Hierarchical KV Cache Compression for LLMs

2024324064 Joshua Dela Rosa
2025313067 Mengjie Li

2025-12-03 (수)

Contents

Transformers and Attention

- Transformers rely heavily in the attention mechanism
- Compares current token to all tokens in the sequence.

Strong connection between “pizza” and “it”

The <u>pizza</u> came out of the <u>oven</u> and **it** tasted good.

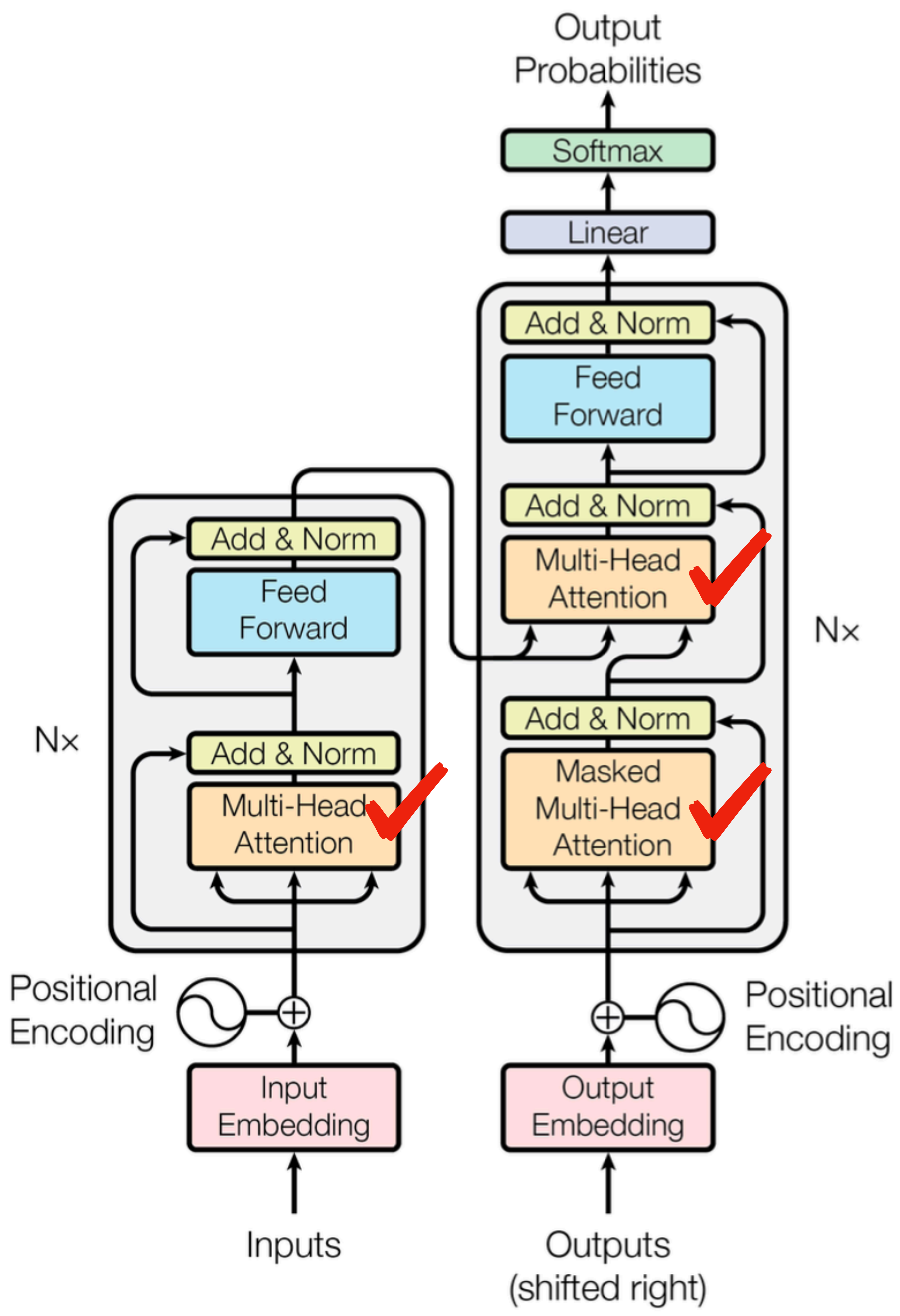
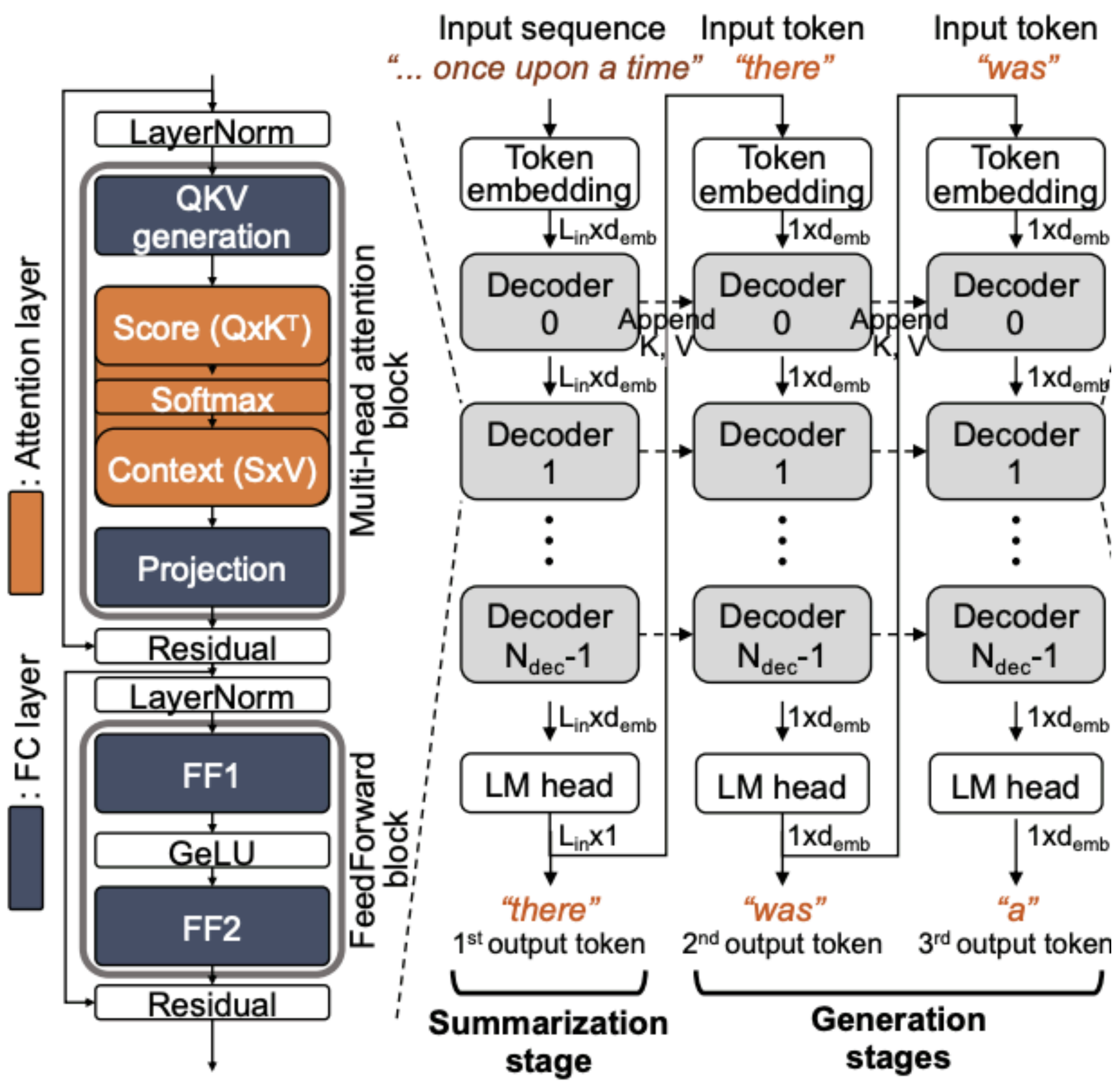


Figure 1: The Transformer - model architecture.

Transformers and Attention

- LLM computation is divided into two phases:
  - **Summarization (Prefill) Stage**: processing of all the tokens in the prompt
  - **Generation (Decode) Stage**: autoregressive generation of tokens



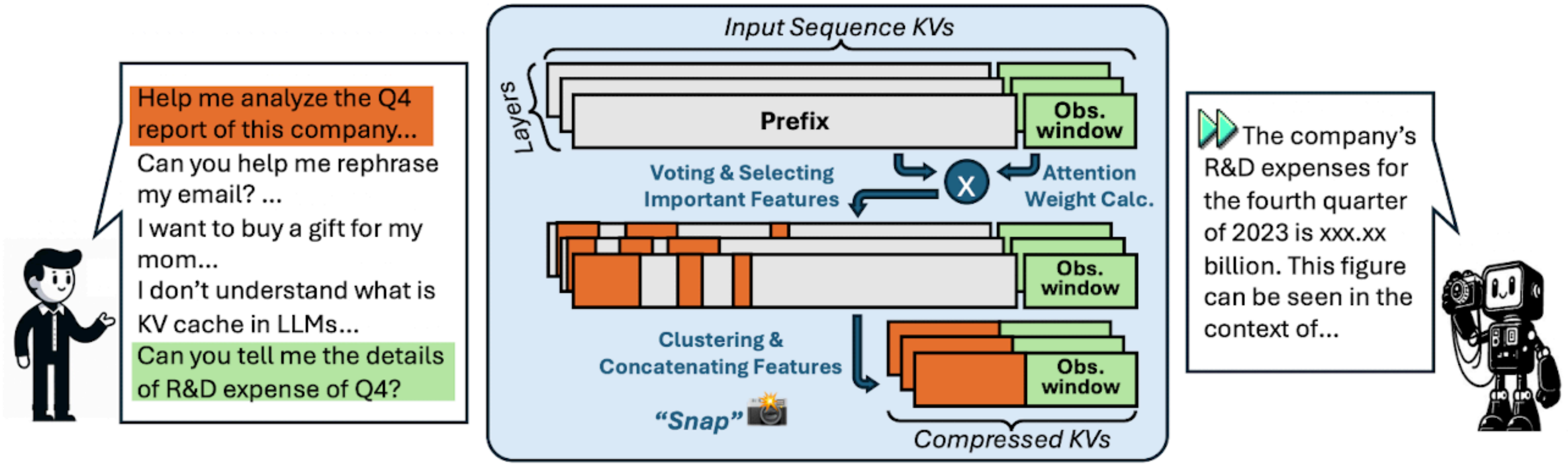
IMG SRC: AttAcc! Unleashing the Power of PIM for Batched TbGM Inference

KV Cache

- To generate a new token, it needs the computed keys and values for all previous tokens.
- For computation efficiency, cache the KV, trade for extra memory.
- Problem:
  - longer context and larger LLMs = larger KV cache
- KV cache compression is a hot topic.
  - Token-level KV cache optimization

Current KV Cache Research

- **SnapKV (NIPS 2024)**
  - the end part of the prompt dictates which tokens of the prompt should be kept



- **AttentionSink (ICLR 2024)**
  - it was observed that a large amount of attention scores are allocated to the initial tokens

Problems with SnapKV

- Why always the end part of the prompt?
  - Suppose a prompt where the question is at the front.
- SnapKV is limited by the assumption that the most important parts are at the end of the prompt.

Example Prompt A

Today was ...
For some reason he fell sick ...
So that was ...

Why was John absent yesterday?

Example Prompt B

In this excerpt, answer this question.
Why was John absent yesterday?

Today was ...
For some reason he fell sick ...
So that was ...

Problems with AttentionSinks

- Highly referenced tokens can exist at the middle.
- AttentionSinks and Context Windows aggressively removes tokens at the middle of the prompt and generated tokens.

Example

Sink

Physics had come...
Albert Einstein...
The **photoelectric effect** ...

<skip>

Evicted

Einstein got his Nobel Prize for his discovery of photoelectric effect ...

<skip>

Context Window

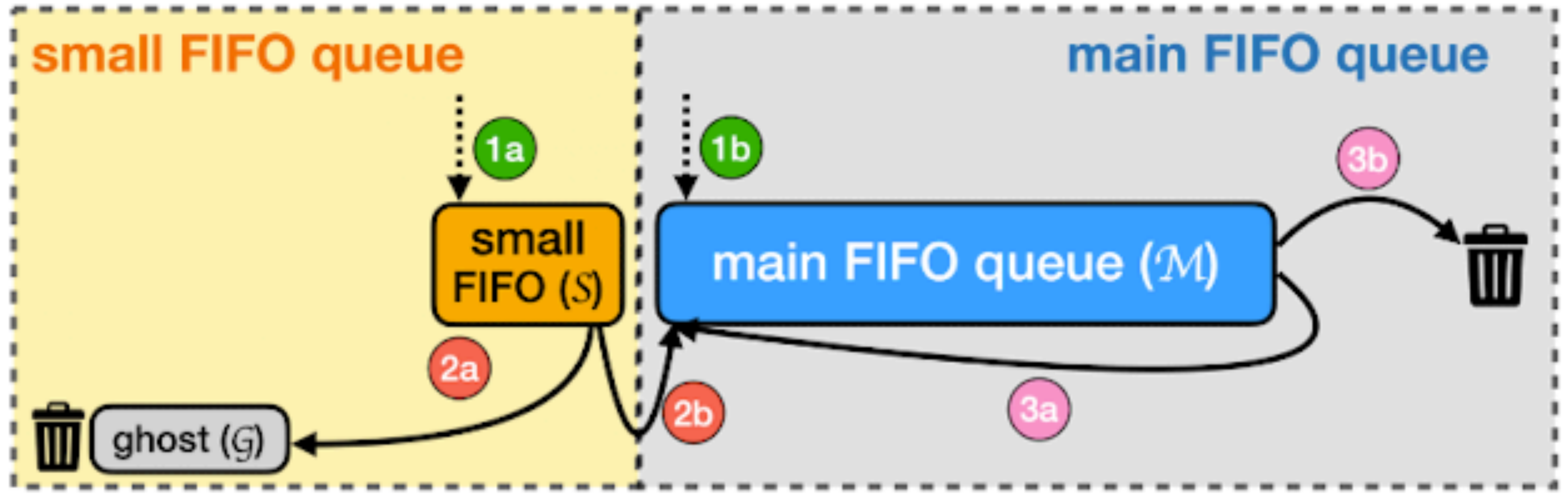
General theory of relativity is Albert Einstein.

What work gave Einstein his Nobel Prize?

Insights from Current Hardware-side Research

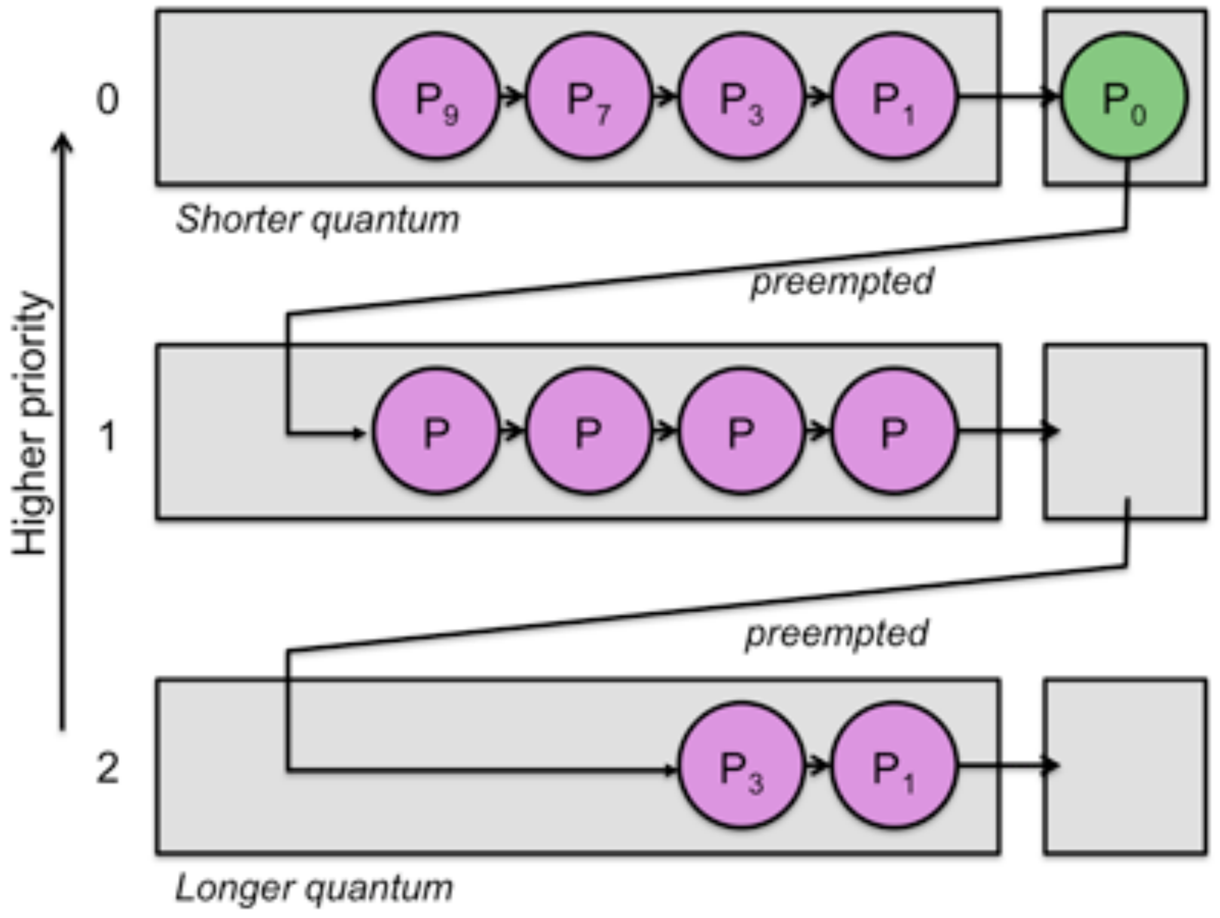
- Hierarchical-based policies are popular in hardware.

- **S3-FIFO (SOSP 2023):**
  filter-out cold objects, keep hot objects
  - Hierarchy between cold and hot



Insert: if not in ghost, insert to small 1a, else insert to main 1b
Evict (small): if not visited, insert to ghost 2a, else insert to main 2b
Evict (main): if visited, insert back 3a, else evict 3b

- **MLFQ (Undergrad OS)**:
  hierarchy based on CPU usage of a program
  - Prioritize programs that use the CPU less



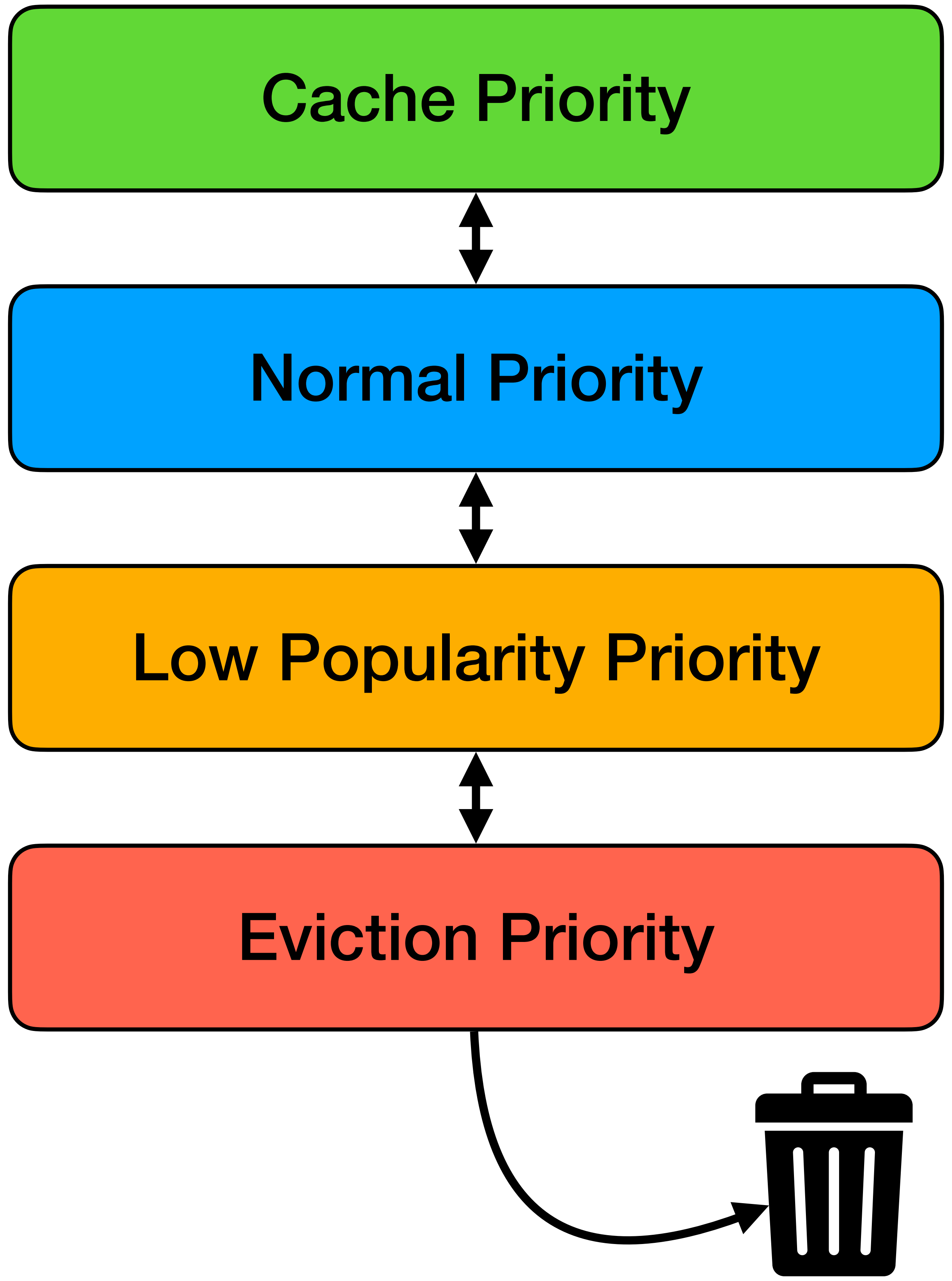
Motivation

- Current approaches (aggressively remove middle parts, or evict based on a small part of the prompt) are static.

- Hierarchical approaches allow for dynamic changes in the token access behavior.

  - Tokens that are highly referenced are placed at a high caching priority.

  - Unpopular tokens move toward the eviction priority.



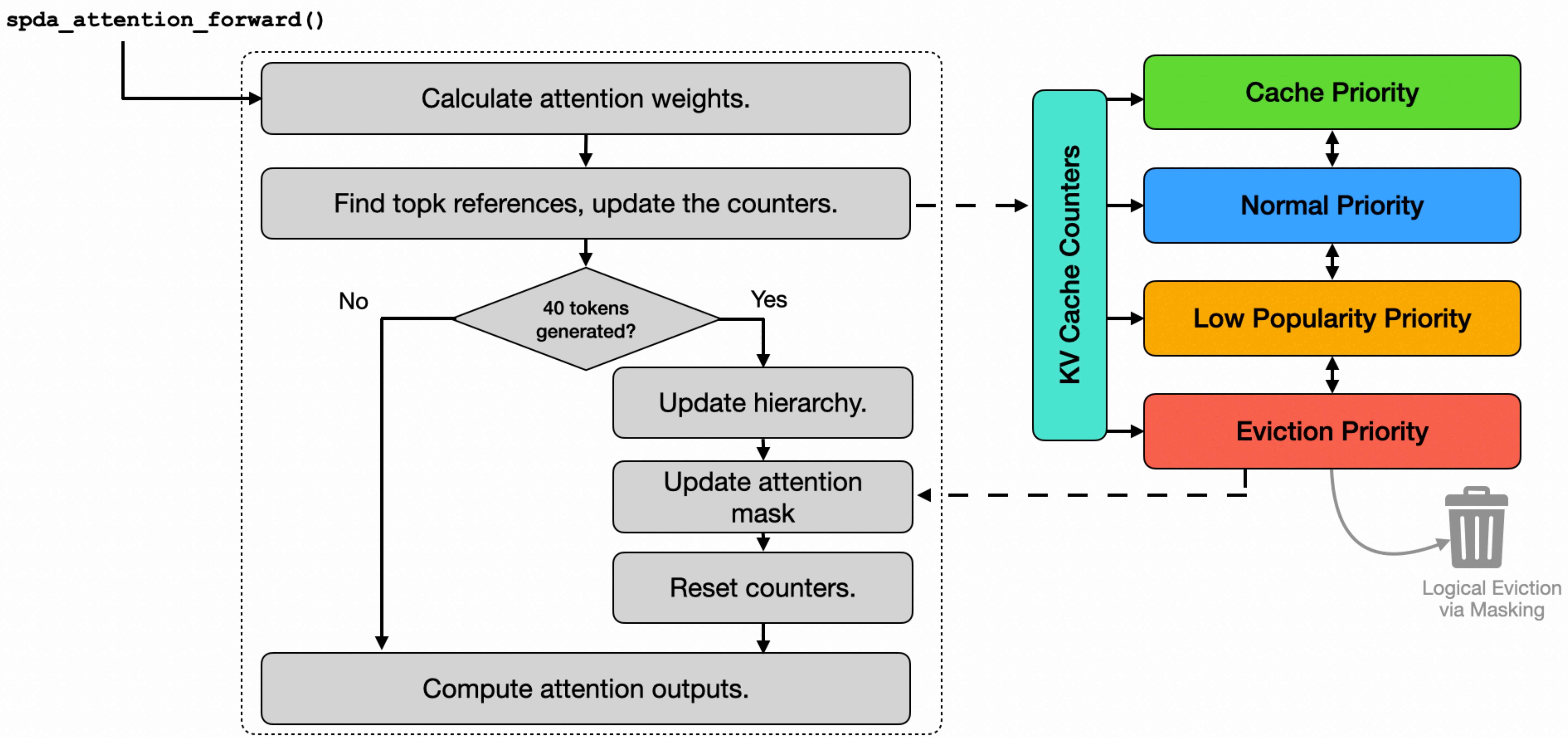
Implementation Setup

- **Interface:** Using HuggingFace
- **LLM:** Llama-2-7b-QNA-Tuned
- **Evaluation Dataset:** Open-source IELTS Reading Comprehension Test
- **Methodology:** Inject logical masking-based hierarchical KV cache management at `spda_attention_forward()` to emulate eviction.
  - Apparently, Pytorch does not have explicit `free()`.
  - llama.cpp was an option but for research purposes, logical eviction is sufficient.

Popularity Definition

- A token is considered popular if its attention weight often places it in the topk keys.
- Based on the number of times a token is part of topk within a profiling window, we place it on its corresponding hierarchy level:
  - **Cache Priority:** 10+ counts
  - **Normal Priority:** 5-9 counts
  - **Low Popularity Priority:** 1-4 counts
  - **Eviction Priority:** 0 counts
- The profiling is done every 40 tokens generated.
- Demotion by one level is performed if it fails to achieve required counts. Else, promotion.

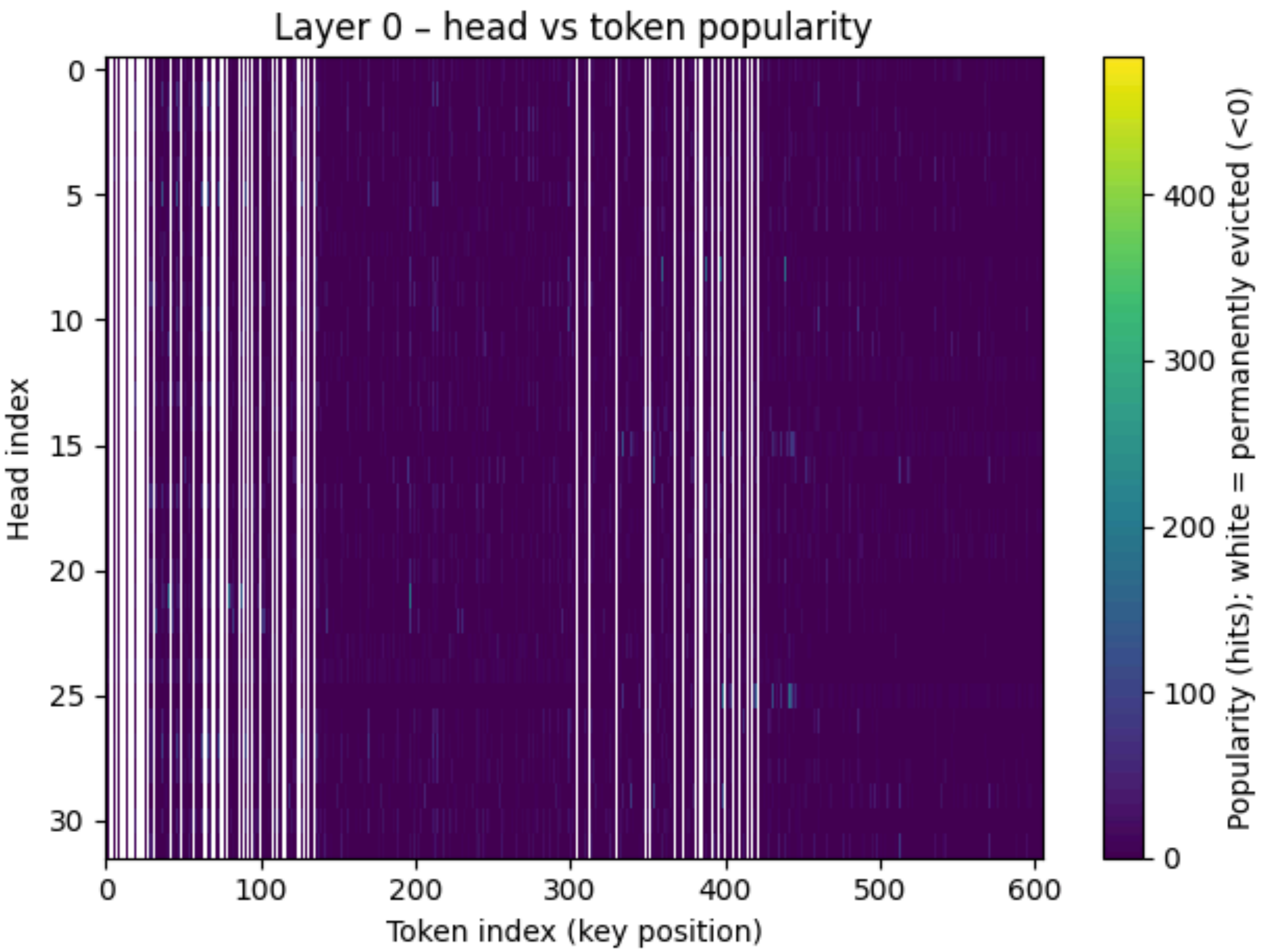
Policy Implementation

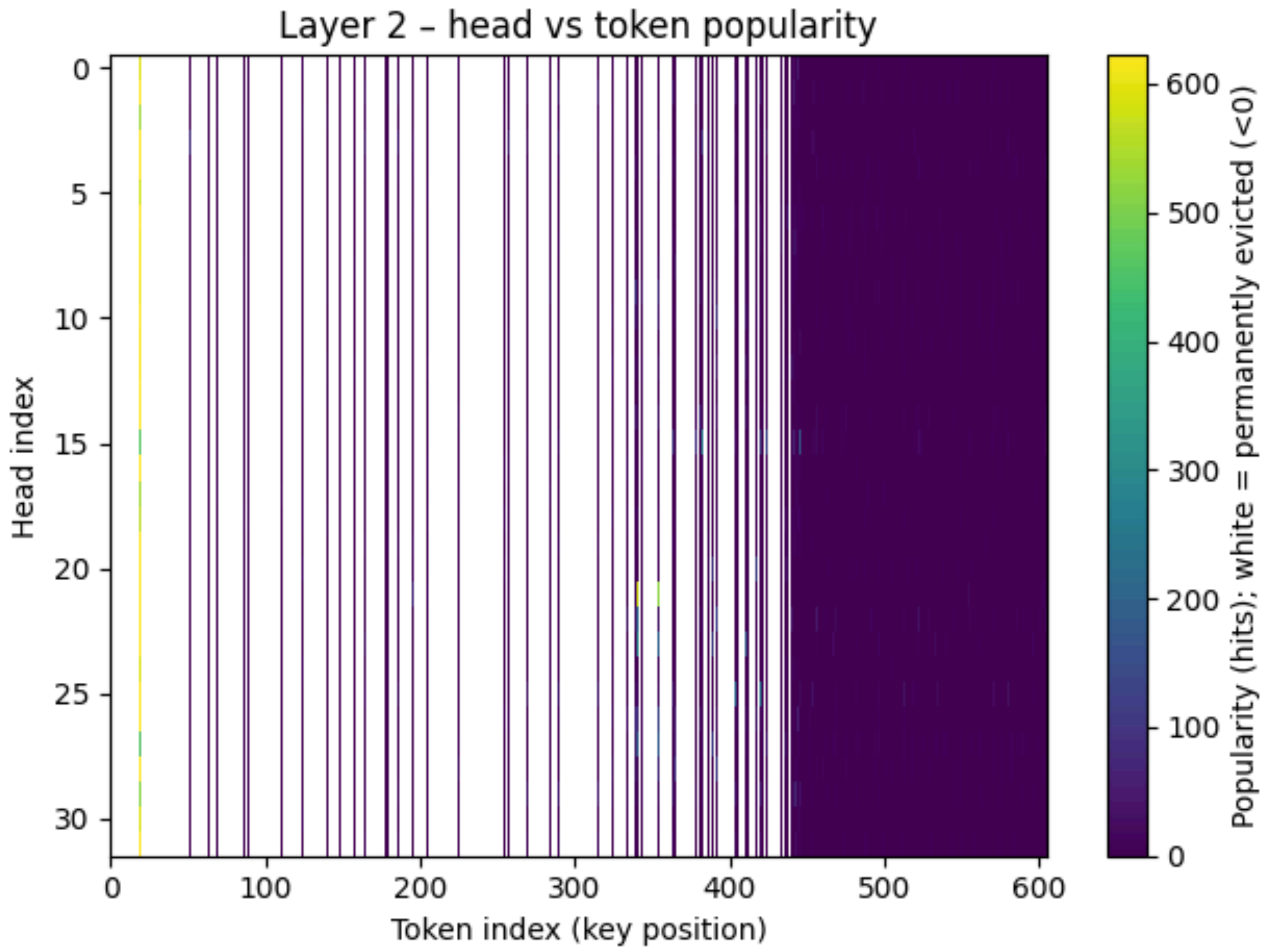


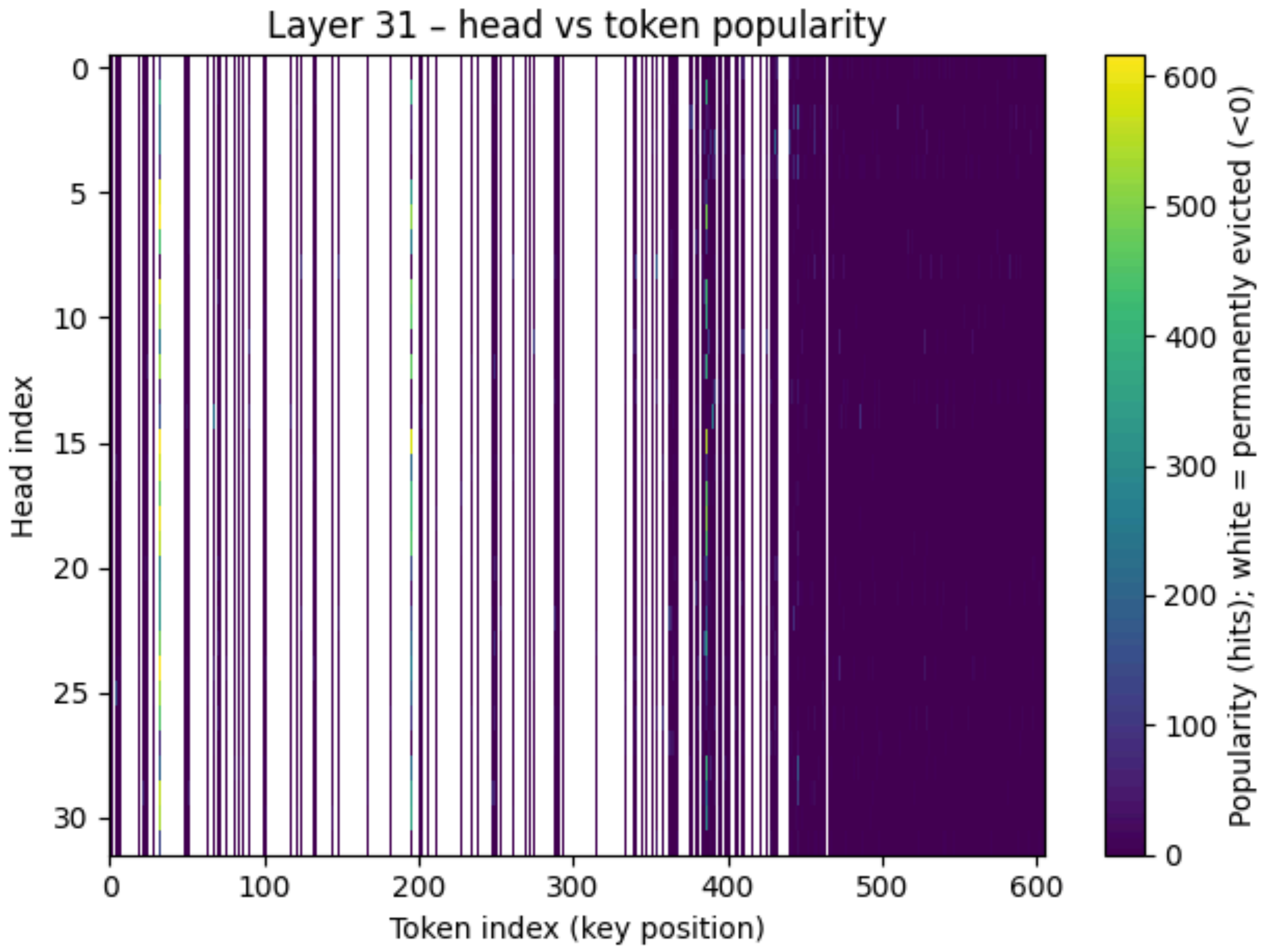
Preliminary Experiments

- **Prompts to test Needle-in-Haystack capability, four questions each:**
    - True or False, Fill in the Blanks, Multiple choice question, Matching Type
- **Position of the question:**
    - Before the passage
    - After the passage
- **Metric:** Quantitatively check if the LLM can answer correctly.
- **Comparison:**
    - No Eviction Baseline
    - Attention Sink (32 attention sink tokens, 256-token context window size)

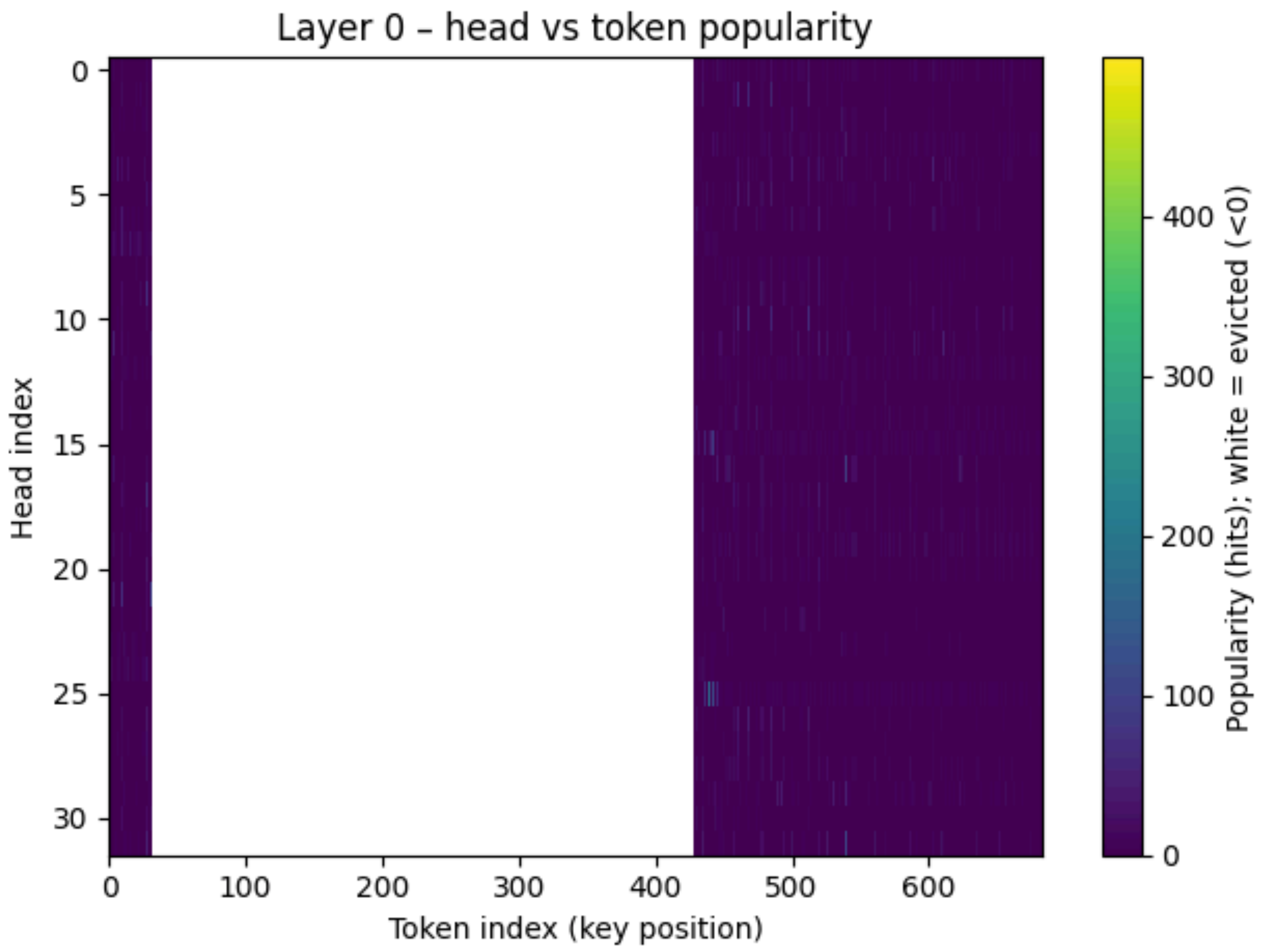
Popularity Plots for Hierarchical Policy

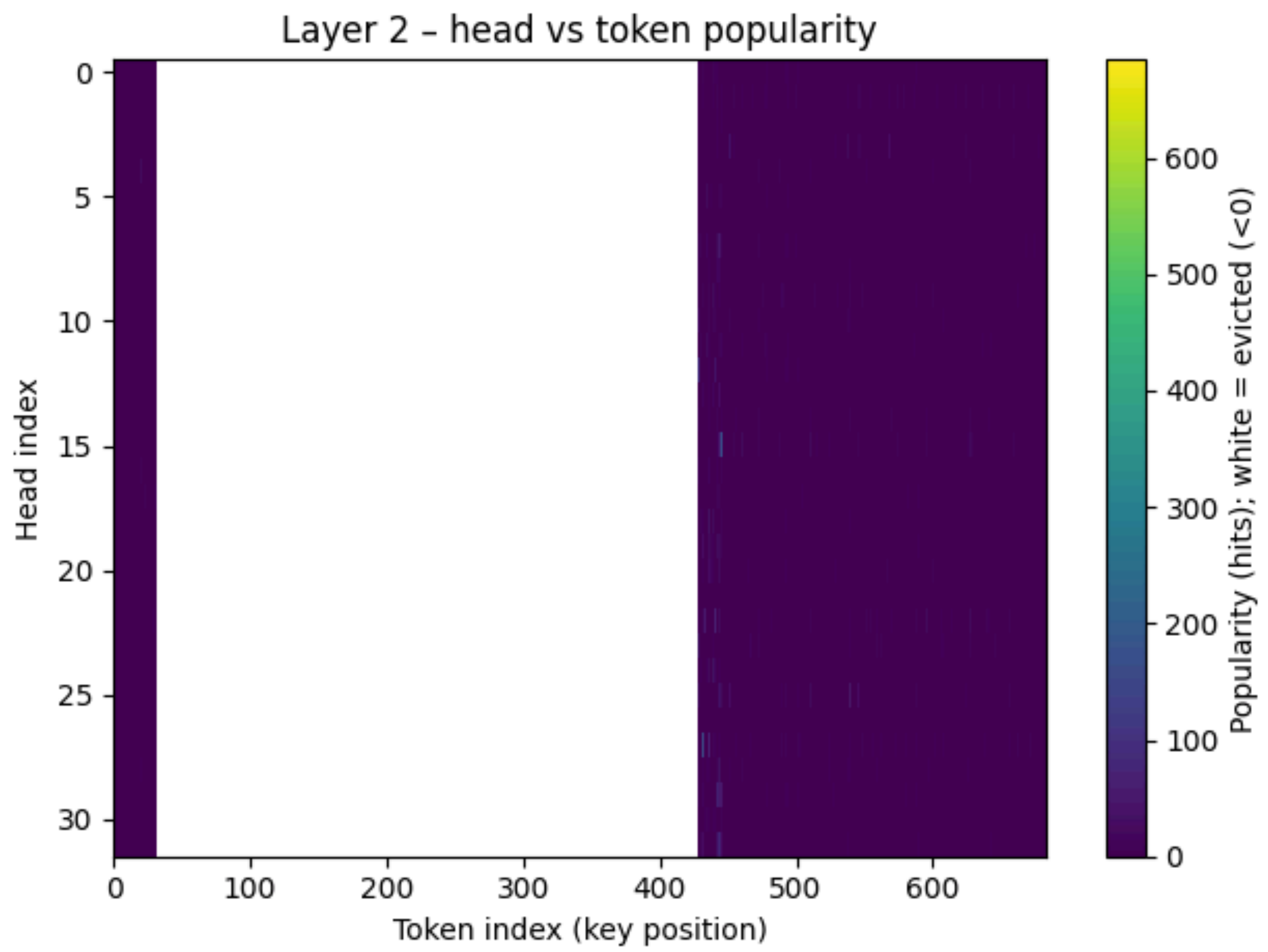


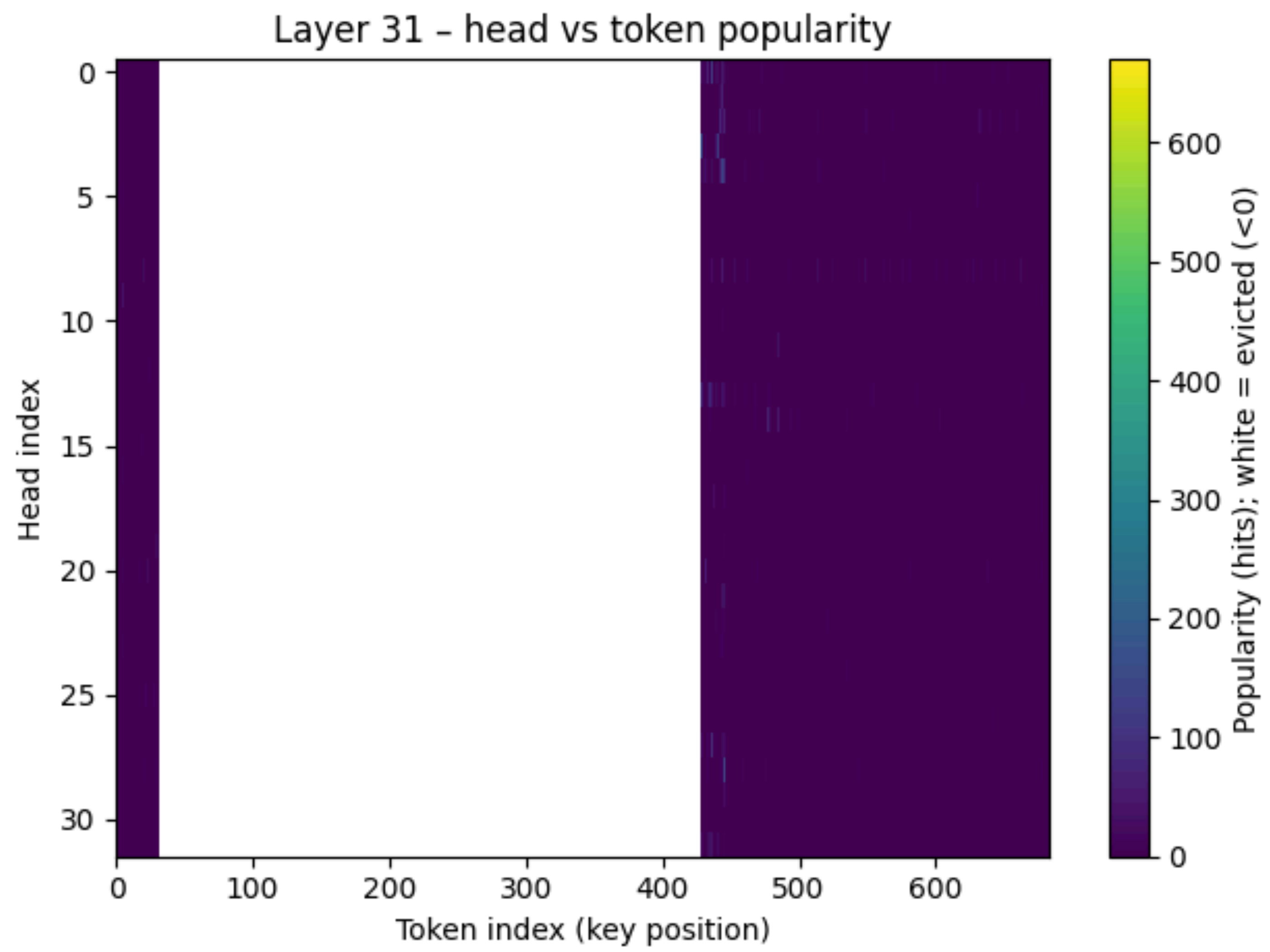




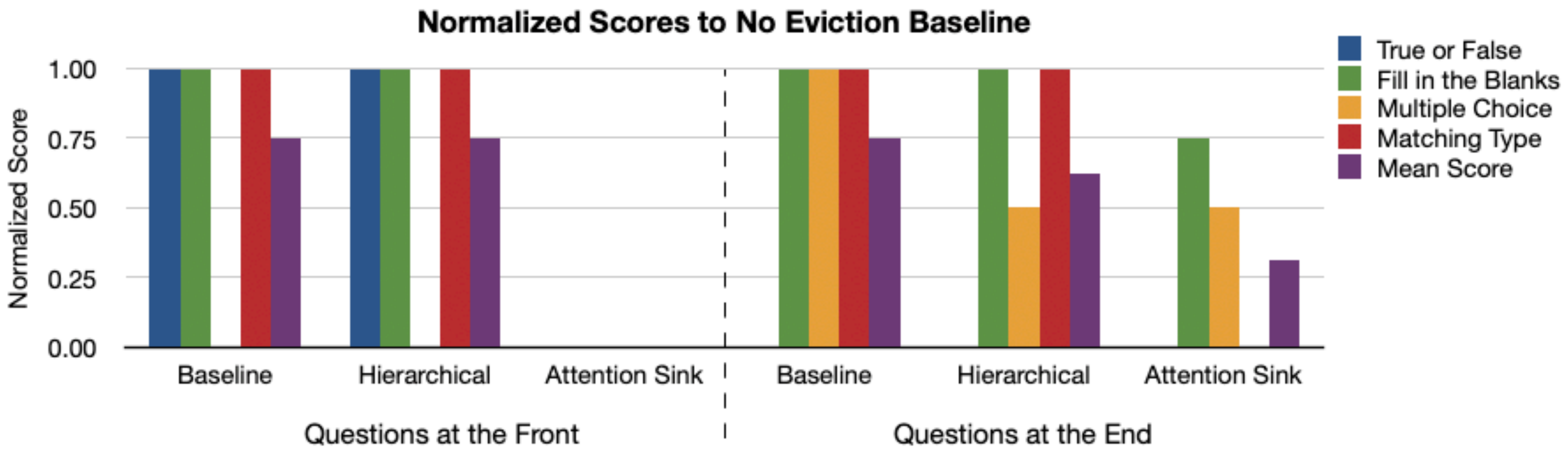
Popularity Plots for AttentionSink







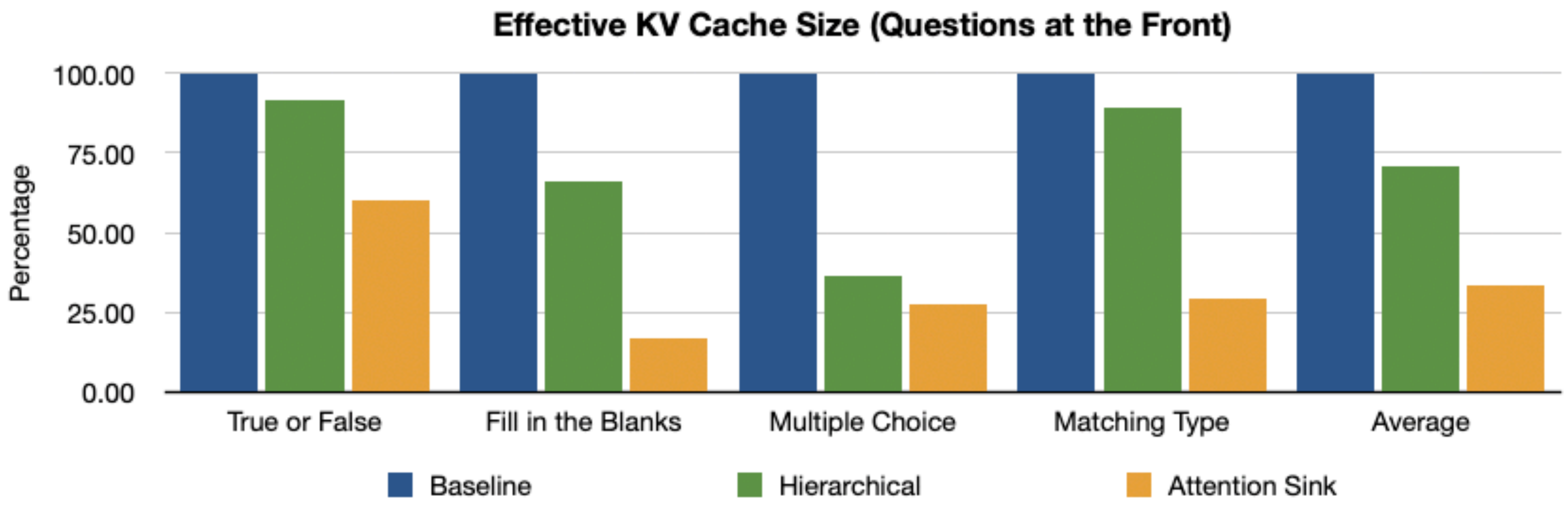
Results

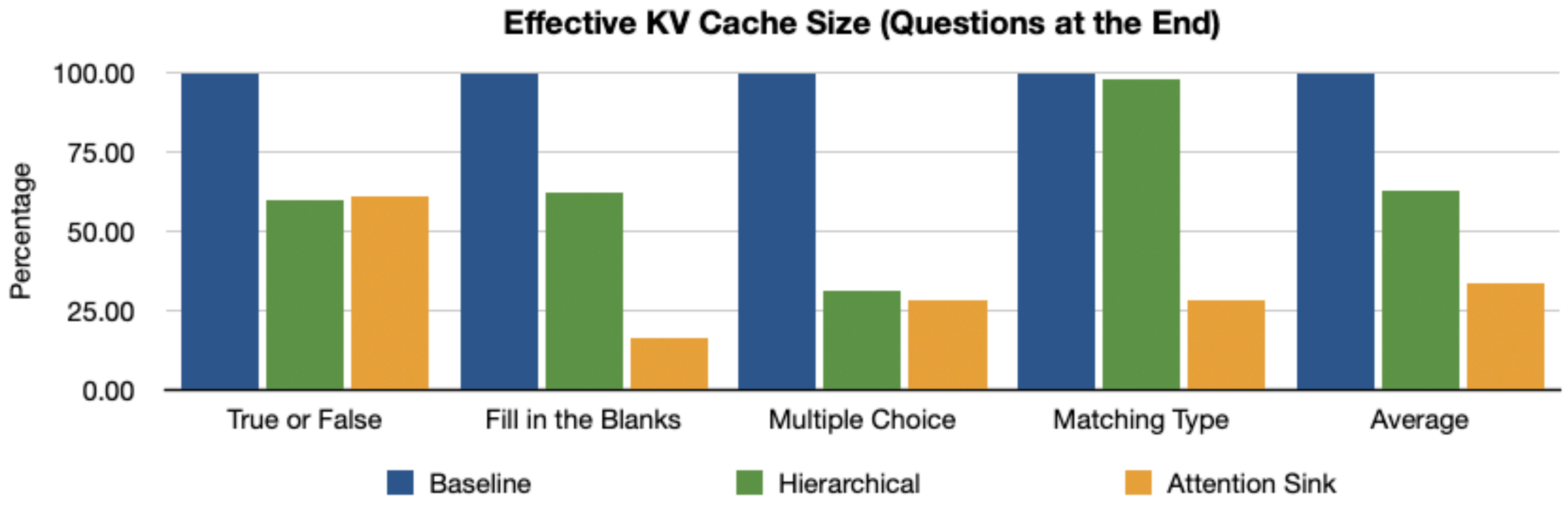


- Hierarchical Policy shows relatively similar scores with Baseline.
- Attention Sink suffers significantly when questions are at the front.
  - Creates its own questions, or wrongly answers one question.

Results

- Hierarchical Policy shows diverse KV cache sizes which demonstrates some its ability to dynamically adapt to diverse workloads.
- Attention Sink aggressively drops all tokens in the middle.





Discussions

- Hierarchical KV Cache Compression Policy performs similarly with baseline with no eviction, demonstrating effectiveness.
- Effective KV Cache size varies among different types of questions which shows inherent ability to dynamically adapt to token popularity patterns.
- Demonstrates weakness of static and aggressive KV Cache compression techniques.
- This compression technique can be a stepping stone for a heterogenous memory system:
    - Highly popular KV entries placed in GPU memory
    - Less popular KV cache entries evicted to CPU memory