

844 A Limitations, Future Work and Impact

845 **Larger Model Sizes, Longer Contexts, and Higher Compression Ratios.** In this work, we focus
846 on models ranging from 1B to 32B parameters, context lengths up to 32K tokens, and compression
847 ratios up to $8\times$. Exploring even larger models, longer contexts, and higher compression ratios remains
848 an exciting avenue for future research.

849 **Integration with Other Attention Mechanisms.** We demonstrated DMS using the standard
850 attention mechanism common to transformer models such as Llama and Qwen2.5. Extending DMS
851 to alternative attention variants, such as Multi-head Latent Attention (DeepSeek-AI, 2024) and Native
852 Sparse Attention (Yuan et al., 2025), represents a promising direction for future investigation.

853 **Broader Impact.** Our approach does not introduce novel risks; however, it may amplify existing
854 concerns associated with large-scale reasoning models. For a detailed analysis of these risks, we refer
855 readers to Zhou et al. (2025). Prior to releasing our top-performing DMS model, we will perform
856 comprehensive evaluations using established benchmarks and safety tests such as PurpleLlama (AI,
857 2023), garak (Derczynski et al., 2024), and Aegis (Ghosh et al., 2024). Furthermore, we plan to
858 distribute the model under an appropriate license restricting its use to research and development.

B Additional Details for Retrofitting

Contrary to DMC, in which append/accumulate decisions are extracted from key vectors, we extract those decisions from queries in Equation 1. By doing so, we affect fewer query–key pairs, because the decision neurons are set to 0 after extraction. We train with a batch size of 1024 according to the original recipe (Touvron et al., 2023). The training context is set to 4096 for the ablations with Llama 3.2 1B Instruct and Llama 2 7B models. We set the training context to 8192 tokens for Llama 3.1 8B and the R1-distilled models, as AIME and MATH 500 tasks benefit from longer context lengths. The details about retrofitting can be found in Table 2 for Qwen-R1 models and in Table 3 for Llama models. Additional common hyperparameters can be found in Table 4. Unless stated otherwise, in all DMS models, eviction is delayed by 16 tokens, resulting in a sliding window of 16 tokens.

Contrary to DMC (Nawrot et al., 2024), we have not employed the third, fine-tuning phase in which the model is additionally trained with a fixed compression ratio, as we found little-to-no benefit from it in DMS.

Table 2: Retrofitting configuration for thinking models. In all tuning stages, we use 8K context length.

| Model Family | #Params | Neuron Decay | | | CR1→CR4 | | | CR4→CR8 | |
|--------------|---------|--------------|-------|------|---------|-------|------|---------|------|
| | | BS | Steps | LR | BS | Steps | LR | Steps | LR |
| Qwen-R1 | 1.5B | 1K | 2K | 1e-5 | 1K | 9K | 1e-5 | 12K | 1e-5 |
| | 7B | 1K | 2K | 1e-6 | 1K | 9K | 1e-6 | 12K | 2e-5 |
| | 32B | 1K | 2K | 1e-6 | 256 | 12K | 1e-5 | 0 | - |

Table 3: Retrofitting Configuration for Llama Models. For all tuning stages, we use a batch size of 1024. The column labeled RF CTX indicates the retrofitting context length. The columns labeled CR+1 specify the number of steps and learning rate schedule used to increment the compression ratio by one. The notation $x \xrightarrow{\cos} y$ indicates that the phase employed a cosine learning rate decay, starting from x at the beginning and decaying to y at the end of the phase. All models except Llama 2 7B and Llama 3.1 8B were retrofitted using logit distillation. For Llama 2 7B and Llama 3.1 8B, standard language modeling loss (LM loss) was used instead.

| Model Family | #Params | Method | RF CTX | Neuron Decay | | CR + 1 | | Tuning | |
|--------------|---------|--------|--------|--------------|------|--------|------|--------|--------------------------------|
| | | | | Steps | LR | Steps | LR | Steps | LR |
| Llama 3.2 | 1B | DMS | 4K | 2K | 1e-6 | 3K | 1e-5 | 0 | - |
| | | DMC | 4K | 2K | 1e-6 | 6K | 1e-5 | 2K | $1e-5 \xrightarrow{\cos} 1e-6$ |
| Llama 2 | 7B | DMS | 4K | 2K | 3e-5 | 3K | 3e-5 | 0 | - |
| | | DMC | 4K | 2K | 3e-5 | 6K | 3e-5 | 2K | $3e-5 \xrightarrow{\cos} 3e-6$ |
| Llama 3.1 | 8B | DMS | 8K | 2K | 3e-5 | 2K | 3e-5 | 0 | - |

Table 4: Additional Training Hyperparameters.

| Hyperparameter | Value |
|----------------|-------|
| Optimiser | AdamW |
| Clip Gradient | 1.0 |
| Weight Decay | 0.1 |
| Adam β_1 | 0.9 |
| Adam β_2 | 0.95 |

For training, we used NVIDIA H100 GPUs. We implement our models using Megatron-LM (NVIDIA, 2024) and train in bfloat16 precision. Optimizer states and gradient accumulation are stored in fp32. Using an unoptimized implementation with tensor parallelism set to 8 and pipeline parallelism set

875 to 2, one retrofitting step for the 32B Qwen-R1 model (batch size 256, context length 8192) takes
876 approximately 18 seconds on 256 NVIDIA H100 GPUs. A single checkpoint of this model, including
877 optimizer states, occupies approximately 430GB of disk space. The project consumed roughly 200K
878 GPU hours, including preliminary experiments.

C Training Data

For the Qwen-R1 models, we utilize logit distillation leveraging the OpenR1-Math-220k dataset. This dataset contains high-quality reasoning traces sampled from DeepSeek R1. To further enhance data quality, we apply a filtering step using Math-Verify (Kydliček and Gandenberger, 2025), retaining only traces resulting in correct mathematical solutions.

For the Llama 3.2 1B Instruct model, the training corpus comprises two main components: (1) a carefully curated set of programming language examples covering languages such as Python, C, and C++, and (2) synthetic data generated by prompting the model. In particular, we utilize the Llama 3.2 1B Instruct model itself to produce completions for the one-dimensional linear algebra subset of the DeepMind mathematics dataset (Saxton et al., 2019), which follows the structured format:

Task format in one-dimensional linear algebra

Solve $aX + b = cX + d$ for X .

Llama 3.2 1B prompt for generating responses

<|start_header_id|>system<|end_header_id|>

Cutting Knowledge Date: December 2023
Today Date: 23 July 2024

You are a helpful

↪ assistant.<|eot_id|><|start_header_id|>user<|end_header_id|>

Given the following problem, reason and give a final answer to the
↪ problem.

Problem: Solve $5*b - 2355 = -50*b - 2740$ for b .

Your response should end with "The final answer is [answer]" where

↪ [answer] is the response to the

↪ problem.<|eot_id|><|start_header_id|>assistant<|end_header_id|>

In contrast with the data mixture for Qwen-R1 models, we do not perform correctness filtering on this synthetic, model-generated dataset.

D Additional Downstream Task Evaluations for Ablations on Model Design

We present the results for a Llama 3.1 8B model trained with DMS (16-token sliding window with delayed eviction) and language modeling loss (Table 5). The model achieved CR 8 \times , maintaining favorable scores. However, we note that a Llama 1B Instruct model trained with language modeling loss deteriorated quickly during training. In addition, Table 6 summarizes evaluations of the 1B Instruct model with different sliding window sizes and delayed/immediate token eviction strategies.

Table 5: Evaluation of DMS-enabled Llama 3.1 8B base models trained with standard language modeling loss instead of logit distillation with 2000 iterations per CR increase. The models were evaluated using 8-shot GSM8K, 5-shot MMLU, 3-shot Needle in a Haystack and Variable Tracking, 0-shot ARC-Challenge, and 0-shot HellaSwag benchmarks.

| Method | ARC-C | GSM8K | HS | MMLU | NIAH | VT |
|---------|-------------|-------------|-------------|-------------|--------------|-------------|
| Vanilla | 49.9 | 56.9 | 75.8 | 63.0 | 100.0 | 99.6 |
| CR2 | | | | | | |
| DMS | 54.0 | 56.6 | 79.5 | 65.0 | 100.0 | 92.0 |
| Quest | 49.9 | 54.9 | 75.8 | 63.0 | 100.0 | 99.6 |
| CR4 | | | | | | |
| DMS | 52.5 | 55.9 | 79.3 | 64.6 | 100.0 | 90.2 |
| Quest | 49.9 | 53.8 | 75.8 | 63.0 | 100.0 | 99.6 |
| CR6 | | | | | | |
| DMS | 54.6 | 54.6 | 79.5 | 63.7 | 99.8 | 82.8 |
| Quest | 49.9 | 51.9 | 75.8 | 63.0 | 100.0 | 99.6 |
| CR8 | | | | | | |
| DMS | 53.8 | 54.3 | 78.8 | 63.5 | 95.6 | 90.2 |
| Quest | 49.9 | 46.6 | 75.8 | 63.0 | 100.0 | 99.6 |

Table 6: Additional evaluations of DMS-enabled Llama 3.2 1B Instruct with different sizes of the sliding window and delayed/immediate token eviction strategies

| Method | ARC-C | GSM8K | HS | MMLU |
|--------------------------|-------|-------|------|------|
| CR2 | | | | |
| DMS + win=0, immediate | 24.8 | 9.2 | 30.0 | 24.6 |
| DMS + win=16, immediate | 31.5 | 40.0 | 44.1 | 47.1 |
| DMS + win=16, delayed | 31.1 | 46.9 | 44.0 | 48.0 |
| DMS + win=64, immediate | 31.1 | 44.0 | 43.9 | 48.1 |
| DMS + win=64, delayed | 31.1 | 45.1 | 43.9 | 48.1 |
| DMS + win=256, immediate | 31.7 | 45.2 | 43.8 | 47.7 |
| DMS + win=256, delayed | 31.1 | 46.2 | 43.9 | 48.0 |
| CR3 | | | | |
| DMS + win=0, immediate | 20.6 | 6.8 | 26.3 | 23.4 |
| DMS + win=16, immediate | 31.7 | 12.8 | 43.6 | 43.0 |
| DMS + win=16, delayed | 30.9 | 46.5 | 43.9 | 45.2 |
| DMS + win=64, immediate | 31.1 | 32.4 | 43.8 | 47.1 |
| DMS + win=64, delayed | 31.1 | 46.2 | 43.9 | 47.8 |
| DMS + win=256, immediate | 31.9 | 31.5 | 43.8 | 46.8 |
| DMS + win=256, delayed | 31.4 | 46.4 | 43.9 | 47.8 |
| CR4 | | | | |
| DMS + win=0, immediate | 21.0 | 1.2 | 26.0 | 23.0 |
| DMS + win=16, immediate | 31.2 | 4.2 | 43.7 | 40.7 |
| DMS + win=16, delayed | 31.9 | 42.3 | 43.9 | 40.2 |
| DMS + win=64, immediate | 30.7 | 21.2 | 43.7 | 45.8 |
| DMS + win=64, delayed | 31.5 | 45.2 | 43.8 | 44.8 |
| DMS + win=256, immediate | 31.2 | 34.0 | 43.8 | 46.7 |
| DMS + win=256, delayed | 31.5 | 44.7 | 43.9 | 47.9 |

899 E Additional Downstream Evaluations for DMC and DMS

900 In Table 7 we show that DMS can extrapolate beyond the retrofitting context length of 4K, whereas
 901 DMC may fail to do so. In Table 8, we show a comparison between Vanilla model, DMS, Quest, and
 902 DMC on Llama 2 7B.

Table 7: Needle in the Haystack and Variable Tracking results for 1B parameter Llama 3.2 Instruct model. We note that in contrast to DMC, DMS can extrapolate beyond the retrofitting context length. Note that on the heavily compressible Variable Tracking task, DMS beats even the vanilla model.

| Method/Task Context | NIAH | | | VT | | |
|------------------------|-------------|-------------|-------------|-------------|-------------|-------------|
| | 3K | 4K | 8K | 3K | 4K | 8K |
| Vanilla | 99.4 | 96.4 | 97.2 | 61.4 | 55.8 | 41.2 |
| CR2 | | | | | | |
| TOVA | 62.8 | 65.2 | 75.0 | 56.0 | 56.2 | 49.8 |
| H2O | 29.0 | 34.0 | 37.0 | 25.6 | 27.4 | 21.4 |
| Quest | 99.2 | 95.8 | 97.4 | 60.0 | 53.0 | 36.4 |
| DMC | 99.0 | 0.0 | 0.0 | 62.4 | 0.0 | 0.0 |
| DMS | 99.0 | 97.8 | 99.4 | 72.0 | 63.2 | 56.0 |
| CR3 | | | | | | |
| TOVA | 25.8 | 40.2 | 41.6 | 38.4 | 45.2 | 40.6 |
| H2O | 16.6 | 17.2 | 19.8 | 15.6 | 17.6 | 13.4 |
| Quest | 99.0 | 95.6 | 97.0 | 60.4 | 50.4 | 31.8 |
| DMC | 99.0 | 1.8 | 0.0 | 46.4 | 0.2 | 1.2 |
| DMS | 99.2 | 93.6 | 24.2 | 76.2 | 69.2 | 58.8 |
| CR4 | | | | | | |
| TOVA | 16.8 | 28.0 | 26.4 | 31.4 | 33.8 | 30.2 |
| H2O | 9.4 | 13.4 | 12.8 | 11.8 | 12.6 | 11.0 |
| Quest | 98.4 | 95.8 | 97.6 | 57.4 | 49.6 | 32.4 |
| DMC | 97.0 | 0.0 | 0.0 | 48.6 | 4.0 | 0.8 |
| DMS | 99.4 | 96.8 | 12.2 | 74.8 | 67.6 | 57.2 |

Table 8: Results for base Llama 2 7B parameter models. Both DMS and DMC were trained using LM-loss without logit distillation. Since these models are not instruction-tuned, we evaluate with 8-shot prompting on GSM8K, 5-shot on MMLU, 1-shot Needle in a Haystack, and zero-shot on ARC-Challenge and HellaSwag. We highlight that DMS used fewer than half the training tokens of DMC, without employing logit distillation or fine-tuning with a fixed compression ratio and cosine learning rate decay, with the latter technique being previously shown to significantly boost model performance in DMC (Nawrot et al., 2024).

| Method | ARC-C | GSM8K | HS | MMLU | NIAH |
|---------|-------|-------|------|------|--------|
| Vanilla | 45.6 | 14.9 | 75.5 | 45.4 | 100.00 |
| CR4 | | | | | |
| DMS | 45.8 | 14.2 | 76.0 | 43.7 | 100.0 |
| Quest | 45.6 | 14.5 | 75.5 | 45.4 | 100.0 |
| DMC | 46.2 | 12.2 | 76.3 | 43.9 | 100.0 |
| CR8 | | | | | |
| DMS | 46.2 | 10.5 | 76.4 | 40.2 | 60.0 |
| Quest | 45.6 | 11.6 | 75.5 | 45.4 | 100.0 |
| DMC | 44.7 | 10.0 | 75.4 | 41.7 | 100.0 |

F Downstream Results Significance

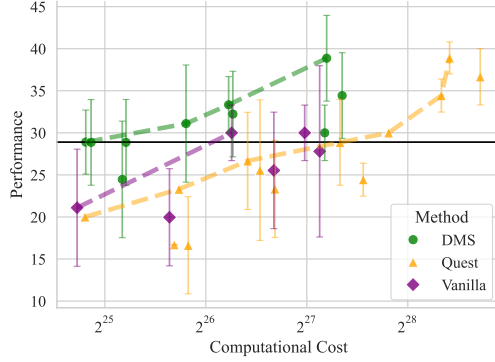
We provide further analysis regarding the statistical significance and robustness of our experimental results. Specifically, we report standard deviations for the Llama 3.2 1B Instruct models in Table 9, and quantify the average Pareto improvement in Tables 10 and 11. To precisely measure the Pareto improvement, we extract Pareto frontiers for DMS, the best KV cache reduction baseline, and the vanilla baseline from Figures 2 and 3. Then, for each task and model size, we identify the largest common budget interval I shared by each pair of methods A and B, and compute the average improvement as:

$$\frac{\int_{x \in I} (A(x) - B(x)) dx}{|I|}$$

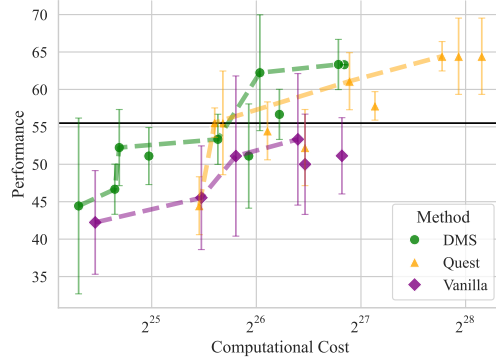
where $A(x)$ and $B(x)$ denote the best accuracy achieved by method A and B, respectively, at budget x . For budget values not explicitly measured, we employ linear interpolation.

Table 9: Results from Table 1 expanded with standard deviation as computed by Language Model Evaluation Harness (Gao et al., 2024).

| Method | ARC-C | GPQA | GSM8K | HS |
|---------|----------------|----------------|----------------|----------------|
| Vanilla | 31.2 \pm 1.4 | 25.0 \pm 2.0 | 44.9 \pm 1.4 | 43.4 \pm 0.5 |
| CR2 | | | | |
| DMS | 31.3 \pm 1.4 | 25.7 \pm 2.1 | 46.6 \pm 1.4 | 43.3 \pm 0.5 |
| TOVA | 29.6 \pm 1.3 | 25.2 \pm 2.1 | 45.0 \pm 1.4 | 42.8 \pm 0.5 |
| H2O | 31.1 \pm 1.4 | 26.8 \pm 2.1 | 44.0 \pm 1.4 | 42.9 \pm 0.5 |
| Quest | 31.2 \pm 1.4 | 25.0 \pm 2.0 | 45.1 \pm 1.4 | 43.4 \pm 0.5 |
| CR3 | | | | |
| DMS | 31.1 \pm 1.4 | 24.6 \pm 2.0 | 45.5 \pm 1.4 | 43.3 \pm 0.5 |
| TOVA | 30.0 \pm 1.3 | 23.7 \pm 2.0 | 40.1 \pm 1.4 | 42.5 \pm 0.5 |
| H2O | 31.2 \pm 1.4 | 24.3 \pm 2.0 | 32.9 \pm 1.3 | 42.1 \pm 0.5 |
| Quest | 31.2 \pm 1.4 | 25.0 \pm 2.0 | 44.7 \pm 1.4 | 43.4 \pm 0.5 |
| CR4 | | | | |
| DMS | 31.1 \pm 1.4 | 24.3 \pm 2.0 | 41.0 \pm 1.4 | 43.4 \pm 0.5 |
| TOVA | 29.0 \pm 1.3 | 23.7 \pm 2.0 | 20.2 \pm 1.1 | 41.8 \pm 0.5 |
| H2O | 27.5 \pm 1.3 | 23.7 \pm 2.0 | 14.7 \pm 1.0 | 41.3 \pm 0.5 |
| Quest | 31.2 \pm 1.4 | 25.0 \pm 2.0 | 39.9 \pm 1.3 | 43.4 \pm 0.5 |



(a) Qwen-r1 1.5B on AIME 24



(b) Qwen-r1 7B on AIME 24

Figure 5: Figure 2 AIME24 task with standard deviation over three seeds marked.

Table 10: Compute–Accuracy Pareto frontier difference. We use linear interpolation for the unknown values of the frontier. NA denotes that the projections of the Pareto frontiers on the budget axis are disjoint.

| Method | AIME 24 | | | MATH 500 | | | GPQA Diamond | | |
|------------------|------------|------------|------------|------------|------------|------------|--------------|------------|------------|
| | 1.5B | 7B | 32B | 1.5B | 7B | 32B | 1.5B | 7B | 32B |
| DMS vs Vanilla | 5.3 | 8.1 | 5.2 | 3.7 | 2.5 | 2.0 | 1.0 | 1.5 | 3.8 |
| Quest vs Vanilla | −2.7 | 5.2 | 2.7 | −1.0 | 0.0 | −0.1 | 1.1 | −0.0 | 2.9 |
| DMS vs Quest | 8.4 | 3.3 | 2.5 | 4.5 | 2.3 | 1.6 | NA | NA | 0.9 |

Table 11: Memory–Accuracy Pareto frontier difference. We use linear interpolation for the unknown values of the frontier. NA denotes that the projections of the Pareto frontiers on the budget axis are disjoint.

| Method | AIME 24 | | | MATH 500 | | | GPQA Diamond | | |
|-----------------|------------|-------------|------------|------------|------------|------------|--------------|------------|------------|
| | 1.5B | 7B | 32B | 1.5B | 7B | 32B | 1.5B | 7B | 32B |
| DMS vs Vanilla | 6.2 | 10.2 | 8.5 | 4.2 | 2.3 | 1.4 | 2.5 | 4.3 | 3.4 |
| TOVA vs Vanilla | 5.3 | −0.2 | 2.6 | 1.3 | −1.4 | 1.8 | −1.1 | −2.1 | 2.1 |
| DMS vs TOVA | 1.1 | 10.7 | 5.9 | 3.2 | 3.7 | −0.1 | NA | 6.1 | 1.3 |

G Evaluation Details

G.1 Implementation of TOVA, H2O, Quest, and DMC

For TOVA (Oren et al., 2024), H2O (Zhang et al., 2023a), and Quest (Tang et al., 2024), we calculate the KV-budget by summing the input length and the maximum generation length, then dividing by the compression ratio. For H2O, the KV-budget is evenly split between the recent cache and the heavy-hitter cache. During evaluation, memory-optimising methods such as TOVA and H2O first perform a standard prefill phase until the KV-budget is reached and subsequently switch to their respective memory-efficient mechanisms.

Quest (Tang et al., 2024), unlike TOVA, H2O, DMC, and DMS, does not reduce the KV-cache memory footprint. Thus, following the authors’ recommendations, we permit Quest to perform prefilling using full dense attention and set the block size to 16. This configuration provides Quest with an advantage over the other methods. Additionally, we employ a separate top-k for each query head, resulting in an increased number of memory transfers for Quest compared to DMS, DMC, TOVA, and H2O. However, the computational cost remains similar. We use this approach as Quest was originally designed for models without GQA, and we wanted to avoid a custom modification that could potentially degrade the performance.

For DMC, we follow the implementation described in the original paper (Nawrot et al., 2024).

G.2 Downstream Tasks

We evaluate all downstream tasks in a zero-shot setting, unless stated otherwise.

For GSM8K (Cobbe et al., 2021), MMLU (Hendrycks et al., 2021a), ARC-Challenge (Clark et al., 2018), and HellaSwag (Zellers et al., 2019), we use the Language Model Evaluation Harness (Gao et al., 2024), version 0.4.3.

For Needle in a Haystack (NIAH) (Kamradt, 2023) and Variable Tracking (VT), we adopt the evaluation implementation provided by RULER (Hsieh et al., 2024) and use the context length defined in the retrofitting procedure. For NIAH, we utilize the essay version with a single needle, whereas for VT, we utilize the version with 40 variable chains and 0 hops, filled with repeating text.

For AIME24,⁵ GPQA \diamond (Rein et al., 2024), and MATH 500 (Lightman et al., 2024), we evaluate models using the Search and Learn framework (version 0.1.0) (Snell et al., 2024; Beeching et al., 2024), with math-parsing functionality derived from MathVerify (version 1.0.0) (Kydliček and Gandenberger, 2025).

For few-shot tasks from Language Model Evaluation Harness we directly utilize the framework to provide the few-shot examples. For RULER (Hsieh et al., 2024), we use the example generator to sample few-shot examples. Below we present remaining prompts that were used during the evaluation, except the prompts to base models, which were set to unaltered task input, and prompts for zero-shot evaluation of instruction tuned models, which were set to task inputs wrapped with HuggingFace tokenizer chat template⁶.

For GSM8K zero-shot evaluation, we adopt the prompt from Meta.

⁵https://huggingface.co/datasets/HuggingFaceH4/aime_2024

⁶https://huggingface.co/docs/transformers/en/chat_templating

GSM8K zero-shot prompt

<|start_header_id|>system<|end_header_id|>

Cutting Knowledge Date: December 2023

Today Date: 23 July 2024

You are a helpful

↪ assistant.<|eot_id|><|start_header_id|>user<|end_header_id|>

Given the following problem, reason and give a final answer to the
↪ problem.

Problem: ___problem_text___

Your response should end with "The final answer is [answer]" where

↪ [answer] is the response to the

↪ problem.<|eot_id|><|start_header_id|>assistant<|end_header_id|>

950

951 For Qwen-R1 AIME 24, MATH 500 and GPQA \diamond we adopt the prompts from Open-R1 repository⁷.

AIME 24 and MATH 500 prompts

<|User|>Solve the following math problem efficiently and clearly.

↪ The last line of your response should be of the following format:

↪ 'Therefore, the final answer is: $\boxed{\text{ANSWER}}$ \$. I hope it is

↪ correct' (without quotes) where ANSWER is just the final number

↪ or expression that solves the problem. Think step by step before

↪ answering.

___problem_text___<|Assistant|><think>

952

GPQA Diamond prompt

<|User|>Answer the following multiple choice question. The last line

↪ of your response should be of the following format: 'Answer:

↪ \$LETTER' (without quotes) where LETTER is one of ABCD. Think step

↪ by step before answering.

___problem_text___<|Assistant|><think>

953

⁷<https://github.com/huggingface/open-r1>

H Influence of KV Cache on Inference Latency

In this section, we provide a simplified analysis estimating the proportion of inference latency introduced by reading from the key-value cache to the entire latency of the step, during a single auto-regressive inference step of an LLM on a GPU. Our calculations are based on the architecture of the Llama 3 model family. Specifically, we derive sample equations for Llama 3.1 8B, parameters of which are listed below.

| Parameter | Value | Description |
|-----------|--------|--------------------------------------|
| n | 32 | Number of layers |
| d | 4096 | Hidden dimension |
| d_{ff} | 14336 | Internal dimension of the MLP layers |
| d_{kv} | 1024 | Dimension of the Key/Value sequences |
| V | 128256 | Vocabulary size |

The estimates can be expressed in terms of batch size B and sequence length L , which determine the number of tokens in the KV cache. The number of floating-point operations (FLOPs) can be approximated as

$$\text{FLOPS}(B, L) \approx nB(6dd_{ff} + 4d^2 + 4dd_{kv} + 4dL) + 2BdV. \quad (2)$$

This calculation considers only major matrix-vector multiplications (assuming two FLOPs per multiply-accumulate operation), omitting minor operations such as normalization and pointwise non-linearities.

Similarly, we estimate the number of reads from the High Bandwidth Memory as:

$$\text{Reads}(B, L) \approx n(6dd_{ff} + 4d^2 + 4dd_{kv} + 4BLd_{kv}) + 2dV \quad (3)$$

assuming 2 bytes per parameter (16-bit precision). Note that only the KV cache ($4nBLd_{kv}$) scales with batch size and sequence length. As a sanity check, we confirm that $\text{Reads}(1, 0)/2 \approx 7.5B$ approximate the model’s parameter count (without $0.5B$ for the input embedding table, which does not have to be entirely read during an inference step). Finally, the approximations for Llama 3.1 8B have the following form:

$$\text{FLOPS}(B, L) \approx 1.45 \cdot 10^9 B + 5.24 \cdot 10^5 BL \quad (4)$$

$$\text{Reads}(B, L) \approx 1.50 \cdot 10^{10} + 1.31 \cdot 10^5 BL. \quad (5)$$

For the remaining calculations, we use the peak performance values for NVIDIA H100 SXM (<https://www.nvidia.com/en-us/data-center/h100/>) for 16-bit calculations without 2:4 sparsity:

| | |
|----------------------------------|--------------|
| BFLOAT16 Tensor Core performance | 989.5 TFLOPS |
| GPU Memory bandwidth | 3.35 TB/s |

Since memory reads are significantly slower than computations, the latency contribution from KV cache reads ($1.31 \times 10^5 BL$ term) dominates at larger sequence lengths and batch sizes. Thus, KV cache size is a critical factor in inference latency for long sequences.

The inference latency per step can be approximated as

$$\max\left(\frac{\text{FLOPS}(B, L)}{989.5 \text{ TFLOPS}}, \frac{\text{Reads}(B, L)}{3.35 \text{ TB/s}}\right), \quad (6)$$

assuming ideal overlap between computation and memory operations. Approximating KV cache reads as $4nBLd_{kv}$ and following the same calculations for other Llama and Qwen models, we visualize the fraction of latency contributed by KV cache reads to the latency of entire inference steps (Figure 6).

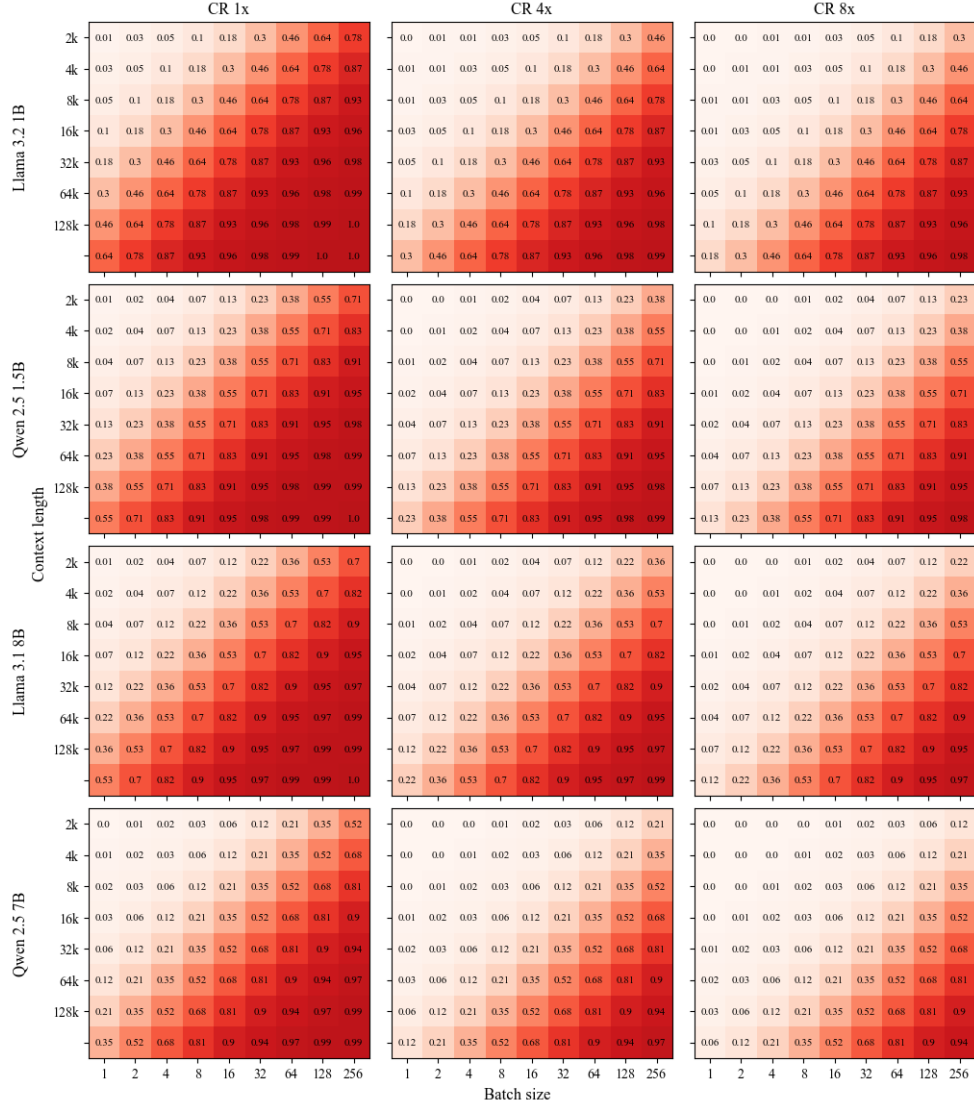


Figure 6: Percentage of total latency attributed to KV cache reads. Those reads clearly dominate latency as batch size and sequence length increase. When the KV cache is compressed (CR 4 \times and 8 \times), more tokens can be accommodated before the latency of reading the KV cache becomes an issue.