PARTREP: Learning What to Repeat for Decoder-only LLMs

Anonymous ACL submission

Abstract

While decoder-only LLMs excel at a vast array of natural language tasks, it suffers from an asymmetric information flow induced by causal attention: later tokens are richer in contextual grounding than earlier ones. A simple and effective remedy is prompt repetition—just appending a second copy of prompt before generation can redistribute grounding across positions and improve reasoning performance. However, full repetition of the original prompt doubles the KV cache footprint and quadruples attention cost during prefill, making it impractical for long-context settings. We propose **PARTREP**, a selective augmentation method that appends only the most informative tokens—rather than the entire prompt. We use token-wise negative log-likelihood (NLL) as a selection signal, motivated by the hypothesis that less predictable tokens are less recoverable from surrounding context and therefore benefit more from late-position repetition. To avoid the heavy cost of a full forward pass for scoring, we train a lightweight gate that predicts high-NLL tokens from early-layer hidden states, enabling token selection during mid-prefill via early exit. Across eight benchmarks (including MMLU, GSM8K, and RULER) and three model families (Qwen2.5, Llama3.2, Gemma4), PARTREP retains most of the gains of full repetition while using only 59.4% of its KV cache and 79.0% of its prefill FLOPs.

1 Introduction

Decoder-only large language models (LLMs) have become the dominant architecture in natural language processing, driving significant advancements across a wide range of generative and analytical tasks (Huang et al., 2023; Achiam et al., 2023). A defining feature of these models is causal attention, which allows each token to attend to all preceding tokens, while masking out the future ones. Although this design enables effective autoregressive generation, it also induces an asymmetric information flow: later tokens have richer contextual grounding than earlier ones (Springer et al., 2024; BehnamGhader et al., 2024). This asymmetry has been linked to several documented limitations, including premise order sensitivity in reasoning (Chen et al., 2024), option ordering bias (Pezeshkpour and Hruschka, 2024; Wei et al., 2024; Ok and Lee, 2026), and the "lost in the middle" phenomenon (Liu et al., 2024). This raises a natural question: How can we redistribute contextual grounding more evenly?

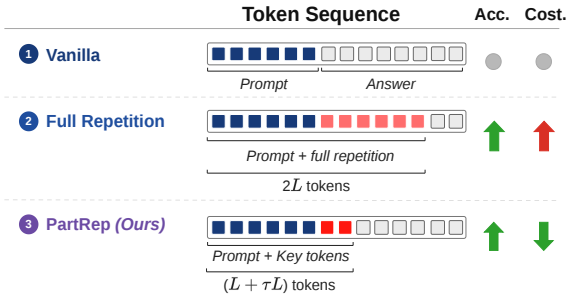

Figure 1: **Overview of token repetition strategies**. Vanilla prompting uses the original prompt once. Full repetition appends the entire prompt, improving accuracy at the cost of doubling the input length to $2L$. **PARTREP**, our method, appends only selected key tokens, producing a shorter sequence of length $(L + \tau L)$ that preserves the accuracy gains of repetition while reducing computational cost.

A surprisingly effective remedy for this asymmetry is prompt repetition: presenting the same prompt twice before generation. By placing a second copy of the prompt at the end of the input, every original token gains a later "echo" that can attend back over the full context, effectively redistributing contextual grounding across the sequence. Despite requiring no parameter updates or architectural changes, this simple intervention has been shown to improve reasoning performance across diverse tasks (Leviathan et al., 2025; Xu et al., 2024; Springer et al., 2024).

However, it also introduces several limitations: Appending a full copy of the prompt doubles the KV cache footprint and quadruples self-attention FLOPs. These overheads become increasingly prohibitive as the prompt length grows, rendering this approach un-scalable for long contexts.

To this end, we present **PartRep**, a method to selectively repeat only the most critical tokens—rather than duplicating the entire prompt (Figure 1). PartRep scores each input token by its informational importance, then appends only the highest-scoring tokens before generation. As the scoring criterion, we adopt the negative log-likelihood (NLL) of each token as a proxy for information density: tokens with high NLL (i.e., high surprisal) are those the model finds least predictable from preceding context, indicating that their content is not redundantly encoded by surrounding tokens. As we show in Section B.1, repeating such tokens yields the greatest marginal benefit, as their echoes inject information that would otherwise remain underrepresented in later positions' attention.

Since exact NLL computation requires a full forward pass and would undermine our efficiency goal, we instead train a lightweight gate to predict the top-τ high-NLL tokens directly from early-layer hidden states, enabling selection during mid-prefill. The selected tokens are then appended to the original prompt via a short natural-language bridge (e.g., "*Pay attention to these key tokens...*"), after which a single forward pass produces the final output.

We demonstrate the robustness of our approach across 8 benchmarks, including GSM8K and RULER, and a diverse set of decoder-only LLMs encompassing Qwen2.5, Llama3.2, and Gemma4 families. PartRep preserves the accuracy benefits of full repetition, while requiring only 59.4% of the KV cache budget and 79.0% of the prefill FLOPs.

We summarize our contributions as follows:

- We propose PartRep, a selective prompt augmentation method that approximates the benefits of full repetition at a fraction of its pre-fill compute and KV cache memory cost.
- We show that token-wise NLL provides a principled importance signal for token selection, and design an efficient gating mechanism that approximates it without a full forward pass.
- We empirically validate PartRep across eight benchmarks and three model families, showing that it consistently matches or surpasses full repetition across the setups, while substantially reducing prefill compute and memory overhead.

2 Related work

Prompt repetition. Several studies have shown that prompt repetition can improve LLM performance. Xu et al. (2024) found that "re-reading" a question before chain-of-thought reasoning improves LLM performance, arguing that repetition enables causally masked models to partially emulate bidirectional attention and mitigate limitations of decoder-only architectures. Arora et al. (2024) extended similar ideas to recurrent language models, while Springer et al. (2024) formalized the phenomenon through "echo embeddings," showing that representations from repeated tokens substantially outperform single-pass embeddings on retrieval and similarity tasks. At larger scale, Leviathan et al. (2025) observed consistent gains from prompt repetition across both open- and closed-weight models on non-reasoning mode. Although prior work has primarily focused on full-prompt repetition, related findings on partial (option-level) repetition by Ok and Lee (2026) further suggest that the benefits arise from restoring information pathways are blocked by causal attention. However, option-level repetition is limited to multiple-choice settings and does not generalize to the open-ended generation tasks prevalent in real-world LLM usage. In contrast, we introduce a task-agnostic framework that selectively repeats only the most informative tokens within an arbitrary prompt, avoiding both the doubled KV cache cost of full repetition and the multiple-choice constraint of option-level repetition.

KV cache eviction. A number of prior works propose to evict KV cache of *less important* tokens to mitigate the high memory I/O cost of long-context LLM inference. For instance, H_2O (Zhang et al., 2023) retains the tokens that frequently receive high attention during the next-token prediction, while KVzip (Kim et al., 2025) asks the model to reconstruct the prompt and utilize the attention during this process. More recent works focus on avoiding the computation of full attention matrices for each token during the prefill stage, to reduce the latency overhead. For example, FastKVzip (Kim et al., 2026) and KVzap (Jegou and Jeblick, 2026) introduce lightweight predictors trained to predict the token importance based on the hidden states of early transformer layers, where the token importance is estimated by prior methods (e.g., KVzip).

We adopt a similar predictor-based strategy, but for a fundamentally different objective. Eviction es-

timates which tokens can be safely discarded, while partial repetition estimates which tokens yield additional benefit when re-attended to — and these two notions of importance are not duals of each other. We elaborate in section 3.1, where we show, for example, that the final prompt token is typically essential under eviction yet least useful to repeat.

Summarization. Another line of work reduces memory cost by compressing the input prompt itself. LLMLingua (Jiang et al., 2023) uses a compact language model to identify and remove *less important* tokens before feeding a summarized prompt to the main model. In this sense, summarization can be viewed as a text-level analogue of KV cache eviction: instead of discarding key-values after prefill, it removes redundant tokens before they enter the model. Our work similarly estimates token importance and constructs a shortened textual representation as the form of a summary. However, the objective differ fundamentally. LLMLingua assumes that the original prompt contains inherent redundancy and compresses it to reduce memory consumption. In contrast, our goal is not merely to shorten the prompt, but to approximate the accuracy gains of *redundant* full repetition under constrained memory budgets.

3 Problem formulation

Suppose that we are given an initial ***user prompt*** as an input, formulated as a sequence of tokens

$$\mathbf{t} = (t_1, t_2, \ldots, t_L) \tag{1}$$

with length L. For decoder-only LLM, the computational complexity of prefill self-attention scales quadratically as $O(L^2)$, and the stored key-value (KV) cache scales linearly as $O(L)$.

The ***full repetition*** (Leviathan et al., 2025) is the idea of repeating the full prompt. In other words, the prompt is duplicated to form

$$\mathbf{t}_{\text{FR}} = \mathbf{t} \oplus \mathbf{t}, \tag{2}$$

where $\oplus$ denotes sequence concatenation. It has been known that prefilling $\mathbf{t}_{\text{FR}}$ enables critical pseudo-bidirectional attention for the sequence, thereby increasing the accuracy of the model. On the other hand, the prefill complexity gets quadruples and the KV cache footprint doubles.

Extending this paradigm, we formulate the problem of ***partial repetition,*** which aims to achieve the benefit of repetition without inheriting its prohibitive memory and latency costs. Our goal is to isolate an *important, highly informative* subset of the original prompt such that appending only this subset—rather than the full prompt—suffices to improve model accuracy. Precisely, we consider the prompt structured as

$$\mathbf{t}_{\text{PR}} = \mathbf{t} \oplus \mathbf{t}_{\text{part}} \tag{3}$$

where $\mathbf{t}_{\text{part}}$ denotes the *keyword subsequence* of $\mathbf{t}$. Given this prompt structure, our goal is find a token selector $f(\mathbf{t}) = \mathbf{t}_{\text{part}}$ which maximizes the model accuracy. More concretely, our goal is to solve

$$\max_f \ \text{acc}(\mathbf{t} \oplus f(\mathbf{t})) \tag{4}$$

where $\text{acc}(Q)$ denotes the accuracy of the base LLM prompted by Q, subject to the constraints

$$f(\mathbf{t}) \subset \mathbf{t}, \qquad |f(\mathbf{t})| \leq \tau \cdot |\mathbf{t}|, \tag{5}$$

where $\subset$ denotes the subsequence relation (instead of a subset), and $\tau \in (0, 1)$ is a repetition threshold enforced to meet the given memory budget.

The primary advantage of formulating as a *token selection* (i.e., imposing $f(\mathbf{t}) \subset \mathbf{t}$) is computational. The predictor f can be viewed as making a binary decision on each input token, acting as a "gate" that accepts or rejects each token. As the output dimension is simple, we can expect f to be implementable with a lightweight model. Indeed, we implement this with a two-layer MLP with attention, on top of the target model's hidden state. The same may not be possible if we let the output of $f(\mathbf{t})$ to lie in the general vocabulary space, e.g., textual summary of $\mathbf{t}$, or continuous token space.

Efficiency. Through compressing the repeated prompt, we can expect the prefill self-attention computation to be proportional to $(1 + \tau)^2 L^2$, which is lower than $4L^2$ of the full repetition. Likewise, the KV cache will be proportional to $(1+\tau)L$ which is less than $2L$ of the full repetition. Note, however, that the partial repetition introduces additional inference cost for the token selection procedure $f(\cdot)$. Fortunately, as we will see in Section 6.2, the cost is small.

3.1 Comparison with KV cache eviction

Recall that the task of KV cache eviction is about deciding "what to discard" (Zhang et al., 2023). The task of partial repetition, i.e., deciding "what

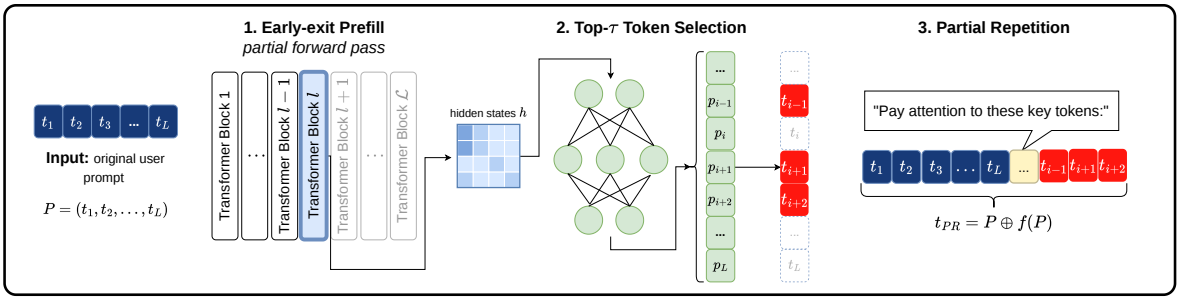

Figure 2: **Inference procedure of the proposed PartRep**. We first prefill the LLM with the original prompt, then pass its early-layer hidden states through the gating module. Next, we select the top-τ fraction of tokens, and repeat the selected tokens (i.e., append it after the original prompt), then continue with the prefill.

to repeat," is similar in the sense that we need to estimate the importance of each token to reduce the KV cache cost. However, the tasks fundamentally differ from each other, in two aspects.

First, tackling partial repetition via KV cache eviction is computationally suboptimal. To see this, consider applying KV cache eviction methods on fully repeated prompt as a mean of partial repetition. In this case, we must still perform prefill with $2L$ tokens, thus losing any computational advantage in the prefill computation.

Second, the notion of *importance* is different in two tasks. To see this, consider the last token of the prompt. In partial repetition, this is the least important token to be repeated, as the last token in the original prompt already attended to all other tokens. In KV cache eviction, however, the last token (of the repeated prompt) are typically treated as essential (Zhang et al., 2023).

4 Algorithm

We develop **PartRep**, a learning-based method that captures the benefits of repetition without inheriting its memory and latency cost by repeating only the highly *important* tokens.

To select these tokens, we first motivate negative log-likelihood (NLL) as a principled proxy for information density (Section 4.1). Since calculating the exact token-level NLL at runtime is prohibitively expensive, we introduce a lightweight gate trained to predict these scores from early-layer hidden states (Section 4.2). Section 4.3 describes the offline procedure used to train the gate across diverse domains. At inference, selected tokens are appended to the original prompt before the model's second forward pass. Figure 2 illustrates the overall PartRep pipeline, with details in Section 4.4.

4.1 Token importance scoring

To identify and retain only critical tokens, we use the negative log-likelihood (NLL) of the next-token prediction as a direct proxy of the predicted tokens' information density (Jaeger and Levy, 2006). In a standard autoregressive language model, the probability of each token t_i conditioned on all preceding tokens $t_{<i}$ is predicted. The NLL for a token is:

$$\mathrm{NLL}(t_i) = -\log P(t_i \mid t_{<i}) \tag{6}$$

where $P(\cdot|\cdot)$ denotes the next-token probability predicted by the model. Tokens with high prediction probabilities yield low NLL scores, indicating high redundancy with previous context and low information added. Conversely, "surprising" tokens with low probabilities yield high NLL scores. Our method leverages this intrinsic property by defining the predictor $f(\cdot)$ to isolate the token subset carrying the highest information density within the repetition threshold τ:

$$f^\star(\mathbf{t}) = \{t_i \in \mathbf{t} \mid \mathrm{NLL}(t_i) \geq \eta_\tau\}. \tag{7}$$

Here, η_τ is the threshold determined to meet the required budget for each sample, i.e., τ-quantile.

Rationale. The NLL-based selector (7) can be viewed as an approximate solution for a more general optimization problem. In particular, consider the following *information maximization* principle:

$$\max_f H(f(\mathbf{t})|\mathbf{t}), \tag{8}$$

where $H(\mathbf{t}_2|\mathbf{t}_1)$ denotes the conditional entropy of the model generating a sequence $\mathbf{t}_2$ conditioned on the context $\mathbf{t}_1$.[1] In other words, we are selecting a

[1]Note that this is different from the usual conditional entropy, where we expect $H(f(X)|X) = 0$ for any deterministic $f(\cdot)$. This discrepancy is due to the fact that we consider conditional entropy of a token sequence given its context, modeled by an LLM, rather than modeling X and $f(X)$ as random variables themselves.

subsequence $f(\mathbf{t})$ that can maximize the information added by the repetition.

Equation (7) approximately solves this problem, where the approximation is that we also condition on the tokens that may be discarded later. Through this approximation, we can save computation—otherwise, we will need to add each context word iteratively, through multiple rounds.

4.2 Gating mechanism

While $f^\star$ can select critical tokens in a principled way, computing it requires much prefill computation to pass the prompt through the LLM. To alleviate this overhead, we employ a *learning-based approach*: We train a small, lightweight gating module that can approximate $f^\star$ on-the-fly.

Precisely, our gating mechanism works as a composition of two different functions:

$$f(\mathbf{t}) = g \circ \phi_{\text{LLM}}(\mathbf{t}). \tag{9}$$

Here, $\phi_{\text{LLM}}(\mathbf{t})$ denotes the hidden states of the original prompt $\mathbf{t}$ extracted from the base LLM at layer $l^\star$, while $g(\cdot)$ is a lightweight token-wise gating function. By leveraging $\phi_{\text{LLM}}(\mathbf{t})$, we can keep $g(\cdot)$ lightweight while minimizing computational overhead. In particular, these features reuse computations from the original prefill stage and therefore incur no additional computational cost.

Architecture. The token-wise gating function $g(\cdot)$ takes a similar structure with Kim et al. (2026), motivated by its success in KV cache eviction. In a nutshell, we first apply a two-layer MLP to map raw hidden states—a highly entangled mixture of syntactic, semantic, and positional signals—onto compact query space. Then, we use attention module to compare the query to the keys that represent latent signatures of high-NLL tokens, to compute the final repetition probability.

Concretely, given some hidden state $\mathbf{h} \in \mathbb{R}^d$, we first compute the query

$$\mathbf{q} = W_2\, \text{SiLU}(W_1 \mathbf{h}), \tag{10}$$

where $W_1 \in \mathbb{R}^{2d_h \times d}, W_2 \in \mathbb{R}^{d_h \times 2d_h}$ are weight matrices. We fix $d_h = 64$ across models to maintain efficiency. The query $\mathbf{q}$ is then evaluated against s learnable keys $K \in \mathbb{R}^{s \times d_h}$ to output the corresponding weighted sum of values $\mathbf{v} \in \mathbb{R}^s$:

$$g(\mathbf{h}) = \sigma\left(\mathbf{v}^\top \text{Softmax}\left(\mathbf{q}K^\top/\sqrt{d_h}\right)\right). \tag{11}$$

Here, $\sigma(\cdot)$ is the sigmoid activation function to generate the final probability score.

4.3 Training the gating module

Now we describe how we train the gating function $f = g \circ \phi_{\text{LLM}}$ (Equation 9) using the NLL supervision (Equation 6). To avoid prohibitive training cost, we train only the gating module g and keep the base feature extractor ϕ_{LLM} frozen.

Dataset construction. We first construct a training dataset by sampling the prompts $\{\mathbf{t}_i\}_{i=1}^n$ and generating the corresponding feature-NLL pairs

$$\mathcal{D} = \{(\phi_{\text{LLM}}(\mathbf{t}_i), \text{NLL}(\mathbf{t}_i))\}_{i=1}^n, \tag{12}$$

where $\text{NLL}(\mathbf{t}_i)$ denotes the sequence of token-wise NLL scores. At this stage, we use the NLL itself as the label, instead of $f^\star(\mathbf{t}_i)$. This allows us to reuse the constructed dataset for different repetition thresholds η_τ. As this labeling can utilize the batched inference, it can be done efficiently.

For the generality of the learned gating function, we draw sample prompts $\{\mathbf{t}_i\}_{i=1}^n$ from a general educational corpora, rather than domain-specific prompt sets. In particular, we stream and process sequences ranging from 10 to 1024 tokens in length, compiling a robust dataset of 3 million training tokens sampled from FineWeb-Edu (2 million) and Ultra-FineWeb-Edu (1 million).

Training procedure. Given a sample prompt $\mathbf{t}$ of length L, training g is formulated as L token-wise binary classification tasks, where the ground-truth labels are provided by the top-τ NLL selector $f^\star$ (for a designated τ). Specifically, we generate binary labels as

$$\mathbf{y} = \left(\mathbf{1}\{t_1 \in f^\star(\mathbf{t})\}, \ldots, \mathbf{1}\{t_L \in f^\star(\mathbf{t})\}\right), \tag{13}$$

where $\mathbf{1}$ denotes the indicator function. Then, we train with the sample-wise loss

$$\ell(\mathbf{t}) = \frac{1}{L}\sum_{j=1}^{L} \lambda_j\, \ell_{\text{CE}}([f(\mathbf{t})]_j, y_j), \tag{14}$$

where $[\cdot]_j$ denotes the jth entry of a vector. Here, λ_j is the weight determined to mitigate the effect of class imbalance. In particular, the positive-labeled tokens (i.e., y_j = 1) are upweighted by

$$\lambda_j = (1-\tau)/\tau, \tag{15}$$

representing the ratio between negative and positive tokens in the training set. For the negative-labeled tokens (i.e., $y_j = 0$), we simply use $\lambda_j = 1$.

Method	Avg KV Cache	Avg Prefill FLOPs	ARC	OBQA	MMLU	MedQA	SciQ	MMLU-Pro	GSM8K	Avg.
No Repetition	235.5	1.549 T	79.8	76.6	62.5	47.5	90.3	27.8	84.2	66.9
Appending summary										
+ Naïve Summary	277.4	3.676 T	81.7	76.5	60.8	50.9	91.6	28.6	85.5	67.9
+ LLMLingua	277.4	2.105 T	81.5	77.8	61.0	49.3	90.7	27.7	84.0	67.4
Full Repetition	471.1	3.140 T	82.8	78.0	60.3	51.0	92.9	27.3	87.3	68.5
Compressing full. rep.										
+ Echo Eviction	235.5	3.152 T	81.3	69.6	60.3	50.7	90.1	22.0	76.6	64.4
+ H2O Eviction	270.8	3.162 T	75.0	76.2	60.5	50.6	77.7	27.4	72.9	62.9
+ LLMLingua Comp.	78.6	1.130 T	15.2	18.8	14.8	12.2	20.6	10.5	12.2	14.9
PARTREP (ours, $\tau = 0.15$)	280.4	2.481 T	81.4	77.8	63.2	50.3	92.2	27.8	85.5	68.3

Table 1: Accuracy comparison of various prompt repetition methods across seven benchmarks, on Qwen 2.5-3B. Colored cells in red and blue indicates performance drops and gains relative to the "No Repetition," respectively.

4.4 Inference

Connecting prompt. Given an input prompt $\mathbf{t}$, the model performs an initial forward pass and pauses at the target layer $l^\star$. Hidden states at this layer are used to compute $f(\mathbf{t})$, and the selected tokens are then appended to the original prompt, connected through a trigger: "`\nPay attention to these key tokens:\n`". The model then performs the full forward pass over the extended prompt $\mathbf{t} \oplus f(\mathbf{t})$, reusing the hidden layer features of the original prompt $\mathbf{t}$.

Token windowing. PARTREP directly appends tokens selected by the gate, which is effective when those tokens alone provide sufficient cues for downstream prediction. However, token-level repetition can be insufficient in two cases. First, subword tokenization may split an informative word into multiple fragments, so repeating only one selected fragment yields an incomplete lexical cue. Second, when the answer depends on local compositional structure, such as relations expressed within a sentence, repeating isolated salient tokens may discard the context needed to interpret them correctly.

To handle these cases, we introduce token windowing as an optional post-selection expansion strategy. When enabled, each selected token can be expanded to include either full word containing it or a broader local context window. This option allows PARTREP to preserve lexical integrity or short-range relational structure when isolated token repetition is insufficient.

5 Experiment

5.1 Experimental setup

Benchmarks. We evaluate on total eight tasks. Five benchmark datasets assess the general knowledge and scientific retrieval capabilities: ARC-Challenge (Clark et al., 2018); OpenBookQA (Mihaylov et al., 2018); MMLU (Hendrycks et al., 2020); MedQA (Jin et al., 2021); SciQ (Welbl et al., 2017). Two benchmark datasets assess more complex reasoning abilities: MMLU-Pro (Wang et al., 2024); GSM8K (Cobbe et al., 2021). One benchmark dataset assess long-context capabilities: RULER (Hsieh et al., 2024).

At the inference stage, we use complete test sets for all benchmarks, except for RULER where we use a subset of 100 questions per each tasks, 1300 questions in total.

Models. We conduct experiments on three different instruction-finetuned LLMs: Qwen 2.5-3B (Qwen et al., 2024); Llama 3.2-3B (Grattafiori et al., 2024); Gemma 4-E4B (Google, 2026).

Notably, Gemma 4-E4B employ a hybrid mechanism alternating between local sliding window and global attention layers. For this model, we strictly constrain our extraction point to coincide with a global attention layer to prevent an information bottleneck within the extracted hidden states.

Baselines. We compare our method against seven standard baselines with repetition. Three of them are the method that does not involve full repetition.

- *No repetition.* This is the vanilla setup that uses the user prompt as is without any repetition.
- + *Naïve summary.* After the original prompt, we append the summary of the prompt generated by the model itself.
- + *LLMLingua.* Same, but using LLMLingua instead of summarization (Jiang et al., 2023).

Other four are the methods based on full repetition.

- *Full repetition.* We repeat the whole prompt twice (Leviathan et al., 2025).
- + *Echo eviction.* After processing fully repeated prompt, we remove the hidden states of the first

half (Springer et al., 2024).
- + *H2O eviction.* Same, but we use H2O to select tokens to evict (Zhang et al., 2023).
- + *LLMLingua.* We apply LLMLingua on the fully repeated prompt (Jiang et al., 2023).

Evaluation metrics. Our evaluation focuses on assessing the tradeoffs across three dimensions:
- *Accuracy.* Accuracy on the target task.
- *KV cache.* The average number of prompt tokens whose layerwise KV cache is stored.
- *Prefill compute.* The computation required for prefilling the prompt, measured in FLOPs.

Note that we account for algorithm-specific computational overhead, including prompt processing and, when applicable, the additional cost of token selection, summarization, or cache eviction.

Implementation details. See Appendix A.1 for training and Appendix A.2 for inference details.

5.2 Main results

Table 1 reports the accuracy of Qwen 2.5-3B under different repetition strategies. We first observe that full repetition provides the strongest gain with over 1.6%p increase over vanilla, confirming that repeating prompt benefits decoder-only LLMs.

PARTREP closely matches this gain with an average accuracy of 68.3%, while repeating only a small subset of the prompt. Critically, our method provides consistent gains over benchmarks, even on tasks where full repetition drops the performance, e.g., MMLU and MMLU-Pro. This may be due to the fact our method can help avoid the accuracy degradations from having too long context (e.g., lost-in-the-middle phenomenon).

Summary-based alternatives also perform strong, but remain below our method. This suggests that compressing prompt retain useful information, but replacing the context with abstract summary is less effective than preserving the original prompt and appending selected tokens.

5.3 Other LLM architectures

In Table 2, we provide additional evaluations on diverse LLM architectures with varying baseline capabilities: Llama 3.2-3B and Gemma 4-E4B. In particular, we evaluate on the MMLU benchmark. The results confirm that our method achieves consistent gain over various architectures. Notably, under the strict constraints of Gemma models' hybrid sliding-window and global attention mechanisms, our method stays reliable.

Method	Qwen 2.5	Llama 3.2	Gemma 4
No Repetition	62.5	54.1	74.0
+ Naïve Summary	60.8	52.7	74.2
+ LLMLingua	61.0	53.5	76.1
Full Repetition	60.3	54.8	72.9
+ Echo Eviction	60.3	53.2	74.5
+ H2O Eviction	60.5	52.6	72.3
+ LLMLingua Comp.	14.8	11.8	24.4
PARTREP (ours)	63.2	54.3	75.8

Table 2: Accuracy comparison across various LLM architectures, measured on MMLU benchmark.

	Context Length		
Method	**4k**	**8k**	**16k**
No Repetition	77.7	80.0	72.3
Full Repetition	79.2	82.3	72.3
LLMLingua	69.2	73.1	63.5
PARTREP (ours, $\tau = 0.3$)	**80.0**	**83.8**	**73.2**

Table 3: **Accuracy comparison on the RULER long-context benchmark at varying context lengths**. For PartRep, we use $\tau = 0.3$ with local-context token window expansion.

5.4 Long-context tasks

We further evaluate our method on long-context settings using the RULER benchmark (Table 3) and Qwen2.5-3B. Results confirm that repetition remains beneficial even when input context becomes substantially longer.

PARTREP achieves strongest performance across all evaluated context length, although the gap gets narrower as the number of token grows. This implies that selectively repeating "only" informative tokens remains effective in long-context scenarios. In contrast, full repetition may just repeat redundant information and suffer from the drawbacks of long context, and compressing the prompt (i.e., using LLMLingua) may discard critical structural cues that can only be preserved by keeping the original context intact.

6 Analysis

6.1 Ablation studies

Gating module. In Table 4, we compare the effectiveness of the two-layer MLP (Eq.10) against alternative architectures: a lighter linear layer, and a heavier transformer-based architecture. We use ARC-Challenge benchmark for this comparison. We observe that our architecture achieves a nice

Gating module	Accuracy (%)	Gate FLOPs
No Repetition	79.8	0 k
Linear	80.7	270.6 k
2-layer MLP (ours)	**81.4**	549.0 k
Transformer-based	81.3	654.3 k

Table 4: **Ablations on gating module architecture**. Our choice (2-layer MLP) achieves favorable accuracy with minimal computational overhead.

Window Expansion	Accuracy	KV cache
No windowing	61.5	90.1
1 word	64.6	94.5
+1 word (left), +1 word (right)	66.0	110.2

Table 5: **Effect of token-window expansion for handling fragmented subword selections**. Adding local word context improves accuracy from 61.5 to 66.0 while introducing only a moderate KV-cache increase.

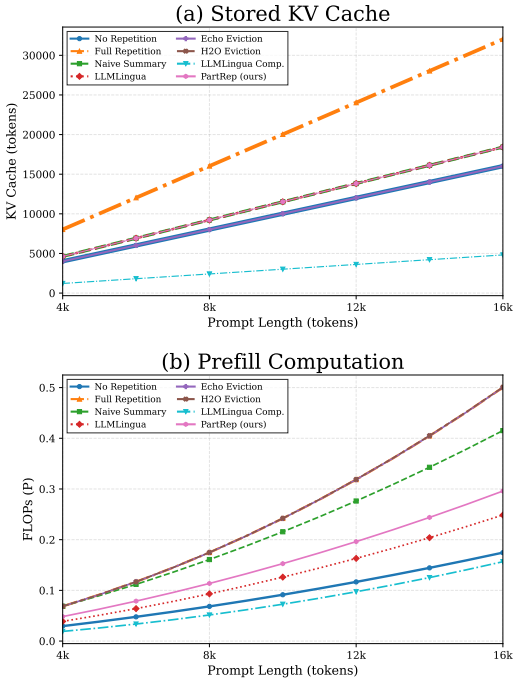

Figure 3: **Memory and compute scaling with prompt length**. The panel (a) reports the number of stored KV cache tokens, and the panel (b) reports the estimated prefill FLOPs. As prompt length increases, Full Repetition incurs substantially larger overhead, whereas Partial Repetition maintains lower KV cache usage and prefill compute by appending only selected informative tokens.

trade-off point of accuracy and computation. Simpler models achieve lower accuracy, while the benefits considering more complicated architectures gets saturated over this scale.

Token windowing. We also ablate the token windowing on the Llama 3.2-3B, which uses subword tokenization. Table 5 reports the effect of expanding selected token spanned on ARC-Challenge. As shown in Table 5, whole-word expansion improves accuracy from 61.5 to 64.6 with a small increase in KV cache, while adding one neighboring word on each side further improves accuracy to 66.0 with moderate increase in KV cache.

Other ablations. We provide more ablations on:

- The scoring criterion for $f^\star$ (Appendix B.1)
- The early exit layer index $l^\star$ (Appendix B.2)
- The repetition budget τ (Appendix B.3)
- The connecting prompt (Appendix B.4)
- The number of learnable keys s (Appendix B.5)

6.2 Efficiency comparison

Figure 3 exhibits the KV cache and FLOPs required by each baseline as prompt length scales. While full repetition maximizes reasoning accuracy, the curve shows it scales prohibitively, doubling the KV cache footprint and inflating prefill complexity to $O(4L^2)$ FLOPs. Meanwhile, as prompt gets longer, PARTREP's KV cache growth turns more and more negligible. As for the complexity, although the gate's token selection procedure introduces additional computational overhead, our empirical results demonstrate that the total required FLOPs still remain substantially lower than what is required in full repetition.

7 Conclusion

We propose PARTREP, a selective prompt augmentation method that approximates the benefits of full repetition at a fraction of its compute and memory cost. Across eight benchmarks and three model families, PARTREP consistently improves over vanilla prompting and remains competitive with full repetition across reasoning, knowledge, and long-context tasks, requiring only 59.4% of the KV cache footprint and 79% of the prefill FLOPs of the full repetition. Our ablation studies further demonstrate that high-NLL token supervision, lightweight gating, and local token-window expansion are effective design choices to balance accuracy and inference efficiency. We believe that PARTREP provides a practical approach for improving decoder-only LLM inference, particularly in long-context scenarios where efficient use of KV cache and prefill computation is essential.

Limitations

One limitation of the PARTREP is that it requires an offline training for each target LLM, instead of being applicable zero-shot or transferrable across architectures. This undermines the applicability of our framework to the scenarios where we do not have much training budget.

Another key limitation is its applicability to different data modalities, e.g., images. Our approach relies on an implicit assumption that each token can convey a meaningful signal by itself. However, as repeating only a few tokens from an image and concatenating them as a sequence may produce highly distorted image, our method may not generalize well to vision-language models.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Simran Arora, Aman Timalsina, Aaryan Singhal, Benjamin Spector, Sabri Eyuboglu, Xinyi Zhao, Ashish Rao, Atri Rudra, and Christopher Ré. 2024. Just read twice: closing the recall gap for recurrent language models. *arXiv preprint arXiv:2407.05483*.

Parishad BehnamGhader, Vaibhav Adlakha, Marius Mosbach, Dzmitry Bahdanau, Nicolas Chapados, and Siva Reddy. 2024. Llm2vec: Large language models are secretly powerful text encoders. *Preprint*, arXiv:2404.05961.

Xinyun Chen, Ryan A Chi, Xuezhi Wang, and Denny Zhou. 2024. Premise order matters in reasoning with large language models. *arXiv preprint arXiv:2402.08939*.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Google. 2026. Gemma 4 model card. Last updated: 2026-04-18.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

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

Yunpeng Huang, Jingwei Xu, Junyu Lai, Zixu Jiang, Taolue Chen, Zenan Li, Yuan Yao, Xiaoxing Ma, Lijuan Yang, Hao Chen, and 1 others. 2023. Advancing transformer architecture in long-context large language models: A comprehensive survey. *arXiv preprint arXiv:2311.12351*.

T Jaeger and Roger Levy. 2006. Speakers optimize information density through syntactic reduction. *Advances in neural information processing systems*, 19.

Simon Jegou and Maximilian Jeblick. 2026. Kvzap: Fast, adaptive, and faithful kv cache pruning. *arXiv preprint arXiv:2601.07891*.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023. Llmlingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pages 13358–13376.

Di Jin, Eileen Pan, Nassim Oufattole, Wei-Hung Weng, Hanyi Fang, and Peter Szolovits. 2021. What disease does this patient have? a large-scale open domain question answering dataset from medical exams. *Applied Sciences*, 11(14):6421.

Jang-Hyun Kim, Dongyoon Han, and Sangdoo Yun. 2026. Fast kvzip: Efficient and accurate llm inference with gated kv eviction. *arXiv preprint arXiv:2601.17668*.

Jang-Hyun Kim, Jinuk Kim, Sangwoo Kwon, Jae W Lee, Sangdoo Yun, and Hyun Oh Song. 2025. Kvzip: Query-agnostic kv cache compression with context reconstruction. *arXiv preprint arXiv:2505.23416*.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2025. Prompt repetition improves non-reasoning llms. *arXiv preprint arXiv:2512.14982*.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the association for computational linguistics*, 12:157–173.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *Proceedings of the 2018 conference on*

empirical methods in natural language processing, pages 2381–2391.

Hyunjong Ok and Jaeho Lee. 2026. Lost in the prompt order: Revealing the limitations of causal attention in language models. *arXiv preprint arXiv:2601.14152*.

Pouya Pezeshkpour and Estevam Hruschka. 2024. Large language models sensitivity to the order of options in multiple-choice questions. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 2006–2017.

A Yang Qwen, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengpeng Li, Dayiheng Liu, Fei Huang, Haoran Wei, and 1 others. 2024. Qwen2. 5 technical report. *arXiv preprint*.

Jacob Mitchell Springer, Suhas Kotha, Daniel Fried, Graham Neubig, and Aditi Raghunathan. 2024. Repetition improves language model embeddings. *arXiv preprint arXiv:2402.15449*.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, and 1 others. 2024. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems*, 37:95266–95290.

Sheng-Lun Wei, Cheng-Kuang Wu, Hen-Hsen Huang, and Hsin-Hsi Chen. 2024. Unveiling selection biases: Exploring order and token sensitivity in large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 5598–5621.

Johannes Welbl, Nelson F Liu, and Matt Gardner. 2017. Crowdsourcing multiple choice science questions. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pages 94–106.

Xiaohan Xu, Chongyang Tao, Tao Shen, Can Xu, Hongbo Xu, Guodong Long, Jian-Guang Lou, and Shuai Ma. 2024. Re-reading improves reasoning in large language models. In *Proceedings of the 2024 conference on empirical methods in natural language processing*, pages 15549–15575.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, and 1 others. 2023. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710.

A Implementation Details

A.1 Training Details

We train our gate for 25 epochs using Adam optimizer with learning rate of $1e{-}3$ and batch size of 4096. We also apply a CosineAnnealingLR scheduler and use early stopping based on validation loss with a patience of 5 epochs.

A.2 Inference Details

All experiments are performed with greedy decoding single runs and bfloat16 precision. Maximum output tokens are set to 1000 for most of the benchmarks, except GSM8K which allows up to 3000. We infer using vLLM on 6 out of 8 prompt repetition methods, except for Echo Eviction and H2O Eviction which use manual KV decoding.

B Additional Ablations

This appendix provides additional analyses of the design choices behind PartRep. We study five components of the method: the target scoring criterion for token selection, the early-exit layer used for token scoring, the repetition budget, the connector prompt used to append selected tokens, and the effect of the number of learnable keys. Unless otherwise specified, each ablation varies only one component while keeping all remaining settings fixed to the default configuration used in the main experiments. We conduct all ablation experiments on ARC-Challenge, with Qwen2.5-3B.

B.1 Token Scoring Criterion

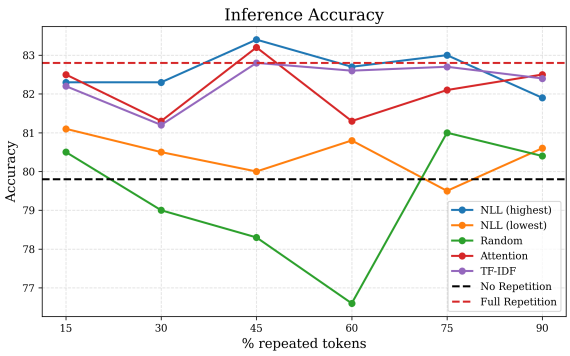

Figure 4: **Comparison of token scoring criteria under different repetition scoring criteria**. Selecting tokens with the highest NLL consistently yields strong performance, supporting NLL as an effective supervision signal for the gating module.

As our method uses token-wise negative log-likelihood (NLL) as the target importance signal for training the gating module, we first study that is a valid choice comparing against several considerable choices. We compare the highest NLL option with low-NLL tokens, random selection, attention-based selection, and TF–IDF-based selection. For TF–IDF-based selection, we use training dataset to measure the IDF, and target prompt to measure the TF.

Figure 4 shows that selecting high-NLL tokens provides the most reliable accuracy improvements across repetition budgets and achieves the strongest overall performance. In contrast, repeating low-NLL tokens or randomly selected tokens is substantially less effective, indicating that the benefit does not arise merely from adding extra tokens. Attention and TF-IDF scores show promising results, but we choose high-NLL since they are more principled solution as we have shown in Section 4.1, and they consistently match or surpass those scores. These results support our hypothesis that tokens which are difficult to predict from preceding context contain information that is particularly valuable to reintroduce at later positions.

Connector Prompt	Example Query
Verbal	`"\nPay attention to these key tokens:\n{tokens}"`
ChatML	`"\n<|im_start|>{tokens}<|im_end|>"`
Verbal+ChatML	`"\n<|im_start|>Pay attention to these key tokens:\n{tokens}<|im_end|>"`
Raw	`"\n{tokens}"`
XML	`"\n<keyword>{tokens}</keyword>"`
Verbal (Technical)	`"\nTokens with high Negative Log-Likelihood:\n{tokens}"`

Table 6: Connector prompt templates evaluated for appending the selected token subsequence to the original prompt.

B.2 Early-exit Layer Choice

We study which intermediate layer should be used to extract the hidden states consumed by the gating module. Extracting features from very shallow layers reduces selection latency, but these representations may not yet contain sufficient semantic information for identifying high-information tokens. Conversely, extracting from deeper layers increases latency and reduces the computational advantage of early exit.

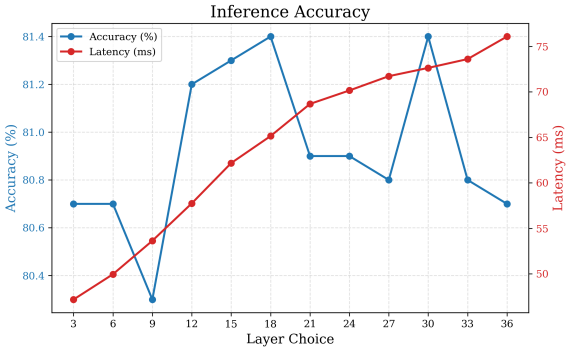

Figure 5: **Effect of the early-exit layer used by the gating module**. Accuracy improves when the gate receives sufficiently contextualized hidden states, whereas latency increases monotonically with extraction depth. We select layer 18 as the default operating point.

Figure 5 shows that accuracy improves substantially once moderately deep representations are used, while later extraction points provide limited or no additional benefit despite higher latency. In particular, layer 18 achieves the best accuracy–latency trade-off. We therefore use this layer as the default extraction point in the main experiments.

B.3 Repetition Budget

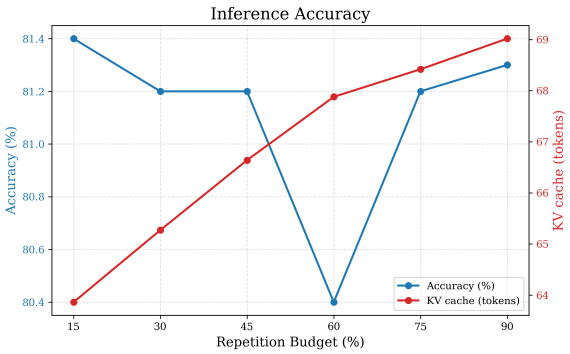

Figure 6: **Effect of the repetition ratio** τ. Larger budgets increase the KV-cache footprint, while accuracy is non-monotonic, suggesting that selectively repeating a compact set of informative tokens is preferable to indiscriminately increasing the repeated context.

We next examine the effect of the repetition budget τ, which controls the fraction of original prompt tokens appended by Partial Repetition. Increasing τ provides the model with more repeated information, but also increases the KV-cache footprint and prefill cost. Thus, the optimal operating point should preserve the benefit of repetition without approaching the overhead of full repetition.

As shown in Figure 6, accuracy does not improve monotonically with a larger repetition budget. We conjecture that this non-monotonic trend is partly associated with the verbal prompt trigger used to introduce the repeated tokens. The trigger frames the appended subsequence as a compact set of salient cues; however, as τ increases, the repeated subsequence may contain increasingly redundant or weakly informative tokens, making this framing less precise and diluting the salience of genuinely useful tokens.

The selected operating point, $\tau = 0.15$, achieves strong accuracy while retaining a substantially smaller KV-cache footprint than full repetition.

B.4 Connecting Prompt

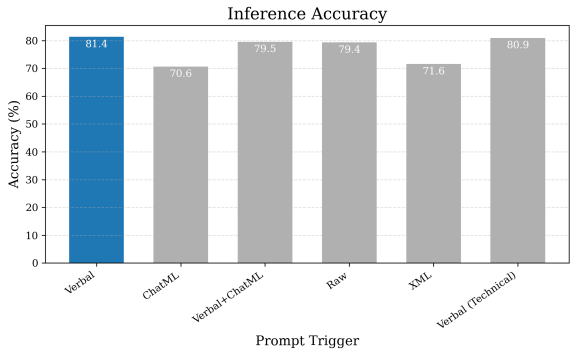

Figure 7: **Comparison of connector prompts used to append the selected tokens in ARC**. A simple natural-language instruction, "Pay attention to these key tokens:", performs best and is used as the default connector.

After selecting informative tokens, Partial Repetition appends them to the original prompt through a short connector string. We evaluate whether the form of this connector affects the usefulness of the repeated tokens. Specifically, we compare a natural-language verbal instruction, structured wrappers such as ChatML and XML, a raw token concatenation baseline, and a more technical verbal instruction. See example query in table 6.

Figure 7 shows that the simple verbal connector, "Pay attention to these key tokens:", achieves the strongest accuracy. This suggests that explicitly framing the appended subsequence as task-relevant information helps the model interpret the repeated tokens, whereas raw concatenation or unnecessarily structured wrappers are less effective. We therefore adopt the verbal connector throughout the main experiments.

Num. learnable keys	Accuracy (%)	Gate FLOPs
No Repetition	79.8	0 k
32	80.1	544.8 k
64 (**ours**)	**81.4**	549.0 k
128	80.6	557.3 k

Table 7: **Ablations on the number of learnable keys used for the gating module**. Our choice (64) achieve better accuracy than using less (32) or more (128) keys.

B.5 Number of learnable keys.

In Table 7, we test various choices on the number of learnable keys for the gating module (Eq. 11), on the ARC-Challenge dataset. We observe that our choice of 64 achieves the sweet spot, where increasing or decreasing it only degrades the accuracy.

C Qualitative results

This appendix provides qualitative examples of how various prompt repetition methods reconstruct the final input prompt given to the model. We use examples from the ARC-Challenge dataset and run them on Qwen2.5-3B. For KV cache eviction method, we represent tokens with evicted red. The original prompt is as below.

> Farmers in Wyoming were concerned because some of their chickens were being preyed upon by hawks that lived in areas around their ranches. The farmers grouped together and hunted the hawks until they were no longer in their area. Which would most likely happen next?
> A. The chicken population would go down.
> B. Populations of mice and rats would increase.
> C. Another bird of prey would replace the hawk.
> D. The chickens would have a lower rate of disease.
> Reply with one letter in the format:
> The answer is:

C.1 No Repetition

This method utilizes the vanilla setup, where the original ARC prompt is given to the model without any additional compressed context or repetition.

C.2 Full Repetition

This method repeats the entire original prompt twice. It gives the model a complete second look over the prompt, but at the cost of doubling the prompt length.

> Farmers in Wyoming were concerned because some of their chickens were being preyed upon by hawks that lived in areas around their ranches. The farmers grouped together and hunted the hawks until they were no longer in their area. Which would most likely happen next?
> A. The chicken population would go down.
> B. Populations of mice and rats would increase.
> C. Another bird of prey would replace the hawk.
> D. The chickens would have a lower rate of disease.
> Reply with one letter in the format:
> The answer is:

C.3 PartRep (Ours)

This method keeps the original prompt in Section C.1 intact and appends only *important* tokens chosen by our method at the end.

> Pay attention to these key tokens:
> Wyoming were concerned some chickens prey grouped hunted until would Pop mice Another replace
> Reply with one letter in the format:
> The answer is:

C.4 Naïve Summary

This method appends a short summary generated by the model to the original prompt in Section C.1. Summary length is restricted to τ tokens, keeping it on a par with PartRep in terms of KV-cache.

> Here is the summary of the prompt:
> Farmers in Wyoming are concerned about hawks preying on their chickens,

C.5 LLMLingua

This method also appends a short summary to the original prompt in Section C.1, however, the prompt is compressed by Microsoft's LLMLingua instead of the model itself. Summary length is also restricted to τ to keep fair comparison.

> Here is the summary of the prompt:
> Wyoming chickens preyed..?
> chicken.
> mice.

C.6 LLMLingua Comp.

This method compresses the fully repeated prompt in Section C.2 using LLMLingua. We only feed the compressed prompt to the main model, hence KV-cache memory remains low.

Wyoming chickens preyed..?
chicken.
mice rats.
..
..
preyed..?
chicken population.
mice rats.
..
..
Reply with one letter in the format:
The answer is:

C.7 Echo Eviction

This method first applies Full Repetition as in Section C.2, then evicts the KV-cache that corresponds with the original prompt (Section C.1). Therefore the textual prompt given to the model is identical to Full Repetition, yet only the repeated part is retained for answer generation.

Farmers in Wyoming were concerned because some of their chickens were being preyed upon by hawks that lived in areas around their ranches. The farmers grouped together and hunted the hawks until they were no longer in their area. Which would most likely happen next?

A. The chicken population would go down.

B. Populations of mice and rats would increase.

C. Another bird of prey would replace the hawk.

D. The chickens would have a lower rate of disease.

Reply with one letter in the format:

The answer is:
Farmers in Wyoming were concerned because some of their chickens were being preyed upon by hawks that lived in areas around their ranches. The farmers grouped together and hunted the hawks until they were no longer in their area. Which would most likely happen next?
A. The chicken population would go down.
B. Populations of mice and rats would increase.
C. Another bird of prey would replace the hawk.
D. The chickens would have a lower rate of disease.
Reply with one letter in the format:
The answer is:

C.8 H2O Eviction

This method also starts from the fully repeated prompt as in Section C.2, but further applies H2O to retain only selected heavy-hitter tokens in the KV-cache. Number of retained tokens is set to $(L + \tau)$ to match the length of PartRep.

Farmers in Wyoming were concerned because some of their chickens were being preyed upon by hawks that lived in areas around their ranches. The farmers grouped together and hunted the hawks until they were no longer in their area. Which would most likely happen next?
A. The chicken population would go down.
B. Populations of mice and rats would increase.
C. Another bird of prey would replace the hawk.
D. The chickens would have a lower rate of disease.
Reply with one letter in the format:
The answer is:
Farmers in Wyoming were concerned because some of their chickens were being preyed upon by hawks . The farmers grouped together and hunted the hawks until they were no longer in their area. Which would most likely happen next?
A. The chicken population would go down.
B. Populations of mice and rats would increase.
C. Another bird of prey would replace the hawk.
D. The chickens would have a lower rate of disease.
Reply with one letter in the format:
The answer is:

D Resources

All experiments are performed on 8×NVIDIA GeForce RTX 4090 GPUs.

E Artifact Licenses and Intended Use

In this section, we provide the licenses of the used open-weight models, datasets, and benchmarks that is used in our study. Our use of these artifacts follows to their original licenses and is entirely consistent with their intended use for academic research.

E.1 Models

- Qwen2.5-3B Instruct: Qwen RESEARCH LICENSE AGREEMENT
- Llama3.2-3B Instruct: Llama 3.2 Community License Aggrement
- Gemma4-E4B it: Apache License 2.0

- llmlingua-2-xlm-roberta-large-meetingbank: MIT License

E.2 Datasets and Benchmarks

- ARC: Creative Commons Attribution Share Alike 4.0
- OpenbookQA: Apache License 2.0
- SciQ: Creative Common Attribution Non Commercial 3.0
- MedQA: Creative Commons Attribution 4.0
- MMLU: MIT License
- MMLU-Pro: MIT License
- GSM8K: MIT License
- Nvidia RULER: Apache License 2.0
- FineWeb-Edu: Open Data Commons License Attribution family
- Ultra-FineWeb-EDU: Apache License 2.0