

顶点课程个人报告

第四组

范一泽

日期：2026年1月11日

摘要

本文为顶点课程结题报告，主要介绍个人在本次顶点课程项目实践过程中的探索与成果。

课程中，我们组的项目包括模型微调、推理优化、APP 开发、科研探索等几个方面，最终我们成功完成了这几部分。其中，我负责科研探索、模型微调以及评估工作。

关键词：大语言模型，端侧部署，推理优化

1 研究内容

本人在小组项目中主要负责探索 KV cache 的优化方法、参与微调工作以及评估工作。

我探索了当前比较主流的优化方法，包括 StreamingLLM、SnapKV、H2O、PyramidInfer、Gem-Filter 以及 FastKV，对他们原文的实现原理进行分析、本地实现以及对比评测。

对于微调，我们使用 Llamafactory 框架对我们的模型进行特定领域微调，具体采用 lora 技术，使用 MedChatZH 数据集（为医学领域中文对话指令数据集），使我们的模型适应下游医疗对话。

2 实现过程

2.1 KV cache 优化方法探索

现有的 KV cache 优化策略大体可分为两类，一类是 Decoding-Only，一类是 Prefill-Aware。

Decoding-Only. 这类方法顾名思义，只对 decode 阶段起到加速作用，prefill 阶段仍需计算全量 KV，因此面对超长 prompt 时仍有压力。

- **StreamingLLM:** 发现模型除了在最近位置的注意力得分高之外，对于历史 tokens 中初始位置的几个 tokens 很受关注。于是将其注意力稀疏为：Attention Sink + Sliding Window
- **SnapKV:** 通过当前位置前面的一段 observation window 计算 prefix 中哪些位置最重要，具体为：window 内的各 token query 在各个注意力头上对 prefix 中各 token 的 attention 权重求和，选出 top-k 个 tokens，最终保留的 tokens 为：Top-k important prefix + Observation window

- **H2O:** 将 cache 淘汰问题定义为优化问题，优化目标为当前 cache 内所有 tokens 累计 attention 得分最高。H2O 采用贪心策略：每一步生成后，如果超出 cache，则淘汰 cache 内累计得分最低 tokens

Prefill-Aware. 这类方法则针对 prefill 阶段进行优化，上面的三个方法无法减少 prefill 阶段的延迟，下面的三个方法则致力于在 prefill 阶段内各层进行逐层筛选，以减少计算量。

- **PyramidInfer:** 每个注意力层只保留 N_l 个 tokens，每层淘汰的 tokens 不参与下一层的 kv 计算，由于 N_l 逐层递减，因此 prefill 阶段各层的计算量会逐层减少，降低了延迟
- **GemFilter:** 该方法主题思想为在 prefill 前进行一次性筛选，只允许选中的 tokens 参与接下来的 prefill，具体实现为：将模型前 r 层单独拿出来做一次前向，用第 r 层最后一个 token 的 query 与前面的 tokens 计算 attention，选出 top-k 用于后面的 prefill。减少了 prefill 阶段所有层的计算量
- **FastKV:** 考虑到模型的浅层注意力分布很分散，深层则逐渐集中到少数 tokens，前两个方法过早丢弃浅层信息，于是 FastKV 在 prefill 阶段标定一个模型中间层命名为 TSP 层，TSP 层之前全量计算所有 tokens 的 kv 以及注意力，并传给后面的层，到 TSP 层时基于前面计算的注意力得分选出 top-k，TSP 之后的层均以 TSP 层选出的 token 位置作为初始 cache，后续按各层自己的 retention rate 独立更新。这样较深的层能看到 TSP 之前浅层的全部信息，防止早期聚焦模式不稳定而误弃重要 token

我对这些方法做了 bench 评估以及系统性能评测，结果在第3章。经过我们的分析，我们发现这些方法都聚焦于如何更科学的计算出哪些 tokens 应该被保留，而哪些应该被淘汰，但对于被淘汰的 tokens 则是完全舍弃，我们认为在端侧部署条件下，上下文长度受限，面对长 prompt 以及长对话，模型很容易丢失记忆，因此要利用好每个 token，尽量利用到每一份的信息。由于是在 llama.cpp 框架实现，考虑到难度较大，于是我们决定先在 StreamingLLM 的方法基础上添加我们的实现，日后再慢慢探究其他方法的适配。方法主体思想为对淘汰的 tokens 进行收集，每到一定阶段压缩成 summary 进行回插，以防止记忆丢失，更多细节请见团体报告。

2.2 模型微调

在模型微调工作中，我主要参与了数据集的选定、预处理，以及微调后模型的评估工作。我将 huggingface 下载的数据转换适配 Qwen3 模板，考虑到模型较小，为了防止破坏模型的原生能力，我们采用 lora 微调，并 merge 进模型权重之中，随后我在 MedChatZH 的官方测试集上对模型进行了评估，具体指标为 $F1_{BERT}$ 。

3 成果展示

3.1 KV cache 方法评测与分析

我对调研的现有方法分别评估了单文档问答、多文档问答、文本摘要、few-shot 以及代码任务的表现，如图1左侧所示。可以发现几乎所有方法的各任务表现均低于全 KV，SnapKV 和 FastKV 表现稍突出。随后，我对各个方法的系统性能进行了评测，具体为各个上下文长度的

各优化方法任务评估							各优化方法系统性能评测						
Method	narrativeqa	2wikimqa	gov_report	trec	passage_count	lcc	Stage	Context 8192		Context 32768		Context=131072	
	F1	F1	Rouge-L	Acc	Edit Sim	Acc		Lantency	MaxMem	Lantency	MaxMem	Lantency	MaxMem
fullkv	30.21	46.65	35.13	73	8.91	63.38	prefill	0.83s	2.44GB	4.48s	5.27GB	30.25s	16.60GB
streamingllm	25.96	37.93	25.39	58.5	7.91	59.29	e2e	10.95s	2.44GB	16.26s	5.27GB	53.77s	16.60GB
snapkv	31.49	44.38	28.86	70.5	7.93	60.46	prefill	1.00s	2.41GB	4.44s	5.15GB	30.20s	16.11GB
h2o	9.32	32.65	31.25	66.5	3.41	56.31	e2e	12.33s	2.41GB	15.16s	5.15GB	41.72s	16.11GB
gemfilter	22.91	36.29	28.28	60.5	4.9	33.09	prefill	0.69s	2.42GB	3.62s	5.19GB	30.58s	16.28GB
fastkv	30.54	46.3	28.19	73	7.36	60.11	e2e	13.44s	2.42GB	16.40s	5.19GB	43.29s	16.28GB
							prefill	1.81s	19.02GB	-	OOM	-	OOM
							e2e	13.99s	19.02GB	-	OOM	-	OOM
							prefill	0.53s	2.41GB	1.95s	5.14GB	14.37s	16.06GB
							e2e	10.81s	2.41GB	12.21s	5.14GB	24.62s	16.06GB
							prefill	0.90s	19.18GB	-	OOM	-	OOM
							e2e	17.33s	19.18GB	-	OOM	-	OOM
							prefill	0.41s	2.41GB	2.06s	5.15GB	16.55s	16.11GB
							e2e	6.87s	17.11GB	8.64s	20.19GB	23.24s	32.57GB

图 1: 对各方法的任务评估和系统性能评测。

prefill 和 end-to-end 延迟以及显存峰值占用，如图1右侧所示。可以发现下面三个 prefill-aware 方法的 prefill 阶段延迟都有所下降，而 decoding-only 的方法则几乎无变化。此外他们的端到端延迟都低于全 KV，峰值显存也略有减少。

3.2 模型微调效果评估

我使用 MedChatZH 官方测试集，共 500 条数据，对我们的模型进行了评估，采用 $F1_{BERT}$ 指标评估语义一致性。模型得分从微调前的 0.63 提升至微调后的 0.70，实现了约 11% 的提升。

4 个人总结

在本次顶点课程项目中，我主要负责 KV cache 推理优化方法的调研与评测，并参与了模型微调与效果评估工作。在推理优化方面，我系统分析并对比实现了多种主流 KV cache 方法，结合任务效果与系统性能评测，总结了不同方法在端侧长上下文场景下的优势与局限，为团队最终方案选择提供了实验依据。在模型微调方面，我参与了数据集处理以及评估工作，实验结果表明微调后模型在医疗对话任务上的语义一致性显著提升。

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/6ceefab15572587b78ecfcbb2827f8-Paper-Conference.pdf.