HOLD ONTO THAT THOUGHT: ASSESSING KV CACHE COMPRESSION ON REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) have demonstrated remarkable performance on long-context tasks, but are often bottlenecked by memory constraints. Namely, the KV cache, which is used to significantly speed up attention computations, grows linearly with context length. A suite of compression algorithms has been introduced to alleviate cache growth by evicting unimportant tokens. However, several popular strategies are targeted towards the prefill phase, i.e., processing long prompt context, and their performance is rarely assessed on reasoning tasks requiring long decoding. In particular, short but complex prompts, such as those in benchmarks like GSM8K and MATH500, often benefit from multi-step reasoning and self-reflection, resulting in thinking sequences thousands of tokens long. In this work, we benchmark the performance of several popular compression strategies on long-reasoning tasks. For the non-reasoning Llama-3.1-8B-Instruct, we determine that no singular strategy fits all, and that performance is heavily influenced by dataset type. However, we discover that H2O and our decoding-enabled variant of SnapKV are dominant strategies for reasoning models, indicating the utility of heavy-hitter tracking for reasoning traces. We also find that eviction strategies at low budgets can produce longer reasoning traces, revealing a tradeoff between cache size and inference costs.

1 INTRODUCTION

Large language models (LLMs) have demonstrated remarkable performance on complex NLP tasks that require multi-step reasoning. Unlike summarization tasks (Bai et al., 2023; Fabbri et al., 2021) and keyword tracking tasks (Hsieh et al., 2024), which scale task complexity with context length, reasoning benchmarks challenge models to generate answers that are not clearly contained in the prompt. Such tasks include reading comprehension (Dua et al., 2019; Yu et al., 2020), commonsense reasoning (Zellers et al., 2019; Talmor et al., 2018; Geva et al., 2021), first-order logic (Han et al., 2022; Kwon et al., 2025), and mathematical problem-solving (Cobbe et al., 2021).

Reasoning benchmarks differ from long-context tasks in that they normally compel the LLM to provide answers that are longer than the question itself. This can pose a serious resource problem for the LLM, as past token key and value embeddings are maintained in memory to avoid redundant attention calculations. This key-value (KV) cache grows linearly with sequence length, which can result in memory blowup for older or single-GPU setups. Furthermore, specialized reasoning models such as DeepSeek-R1 (Guo et al., 2025) and the Llama-Nemotron series are known to output excessively long reasoning traces (Cai et al., 2025; Fatemi et al., 2025) which outnumber the length of the prompt itself by hundreds to thousands of tokens.

A defining characteristic of reasoning benchmarks is that LLM responses corresponding to their queries often far exceed the length of the input question. This can pose a serious resource bottleneck: past token key and value embeddings must be stored in VRAM to avoid redundant attention calculations. On resource-constrained systems, the resulting linear memory growth can quickly lead to exhaustion. The problem is further amplified by reasoning-focused models such as the DeepSeek-R1 (Guo et al., 2025) and NVIDIA Llama-Nemotron series, which are known for producing exceptionally verbose reasoning traces spanning thousands of tokens (Cai et al., 2025; Fatemi et al., 2025).

To address the memory demands of long sequences, numerous KV cache compression methods have been proposed. These techniques generally maintain a fixed KV cache size by selectively discarding

tokens deemed "unimportant". However, defining token importance" is non-trivial, and different approaches rely on distinct heuristics: attention scores (Zhang et al., 2023; Liu et al., 2023; Li et al., 2024), cosine similarity (Liu et al., 2024a; Han et al., 2023), embedding norms (Devoto et al., 2024), and head-specific token-type preferences (Ge et al., 2023). Despite this variety, most evaluations of cache compression have focused on long-context benchmarks such as LongBench (Bai et al., 2023) and RULER (Hsieh et al., 2024), or on heterogeneous batteries like LM Eval Harness ((Gao et al., 2024)), rather than tasks where the generation length, not the prompt, dominates memory usage.

In this work, we conduct a comprehensive assessment of the major state-of-the-art KV cache compression strategies across eight reasoning benchmarks: FOLIO (Han et al., 2022), DROP (Dua et al., 2019), GSM8K (Cobbe et al., 2021), MATH-500 (Lightman et al., 2023), ReClor (Yu et al., 2020), StrategyQA (Geva et al., 2021), CommonSenseQA (Talmor et al., 2018), and OpenBookQA (Mihaylov et al., 2018). Together, these benchmarks span four critical reasoning categories: reading comprehension, common sense, logical reasoning, and mathematical reasoning. We evaluate these strategies on Llama-3.1-8B-Instruct as well as four reasoning models: Llama-3.1-Nemotron-Nano-8B-v1, DeepSeek-R1-Distill-Llama-8B, and DeepSeek-R1-Distill-Qwen-7B/14B. By focusing on long-generation rather than long-prompt scenarios, our study fills a notable gap in the existing literature. Our primary contributions are threefold:

A comprehensive benchmark: We conduct a comprehensive evaluation of major KV cache compression strategies, including StreamingLLM, H2O (Zhang et al., 2023), a decoding-enabled SnapKV (Li et al., 2024), R-KV (Cai et al., 2025), and KNorm (Devoto et al., 2024), across a suite of eight benchmarks spanning mathematical, logical, and commonsense reasoning. We evaluate over several realistic settings, cache, and max token budgets for a single-GPU system.

Renewed attention for attention-based compression: Our analysis reveals that classical attention-based "heavy-hitter" strategies, which evict tokens based on accumulated attention scores, significantly outperform other methods, even defeating full-cache reasoning occasionally. Namely, this includes H2O and our novel and simple extension of SnapKV (prompt-only compression method) to a decoding-enabled variant, SnapKV-**D**ecoding. Both methods, especially SnapKV-D, win over *all* budgets and datasets for reasoning models.

A library for analyzing decoding compression: We implement a fork of the NVIDIA `kvpress`[1] library, which adds support for decoding phase compression. The current `kvpress` is restricted to prefill phase (prompt) compression, which is only suitable for non-reasoning models and long prompt tasks such as LongBench and RULER. Furthermore, extend decoding-phase support for all 25+ compression strategies present within `kvpress`, providing a open-source playground for analyzing end-to-end KV cache compression strategies.

1.1 PRELIMINARIES

In this section, we briefly review the concepts of large language models, LLM inference and autoregressive generation, the KV cache, and the chain-of-thought (CoT) reasoning.

Transformer Architectures and Autoregressive Generation. Modern Large Language Models (LLMs) predominantly operate as autoregressive, decoder-only Transformers (Vaswani et al., 2017; Radford et al., 2019; Achiam et al., 2023; Touvron et al., 2023). This architecture generates text sequentially, producing one token at a time by conditioning on the entire preceding sequence of tokens, which includes both the initial prompt and any previously generated output (Brown et al., 2020). Importantly, the model's ability to maintain coherent and contextually relevant generation over time is crucial to its capabilities, especially in tasks requiring reasoning or narrative development (Lee et al., 2024; Zhang et al., 2025).

Self-Attention Mechanism and the KV Cache Bottleneck. During generation, a query (q) vector for the current token is *attends* to a series of Key (k) and Value (v) vectors corresponding to every token in the preceding context. In this process, notably, for the generation of every new token, the entire sequence of Key and Value vectors for *all* previous tokens should be accessed. To avoid recomputing these K-V pairs at each step, they are stored in the Key-Value (KV) cache, the size of which grows linearly with the sequence length (n), resulting in an $O(n)$ memory complexity that creates a significant bottleneck. Formally, for a sequence of n tokens, we denote the query cache $Q_l^h \in \mathbb{R}^{n \times d}$,

[1] https://github.com/NVIDIA/kvpress

key cache $K_l^h \in \mathbb{R}^{n \times d}$, and value cache $V_l^h \in \mathbb{R}^{n \times d}$, where d is the embedding dimension, l is the layer, and h denotes a head for multi-head attention layers (Vaswani et al., 2017). The dot-product self-attention mechanism is defined as $A_l^h(Q_l^h, K_l^h, V_l^h) = \text{softmax}(Q_l^h(K_l^h)^\top/\sqrt{d})V_l^h$. To avoid linear scaling with sequence length, *token eviction* methods, the key focus of work, discard embeddings of previous tokens which are no longer "important" to the current decoding step.

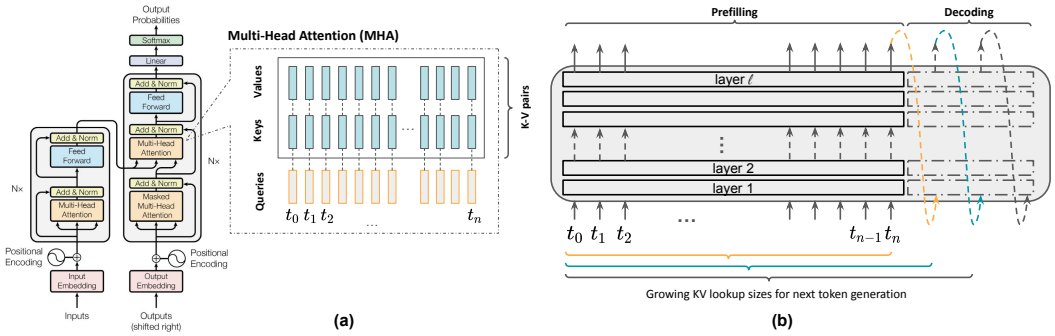


Figure 1: **Overview of the Transformer Decoder Architecture and the Inference Bottleneck.** (a) The standard Transformer decoder architecture (left) and the Multi-Head Attention (MHA) mechanism (right). In MHA, Query vectors representing the current context attend to a sequence of Key-Value (K-V) pairs from all previous tokens. Such K-V pairs form the basis of the KV cache. (b) The two-phase inference process in autoregressive generation. During Prefilling, the tokens in the input context are processed in parallel to populate the initial KV cache across all layers. During Decoding, each new token is generated sequentially. This requires recomputing the entire set of the preceding KV entries at each step, causing the lookup size to grow linearly with the sequence length.

Figure 1 (a) exhibits the Q, K, and V vectors along with the self-attention mechanism. Figure 1 (b) demonstrate the decoding KV cache bottleneck on memory.

Chain-of-Thought and Multi-Step Reasoning. While many long-context applications involve processing long prompts, a critical class of tasks requires long-form generation from short and complex prompts. Prompting strategies such as Chain-of-Thought (CoT) encourage models to externalize their reasoning process, generating intermediate "thinking" steps that can extend for hundreds or thousands of tokens to solve a problem (Wei et al., 2022; Wang et al., 2022). Benchmarks like GSM8K (Cobbe et al., 2021) are representative of this domain, where the path to the correct answer necessitates a lengthy, self-generated chain of reasoning that stresses the models' decoding-phase memory limits.

2 Related Work

2.1 KV Cache Compression

KV cache compression is a rich field of study composed of strategies ranging from quantization (Hooper et al., 2024; Ashkboos et al., 2024; Liu et al., 2024b) to offloading methods that move the entire cache to the CPU which is significantly less memory bound (Sun et al., 2024; Chen et al., 2024; Tang et al., 2024). However, in this work, we are focused on strategies which maintain a constant cache size, thus permitting arbitrary generation length.

2.1.1 Token Eviction

A primary line of research for mitigating the memory burden of the KV cache involves *token eviction*. These methods aim to reduce the cache size by selectively removing or merging tokens deemed less important. To achieve this, multiple approaches have been developed, including recency-based approaches such as simple sliding window (Beltagy et al., 2020), importance-based methods that retain "attention sinks" or heavy-hitter tokens from the prompt (Xiao et al., 2023; Zhang et al., 2023; Li et al., 2024; Liu et al., 2023), dynamically adjustment of KV caches per layer for optimal efficiency-utility balancing (Cai et al., 2024), redundancy-aware techniques that merge semantically

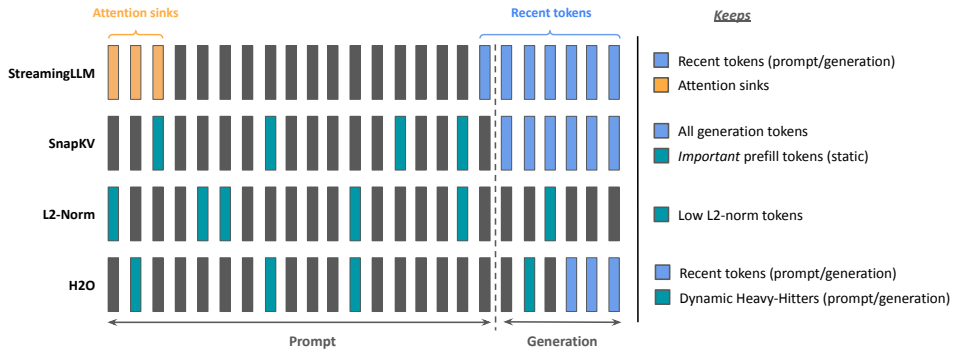


Figure 2: A Conceptual Comparison of Token Retention Strategies in Different KV Cache Compression Methods. Each row illustrates a method's logic for retaining tokens (colored) versus evicting them (gray) from the KV cache during a long sequence divided into a prefill and decoding phase.

similar states (Cai et al., 2025). Figure 2 provides a conceptual comparison of the most important approaches we cover in this work.

StreamingLLM (Xiao et al., 2023) is based on the critical observation that in autoregressive models, a small number of initial tokens act as "attention sinks," consistently receiving a large proportion of attention scores regardless of their semantic relevance. StreamingLLM's strategy is therefore to permanently cache the KV states of the first few (e.g., four) tokens, which serve as the attention sinks, and combine them with a sliding window of the most recent tokens. **H2O** (Zhang et al., 2023) dynamically identifies important or "heavy hitter" tokens based on their cumulative attention scores received during generation. The H2O cache is composed of two parts: a budget for the most recent tokens and a budget for the H2 tokens. **SnapKV** (Li et al., 2024) focuses primarily on compressing the KV cache of the initial prompt during the prefill stage. SnapKV uses a small "observation window" at the end of the prompt to predict importance. The attention scores from queries in this observation window are aggregated to "vote" for important positions (heavy hitters) in the prefix. A distinct and computationally efficient approach, which we refer to as the **KNorm** strategy (Devoto et al., 2024), bypasses the need for attention scores entirely. Specifically, the authors observe that tokens whose key vectors have a low L_2 norm consistently attract high attention scores from subsequent queries.

2.2 Benchmarking Reasoning

GSM8K (Grade School Math 8K) is a widely-used dataset of grade-school level math word problems that require a sequence of elementary arithmetic operations to solve (Cobbe et al., 2021). More advanced challenges are drawn from the **MATH-500** dataset (Lightman et al., 2023), which contains competition-level problems across algebra, geometry, and number theory. **ReClor** (Yu et al., 2020) is a reading comprehension dataset built from GMAT and LSAT logical reasoning questions. Similarly, **LogiQA** (Liu et al., 2020) provides multiple-choice questions from civil service exams that require a deep understanding of logical puzzles and deductions. For evaluating capabilities in more formal systems, the **FOLIO** (Han et al., 2022) dataset assesses natural language reasoning in the context of First-Order Logic (FOL). Beyond formal and mathematical logic, a significant portion of research focuses on commonsense reasoning. **StrategyQA** (Geva et al., 2021) tests a model's ability to infer the implicit reasoning steps needed to answer a yes/no question by asking for the underlying strategy. Another tested benchmarks is **CommonsenseQA**, which tests a model's ability to reason with general world knowledge. Finally, the integration of textual understanding with quantitative skills is measured by benchmarks such as **DROP** (Dua et al., 2019). This reading comprehension dataset is unique in that answering its questions requires performing discrete operations like counting, sorting, or simple arithmetic directly on the information presented in a context paragraph.

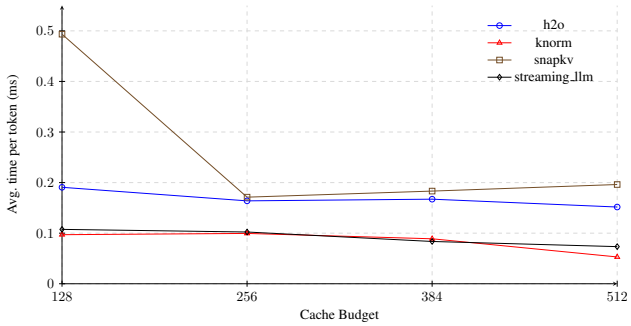


Figure 3: **Latency vs Budget.** Average generation time per token (ms) versus KV-cache budget for eviction strategies. KNorm and StreamingLLM speed up markedly with larger budgets, H2O improves more modestly, while SnapKV-D is slow at small budgets.

3 Experiments & Analysis

3.1 Setup

KV Compression Methods. We test **H2O**, **StreamingLLM**, **KNorm**, our own decoding-variant of SnapKV which we call **SnapKV-D**, and **ShadowKV** (Sun et al., 2024). We note that ShadowKV uses the CPU to offload the cache and thus is not a true compression strategy. However, offloading strategies represent an important class of compression methods; thus, we include them as a baseline.

Models. For our core experiments, we test the base, non-reasoning Llama-3.1-8B-Instruct and three reasoning models: DeepSeek-R1-Distill-Qwen-7B, Nemotron-Nano-8B-v1 and DeepSeek-R1-Distill-Llama-8B. For a larger model comparison, we use DeepSeek-R1-Distill-Qwen-14B.

Datasets. We divide our benchmark into 4 distinct groups: **(1) Reading Comprehension:** DROP, ReClor; **(2) Logical Reasoning:** StrategyQA, FOLIO; **(3) Commonsense Reasoning:** OpenBookQA (OBQA), CommonsenseQA (CSQA); **(4) Math Reasoning:** MATH-500, GSM8K. For each dataset, we randomly sample 100 questions for two different seeds.

Performance. For benchmarking the individual compression strategies, we use the NVIDIA kvpress library, which natively provides most of the targeted algorithms. We provide each dataset to each model over the cache budgets $\{128, 256, 384, 512\}$. Each model is allowed to generate a maximum of 2048 new tokens via greedy decoding. This token limit is enforced to better simulate a resource-constrained setting for inference, but also based on mean generation lengths reported in Table 3. However, we do perform a max token ablation to study its effect on dominant method.

We use author-recommended hyperparameters for all methods. Accuracy benchmarks were performed on an HPC cluster using an NVIDIA RTX A6000 48GB GPU. Latency benchmarks were performed on an NVIDIA H100 PCIe 80GB GPU.

3.2 Latency Experiment

Although this benchmark is primarily concerned with accuracy, we assess the latency of our tested methods in Figure 3 to gather a more complete picture of efficiency.

3.3 Max Token Ablation

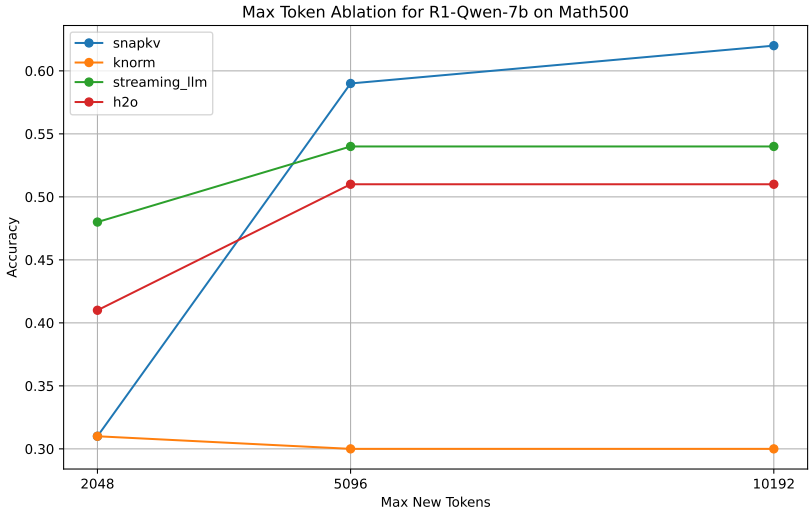
As explained in setup, we chose a max token length of 2048 both because we find that the mean token length over datasets is under this budget and to better assess performance in a compute-bound setting. However, we study the effect of max token generation on method performance under a fixed budget of 1024 for MATH500 for R1-Distill-Qwen7B. We find that performance improves significantly for all methods initially, but then SnapKV-D overtakes all methods for all other max token limits.

Table 1: Strategy versus Accuracy. Cache budgets = [128, 256, 384, 512]. Llama-3.1-8B-Instruct=ML, Deepssek-R1-Distill-Qwen-7B=DQ,=DQ,-Nemotron-Nano-8B-v1==LN, DeepSeek-R1-Distill-Llama-8B=DL.

Benchmark	ML				DQ				LN				DL			
	128	256	384	512	128	256	384	512	128	256	384	512	128	256	384	512
GSM8K full	0.88				0.70				0.64				0.70			
shadowkv	0.32				0.47				0.44				0.51			
h2o	**0.63**	**0.77**	0.82	0.83	0.21	0.44	0.51	0.52	0.22	0.45	0.52	0.57	0.37	0.53	0.62	0.61
knorm	0.07	0.53	0.73	0.82	0.00	0.00	0.08	0.15	0.01	0.02	0.09	0.18	0.00	0.09	0.19	0.28
snapkv	0.53	0.55	0.56	0.53	**0.67**	**0.67**	**0.70**	**0.71**	**0.65**	**0.63**	**0.66**	**0.66**	**0.72**	**0.72**	**0.74**	**0.72**
streaming_llm	0.26	0.75	**0.84**	**0.87**	0.02	0.19	0.32	0.44	0.03	0.20	0.40	0.53	0.06	0.25	0.39	0.56
Math500 full	0.39				0.47				0.45				0.46			
shadowkv	0.22				0.33				0.28				0.34			
h2o	**0.23**	**0.24**	0.26	**0.31**	0.14	0.21	0.29	0.31	0.16	0.24	0.31	0.33	0.20	0.31	0.36	0.36
knorm	0.03	0.15	0.21	0.23	0.00	0.01	0.03	0.05	0.01	0.01	0.03	0.06	0.00	0.01	0.02	0.06
snapkv	0.20	0.21	0.19	0.20	**0.38**	**0.36**	**0.36**	**0.32**	**0.41**	**0.44**	**0.45**	**0.43**	**0.42**	**0.44**	**0.41**	**0.41**
streaming_llm	0.07	0.20	**0.27**	0.24	0.03	0.12	0.19	0.26	0.02	0.13	0.22	0.34	0.03	0.09	0.21	0.29
CSQA full	0.77				0.67				0.51				0.75			
shadowkv	0.20				0.20				0.20				0.20			
h2o	0.67	0.70	**0.73**	**0.76**	0.44	0.61	**0.60**	**0.64**	0.47	0.49	**0.52**	0.51	0.48	0.72	0.73	**0.73**
knorm	0.41	**0.72**	**0.73**	0.73	0.05	0.13	0.30	0.42	0.36	0.40	0.44	0.46	0.05	0.28	0.54	0.66
snapkv	**0.70**	0.64	0.71	0.72	**0.65**	**0.62**	0.59	0.61	**0.49**	**0.50**	0.51	**0.53**	**0.74**	**0.73**	**0.74**	**0.73**
streaming_llm	0.24	0.65	0.70	0.74	0.08	0.14	0.31	0.48	0.36	0.44	0.46	0.50	0.04	0.14	0.35	0.50
OBQA full	0.84				0.78				0.64				0.84			
shadowkv	0.31				0.31				0.31				0.31			
h2o	**0.83**	**0.81**	0.83	**0.82**	0.41	0.62	**0.68**	0.67	0.59	0.58	0.58	0.61	0.45	0.78	**0.84**	**0.84**
knorm	0.40	0.77	**0.84**	0.77	0.05	0.05	0.23	0.34	0.31	0.43	0.46	0.57	0.04	0.26	0.57	0.70
snapkv	0.73	0.77	0.72	0.76	**0.71**	**0.75**	**0.68**	**0.76**	**0.68**	**0.63**	**0.66**	**0.66**	**0.82**	**0.83**	0.83	0.81
streaming_llm	0.17	0.70	0.80	0.80	0.03	0.12	0.29	0.36	0.35	0.46	0.50	0.61	0.06	0.16	0.34	0.53
ReClor full	0.60				0.45				0.48				0.51			
shadowkv	0.27				0.27				0.27				0.27			
h2o	0.32	0.56	**0.60**	0.58	0.01	0.04	0.18	0.28	0.20	0.22	0.35	**0.40**	0.03	**0.08**	**0.23**	**0.38**
knorm	0.01	0.19	0.46	**0.59**	0.00	0.00	0.01	0.01	0.01	0.03	0.07	0.07	0.00	0.00	0.02	0.10
snapkv	**0.53**	**0.57**	0.58	0.55	**0.45**	**0.39**	**0.40**	**0.43**	**0.42**	**0.42**	**0.42**	0.37	**0.50**	**0.08**	0.05	0.05
streaming_llm	0.05	0.21	0.59	0.58	0.00	0.01	0.01	0.04	0.03	0.06	0.09	0.14	0.00	0.00	0.01	0.06
DROP full	0.15				0.16				0.11				0.14			
shadowkv	0.28				0.14				0.11				0.09			
h2o	0.12	**0.14**	**0.17**	**0.17**	0.04	0.07	0.10	0.10	0.05	0.06	0.10	0.09	0.06	0.07	0.10	0.11
knorm	0.01	0.08	0.13	0.13	0.00	0.01	0.01	0.03	0.01	0.01	0.02	0.03	0.00	0.01	0.01	0.05
snapkv	**0.15**	0.12	0.11	0.12	**0.13**	**0.11**	**0.12**	**0.16**	**0.11**	**0.11**	**0.12**	**0.10**	**0.17**	**0.15**	**0.15**	**0.16**
streaming_llm	0.09	0.11	0.15	0.16	0.04	0.05	0.08	0.13	0.03	0.02	0.06	0.08	0.02	0.02	0.09	0.13
StrategyQA full	0.83				0.67				0.89				0.74			
shadowkv	0.68				0.60				0.65				0.80			
h2o	0.77	0.83	0.86	**0.87**	0.31	0.58	**0.69**	**0.65**	0.72	0.83	**0.84**	0.83	0.24	0.64	**0.73**	**0.74**
knorm	0.50	**0.84**	0.85	0.85	0.01	0.12	0.41	0.54	0.39	0.51	0.65	0.73	0.06	0.34	0.54	0.67
snapkv	**0.78**	0.78	0.81	0.76	**0.60**	**0.59**	0.57	0.63	**0.83**	**0.85**	**0.84**	**0.84**	**0.68**	**0.66**	0.64	0.68
streaming_llm	0.14	0.74	**0.87**	0.81	0.01	0.06	0.23	0.41	0.24	0.40	0.50	0.68	0.03	0.13	0.37	0.53
FOLIO full	0.51				0.36				0.36				0.47			
shadowkv	0.33				0.33				0.33				0.33			
h2o	0.22	**0.43**	0.41	0.43	0.03	0.21	0.23	0.23	0.22	0.36	0.35	0.37	0.07	0.37	0.41	**0.46**
knorm	0.02	0.28	0.39	0.38	0.00	0.01	0.03	0.05	0.03	0.04	0.07	0.13	0.00	0.03	0.11	0.21
snapkv	**0.44**	0.40	**0.45**	**0.46**	**0.30**	**0.25**	**0.31**	**0.29**	**0.38**	**0.42**	**0.41**	**0.41**	**0.46**	**0.45**	**0.49**	**0.46**
streaming_llm	0.03	0.09	0.25	0.35	0.00	0.01	0.02	0.03	0.03	0.03	0.06	0.15	0.00	0.01	0.04	0.09

Table 2: **R1-Distill-Qwen14B**. Cache budgets = [128, 256, 384, 512]. We examine the performance of various compression methods for a larger reasoning model. Winner per budget in bold.

Method	GSM8K				MATH500			
	128	256	384	512	128	256	384	512
full	0.81				0.47			
shadowkv	0.53				0.38			
h2o	0.33	0.56	0.62	0.64	0.2	0.27	0.31	0.31
knorm	0	0.02	0.08	0.21	0	0	0	0.02
snapkv	**0.80**	**0.82**	**0.81**	**0.78**	**0.43**	**0.44**	**0.42**	**0.45**
streaming_llm	0.07	0.27	0.5	0.59	0.02	0.17	0.26	0.35



3.4 Large Model Comparison

We determine whether our observed trends hold for a larger reasoning model, R1-Distill-Qwen-14B. We examine the performance of all methods on the more challenging GSM8K and MATH500. Unsurprisingly, base accuracies do improve, but more importantly, we observe that again, the heavy-hitter methods H2O and SnapKV-D outperform their competitors by a significant margin indicating that larger reasoning models still benefit from attention-based eviction.

3.5 Effects of Cache Budget on Output Length

In this section, we display the mean number of output tokens in Figure 4 for all tested models and strategies on GSM8K, a dataset with grade-school math questions which requires multi-step reasoning. As we can see, it is possible for models with small budgets to eventually generate longer answers than the full cache itself. At lower budgets, the removal of critical tokens can result in longer, less coherent reasoning traces. We demonstrate this phenomenon in Section A.2.

3.6 Cache Budget vs Output Length

We study the effects of cache budget on output generation lengths in Figure 4. Fascinatingly, lower budgets are capable of triggering longer reasoning traces, revealing a hidden tradeoff between cache budget and inference costs specifically for reasoning models. KNorm, arguably the lowest performing strategy, tends to cause the greatest elongation of outputs. In Section A.2, we examine one such non-terminating output that demonstrates repetitive, dead-end chain-of-thought.

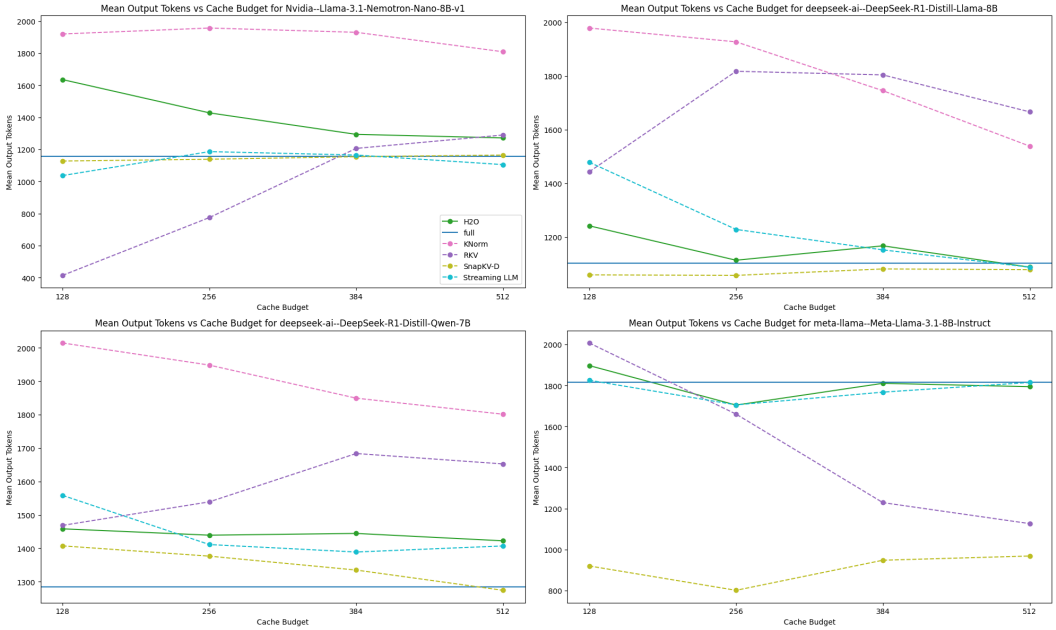
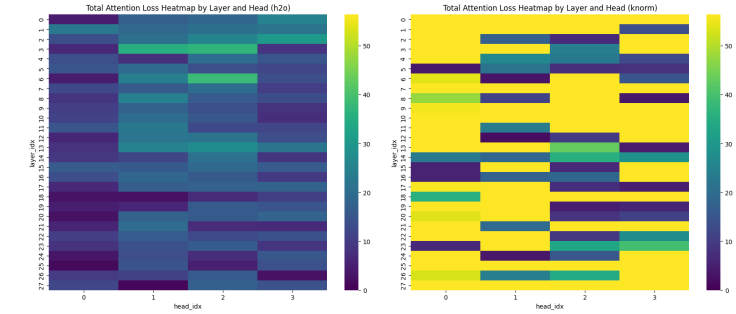
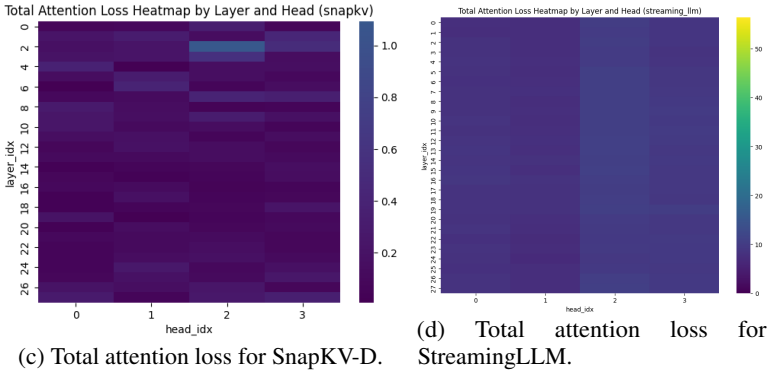


Figure 4: **Budget vs Output Length**. We observe that several compression methods, especially at lower budgets, ultimate produce longer outputs than the base full cache model.



(a) Total attention loss for H2O. (b) Total attention loss for KNorm.



(c) Total attention loss for SnapKV-D. (d) Total attention loss for StreamingLLM.

Figure 5: **Attention Loss Heatmaps**. We visualize attention loss at every compression step for a question in GSM8K. The attention loss over each head is summed up over every layer. We observe that higher performance correlates with less attention loss.

3.7 Attention as an Indicator of Performance

All eviction methods tested propose to capture important tokens via ad-hoc strategies either explicitly or implicitly relating to attention: H2O examines at accumulated attention across the entire sequence,

SnapKV examines attention with regards to an observation window, KNorm uses small key norms as a proxy for high-attention, StreamingLLM retains recent tokens and the sink (initial) to effectively approximate the attention distribution. We examine how much attention is actually lost through these various compression methods. For this study, we compare the absolute difference between the attention scores of each head pre- and post-eviction for GSM8K, which we refer to as attention loss following other recent literature (Liu et al., 2024a; Devoto et al., 2024). The trend is striking: in order of least to most attention loss: SnapKV-D, H2O, StreamingLLM, and KNorm. *This correlates with performance.*

3.8 Ablation Results

In this section, we review the high-level performance trends of our selected KV cache compression strategies.

Attention is the most versatile estimator for reasoning models. SnapKV-D and H2O are the most dominant, significantly outcompeting nearly all compression strategies across all budget constraints and datasets for our reasoning models. These methods rely on accumulated attention scores to determine the most important tokens to retain. (i.e., "heavy hitters"). While both maintain a recency window, H2O is focused on heavy hitters with regard to the current token, while SnapKV (and consequently, SnapKV-D) finds heavy hitters with respect to an observation window at the end of the current sequence. The latter approach is more effective, routinely defeating H2O. The utility of the observation window was previously known to work well for prompt compression, but not for long decoding phases. *We find that this trend generalizes for model size, max token settings, and budgets.*

No singular strategy is dominant for the non-reasoning Llama-3.1-8B-Instruct. For models that do not produce reasoning traces, the optimal choice of strategy is dataset-dependent. For example, while StreamingLLM excels at GSM8K, it is less effective on all other task types. While SnapKV-D and H2O are capable of winning 2/4 settings for several datasets, other methods, such as R-KV and KNorm, can convincingly defeat them over the remaining budgets.

Eviction lags full cache performance for reasoning models. According to Table 1, all compression strategies can defeat the full cache performance of Llama-3.1-8B-Instruct on at least one setting (with H2O and SnapKV-D frequently achieving this). However, for reasoning models, this trend only holds true for SnapKV-D. While H2O is still second best compared to other strategies, it significantly lags full cache performance on nearly every dataset. As noted in Figure 4, H2O results in significantly longer reasoning traces than SnapKV-D, which occasionally do not terminate. It is possible that allowing a less restrictive constraint on the maximum number of new tokens can alleviate this performance drop, though this would result in longer inference.

Cache compression can cost more computation. Interestingly, according to Figure 4, eviction strategies can result in more "talkative" reasoning models, generating noticeably longer sequences compared to the full cache setting, while this does not occur for Llama-3.1-8B-Instruct. In Section A.2, we show this phenomenon at work, where KNorm results in long circular babble for Deepseek-R1-Distill-Llama-8B that never produces an answer. This is problematic for resource-constrained settings: while cache eviction can reduce peak memory usage, it can result in significantly longer auto-regression. At lower budgets, eviction occurs more frequently over shorter stretches of context, resulting in the eviction of critical reasoning tokens, which can result in longer reasoning.

4 Conclusion

In this work, we comprehensively assessed the performance of several popular KV cache compression strategies on reasoning tasks for the non-reasoning Llama-3.1-8B-Instruct and several popular reasoning models. For the non-reasoning model, we find that no singular method is dominant. However, for reasoning models, we demonstrate that attention-based eviction methods such as H2O and SnapKV-D perform extraordinarily well on a variety of reasoning tasks, even occasionally exceeding full cache performance. Furthermore, this generalizes to a larger model, R1-Distill-Qwen-14B. We also discovered that it is possible, especially at lower budgets, for compression strategies to produce longer reasoning traces, thus revealing an under-considered tradeoff between memory and inference costs. For future work, even larger models should be assessed, along with larger cache budgets, to fully assess the limits of the cache compression/performance tradeoff.

5 Ethics Statement

We do not anticipate any notable negative societal impacts stemming from this results discussed in this work. However, we do note that KV cache compression is capable of altering outputs and thus must be exercised with care in sensitive domains to ensure that content is not produced which significantly deviates from uncompressed models.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. Quarot: Outlier-free 4-bit inference in rotated llms. *Advances in Neural Information Processing Systems*, 37:100213–100240, 2024.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. Longbench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*, 2023.

Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, et al. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*, 2024.

Zefan Cai, Wen Xiao, Hanshi Sun, Cheng Luo, Yikai Zhang, Ke Wan, Yucheng Li, Yeyang Zhou, Li-Wen Chang, Jiuxiang Gu, et al. R-kv: Redundancy-aware kv cache compression for training-free reasoning models acceleration. *arXiv preprint arXiv:2505.24133*, 2025.

Zhuoming Chen, Ranajoy Sadhukhan, Zihao Ye, Yang Zhou, Jianyu Zhang, Niklas Nolte, Yuandong Tian, Matthijs Douze, Leon Bottou, Zhihao Jia, et al. Magicpig: Lsh sampling for efficient llm generation. *arXiv preprint arXiv:2410.16179*, 2024.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Alessio Devoto, Yu Zhao, Simone Scardapane, and Pasquale Minervini. A simple and effective l_2 norm-based strategy for kv cache compression. *arXiv preprint arXiv:2406.11430*, 2024.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs. *arXiv preprint arXiv:1903.00161*, 2019.

Alexander R Fabbri, Wojciech Kryściński, Bryan McCann, Caiming Xiong, Richard Socher, and Dragomir Radev. Summeval: Re-evaluating summarization evaluation. *Transactions of the Association for Computational Linguistics*, 9:391–409, 2021.

Mehdi Fatemi, Banafsheh Rafiee, Mingjie Tang, and Kartik Talamadupula. Concise reasoning via reinforcement learning. *arXiv preprint arXiv:2504.05185*, 2025.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 07 2024. URL `https://zenodo.org/records/12608602`.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. Model tells you what to discard: Adaptive kv cache compression for llms. *arXiv preprint arXiv:2310.01801*, 2023.

Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies. *Transactions of the Association for Computational Linguistics*, 9:346–361, 2021.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Insu Han, Rajesh Jayaram, Amin Karbasi, Vahab Mirrokni, David P Woodruff, and Amir Zandieh. Hyperattention: Long-context attention in near-linear time. *arXiv preprint arXiv:2310.05869*, 2023.

Simeng Han, Hailey Schoelkopf, Yilun Zhao, Zhenting Qi, Martin Riddell, Wenfei Zhou, James Coady, David Peng, Yujie Qiao, Luke Benson, et al. Folio: Natural language reasoning with first-order logic. *arXiv preprint arXiv:2209.00840*, 2022.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Yakun S Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *Advances in Neural Information Processing Systems*, 37:1270–1303, 2024.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.

Yejin Kwon, Daeun Moon, Youngje Oh, and Hyunsoo Yoon. Logicqa: Logical anomaly detection with vision language model generated questions. *arXiv preprint arXiv:2503.20252*, 2025.

Seungpil Lee, Woochang Sim, Donghyeon Shin, Wongyu Seo, Jiwon Park, Seokki Lee, Sanha Hwang, Sejin Kim, and Sundong Kim. Reasoning abilities of large language models: In-depth analysis on the abstraction and reasoning corpus. *ACM Transactions on Intelligent Systems and Technology*, 2024.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37:22947–22970, 2024.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2023.

Jian Liu, Leyang Cui, Hanmeng Liu, Dandan Huang, Yile Wang, and Yue Zhang. Logiqa: A challenge dataset for machine reading comprehension with logical reasoning. *arXiv preprint arXiv:2007.08124*, 2020.

Minghui Liu, Tahseen Rabbani, Tony O'Halloran, Ananth Sankaralingam, Mary-Anne Hartley, Furong Huang, Cornelia Fermüller, and Yiannis Aloimonos. Hashevict: A pre-attention kv cache eviction strategy using locality-sensitive hashing. *arXiv preprint arXiv:2412.16187*, 2024a.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Advances in Neural Information Processing Systems*, 36:52342–52364, 2023.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. *arXiv preprint arXiv:2402.02750*, 2024b.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*, 2018.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Hanshi Sun, Li-Wen Chang, Wenlei Bao, Size Zheng, Ningxin Zheng, Xin Liu, Harry Dong, Yuejie Chi, and Beidi Chen. Shadowkv: Kv cache in shadows for high-throughput long-context llm inference. *arXiv preprint arXiv:2410.21465*, 2024.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. Commonsenseqa: A question answering challenge targeting commonsense knowledge. *arXiv preprint arXiv:1811.00937*, 2018.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context llm inference. *arXiv preprint arXiv:2406.10774*, 2024.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2023.

Weihao Yu, Zihang Jiang, Yanfei Dong, and Jiashi Feng. Reclor: A reading comprehension dataset requiring logical reasoning. *arXiv preprint arXiv:2002.04326*, 2020.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

Jinghan Zhang, Xiting Wang, Weijieying Ren, Lu Jiang, Dongjie Wang, and Kunpeng Liu. Ratt: A thought structure for coherent and correct llm reasoning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 26733–26741, 2025.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023.

A Appendix

A.1 Generation Lengths

In Table 3, we report the mean generation lengths for all methods across and models on MATH-500, the dataset which elicits the longest responses. To keep these tables concise, we averaged output lengths over all budgets.

Table 3: Mean output tokens generated by different models under various strategies for Math500.

Strategy	Model	Mean Output Tokens
full	Nvidia--Llama-3.1-Nemotron-Nano-8B-v1	1616.275
full	deepseek-ai--DeepSeek-R1-Distill-Llama-8B	1727.18
full	deepseek-ai--DeepSeek-R1-Distill-Qwen-7B	1728.84
h2o	Nvidia--Llama-3.1-Nemotron-Nano-8B-v1	1753.4075
h2o	deepseek-ai--DeepSeek-R1-Distill-Llama-8B	1763.805
h2o	deepseek-ai--DeepSeek-R1-Distill-Qwen-7B	1767.7275
knorm	Nvidia--Llama-3.1-Nemotron-Nano-8B-v1	1987.80875
knorm	deepseek-ai--DeepSeek-R1-Distill-Llama-8B	2001.07625
knorm	deepseek-ai--DeepSeek-R1-Distill-Qwen-7B	1967.4575
snapkv	Nvidia--Llama-3.1-Nemotron-Nano-8B-v1	1667.895
snapkv	deepseek-ai--DeepSeek-R1-Distill-Llama-8B	1790.165
snapkv	deepseek-ai--DeepSeek-R1-Distill-Qwen-7B	1794.0275
streaming_llm	Nvidia--Llama-3.1-Nemotron-Nano-8B-v1	1375.70375
streaming_llm	deepseek-ai--DeepSeek-R1-Distill-Llama-8B	1655.0375
streaming_llm	deepseek-ai--DeepSeek-R1-Distill-Qwen-7B	1698.25625

A.2 A Verbose, Wrong Answer

Below, we present a verbose, incorrect answer generated for Deepseek-R1-Distill-Llama-8B via KNorm on a budget of 256 tokens.

```
Define:

- Let a be the value of a small button (SB).
- Then, 1 SS = a
- 1 large button (LB) is worth 3 small buttons, so LB = 3a
- Alternatively, 1 LB = 3a
- 1 LS can be converted to 1 LB or 3 SS, so 1 LS = 1 LB or 3*(1a)? Wait,
    small stickers themselves are worth 1a.

Hold on, perhaps clarifying units.

Wait, sorry, the problem states:

"Lorraine has ... stickers and large buttons..."

Wait, maybe it's better to model it with equations.

Let me denote:

- Value of 1 small sticker = x
- Then, 1 medium sticker would be higher, but in our case, 1 large
    sticker (LS) is worth either 1 large button (LB) or 3 small buttons (
    SB).

But we don't know whether a small button is same as a small sticker? Let'
    s see.

The problem says:
```

```
- A large button is worth three small stickers. So 1 LB = 3 SS.

So, 1 SS = 1/3 LB.

Or, alternatively, 1 LB = 3 SS.

So, yeah, 1 LB = 3 SS.

On the flip side, 1 LS can be either 1 LB or 3 SS.

But note that 1 LB is equal to 3 SS.

So, the question becomes:

Traders use sticker-based currencies.

Given that, let me
```

A.3 LLM USAGE

We used LLMs to stylize tables and figures. We also used LLMs to polish grammar but not to produce any writing itself.