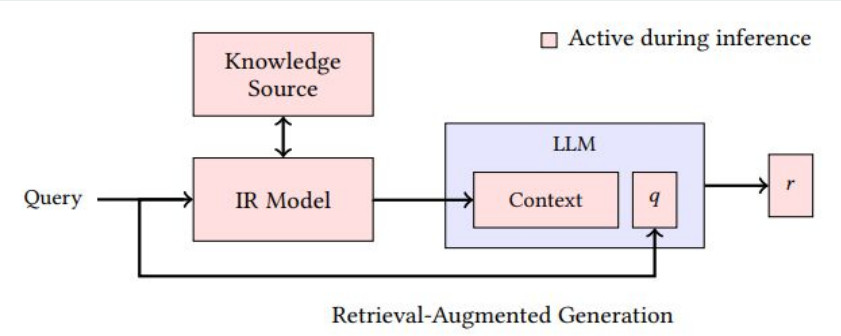
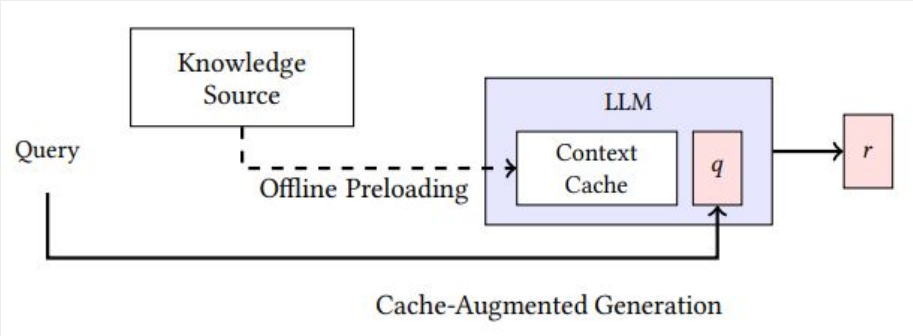
專題期末報告
Cache-Augmented Generation

111550170 卓建均

介紹

為了提升知識型任務的推論效率與準確率，以及消除RAG(Retrieval-Augmented Generation)在知識檢索上的延遲和誤差，本專題以CAG(Cache-Augmented Generation)方法取代RAG，研究並探討在不同模型與資料集下，CAG在推論速度和答案品質上的表現，並分析其在實際應用中的優勢與限制。





Cache-Augmented Generation(CAG)

目標:利用長文LLM的擴展上下文功能，讓語言模型能一次性整合所有知識，實現不需要即時檢索的高效問答

CAG的架構分為三個階段:
1. Knowledge Preloading
2. Inference
3. Cache Reset

Knowledge Preloading

這個階段會先蒐集目標應用相關的文件集合D，然後對其做預處理跟格式化，讓他放入LLM的extended context window。接著LLM將D編碼轉換成預計算的KV cache，這個KV cache封裝了LLM處理知識後的推論狀態並儲存在硬體或記憶體中，只需要一次計算成本就能無限使用。

```
knowledges = f"""
<|begin_of_text|>
<|start_header_id|>system<|end_header_id|>
You are an assistant for giving short answers based on given context.<|eot_id|>
<|start_header_id|>user<|end_header_id|>
Context information is bellow.
---------------------------------------------
{documents}
---------------------------------------------
{answer_instruction}
Question:
"""
```

Inference

在推論這個階段會將前一步計算好的KV cache與用戶的query一起載入，LLM利用cache中的知識context結合用戶問題，生成新答案。此步驟省略了即時檢索，一定程度了消除延遲與檢索錯誤的產生。

```
# 載入預先計算的 KV cache
kv_cache = torch.load("cache_knowledges.pt")

# 把用戶的query包裝成prompt並轉換成token ID
question_prompt = f"{question}<|eot_id|>\n<|start_header_id|>assistant<|end_header_id|>"
input_ids = tokenizer.encode(question_prompt, return_tensors="pt").to(model.device)

# 推論時直接利用快取生成答案
with torch.no_grad():
    output = model(input_ids, past_key_values=kv_cache, use_cache=True)
    answer = tokenizer.decode(output[0], skip_special_tokens=True)
```

Cache Reset

當知識庫內容有更新、擴充或更換的時候，要重新進行一次knowledge preloading，確保模型使用的KV cache內容是最新的。先清除舊的KV cache，再重新將知識集D編碼成新的KV cache，以維持推論正確性與知識時效性。

實驗設計

目的：系統性比較CAG與RAG兩種方法在知識型任務上的表現

資料集：SQuAD-train、HotPotQA-train
模型：Llama-3.2-8B-Instruct、Llama-3.2-1B-Instruct、TinyLlama-1.1B-chat-v1.0
評分指標：BertScore、Inference Time

實驗結果

System	Top-k	SQuAD-train Bertscore	HotPotQA-train Bertsocre
Sparse RAG Llama-3.2-8B-Instruct	1	0.72144	0.6788
	3	0.7616	0.7626
	5	0.7608	0.7676
	10	0.7584	0.7521
Dense RAG Llama-3.2-8B-Instruct	1	0.6216	0.7164
	3	0.7106	0.7582
	5	0.7334	0.7481
	10	0.7586	0.7576
CAG Llama-3.2-8B-Instruct		0.7695	0.7951
CAG Llama-3.2-1B-Instruct		0.2474	0.2948
CAG TinyLlama-1.1B-chat-v1.0		0.3977	0.4785
Sparse RAG Llama-3.2-1B-Instruct	1	0.4496	0.4464
	2	0.5548	0.4952
	3	0.5196	0.4244
	4	0.4073	0.4099
	5	0.3356	0.383
Sparse RAG TinyLlama-1.1B-chat-v1.0	1	0.3764	0.2937
	2	0.373	0.3071
	3	0.3238	0.3354

實驗結果

System	Retrieval	Generation
Sparse RAG, Top-3 Llama-3.2-8B-Instruct	0.0008	0.7406
Sparse RAG, Top-10 Llama-3.2-8B-Instruct	0.0012	1.5595
Dense RAG, Top-3 Llama-3.2-8B-Instruct	0.4849	1.0093
Dense RAG, Top-10 Llama-3.2-8B-Instruct	0.3803	2.6608
CAG Llama-3.2-8B-Instruct		0.8512
In-Context Learning		9.3197
CAG Llama-3.2-1B-Instruct	1.04E-06	0.1645
CAG TinyLlama-1.1B-chat-v1.0	9.18E-07	5.2076
Sparse RAG Llama-3.2-1B-Instruct	0.0254	0.3839
Sparse RAG TinyLlama-1.1B-chat-v1.0	0.0331	3.5327

結論

效率方面:
CAG方法在所有實驗中推論速度明顯快於RAG，尤其在知識庫規模較大或需要大量重複檢索的時候，CAG優勢更明顯。RAG則因為每次都需要即時檢索導致推論延遲較高

準確率方面:
RAG在部份情況下(特別是小模型或是複雜資料集)BertScore略高於CAG，但CAG的分數也十分接近且架構更簡單，CAG在知識庫完整預載入時，有效整合所有知識避免檢索遺漏