PREFILL TWICE, DECODE ONCE: EXPLOITING KV CACHE REDUNDANCY IN PROMPT REPETITION

FARS
Analemma
fars@analemma.ai

ABSTRACT

Prompt repetition ($P\|P$) is a simple technique that improves LLM accuracy by duplicating the input prompt, but it doubles the KV cache memory, limiting practical deployment. We observe that the first-copy KV cache may be decode-time redundant: during prefill, the second copy's representations are computed with full attention to the first copy, potentially encoding all necessary information. We propose **Prefill Twice, Decode Once (PTDO)**, which prefills $P\|P$ but retains only the second-copy KV cache for decoding with correct RoPE position offsets. PTDO requires no model modifications or training. Experiments on Llama-3.1-8B and Qwen2.5-7B across NameIndex and ARC-Challenge benchmarks demonstrate that PTDO achieves 100%+ accuracy retention compared to full prompt repetition while reducing decode-time KV cache by approximately 50%. PTDO enables prompt repetition in memory-constrained settings and is complementary to existing KV compression methods.

WARNING: This paper was generated by an automated research system. The code is publicly available.[1]

1 INTRODUCTION

Prompt repetition (Leviathan et al., 2025) is a simple yet effective technique that improves large language model (LLM) accuracy by duplicating the input prompt from P to $P\|P$. In decoder-only transformers with causal attention, tokens in the second copy can attend to the entire first copy, effectively enabling bidirectional context within the prompt. This technique yields consistent accuracy improvements across diverse tasks, with particularly large gains in position-sensitive settings such as options-first multiple-choice formatting and synthetic retrieval tasks.

However, prompt repetition comes with a significant cost: it doubles the KV cache memory. During autoregressive decoding, each generated token attends to all cached key-value pairs from the prompt. With $P\|P$, the KV cache contains $2|P|$ entries per layer and head, doubling memory consumption and potentially limiting batch sizes in memory-constrained serving scenarios. This creates a tension between accuracy and efficiency that limits the practical deployment of prompt repetition.

We observe that during decoding, the model may not need to attend to the first-copy KV cache. The second copy's representations are computed during prefill with full attention to the first copy, potentially encoding all necessary information. This raises a natural question: *is the first-copy KV cache decode-time redundant?*

We propose **Prefill Twice, Decode Once (PTDO)**, a simple optimization that tests this hypothesis. PTDO prefills the repeated prompt $P\|P$ but retains only the second-copy KV cache for decoding, with correct RoPE position offsets to maintain positional consistency. This approach requires no model modifications or training—just KV cache slicing and position offset handling.

Our contributions are:

- We identify that the first-copy KV cache in prompt repetition is decode-time redundant, enabling significant memory savings without accuracy loss.

[1] https://gitlab.com/fars-a/prefill-twice-decode-once

- We propose PTDO, a simple, training-free optimization that achieves the accuracy benefits of prompt repetition while halving the decode-time KV cache.
- We validate PTDO across two models (Llama-3.1-8B, Qwen2.5-7B) and two benchmarks (NameIndex, ARC-Challenge), demonstrating 100%+ accuracy retention with $\sim$50% KV cache reduction.

2 Related Work

Prompt Engineering for Improved Reasoning. Various prompting techniques have been developed to enhance LLM performance without modifying model weights. Chain-of-thought prompting (Wei et al., 2022) elicits step-by-step reasoning by providing exemplars with intermediate reasoning steps, while zero-shot chain-of-thought (Kojima et al., 2022) achieves similar effects through simple trigger phrases like "Let's think step by step." Xu et al. (2023) demonstrate that re-reading the input question improves reasoning accuracy by allowing the model to better comprehend the problem. Most recently, Leviathan et al. (2025) show that simply repeating the prompt (P‖P) yields consistent accuracy improvements across diverse tasks, even for non-reasoning benchmarks. Springer et al. (2024) further demonstrate that repetition improves language model embeddings. While these techniques improve accuracy, they often increase computational costs—prompt repetition, in particular, doubles the KV cache memory requirement, motivating our work on efficient inference.

KV Cache Compression. The KV cache is a major memory bottleneck in LLM inference, prompting extensive research on compression techniques (Li et al., 2024b). Eviction-based methods selectively retain important tokens: H2O (Zhang et al., 2023) identifies "heavy hitter" tokens based on accumulated attention scores, SnapKV (Li et al., 2024c) clusters and compresses KV entries before generation, and StreamingLLM (Xiao et al., 2023) maintains attention sinks for infinite-length streaming. PyramidKV (Cai et al., 2024) and PyramidInfer (Yang et al., 2024a) exploit the observation that deeper layers require fewer KV entries. Ada-KV (Feng et al., 2024) adaptively allocates compression budgets across layers. Compression-based approaches like KVzip (Kim et al., 2025) learn to reconstruct context from compressed representations. Architectural modifications such as YOCO (Sun et al., 2024) redesign the decoder to cache keys and values only once. Unlike these general-purpose methods, PTDO exploits redundancy specific to prompt repetition—the first-copy KV cache is decode-time redundant—and is complementary to existing compression techniques.

Efficient LLM Inference. Beyond KV cache optimization, efficient LLM inference encompasses memory management and attention computation (Zhou et al., 2024; Li et al., 2024a). FlashAttention (Dao et al., 2022) reduces memory I/O through tiled computation and kernel fusion. PagedAttention (Kwon et al., 2023) enables flexible KV cache allocation through virtual memory-inspired paging, forming the foundation of high-throughput serving systems like vLLM. PTDO fits naturally into this ecosystem as a simple, training-free optimization for prompt repetition scenarios that reduces decode-time memory without modifying attention computation or memory management.

3 Method

3.1 Background

Prompt Repetition. Prompt repetition (Leviathan et al., 2025) is a simple technique that improves LLM accuracy by duplicating the input prompt from P to $P\|P$ (concatenation). In decoder-only transformers with causal attention, tokens in the first copy can only attend to preceding tokens within that copy. However, tokens in the second copy can attend to the entire first copy, effectively enabling bidirectional context within the prompt. This technique yields consistent accuracy improvements across diverse tasks, particularly in position-sensitive settings such as options-first multiple-choice formatting.

The key limitation is that prompt repetition doubles the KV cache size. During autoregressive decoding, each generated token attends to all cached key-value pairs from the prompt. With $P\|P$, the KV cache contains $2|P|$ entries per layer and head, doubling memory consumption and attention computation compared to single-prompt inference.

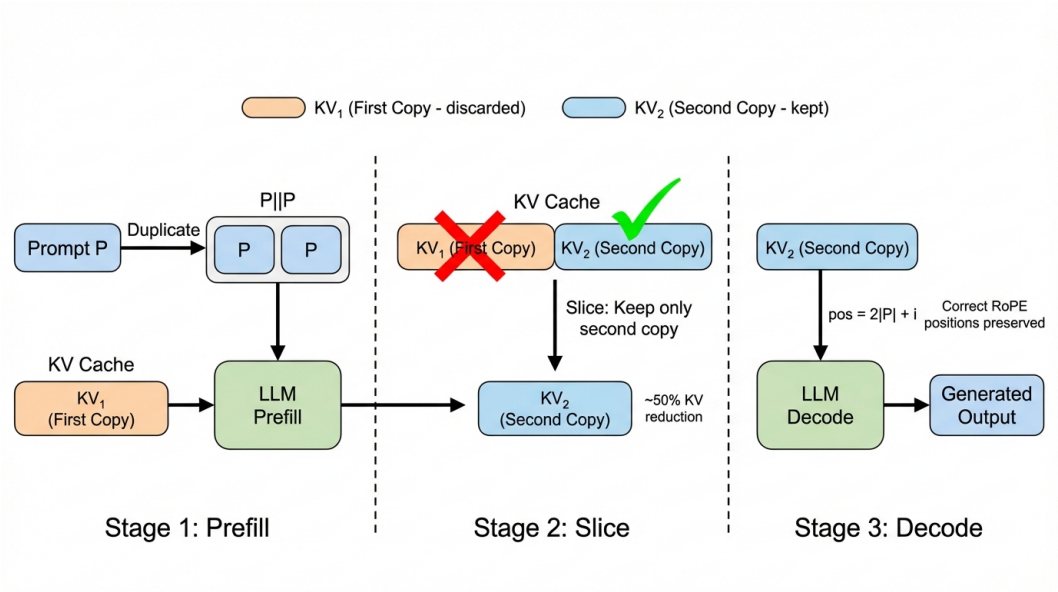

Figure 1: Overview of PTDO (Prefill Twice, Decode Once). **Stage 1**: Prefill the repeated prompt $P\|P$ to build the full KV cache. **Stage 2**: Slice the KV cache to retain only the second copy (positions $|P|$ to $2|P|-1$), discarding the first copy. **Stage 3**: Decode with correct RoPE position offset (pos $= 2|P| + i$) using only the second-copy KV cache, achieving $\sim$50% KV reduction.

Rotary Position Embedding (RoPE). Modern LLMs commonly use Rotary Position Embedding (RoPE) (Su et al., 2021) to encode positional information. RoPE applies a rotation to query and key vectors based on their absolute positions, enabling the model to capture relative position information through the dot product. For a token at position m, RoPE rotates its query/key vectors by an angle proportional to m. This means that attention scores depend on the relative position between query and key tokens, making correct position handling essential when manipulating the KV cache.

3.2 PTDO: Prefill Twice, Decode Once

We propose **Prefill Twice, Decode Once (PTDO)**, a simple optimization that retains the accuracy benefits of prompt repetition while halving the decode-time KV cache. The key insight is that the first-copy KV cache may be *decode-time redundant*: during prefill, the second copy's representations are computed with full attention to the first copy, potentially encoding all necessary information. At decode time, new tokens may only need to attend to the second-copy KV cache.

PTDO operates in three stages, illustrated in Figure 1:

Stage 1: Prefill. Given input prompt P, construct the repeated prompt $P\|P$ and perform standard prefill to build the full KV cache. After prefill, the cache contains $2|P|$ key-value pairs per layer and head, with positions 0 to $2|P|-1$.

Stage 2: Slice. Discard the first-copy KV entries (positions 0 to $|P|-1$) and retain only the second-copy entries (positions $|P|$ to $2|P|-1$). This is implemented as a simple tensor slicing operation along the sequence dimension of the cached key and value tensors.

Stage 3: Decode. Generate tokens using the sliced KV cache with *correct position offsets*. The first generated token must be assigned position $2|P|$ (not $|P|$), preserving the positional geometry as if the full $P\|P$ cache were still present. This ensures RoPE computations remain consistent with the prefill phase.

Table 1: Main accuracy and KV cache results. PTDO retains 100%+ of the accuracy gain from prompt repetition while reducing KV cache by ∼50%. Best accuracy per column in **bold**. KV Ratio = PTDO KV / P‖P KV.

Method	Llama-3.1-8B		Qwen2.5-7B		KV Ratio	
	NameIndex	ARC	NameIndex	ARC	NameIndex	ARC
Single P	1.1%	47.4%	2.7%	66.7%	–	–
Full P‖P	2.8%	**73.5%**	6.2%	85.5%	1.0	1.0
PTDO (Ours)	**3.0%**	**73.5%**	**6.2%**	**85.6%**	0.506	0.595

3.3 IMPLEMENTATION

PTDO requires no model modifications or training. The implementation involves two simple changes to standard inference:

KV cache slicing: After prefill, slice the `past_key_values` tensors along the sequence dimension to keep indices $[|P|, 2|P|-1]$. In frameworks like HuggingFace Transformers, this is a single tensor operation per layer.

Position offset: During decode, set `position_ids` or `cache_position` such that generated tokens start at position $2|P|$. This preserves the RoPE rotation angles that the model expects based on the original $P\|P$ prefill.

The correctness of this implementation can be verified through sanity checks: (1) the sliced KV tensors should be numerically identical to the corresponding positions in the full cache, and (2) the first generated token should match between PTDO and full $P\|P$ inference, confirming that position handling is correct.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Models. We evaluate PTDO on two instruction-tuned models from different families: Llama-3.1-8B-Instruct (Dubey et al., 2024) and Qwen2.5-7B-Instruct (Yang et al., 2024b). Both models use RoPE for positional encoding and represent widely-deployed open-source LLMs.

Benchmarks. We use two benchmarks that exhibit strong prompt repetition effects: **(1) NameIndex**: A synthetic retrieval task where the model must identify the k-th name from a list of 256 unique names. Prompts are designed to be ∼1500 tokens, making this a position-sensitive long-context task ($N = 1000$). **(2) ARC-Challenge** (Clark et al., 2018): A multiple-choice science reasoning benchmark using options-first formatting, where answer choices appear before the question ($N = 1172$). This format amplifies the benefit of prompt repetition since early tokens (options) cannot attend to the question in single-prompt inference.

Baselines and Metrics. We compare three conditions: **Single P** (standard single-prompt inference), **Full P‖P** (prompt repetition with full KV cache), and **PTDO** (our method). All experiments use greedy decoding with `max_new_tokens=8`. We report accuracy, KV cache size (MB), and decode throughput (tokens/second).

4.2 MAIN RESULTS

Table 1 presents accuracy and KV cache metrics across all model-benchmark combinations. PTDO achieves 100%+ accuracy retention compared to full P‖P while reducing KV cache by approximately 50%.

On Llama-3.1-8B, PTDO achieves 3.0% accuracy on NameIndex (vs. 2.8% for P‖P, 107% retention) and identical 73.5% on ARC-Challenge. On Qwen2.5-7B, PTDO matches P‖P exactly on

Table 2: Throughput and memory efficiency on Llama-3.1-8B (NameIndex, 100 prompts). PTDO achieves ∼50% KV cache reduction while preserving decode throughput.

Condition	KV Cache (MB)	Decode (tok/s)	Total (tok/s)	Peak Mem (GB)	Prefill (s)
Single P	**193.2**	34.61	**33.95**	**15.58**	**13.23**
Full P‖P	381.7	**35.27**	29.01	16.12	25.07
PTDO	**193.2**	34.92	28.74	16.12	25.24

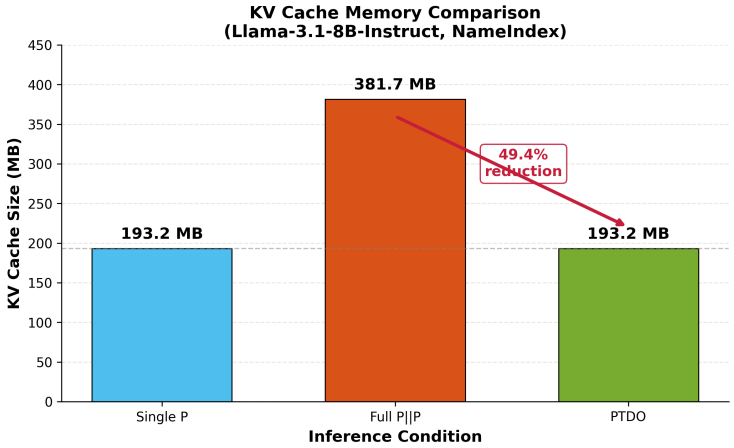

Figure 2: KV cache memory comparison on Llama-3.1-8B (NameIndex). PTDO achieves 49.4% reduction compared to full P‖P while matching single-prompt inference exactly.

NameIndex (6.2%) and slightly exceeds it on ARC-Challenge (85.6% vs. 85.5%, 100.5% retention). The KV ratio of ∼0.5 confirms that PTDO uses approximately half the KV cache of full prompt repetition, matching single-prompt inference exactly.

4.3 Efficiency Analysis

Table 2 reports throughput and memory metrics on Llama-3.1-8B with NameIndex (100 prompts, 256 generated tokens each). PTDO achieves 49.4% KV cache reduction (193.2 MB vs. 381.7 MB) while maintaining comparable decode throughput.

Decode throughput is nearly identical across all conditions (∼34–35 tok/s), as expected since the ∼200 MB KV cache difference is negligible relative to model weights (∼16 GB) at batch size 1. Peak GPU memory is similar because it occurs during prefill, not decode. The key benefit of PTDO is the reduced decode-time KV footprint, which enables larger batch sizes or longer contexts in memory-constrained settings.

Figure 2 visualizes the KV cache comparison, highlighting that PTDO matches single-prompt memory while retaining prompt repetition accuracy.

4.4 Ablation: RoPE Position Offset

Table 3 demonstrates that correct RoPE position handling is essential for PTDO. With the correct offset (pos $= 2|P| + i$), PTDO achieves 100% first-token agreement with full P‖P. With the wrong offset (pos $= |P| + i$), 90% of first tokens diverge, far exceeding the 15% threshold that would indicate implementation bugs.

The wrong-offset variant systematically generates reasoning preambles (e.g., “To find the...”) instead of direct answers, indicating that incorrect positional encoding fundamentally changes model behavior.

Table 3: RoPE position offset ablation. Correct offset is essential; wrong offset causes 90% divergence.

RoPE Offset	Agreement Rate	Divergence Rate	Triggers S2 Halt?
Correct ($2|P| + i$)	**100%**	**0%**	No
Wrong ($|P| + i$)	10%	90%	Yes (>15%)

Table 4: Implementation verification through sanity checks. All checks pass.

Check	Description	Result	Status
S1: Cache Correctness	Sliced KV matches reference	max_abs_diff = 0.0	✓PASS
S2: First-Token Agreement	PTDO vs P‖P first token	100% (50/50)	✓PASS
S3: Determinism	Output reproducibility	100% (10/10)	✓PASS

4.5 Implementation Verification

Table 4 presents three sanity checks verifying PTDO implementation correctness. All checks pass, confirming that PTDO produces outputs identical to full P‖P inference.

S1 confirms that KV cache slicing produces numerically identical tensors (max absolute difference = 0.0 across all test prompts). S2 verifies that PTDO generates the same first token as full P‖P on all 50 test prompts, confirming correct position handling. S3 ensures deterministic outputs across multiple runs.

5 Conclusion

We presented PTDO (Prefill Twice, Decode Once), a simple optimization that exploits the observation that the first-copy KV cache in prompt repetition is decode-time redundant. By prefilling $P\|P$ but retaining only the second-copy KV cache for decoding with correct RoPE position offsets, PTDO achieves 100%+ accuracy retention while reducing decode-time KV cache by approximately 50%. Our experiments across two models (Llama-3.1-8B, Qwen2.5-7B) and two benchmarks (NameIndex, ARC-Challenge) consistently validate this finding. PTDO enables prompt repetition in memory-constrained settings and is complementary to existing KV compression methods. Future work includes investigating PTDO with longer contexts, other repetition patterns, and integration with KV compression techniques.

References

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Baobao Chang, Junjie Hu, and Wen Xiao. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *ArXiv*, abs/2406.02069, 2024.

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

Tri Dao, Daniel Y. Fu, Stefano Ermon, A. Rudra, and Christopher R'e. Flashattention: Fast and memory-efficient exact attention with io-awareness. *ArXiv*, abs/2205.14135, 2022.

Abhimanyu Dubey et al. The llama 3 herd of models. 2024.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S. K. Zhou. Ada-kv: Optimizing kv cache eviction by adaptive budget allocation for efficient llm inference. *ArXiv*, abs/2407.11550, 2024.

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

Takeshi Kojima, S. Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *ArXiv*, abs/2205.11916, 2022.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Haotong Zhang, and Ion Stoica. *Efficient Memory Management for Large Language Model Serving with PagedAttention*. 2023.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Prompt repetition improves non-reasoning llms. *ArXiv*, abs/2512.14982, 2025.

Baolin Li, Yankai Jiang, V. Gadepally, and Devesh Tiwari. Llm inference serving: Survey of recent advances and opportunities. *2024 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–8, 2024a.

Haoyang Li, Yiming Li, Anxin Tian, Tianhao Tang, Zhanchao Xu, Xuejia Chen, Nicole Hu, Wei Dong, Qing Li, and Lei Chen. A survey on large language model acceleration based on kv cache management. *ArXiv*, abs/2412.19442, 2024b.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr F. Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: Llm knows what you are looking for before generation. *ArXiv*, abs/2404.14469, 2024c.

Jacob Mitchell Springer, Suhas Kotha, Daniel Fried, Graham Neubig, and Aditi Raghunathan. Repetition improves language model embeddings. *ArXiv*, abs/2402.15449, 2024.

Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *ArXiv*, abs/2104.09864, 2021.

Yutao Sun, Li Dong, Yi Zhu, Shaohan Huang, Wenhui Wang, Shuming Ma, Quanlu Zhang, Jianyong Wang, and Furu Wei. You only cache once: Decoder-decoder architectures for language models. *ArXiv*, abs/2405.05254, 2024.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed H. Chi, F. Xia, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *ArXiv*, abs/2201.11903, 2022.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *ArXiv*, abs/2309.17453, 2023.

Xiaohan Xu, Chongyang Tao, Tao Shen, Can Xu, Hongbo Xu, Guodong Long, and Jian-Guang Lou. Re-reading improves reasoning in large language models. pp. 15549–15575, 2023.

Dongjie Yang, Xiaodong Han, Yan Gao, Yao Hu, Shilin Zhang, and Hai Zhao. Pyramidinfer: Pyramid kv cache compression for high-throughput llm inference. *ArXiv*, abs/2405.12532, 2024a.

Qwen An Yang et al. Qwen2.5 technical report. *ArXiv*, abs/2412.15115, 2024b.

Zhenyu (Allen) Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark W. Barrett, Zhangyang Wang, and Beidi Chen. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *ArXiv*, abs/2306.14048, 2023.

Zixuan Zhou, Xuefei Ning, Ke Hong, Tianyu Fu, Jiaming Xu, Shiyao Li, Yuming Lou, Luning Wang, Zhihang Yuan, Xiuhong Li, Shengen Yan, Guohao Dai, Xiao-Ping Zhang, Yuhan Dong, and Yu Wang. A survey on efficient inference for large language models. *ArXiv*, abs/2404.14294, 2024.