HACK: Homomorphic Acceleration via Compression of the Key-Value Cache for Disaggregated LLM Inference

Zeyu Zhang
University of Virginia

Haiying Shen
University of Virginia

Shay Vargaftik
VMware Research

Ran Ben Basat
University College London

Michael Mitzenmacher
Harvard University

Minlan Yu
Harvard University

Abstract

Disaggregated Large Language Model (LLM) inference decouples compute-intensive prefill stage from memory-intensive decode stage, allowing low-end, compute-focused GPUs for prefill and high-end, memory-rich GPUs for decode, which reduces cost while maintaining high throughput. However, transmitting Key-Value (KV) data between the two stages can be a bottleneck, especially for long prompts. Additionally, the computational overhead in the two stages is key for optimizing Job Completion Time (JCT), and KV size can become prohibitive for long prompts and sequences. Existing KV quantization methods can alleviate transmission and memory bottlenecks, but they introduce significant dequantization overhead, exacerbating the computation time.

We propose Homomorphic Acceleration via Compression of the KV cache (HACK) for disaggregated LLM inference. HACK eliminates the heavy KV dequantization and directly computes on quantized KV to approximate and reduce the cost of expensive matrix multiplication. Extensive trace-driven experiments show that HACK reduces JCT by up to 70.9% compared to disaggregated LLM inference baseline and by up to 52.3% compared to state-of-the-art KV quantization methods.

1 Introduction

Disaggregated LLM inference improves cost-efficiency by assigning low-end GPUs (e.g., NVIDIA A10G, V100, T4, and L4) to prefill stage and high-end GPUs (e.g., A100 and H100) to decode stage [8, 15, 16, 18, 22, 23, 25, 26]. However, Key-Value (KV) transmission between the two stages can be a bottleneck, since low-end GPU instances often lack high-speed networking for cost savings. For example, AWS's A10G, V100, T4, and L4 instances cost roughly 10–20 times less than A100 instances—which typically offer 400 Gbps bandwidth—but their networks are limited to 10–50 Gbps or lower [9]. Similarly, Tencent Cloud's A100 instances are configured with only 5–50 Gbps bandwidth to cut costs [11]. Computation can also become a bottleneck due to the attention mechanism. Moreover, during the decode stage, GPU memory is constrained by the large volume of cached KV data [22, 26].

KV quantization (e.g., CacheGen [21] and KVQuant [14]) can alleviate transmission and memory bottlenecks. They quantize KV after each iteration before storing it in the cache and then retrieve and dequantize all tokens' KV in the next decode iteration. However, they introduce significant KV dequantization overhead and cannot reduce computation time.

Ideally, arithmetic operations should be executable directly on quantized KV, eliminating dequantization and accelerating computation through smaller data elements. A similar idea has been explored for gradient aggregation [20], but is limited to addition operations and unsuitable for the matrix multiplications required in the attention mechanism. To this end, we propose Homomorphic Acceleration via Compression of the KV cache (HACK) for disaggregated LLM inference. HACK addresses the KV transmission bottleneck by enabling computation on quantized data (INT2/8) while maintaining comparable inference accuracy and reducing computation and memory constraints. HACK is compatible with any quantization method that dequantizes data by linear transformation (e.g., MXFP4/8 [24]). We open-sourced the code of HACK [2].

2 Motivation

We show the networking, computation, and memory bottlenecks and demonstrate the limitations of current KV quantization methods in addressing these issues. The default experiment settings are detailed in §4.

In our measurements with the default experiment settings, KV transmission can contribute up to 42.2% of JCT. Prefill and decode times can reach up to 45.6% and 84.3% of JCT. GPU memory usage can reach up to 93.7%. KV memory access can reach up to 33.1% of JCT.

Although KV quantization (e.g., CacheGen and KVQuant) can reduce KV transmission overhead, memory usage, and KV memory access time, they introduce substantial dequantization costs per decode iteration. In our measurements, CacheGen and KVQuant introduce additional KV dequantization overhead up to 37.9% of JCT, which can be even higher for long-sequences. In addition, they cannot reduce computation time. This highlights the need for a quantization method that simultaneously lowers communication and memory overhead, avoids the cost of dequantization, and reduces computation time.

3 Design

HACK avoids KV dequantization and reduces attention computation time via homomorphic matrix multiplication on quantized data. Fig. 1 illustrates the workflow. The most critical step is Step ②, which quantizes KV to INT2 and query

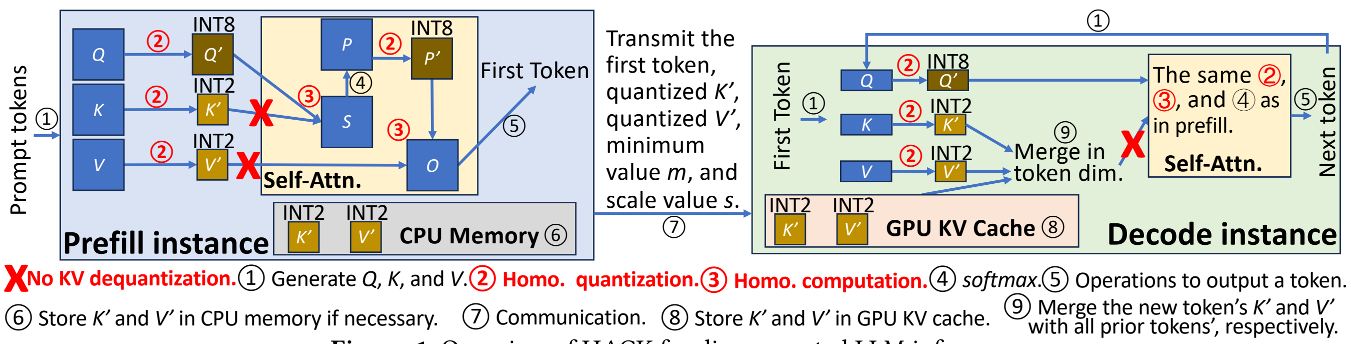
Prompt tokens
①
Q
K
V
②
INT8
Q′
INT2
K′
INT2
V′
P
②
INT8
P′
④
③
S
③
O
Self-Attn.
First Token
⑤
Prefill instance
INT2 INT2
K′ V′
CPU Memory ⑥
Transmit the first token, quantized K′, quantized V′, minimum value m, and scale value s.
⑦
①
First Token
①
Q
K
V
② INT8 Q′
② INT2 K′
② INT2 V′
⑨ Merge in token dim.
The same ②, ③, and ④ as in prefill.
Self-Attn.
⑤
Next token
INT2 INT2
K′ V′
GPU KV Cache ⑧
Decode instance
No KV dequantization. ① Generate Q, K, and V. ② Homo. quantization. ③ Homo. computation. ④ softmax. ⑤ Operations to output a token.
⑥ Store K′ and V′ in CPU memory if necessary. ⑦ Communication. ⑧ Store K′ and V′ in GPU KV cache. ⑨ Merge the new token's K′ and V′ with all prior tokens', respectively.

Figure 1. Overview of HACK for disaggregated LLM inference.

Q to INT8 using homomorphic quantization, followed by Step ③, which performs homomorphic computation on the quantized data. HACK consists of the following components.

Homomorphic quantization for matrix multiplication. Attention primarily involves matrix multiplications. For any matrix multiplication $C = AB$, HACK first quantizes A and B to obtain A' and B'. It then performs the matrix multiplication $C' = A'B'$ to have the quantized output C'. C' is subsequently approximated into C with a minimal overhead. We use an asymmetric 2-bit/8-bit stochastic quantization [19] when performing homomorphic quantization to reduce quantization error. It identifies the minimum (min_i) and maximum (max_i) values of the matrix elements and computes the $scale = \frac{max_i - min_i}{2^2-1}$. Each original value x is quantized to an integer $x' = round(\frac{x - min_i}{scale})$. The stochastic rounding $round(*)$ rounds $*$ to $\lfloor * \rfloor$ with probability $(\lceil * \rceil - *)/\lceil * \rceil - \lfloor * \rfloor)$ and to $\lceil * \rceil$ otherwise. We explain how to estimate C given C'. Let a_{iz} represent the element in the i-th row and z-th column of A, and b_{zj} represent the element in the z-th row and j-th column of B. The matrix multiplication $C = AB$ can then be expressed as $c_{ij} = \sum_z a_{iz}b_{zj}, \forall i, j$. Let m_{a_i} and s_{a_i} denote the minimum and scale values of a_{iz}. Since $a'_{iz} = round(\frac{a_{iz} - m_{a_i}}{s_{a_i}})$ and $b'_{zj} = round(\frac{b_{zj} - m_{b_j}}{s_{b_j}})$, we have $a_{iz} \approx s_{a_i} q_{a_{iz}} + m_{a_i}$ and $b_{zj} \approx s_{b_j} q_{b_{zj}} + m_{b_j}$. Thus, $(AB)_{ij}$ can be extended to:

$$\sum_z a_{iz}b_{zj} \approx s_{a_i}s_{b_j}\sum_z a'_{iz}b'_{zj} + m_{b_j}s_{a_i}\sum_z a'_{iz} + m_{a_i}s_{b_j}\sum_z b'_{zj} + Zm_{a_i}m_{b_j}, \tag{1}$$

where $\{\sum_z a'_{iz}b'_{zj}, \forall i, j\}$ is the quantized matrix multiplication that can be accelerated by INT8 computation. The other terms in Eq. (1) approximate $\sum_z a'_{iz}b'_{zj}$ (C') into $\sum_z a_{iz}b_{zj}$ (C). Eq. (1) is the homomorphic quantization for multiplication.

Summation elimination. We store the sum $\sum_z b'_{zj}$ in Eq. (1) for K and V during decode and reuse them every iteration to avoid the recomputation cost for the decode stage. This only needs a little extra memory, up to ~2.7% of the GPU memory capacity.

Requantization elimination for the last block of V. Quantization is applied to groups of elements. For the value matrix V, each group spans the sequence dimension. During decode, if the last block of V has fewer tokens than the group size, quantization metadata (e.g., the minimum) is undefined, requiring recomputation and requantization at each iteration until the group is full. To avoid this, we store the original FP16 values of the last group V in a cache, which consumes at most 0.51% of the GPU memory capacity.

4 Evaluation

Table 1 lists the AWS GPU instances we use. We use two p4de.24xlarge for decode [22, 26]; ten g5.12xlarge, sixteen p3.8xlarge, sixteen g4dn.12xlarge, ten g6.12xlarge, or two p4de.24xlarge for prefill to avoid underutilizing decode instances [22]. We evaluate HACK using models listed in Table 2 with their Tensor Parallelism (TP) and Pipeline Parallelism (PP) sizes [22, 26] across various datasets (IMDb [17], HumanEval [10], GSM8K [4], arXiv [12], and Cocktail [13]).

Name	GPUs	GPU mem.	Bw	vCPUs	Memory	Purpose
g5.12xlarge	4 A10G	96 GiB	40 Gbps	48	192 GiB	Prefill
p3.8xlarge	4 V100	64 GiB	10 Gbps	32	244 GiB	Prefill
g4dn.12xlarge	4 T4	64 GiB	50 Gbps	48	192 GiB	Prefill
g6.12xlarge	4 L4	96 GiB	40 Gbps	48	192 GiB	Prefill
p4de.24xlarge	8 A100	640 GiB	400 Gbps	96	1152 GiB	Prefill/Decode

Table 1. GPU instances.

Model	A10G, L4	V100, T4	A100
Mistral-v0.3 7B [7]	TP=4, no PP	TP=4, no PP	no TP, no PP
Phi-3 14B [6]	TP=2, PP=2	TP=2, PP=2	no TP, no PP
Yi 34B [1]	TP=4, PP=2	TP=4, PP=2	TP=4, no PP
Llama-3.1 70B [5]	TP=4, PP=2	TP=4, PP=4	TP=4, no PP
Falcon 180B [3]	TP=4, PP=5	TP=4, PP=8	TP=4, PP=2

Table 2. Models.

When achieving 99% of the accuracy of the disaggregated baseline without quantization, HACK provides up to 86% KV size reduction, up to 70.9% JCT reduction over the disaggregated baseline, and up to 52.3% JCT reduction over quantization methods (CacheGen and KVQuant).

Acknowledgements

This work was supported in part by U.S. NSF grants NSF-2421782, NSF-2350425, NSF-2319988, NSF-2206522, Microsoft Research Faculty Fellowship 8300751, Amazon research award, AWS Cloud Credit for Research, and the Commonwealth Cyber Initiative (CCI), an investment in the advancement of cyber research, innovation and workforce development. For more information about CCI, visit cyberinitiative.org. This work was also supported in part by ACE, one of the seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

References

[1] 2025. 01-ai Model Yi. https://huggingface.co/01-ai/Yi-34B-200K. (2025).

[2] 2025. The code of HACK. https://anonymous.4open.science/r/HKVQ. (2025).

[3] 2025. Falcon-180B. https://huggingface.co/tiiuae/falcon-180B. (2025).

[4] 2025. GSM8K. https://huggingface.co/datasets/openai/gsm8k. (2025).

[5] 2025. Meta Llama-3.1. https://llama.meta.com/. (2025).

[6] 2025. Microsoft Phi-3. https://huggingface.co/microsoft/Phi-3-medium-128k-instruct. (2025).

[7] 2025. Mistral-v0.3. https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.3. (2025).

[8] 2025. NVIDIA Dynamo. https://developer.nvidia.com/blog/introducing-nvidia-dynamo-a-low-latency-distributed-inference-framework-for-scaling-reasoning-ai-models/. (2025).

[9] Inc. Amazon Web Services. 2024. Recommended AWS GPU Instances. https://docs.aws.amazon.com/dlami/latest/devguide/gpu.html. (2024).

[10] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. (2021). arXiv:cs.LG/2107.03374 https://arxiv.org/abs/2107.03374

[11] Tencent Cloud. 2024. Tencent Cloud - A100 Instances. https://www.tencentcloud.com/document/product/560/19701#GT4. (2024).

[12] Arman Cohan, Franck Dernoncourt, Doo Soon Kim, Trung Bui, Seokhwan Kim, Walter Chang, and Nazli Goharian. 2018. A Discourse-Aware Attention Model for Abstractive Summarization of Long Documents. (2018). arXiv:cs.CL/1804.05685 https://arxiv.org/abs/1804.05685

[13] Sunhao Dai, Weihao Liu, Yuqi Zhou, Liang Pang, Rongju Ruan, Gang Wang, Zhenhua Dong, Jun Xu, and Ji-Rong Wen. 2024. Cocktail: A Comprehensive Information Retrieval Benchmark with LLM-Generated Documents Integration. (2024). arXiv:cs.IR/2405.16546 https://arxiv.org/abs/2405.16546

[14] Coleman Richard Charles Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Sophia Shao, Kurt Keutzer, and Amir Gholami. 2024. KVQuant: Towards 10 Million Context Length LLM Inference with KV Cache Quantization. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*. https://openreview.net/forum?id=0LXotew9Du

[15] Cunchen Hu, Heyang Huang, Junhao Hu, Jiang Xu, Xusheng Chen, Tao Xie, Chenxi Wang, Sa Wang, Yungang Bao, Ninghui Sun, and Yizhou Shan. 2024. MemServe: Context Caching for Disaggregated LLM Serving with Elastic Memory Pool. (2024). arXiv:cs.DC/2406.17565 https://arxiv.org/abs/2406.17565

[16] Cunchen Hu, Heyang Huang, Liangliang Xu, Xusheng Chen, Jiang Xu, Shuang Chen, Hao Feng, Chenxi Wang, Sa Wang, Yungang Bao, Ninghui Sun, and Yizhou Shan. 2024. Inference without Interference: Disaggregate LLM Inference for Mixed Downstream Workloads. (2024). arXiv:cs.DC/2401.11181 https://arxiv.org/abs/2401.11181

[17] IMDb. 2020. Genre Classification Dataset IMDb. https://www.kaggle.com/datasets/hijest/genre-classification-dataset-imdb. (2020).

[18] Philip Kiely. 2024. NVIDIA A10 vs A10G for ML model inference. https://www.baseten.co/blog/nvidia-a10-vs-a10g-for-ml-model-inference/. (2024).

[19] John R. Klauder. 1983. Stochastic Quantization. In *Recent Developments in High-Energy Physics*, H. Mitter and C. B. Lang (Eds.). Springer Vienna, Vienna, 251–281.

[20] Minghao Li, Ran Ben Basat, Shay Vargaftik, ChonLam Lao, Kevin Xu, Michael Mitzenmacher, and Minlan Yu. 2024. THC: Accelerating Distributed Deep Learning Using Tensor Homomorphic Compression. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. USENIX Association, Santa Clara, CA, 1191–1211. https://www.usenix.org/conference/nsdi24/presentation/li-minghao

[21] Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, Michael Maire, Henry Hoffmann, Ari Holtzman, and Junchen Jiang. 2024. CacheGen: KV Cache Compression and Streaming for Fast Large Language Model Serving. In *Proceedings of the ACM SIGCOMM 2024 Conference (ACM SIGCOMM '24)*. Association for Computing Machinery, New York, NY, USA, 38–56. https://doi.org/10.1145/3651890.3672274

[22] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. 2024. Splitwise: Efficient Generative LLM Inference Using Phase Splitting. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. 118–132. https://doi.org/10.1109/ISCA59077.2024.00019

[23] Ruoyu Qin, Zheming Li, Weiran He, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. 2024. Mooncake: A KVCache-centric Disaggregated Architecture for LLM Serving. (2024). arXiv:cs.DC/2407.00079 https://arxiv.org/abs/2407.00079

[24] Bita Darvish Rouhani, Ritchie Zhao, Ankit More, Mathew Hall, Alireza Khodamoradi, Summer Deng, Dhruv Choudhary, Marius Cornea, Eric Dellinger, Kristof Denolf, Stosic Dusan, Venmugil Elango, Maximilian Golub, Alexander Heinecke, Phil James-Roxby, Dharmesh Jani, Gaurav Kolhe, Martin Langhammer, Ada Li, Levi Melnick, Maral Mesmakhosroshahi, Andres Rodriguez, Michael Schulte, Rasoul Shafipour, Lei Shao, Michael Siu, Pradeep Dubey, Paulius Micikevicius, Maxim Naumov, Colin Verrilli, Ralph Wittig, Doug Burger, and Eric Chung. 2023. Microscaling Data Formats for Deep Learning. (2023). arXiv:cs.LG/2310.10537 https://arxiv.org/abs/2310.10537

[25] Foteini Strati, Sara Mcallister, Amar Phanishayee, Jakub Tarnawski, and Ana Klimovic. 2024. DéjàVu: KV-cache Streaming for Fast, Fault-tolerant Generative LLM Serving. In *Proceedings of the 41st International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (Eds.), Vol. 235. PMLR, 46745–46771. https://proceedings.mlr.press/v235/strati24a.html

[26] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. 2024. DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. USENIX Association, Santa Clara, CA, 193–210. https://www.usenix.org/conference/osdi24/presentation/zhong-yinmin