

顶点课程团队报告

第四组

22230608 范一泽 22230610 曹晨旭 22230616 黄勇鑫

日期：2026 年 1 月 11 日

摘 要

本文为顶点课程结题报告，主要介绍团队在顶点课程项目实践中的探索与成果。摘要部分撰写思路是：在什么背景下，面向什么问题和需求，进行了哪些探索、研究与开发等，最终解决了什么问题，实现了什么效果，具备什么意义。同时还可附上项目开源链接、或者成果公开展示页面之类的东西。

摘要中至少要明确出来如下几个问题：1、部署平台（手机 / GPU / Jetson / CPU）；2、模型规模；3、核心优化手段；4、关键性能指标（至少一个）；5、实际效果。

关键词：大语言模型，端侧部署，推理优化

1 研究背景与意义

1.1 研究背景

近年来，大语言模型（Large Language Models, LLMs）在自然语言理解与生成任务中展现出卓越性能，在个人助理、智能问答、医疗咨询等场景中具有广阔的应用前景。然而，当前主流的大语言模型推理方式高度依赖云端计算资源，存在隐私风险、网络依赖性强以及推理成本较高等问题。

在此背景下，将大语言模型部署至移动端等资源受限设备上开展端侧推理研究，成为推动大模型实际落地的重要方向。端侧推理能够在不依赖网络的情况下完成模型推断，有助于提升系统可用性与安全性，同时降低云端计算压力。

1.2 研究意义

端侧大语言模型部署与推理优化具有以下重要意义：

- **隐私与数据安全需求：**端侧推理避免用户数据上传云端，适用于医疗、政务等高隐私敏感场景，有效降低数据泄露风险。
- **低延迟与高可用性：**端侧推理不依赖网络环境，在弱网或离线条件下仍可稳定运行，显著提升交互响应速度。
- **成本与规模化部署优势：**减少云端推理请求，降低算力成本，提高模型在大规模终端设备上的可扩展性。

1.3 核心技术瓶颈与研究重点

尽管端侧推理具备显著优势，但仍面临诸多技术挑战：

- **算力受限**：移动端计算能力远低于服务器，难以直接承载标准精度的大参数模型。
- **内存与显存约束**：模型参数、KV Cache 以及中间激活占用显著，容易引发内存溢出或频繁内存交换。
- **能耗与发热问题**：长时间高负载推理会导致功耗升高与热降频，影响系统稳定性与用户体验。

因此，本项目聚焦于在端侧算力与内存资源受限条件下，对自回归大语言模型进行高效部署与推理优化。

2 研究内容与思路

2.1 研究目标

本项目旨在面向 Android 平台，探索并实现自回归大语言模型的端侧部署与推理优化方法。在保证模型输出质量的前提下，降低推理过程中的算力与内存开销，并完成可交互的移动端应用系统。

2.2 研究内容

围绕上述研究目标，项目主要包括以下研究内容：

- 在推理与量化层面对大语言模型进行部署优化；
- 采用指令微调方法，构建面向医疗知识问答场景的大语言模型；
- 实现 Android 端手机 APP，在图形化界面中完成与模型的交互；
- 结合工程实践过程，进行一定的科研探索与方法验证。

2.3 技术路线与整体 Pipeline

项目整体技术路线主要包括三条主线：

- **推理优化路线**：StreamingLLM → 引入 Summary 机制 → SnapKV-lite → SSSKV-lite 推理框架；
- **模型微调路线**：基于 MedChatZH 数据集进行中文医疗指令微调；
- **系统实现路线**：完成模型加载、端侧推理与移动端应用的整体集成。

整体 Pipeline 如图1所示。

3 技术实现方案

3.1 KV Cache 优化方法调研

针对端侧推理中 KV Cache 占用过大的问题，已有多种研究提出了优化方法：

- **StreamingLLM[4]**：结合 Attention Sink 与 Sliding Window 机制，保留初始关键 token；

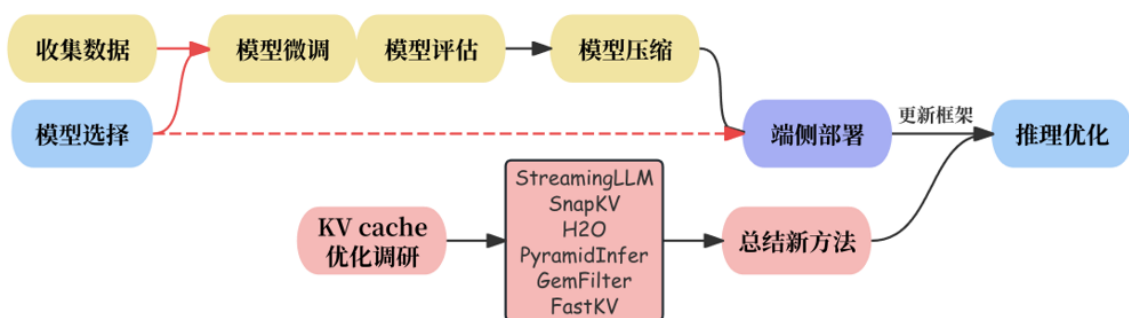


图 1: 项目 Pipeline

- **SnapKV**[2]: 在 Observation Window 中统计注意力权重, 选取 Top-k 重要前缀 token;
- **H2O**[6]: 将 KV Cache 淘汰过程建模为优化问题, 贪心保留累计注意力得分最高的 token。
- **PyramidInfer**[5]: 采用逐层递减的 token 保留策略, 每一层仅保留 N_l 个重要 tokens, 使后续层的 KV 计算规模不断缩小。
- **GemFilter**[3]: 在 prefill 之前单独利用模型前 r 层进行一次筛选, 通过第 r 层最后一个 token 的注意力分布选出 top-k tokens, 仅让其参与完整推理。
- **FastKV**[1]: 进一步考虑不同层注意力分布差异, 在 prefill 阶段设置 TSP 中间层: TSP 之前保持全量信息计算, 避免过早丢弃浅层信息; 在 TSP 层依据累计注意力筛选关键 tokens, 并将其作为后续层的初始 cache, 后续各层再按自身保留率独立更新, 在效率与信息保留之间取得更好的平衡。

上述方法为本项目的推理优化设计提供了重要参考。

3.2 系统总体设计

系统整体架构由模型推理模块与移动端应用模块组成:

- **模型选择**: 采用 Qwen 与 Llama 系列大语言模型;
- **本地推理硬件**: 支持 GPU 与手机端 CPU 推理;
- **应用形式**: 通过 Android APP 提供用户交互界面。

3.3 SSSKV-lite 推理框架实现

本项目提出并实现了 StreamingSummarySnapKV-lite (SSSKV-lite) 推理框架, 核心设计如下。

3.3.1 三段式连续上下文的实现

在实现过程中, 重点保证主推理 context 中的 token 始终以连续块形式存储, 从而使 CPU 上的注意力计算保持顺序内存访问, 提高 cache 命中率。

- **Prefix**: 在初始化阶段写入系统提示, 后续推理过程中保持不变;
- **Summary**: 由摘要生成链路维护, 在更新后触发主 KV cache 重建;
- **Recent**: 在每轮对话后追加新 token, 超过窗口上限时从左侧丢弃。

当 Recent 超过设定的最大长度时，将被丢弃的 token 数量累计记录，为后续摘要触发提供依据。

```
// Check context overflow
if (current_position >= DEFAULT_CONTEXT_SIZE - OVERFLOW_HEADROOM) {
    LOGW("%s: Context full! Triggering compression...", __func__);
    check_and_compress_context();
    if (g_needs_rebuild) {
        if (rebuild_kv_cache() != 0) {
            return nullptr;
        }
    }
}
}
```

图 2: 一次更新逻辑

3.3.2 长期记忆压缩与 KV cache 重建策略

为避免频繁生成摘要带来的额外计算开销，本工作采用基于丢弃累计量的触发机制。当 Recent 滑动窗口持续丢弃的 token 数量累计达到设定阈值后，才触发一次摘要更新流程。

摘要更新流程包括以下步骤：

1. 构造摘要输入（由旧 Summary、被丢弃的部分 Recent 组成）；
2. 在独立的摘要 context 中生成新的语义摘要；
3. 更新 Summary 内容，并标记主推理 context 需要重建；
4. 清空主 KV cache，并重新以 Prefix + Summary + Recent 进行 prefill。

其中，生成 summary 的方式为：先采用启发式策略进行稀疏注意力的稀疏采样，将 old summary+recent discard 采样得到 sparse summary；再调用生成摘要专用的独立 context 进行 decode，生成最终的 new summary。

该“低频重建”策略避免了 token 级 KV 管理所带来的复杂内存访问问题，更适合 Android CPU 的性能特点。

```
// 策略：保留开头 + 结尾 + 均匀采样中间
const int head = g_segments.summary_max / 4;
const int tail = g_segments.summary_max / 4;
const int middle = g_segments.summary_max - head - tail;

// 开头
result.insert(position, result.end(), first, tokens.begin(), last, tokens.begin() + head);

// 均匀采样中间
const int middle_start = head;
const int middle_end = (int) tokens.size() - tail;
const int step = std::max(1, (middle_end - middle_start) / middle);
for (int i = middle_start; i < middle_end && (int) result.size() < head + middle; i += step) {
    result.push_back(tokens[i]);
}

// 结尾
result.insert(position, result.end(), first, tokens.end() - tail, last, tokens.end());

return result;
```

图 3: 稀疏策略

3.3.3 面向 CPU 的 SnapKV-lite 实现

考虑到移动端 CPU 上难以低成本获取精确的注意力权重，且 token 级稀疏 KV 访问容易引发 cache miss，本工作未采用严格的 token-level SnapKV，而是通过启发式稀疏注意力采样实现工程化近似。

具体策略是依据先验知识（如 StreamingLLM 中对于维持注意力的关键 attention sink）我们分别保留 kv 的前后 $\frac{1}{4}$ ，对于中间的进行均匀稀疏采样。仅将这些稀疏 token 作为摘要输入，从而在降低摘要生成成本的同时，保留“重要信息优先”的核心思想。

3.3.4 KV cache 量化策略

在 KV cache 优化方面，采用任务感知的量化策略以平衡性能与质量：

- 主对话推理 context 使用 Q8_0 KV 量化，以保证生成质量；
- 摘要生成 context 使用 Q4_0 KV 量化，优先降低内存占用与计算开销。

摘要 context 在程序初始化阶段完成创建，并配置独立的 batch 与 sampler，从而实现摘要推理过程与主推理流程的资源隔离。

3.4 基于 MedChatZH 的模型微调

3.4.1 设计思路

模型微调部分的设计思路是在原先预训练模型的基础上进行模型解决某个特定领域任务的能力。我们选择了 Qwen-0.6B-Base 模型进行微调，这是因为 Qwen 模型更适配中文数据集任务，同时小参数规模的模型能够降低工程实现与调试成本，以及 Base 模型未经过指令对齐或 RLHF 约束，可塑性更强，能更直观观察到微调效果

观察到医疗知识问答任务在当前大模型工作中开始逐渐流行，我们也尝试从这一角度出发，以提升模型在该方面的能力。我们选用了 MedChatZH 数据集进行微调，并对微调后的结果进行评估和分析。

3.4.2 算法解释

我们使用了比较流行的 LoRA 微调，这是一种用于大规模语言模型的高效微调方法，其核心思想是在不修改原始模型参数的前提下，通过引入少量可训练的低秩参数来改变模型行为。传统的全参数微调需要对模型中所有权重进行更新，这在参数规模达到数十亿时会带来巨大的显存和存储开销，也容易导致模型遗忘原有能力。LoRA 的提出正是为了解决“只想轻微调整模型，却不得不整体重训”的问题。

从模型结构上看，大语言模型中最关键、参数量最大的部分是各种线性映射，例如注意力中的 Q/K/V/O 投影以及 MLP 中的 up/down projection。LoRA 并不直接修改这些权重矩阵，而是假设模型在新任务上的最优改动可以用一个低秩矩阵来近似。

在训练流程上，LoRA 与普通微调几乎一致：前向传播时模型输出等于“原模型输出 + LoRA 分支输出”，反向传播时梯度只流向 LoRA 参数。由于可训练参数极少，LoRA 微调对显存和计算资源的要求大幅降低，并且每个任务只需保存一个很小的 LoRA 适配器文件，而不必保存整份模型权重。在推理阶段，LoRA 可以按需加载或卸载，甚至多个 LoRA 还可以叠加使用。

从机制角度理解，LoRA 本质上是在模型的关键线性映射处学习一个方向性很强、受限于低维子空间的权重扰动。这与推理期的干预方法（如 ITI 或对 MLP 输出加向量扰动）在思想上是相通的：它们都不是重写整个模型，而是沿着少数重要方向对模型行为进行控制。区别在于，

LoRA 通过训练把这种扰动“固化”进权重空间，而 ITI 或 MLP-level 干预是在推理时直接修改激活状态。正因为这种低秩、可控、模块化的特性，LoRA 已成为当前大模型微调中最常用、最实用的方法之一。



图 4: 中文医疗对话数据集 MedChatZH

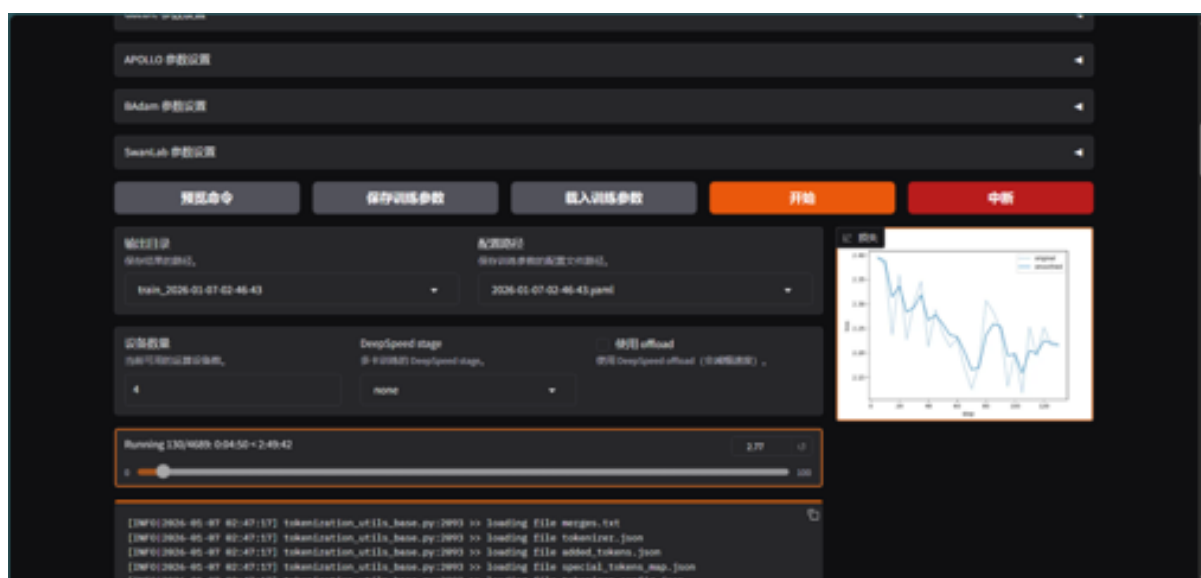


图 5: 训练过程截图

3.5 小组分工

范一泽：科研探索，数据微调及评测，PPT 制作

曹晨旭：APP 开发，推理优化，PPT 制作

黄勇鑫：APP 开发，数据微调，PPT 制作

4 实验与成果展示

4.1 推理优化实验结果

4.1.1 优化前：长上下文下的注意力丧失问题

在未引入上下文压缩与 KV cache 管理机制之前，手机端推理主要依赖全量历史拼接或简单滑动窗口策略。当对话长度持续增长时，系统面临以下问题：

- **KV cache 无界增长**: 历史 token 持续累积, 导致注意力计算复杂度和内存占用快速上升;
- **推理性能显著退化**: 在 Android CPU-only 环境下, 推理延迟随上下文长度线性甚至超线性增长;
- **注意力丧失现象明显**: 当上下文超过模型可承受范围后, 早期关键信息被滑动窗口直接裁剪, 模型无法正确回忆用户约束、设定或早期结论。

在实际对话中, 这种注意力丧失表现为模型“遗忘”早期给出的回答、忽略用户先前明确提出的条件, 甚至在长对话后出现语义漂移, 严重影响交互体验。

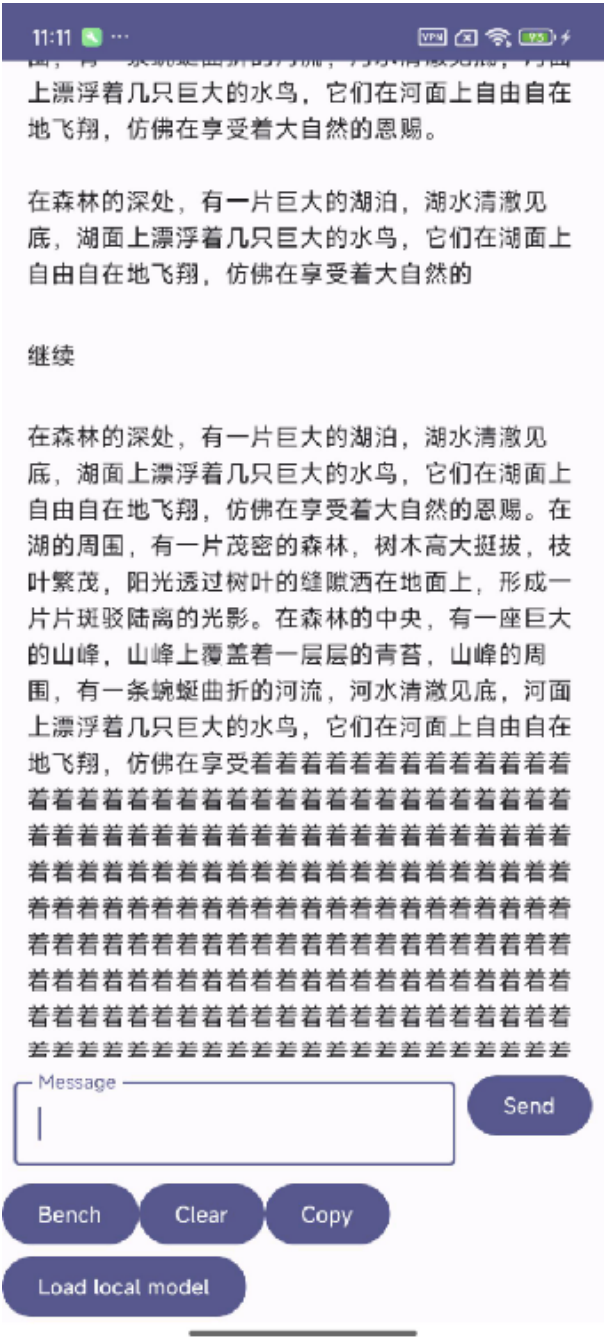


图 6: 优化之前, 简单的 KV 裁剪无法维持注意力

4.1.2 优化后：近似无限长上下文的对话能力

针对上述问题，我们提出并实现了一套基于 StreamingLLM 的上下文优化方案，通过三段式连续上下文结构（Prefix / Summary / Recent）与 KV cache 压缩机制，使系统在有限 KV 资源下具备长期记忆能力。

优化后的系统具有以下显著特征：

- **长期语义信息得以保留**：被裁剪的历史内容不再直接丢弃，而是通过语义摘要形式持续注入上下文；
- **注意力行为更加稳定**：模型能够持续关注早期对话中的关键设定、约束与结论；
- **上下文长度不再成为功能瓶颈**：在理论上，对话轮次可以无限增长，系统仅以固定规模的 KV cache 运行。

尽管底层模型的最大上下文长度有限，但通过稀疏摘要驱动的记忆压缩机制，模型在语义层面实现了“近似无限长上下文”的对话能力。

4.1.3 对比分析：从注意力丧失到长期记忆保持

将优化前后的系统进行对比，可以总结如下变化：

- 从“历史 token 被直接裁剪”转变为“历史语义被持续压缩与保留”；
- 从“对话轮次增加导致模型遗忘”转变为“模型可稳定遵循长期设定”；
- 从“上下文长度受限于 KV cache 容量”转变为“上下文长度与 KV 规模解耦”。

上述改进并非来源于模型参数规模的扩大，而是通过对推理框架与上下文管理方式的优化实现，体现了我们在 **推理框架层面**的技术贡献。

此外，我们创新性地将 SnapKV 的提取关键注意力的思想，通过启发式的稀疏策略应用到了 summary 生成中，不仅能够加快生成速度，还能够保留关键的历史信息，在端侧移动平台做到了效率与质量的双重适配。

4.2 微调实验结果

```
{'loss': 2.2124, 'grad_norm': 0.4910641610622406, 'learning_rate': 5.611095602597871e-10, 'epoch': 2.99, 'num_input_tokens_seen': 102690864, 'train_runtime': 10264.9914, 'train_tokens_per_second': 10003.989}
[INFO]2026-01-07 05:38:54] llamafactory.train.callbacks:143 >> {'loss': 2.1015, 'learning_rate': 1.4028e-10, 'epoch': 3.00, 'throughput': 10004.50}
{'loss': 2.1015, 'grad_norm': 0.44360944628715515, 'learning_rate': 1.4027778362235567e-10, 'epoch': 3.0, 'num_input_tokens_seen': 102801056, 'train_runtime': 10275.5234, 'train_tokens_per_second': 10004.459}
{'train_runtime': 10283.4962, 'train_samples_per_second': 29.173, 'train_steps_per_second': 0.456, 'train_loss': 2.153981921972306, 'epoch': 3.0, 'num_input_tokens_seen': 102887824}
***** train metrics *****
epoch                =          3.0
num_input_tokens_seen =    102887824
total_flos           = 256139439GF
train_loss            =         2.154
train_runtime         = 2:51:23.49
train_samples_per_second =    29.173
train_steps_per_second =         0.456
Figure saved at: saves/Qwen3-0.6B-Base/lora/train_2026-01-07-02-46-43/training_loss.png
[WARNING]2026-01-07 05:39:02] llamafactory.extras.plotting:148 >> No metric eval_loss to plot.
[WARNING]2026-01-07 05:39:02] llamafactory.extras.plotting:148 >> No metric eval_accuracy to plot.
```

图 7：微调结果

上图展示了我们在终端运行之后的结果，包括 tokenNum 以及 trainTime 等一些关键信息。在训练之后，我们从不同角度对模型进行了评估，在这里我展示其中一部分。由于我们的测试集有 500 条数据，所以我们直接进行了人工审查，以下是一些典型样例。

[illegible]

图 8: 典型样例 1

上面是一个经典的情况，我们对这种情况进行了分析：由于 **Base** 预训练模型的训练目标主要是语言建模，而非任务导向的问答生成，在自回归生成过程中容易陷入高概率 **token** 序列循环，所以才会出现上图所示的微调前的结果；而在经过微调后，模型被显式引导学习医疗问答场景中的合理回答方式，包括信息组织结构、回答长度以及自然结束位置。微调过程在概率分布层面降低了对问题复述类 **token** 的依赖，缓解了重复生成问题。从这一例子我们能够直观地感受到模型微调在生成行为层面改善了模型输出的可用性。下面还有部分样例：

[illegible]

图 9: 样例 2

术前输出:

B超检查疑是其胎瘤，是会复发吗？十前做过其胎瘤手术，现在怀孕了，想问一下，B超检查疑是其胎瘤，是会复发吗？十前做过其胎瘤手术，现在怀孕了，想问一下，B超检查疑是其胎瘤，是会复发吗？

B超检查疑是其胎瘤，是会复发吗？十前做过其胎瘤手术，现在怀孕了，想问一下，B超检查疑是其胎瘤，是会复发吗？十前做过其胎瘤手术，现在怀孕了，想问一下，B超检查疑是其胎瘤，是会复发吗？

B超检查疑是其胎瘤，是会复发吗？十前做过其胎瘤手术，现在怀孕了，想问一下，B超检查疑是其胎瘤，是会复发吗？十前做过其胎瘤手术，现在怀孕了，想问一下，B超检查疑是其胎瘤，是会复发吗？

=====
术后输出:

B超检查疑是其胎瘤，是会复发吗？十前做过其胎瘤手术，现在怀孕了，想问一下，B超检查疑是其胎瘤，是会复发吗？如果复发了，会有什么后果，谢谢您好，根据您的描述，您可能患有卵巢肿瘤。卵巢肿瘤是一种常见的妇科疾病，其复发率相对较低，但具体情况因人而异。如果您在怀孕期间进行了卵巢肿瘤手术，那么手术的成功率会提高，但仍存在一定的复发风险。如果复发了，可能会造成卵巢肿瘤的再次生长，影响您的生育能力，甚至可能引起其他并发症。因此，建议您在怀孕期间定期进行妇科检查，及时发现和治疗卵巢肿瘤，以保障您和宝宝的安全。如果您有任何疑问或需要进一步的帮助，请随时与您的医生联系。[\[endof.txt\]](#)

图 10: 样例 3

4.3 目标达成情况

项目在基础、进阶与突破性要求上均实现 100% 完成，在挑战性目标上完成度约为 90%，总体达成预期目标。

1. 基础要求：展示 GPU 平台上大语言模型推理的基础功能，以及在相关数据集上精度测试的结果。
2. 进阶要求：在上述展示结果的基础上添加展示对应优化的效果。例如如果实现了量化算法，则需要展示量化前后的模型参数量对比结果，以及量化前后模型推理的性能对比结果。
3. 突破性要求：展示手机上大语言模型推理的基础功能，在相关数据集上精度测试的结果，以及相关优化的前后对比结果。
4. 挑战性任务：展示手机上大语言模型推理功能改进前后的对比效果。

5 相关讨论

5.1 工作的贡献与创新性

本课题工作对推理优化过程做出了新的尝试，并在微调过程中引入了中文医疗对话数据集。

5.2 主要难点与挑战

本课题的主要难点和挑战集中体现在端侧大语言模型部署所面临的多重资源约束条件下的推理稳定性与效率问题。

5.3 经验与教训

通过该课程，我们感受到端侧场景下模型部署受到算力、内存、能耗等多重因素约束，更强调模型规模控制与推理效率。同时也熟悉了端侧推理框架与模型加载流程，并且掌握了从模型微调到端侧推理的完整步骤。最后，我们还根据已有成果进行新的尝试，体会到了不同策略在端侧应用中的实际价值。

5.4 相关课程支撑目标情况

本科阶段所学习的高等数学、机器学习与程序设计类课程，为理解模型结构、实现推理优化与系统开发提供了重要支撑。总体而言，本课程通过端侧大模型部署与优化这一综合性实践任务，较好体现了顶点课程“综合应用与能力验证”的核心定位。

6 总结与展望

6.1 项目总结

在端侧推理优化方面，本项目对 KV Cache 管理与推理流程进行了新的尝试；在模型训练方面，引入了中文医疗对话数据集进行指令微调。通过本项目的实践，系统性理解了端侧部署场景下面临的算力、内存与能耗约束，并掌握了从模型微调到端侧推理的完整流程。

6.2 未来展望

未来可进一步从以下方向展开研究：

- 更高效的端侧模型压缩与量化策略；
- 面向端侧场景的参数高效微调方法；
- 推理阶段的系统级优化，如 KV Cache 管理与内存调度；
- 构建更加全面的评测与分析体系。

参考文献

- [1] Dongwon Jo et al. “FastKV: Decoupling of Context Reduction and KV Cache Compression for Prefill-Decoding Acceleration”. In: *arXiv preprint arXiv:2502.01068* (2025).
- [2] Yuhong Li et al. “SnapKV: LLM Knows What You are Looking for Before Generation”. In: *arXiv preprint arXiv:2404.14469* (2024).
- [3] Zhenmei Shi et al. “Discovering the gems in early layers: Accelerating long-context llms with 1000x input token reduction”. In: *arXiv preprint arXiv:2409.17422* (2024).
- [4] Guangxuan Xiao et al. “Efficient Streaming Language Models with Attention Sinks”. In: *arXiv* (2023).
- [5] Dongjie Yang et al. “PyramidInfer: Pyramid KV Cache Compression for High-throughput LLM Inference”. In: *Findings of the Association for Computational Linguistics: ACL 2024*. Ed. by Lun-Wei Ku, Andre Martins, and Vivek Srikumar. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 3258–3270. doi: [10.18653/v1/2024.findings-acl.195](https://doi.org/10.18653/v1/2024.findings-acl.195). URL: <https://aclanthology.org/2024.findings-acl.195/>.
- [6] Zhenyu Zhang et al. “H2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models”. In: *Advances in Neural Information Processing Systems*. Ed. by A. Oh et al. Vol. 36. Curran Associates, Inc., 2023, pp. 34661–34710. URL: https://proceedings.neurips.cc/paper_files/paper/2023/file/6ceefa7b15572587b78ecfceb2827f8-Paper-Conference.pdf.