.0	10.4	3.9	-	-	1.4	**2.8-15.0×**	DroidTask: applauncher (prompt length: 600 tokens)	16.2	8.1	3.1	-	-	1.1	**2.8-14.7×**
Gemma-2B		39.4	16.5	-	-	2.5	1.2	**2.1-32.8×**		29.4	12.3	-	-	1.9	0.9	**2.1-32.7×**
Phi-2-2.7B		46.6	25.0	7.4	-	4.2	3.1	**1.4-15.0×**		35.4	19.0	5.9	-	3.2	2.4	**1.3-14.8×**
LlaMA-2-7B		87.7	60.4	10.6	11.1	-	4.8	**2.2-18.3×**		63.7	43.9	7.7	8.2	-	3.6	**2.1-17.7×**
Mistral-7b		122.3	68.6	-	12.0		4.9	**2.4-25.0×**		90.1	50.6	-	8.9	-	3.8	**2.3-23.7×**
Geo-mean (speedup)		**20.2×**	**10.8×**	**2.4×**	**2.4×**	**1.7×**	-			**19.7×**	**10.5×**	**2.5×**	**2.3×**	**1.7×**	-	

**LCPP* and *PI* in the first row represent llama.cpp and PowerInfer-V2, respectively.

实验

- 研究背景
- 问题分析与发现
- 系统结构与原理
- 实验
  - 实验设置
  - 预填充性能
  - 端到端性能
  - 推理准确度
  - 内存消耗
  - 消融实验
- 总结

- 推理准确度 （相比FP16的损失）

Smooth Quant
将outlier平滑到范围内
PerTensor量化

Outlier保持浮点数

PerGroup
量化

Table 5. LLM capability accuracy on `mllm-NPU` and baselines. "SQ": SmoothQuant; "Int8()": LLM.Int8(); "Degrad.": accuracy degradation compared to FP16.

LAMBADA	FP16	SQ	INT8()	K-Quant	Ours	Ours Degrad.
Qwen1.5-1.8B	71.1%	65.6%	71.0%	62.7%	71.7%	+0.6%
Gemma2-2B	59.6%	45.8%	59.2%	56.9%	59.4%	-0.2%
Phi-2-2.7B	72.2%	66.1%	71.7%	59.3%	67.5%	-4.7%
LlaMA-2-7B	87.5%	71.9%	88.0%	15.6%	86.3%	-1.2%
Mistral-7b	84.8%	51.2%	85.3%	23.9%	84.1%	-0.7%
Avg. Degrad.	-	**-14.9%**	**0%**	**-31.3%**	**-1.2%**	

HellaSwag	FP16	SQ	INT8()	K-Quant	Ours	Ours Degrad.
Qwen1.5-1.8B	43.8%	40.9%	43.5%	44.3%	43.8%	0%
Gemma2-2B	46.5%	43.8%	46.1%	45.4%	47.3%	+0.8%
Phi-2-2.7B	48.2%	46.2%	47.7%	47.6%	46.9%	-1.3%
LlaMA-2-7B	52.8%	44.4%	53.1%	50.5%	53.5%	+0.7%
Mistral-7b	57.4%	44.9%	57.9%	57.0%	57.0%	-0.4%
Avg. Degrad.	-	**-5.7%**	**-0.1%**	**-0.8%**	**-0.0%**	

WinoGrande	FP16	SQ	INT8()	K-Quant	ours	Ours Degrad.
Qwen1.5-1.8B	58.3%	51.0%	58.2%	59.0%	59.3%	+1.0%
Gemma2-2B	58.3%	54.8%	59.0%	58.5%	59.5%	+1.2%
Phi-2-2.7B	72.2%	68.9%	72.4%	72.5%	70.2%	-2.0%
LlaMA-2-7B	65.2%	56.9%	66.2%	67.4%	65.1%	-0.1%
Mistral-7b	73.5%	59.1%	73.3%	73.5%	73.1%	-0.4%
Avg. Degrad.	-	**-7.4%**	**+0.3%**	**+0.7%**	**-0.1%**	

OpenBookQA	FP16	SQ	INT8()	K-Quant	ours	Ours Degrad.
Qwen1.5-1.8B	28.8%	23.0%	28.5%	28.0%	26.6%	-2.2%
Gemma2-2B	33.7%	28.0%	34.2%	33.0%	38.4%	+4.7%
Phi-2-2.7B	41.0%	35.9%	40.2%	39.5%	37.7%	-3.3%
LlaMA-2-7B	32.7%	25.0%	32.0%	31.5%	31.1%	-1.6%
Mistral-7b	39.4%	25.6%	39.3%	37.9%	39.3%	-0.1%
Avg. Degrad.	-	**-7.6%**	**-0.3%**	**-1.1%**	**-0.5%**	

MMLU	FP16	SQ	INT8()	K-Quant	ours	Ours Degrad.
Qwen1.5-1.8B	29.7%	27.9%	29.1%	29.8%	30.8%	+1.1%
Gemma2-2B	35.7%	32.1%	35.1%	35.1%	36.4%	+0.7%
Phi-2-2.7B	35.4%	35.3%	35.6%	35.7%	36.7%	+1.3%
LlaMA-2-7B	37.8%	29.2%	38.1%	34.4%	36.9%	-0.9%
Mistral-7b	42.1%	30.9%	41.4%	42.3%	41.0%	-1.1%
Avg. Degrad.	-	**-5.1%**	**-0.3%**	**-0.7%**	**+0.2%**	

实验

- 研究背景
- 问题分析与发现
- 系统结构与原理
- 实验
  - 实验设置
  - 预填充性能
  - 端到端性能
  - 推理准确度
  - 内存消耗
  - 消融实验
- 总结

- 推理准确度 （相比FP16的损失）

Smooth Quant
将outlier平滑到范围内
PerTensor量化

Outlier保持浮点数

PerGroup
量化

Table 5. LLM capability accuracy on `mllm-NPU` and baselines. "SQ": SmoothQuant; "Int8()": LLM.Int8(); "Degrad.": accuracy degradation compared to FP16.

LAMBADA	FP16	SQ	INT8()	K-Quant	Ours	Ours Degrad.
Qwen1.5-1.8B	71.1%	65.6%	71.0%	62.7%	71.7%	+0.6%
Gemma2-2B	59.6%	45.8%	59.2%	56.9%	59.4%	-0.2%
Phi-2-2.7B	72.2%	66.1%	71.7%	59.3%	67.5%	-4.7%
LlaMA-2-7B	87.5%	71.9%	88.0%	15.6%	86.3%	-1.2%
Mistral-7b	84.8%	51.2%	85.3%	23.9%	84.1%	-0.7%
Avg. Degrad.	-	**-14.9%**	**0%**	**-31.3%**	**-1.2%**	

HellaSwag	FP16	SQ	INT8()	K-Quant	Ours	Ours Degrad.
Qwen1.5-1.8B	43.8%	40.9%	43.5%	44.3%	43.8%	0%
Gemma2-2B	46.5%	43.8%	46.1%	45.4%	47.3%	+0.8%
Phi-2-2.7B	48.2%	46.2%	47.7%	47.6%	46.9%	-1.3%
LlaMA-2-7B	52.8%	44.4%	53.1%	50.5%	53.5%	+0.7%
Mistral-7b	57.4%	44.9%	57.9%	57.0%	57.0%	-0.4%
Avg. Degrad.	-	**-5.7%**	**-0.1%**	**-0.8%**	**-0.0%**	

WinoGrande	FP16	SQ	INT8()	K-Quant	ours	Ours Degrad.
Qwen1.5-1.8B	58.3%	51.0%	58.2%	59.0%	59.3%	+1.0%
Gemma2-2B	58.3%	54.8%	59.0%	58.5%	59.5%	+1.2%
Phi-2-2.7B	72.2%	68.9%	72.4%	72.5%	70.2%	-2.0%
LlaMA-2-7B	65.2%	56.9%	66.2%	67.4%	65.1%	-0.1%
Mistral-7b	73.5%	59.1%	73.3%	73.5%	73.1%	-0.4%
Avg. Degrad.	-	**-7.4%**	**+0.3%**	**+0.7%**	**-0.1%**	

OpenBookQA	FP16	SQ	INT8()	K-Quant	ours	Ours Degrad.
Qwen1.5-1.8B	28.8%	23.0%	28.5%	28.0%	26.6%	-2.2%
Gemma2-2B	33.7%	28.0%	34.2%	33.0%	38.4%	+4.7%
Phi-2-2.7B	41.0%	35.9%	40.2%	39.5%	37.7%	-3.3%
LlaMA-2-7B	32.7%	25.0%	32.0%	31.5%	31.1%	-1.6%
Mistral-7b	39.4%	25.6%	39.3%	37.9%	39.3%	-0.1%
Avg. Degrad.	-	**-7.6%**	**-0.3%**	**-1.1%**	**-0.5%**	

MMLU	FP16	SQ	INT8()	K-Quant	ours	Ours Degrad.
Qwen1.5-1.8B	29.7%	27.9%	29.1%	29.8%	30.8%	+1.1%
Gemma2-2B	35.7%	32.1%	35.1%	35.1%	36.4%	+0.7%
Phi-2-2.7B	35.4%	35.3%	35.6%	35.7%	36.7%	+1.3%
LlaMA-2-7B	37.8%	29.2%	38.1%	34.4%	36.9%	-0.9%
Mistral-7b	42.1%	30.9%	41.4%	42.3%	41.0%	-1.1%
Avg. Degrad.	-	**-5.1%**	**-0.3%**	**-0.7%**	**+0.2%**	

实验

- 研究背景
- 问题分析与发现
- 系统结构与原理
- 实验
  - 实验设置
  - 预填充性能
  - 端到端性能
  - 推理准确度
  - 内存消耗
  - 消融实验
- 总结

- 内存消耗
  - 并不会带来非常多的额外内存消耗
  - mllm-NPU比llama.cpp和TFLite多消耗1.32倍的内存

内存消耗
GB

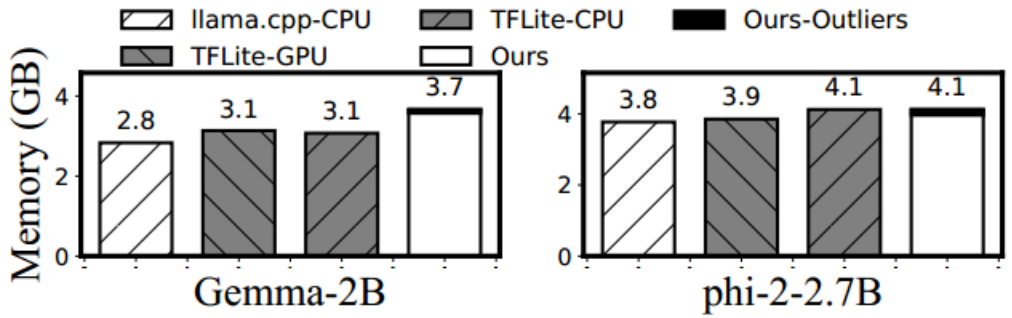


Figure 16. Memory consumption of different baselines (prompt length=512).

实验

- 研究背景
- 问题分析与发现
- 系统结构与原理
- 实验
  - 实验设置
  - 预填充性能
  - 端到端性能
  - 推理准确度
  - 内存消耗
  - 消融实验
- 总结

- 直接将计算卸载到NPU导致性能下降
  （对于变长prompt需要重构和重新优化计算图）
- 步骤①（Chunk）减少图构建和优化延迟 Perfill速度1.46 - 5.09x ↑
- 步骤②（Outlier）Perfill延迟3.91 - 8.68x ↓
- 步骤③（OOE） Perfill延迟18%-44% ↓

预填充速度
token/s

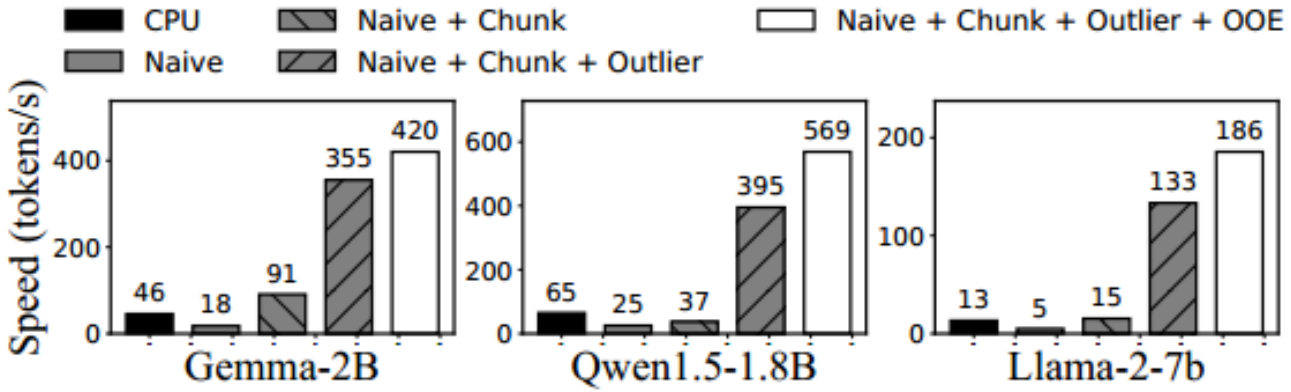


Figure 17. Ablation study of mllm-NPU (prompt length=512).

总结

- 研究背景
- 问题分析与发现
- 系统结构与原理
- 实验
- 总结

- Mllm-NPU:
  - 在移动设备上使用NPU来加速LLM推理的Prefill阶段
- 关键技术:
  - 处理可变长度的prompt → ① Chunk-sharing graph execution (计算图切分)
  - 激活值存在离群值(outlier) → ② Shadow Outlier Execution (离群值处理)
  - 部分计算需要调度到CPU → ③ Out-of-order subgraph execution (乱序调度)
- 实验:
  - 速度、精度有明显优势；因使用NPU，能耗有明显优势
  - 额外内存消耗可接受
  - Decode阶段 后端可替换, 进一步提升性能

Idea

- 基于此：
  - 单任务→多任务，高优先级任务抢占
  - 若继续降低量化bit数，outlier的数量是否会增加，增加后如何处理
- 与此类似：
  - 共享计算子图(①)的思路可以使用在其他领域
    - 比如图像领域不同的任务(分类/识别/语义分割) 共享backbone的计算图
- 由此需要：
  - 更好的Chunk划分方法
  - 计算图缓存与复用
- 实际问题驱动：
  - 量化(②)的部分， 将离群值提取后在CPU上计算的方法可以用在项目里
  - 调度(③)的部分，当NPU在片外的时候，权重传输的开销

请各位老师和同学批评指正

汇报人：姚昌硕

2024年11月15日

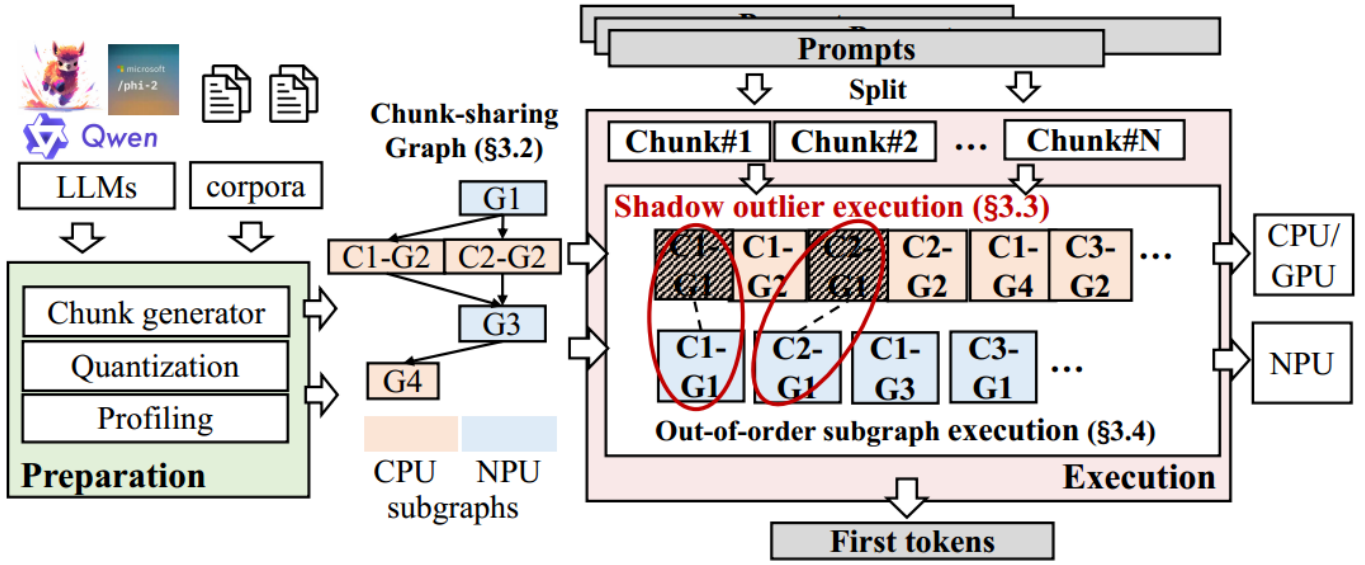


Figure 6. The workflow of `mllm-NPU`.