# Accelerate LLM inference with Asynchronous model offload

Jie Ye Bogdan Nicolae Xian-He Sun Anthony Kougkas
jye20@hawk.iit.edu, bnicolae@anl.gov, sun@iit.edu, and akougkas@iit.edu

Gnosis Research Center @ ILLINOIS TECH

StoreHub

## Introduction

Large language models (LLMs) have demonstrated their effectiveness across diverse generation and data management tasks. Modern LLMs typically come with tens to hundreds of billions of parameters, such as Llama-3.3-70B-Instruct, GPT-3 175 billion parameters, and Llama-3.1-405B. Moreover, once trained, the models must be deployed to handle requests with varying prompt lengths from numerous concurrent users. As a result, serving and deploying these models demands hundreds of gigabytes of memory, which far exceeds the memory capacity of modern GPUs like NVIDIA A100. Therefore, with the fast growth of the parameter size, it becomes increasingly challenging to deploy these models, especially in environments with limited memory.

A common approach to overcome memory constraints is to offload some model parameters to host memory and load them back to GPU memory on-demand during inference. Existing vLLM proposes a naive offloading mechanism that offloads layers starting from block 0 and fetches the blocks synchronously right before performing the computation of that layer. These inefficient data and synchronous data transfers between the GPU and host memory can introduce bottlenecks, severely degrading inference performance.

## Our Approach

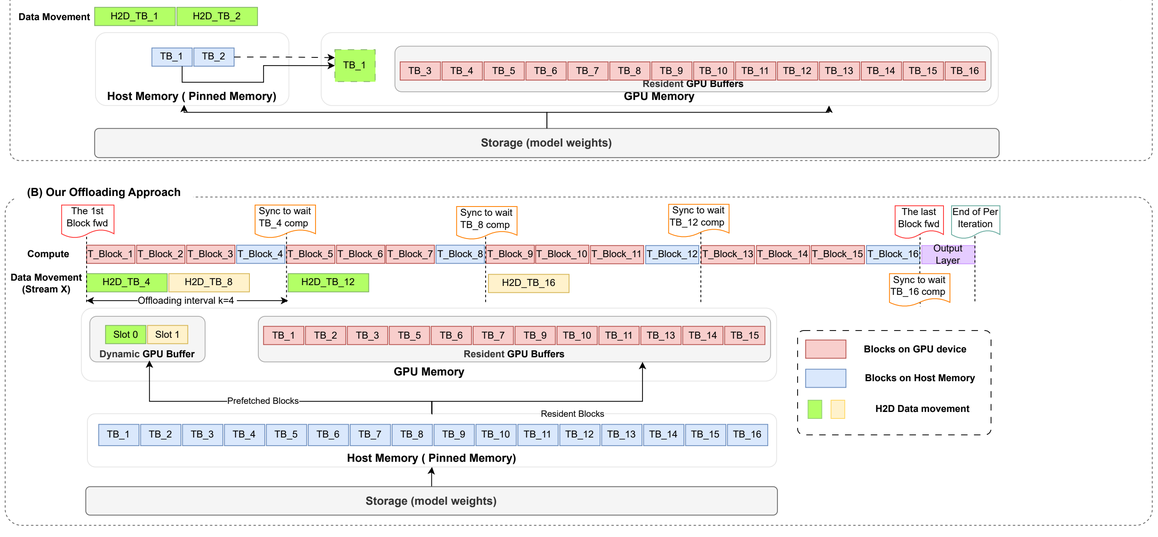


Figure 3: Comparison of vLLM's naive model offloading approach vs. Our asynchronous offloading approach

### vLLM's model Offload Approach:

1. Progressively offloads transformer blocks (0 → N) until reaching the pre-configured offload memory size
2. **Blocking/On-demand fetching:** Offloaded blocks are fetched on-demand before execution, stalling and delaying computation due to long data movement

### Our model Offload Approach:

- **Offload interval:** Selectively offload blocks based on a pre-configured offload interval *k*.
- **Asynchronous Prefetch with a double-buffer:** Enables concurrent computation and data movement by prefetching upcoming layers in advance, optimizing hardware usage.

## Evaluation Method

- **Platform:** ALCF's Polaris. Each Node: 4×A100 GPUs with 40 GB HBM2 on each GPU
- **Models Workloads:** Llama-3.1-8B model with LongBench's Multi-news dataset (200 requests). The average prompt length of each request is 2000, and the max output length is 512.

### Result: Throughput (Naive vs. Our Approach)

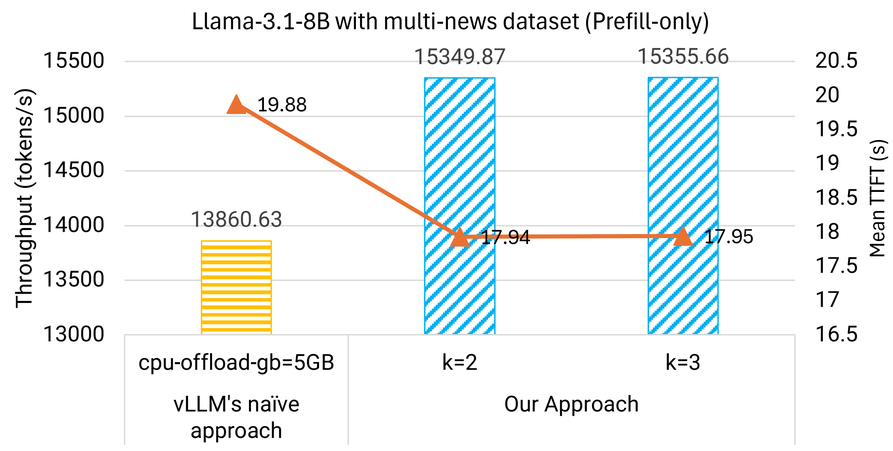


Figure 4: Throughput and TTFT of Prefill-only: vLLM's naive method vs. our approach (gpu_limit=0.8, reqs=200)

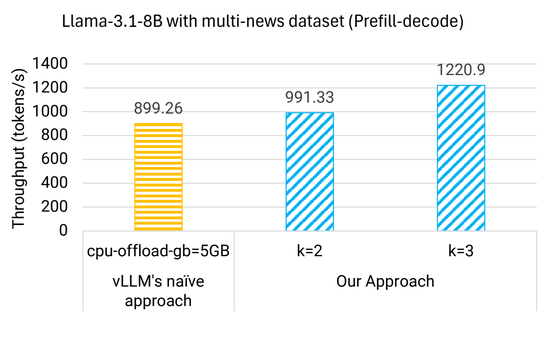


Figure 5: Throughput of Prefill-decode: vLLM's naive method vs. our approach (gpu_limit=0.8, reqs=200)

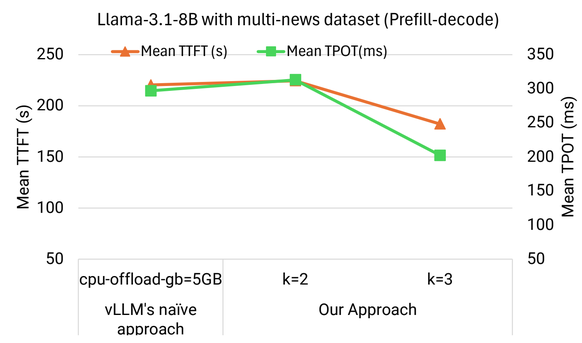


Figure 6: Latency of Prefill-decode: vLLM's naive method vs. our approach (gpu_limit=0.8, reqs=200)

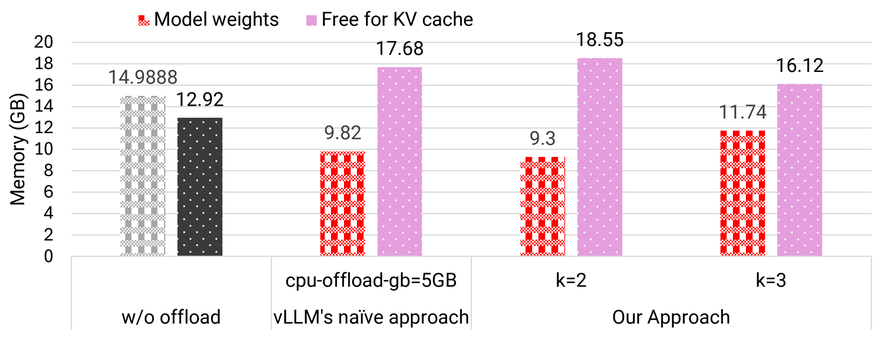


Figure 7: Memory Usage of model weights and max available memory for KV cache: vLLM's naive method vs. our approach (gpu_limit=0.8, reqs=200)

### Observations:

- Compared to vLLM's built-in offload method, our approach delivers at least 1.1x higher inference throughput while saving more GPU memory.

## Conclusion

To summarize, we propose a method to selectively offload the transformer blocks based on the offloading interval k, and asynchronously prefetch the blocks to overlap I/O and computation. The results show we can achieve higher inference throughput while saving more GPU memory than the existing approach.

## Acknowledgments

This material is based upon work supported in part by the National Science Foundation (NSF), Division of Computer and Network Systems (CISE/CNS), under Grant CCRI-CISE 2346504.

# Accelerate LLM inference with Asynchronous model offload

Jie Ye
Illinois Institute of Technology
Chicago, USA
jye20@hawk.iit.edu

Bogdan Nicolae
Argonne National Laboratory
Lemont, IL, USA
bnicolae@anl.gov

Xian-He Sun
Illinois Institute of Technology
Chicago, IL, USA
sun@iit.edu

Anthony Kougkas
Illinois Institute of Technology
Chicago, IL, USA
akougkas@iit.edu

## ABSTRACT

Large language models (LLMs) have shown remarkable capabilities in various domains but their deployment remains challenging due to their massive memory requirements. While much research focuses on scaling pre-training, efficient inference is equally critical, particularly under memory constraints. A common solution is offloading model parameters to host memory and fetching them back during inference. However, existing approaches, like vLLM's synchronous offloading, introduce significant latency due to inefficient data transfers between GPU and host memory, degrading inference performance. In this work, we propose a novel offloading strategy that (1) selectively offloads transformer blocks with configurable granularity, enabling flexible memory management, and (2) leverages asynchronous prefetching via a separate CUDA stream to overlap I/O and computation, minimizing idle time. Our approach reduces inference latency by optimizing data movement, making LLM deployment more efficient in memory-constrained environments.

## 1 INTRODUCTION

Large language models (LLMs) have demonstrated their effectiveness across diverse generation and data management tasks. Modern LLMs typically come with tens to hundreds of billions of parameters, such as Llama-3.3-70B-Instruct [1], GPT-3 175 billion parameters [2], and Llama-3.1-405B [3]. Pre-training of LLMs and transformers is known to take weeks if not months even using the powerful HPC systems. For this reason, a large part of the existing work focuses on exploring how to scale pre-training. However, inferences are an equally important problem: once pre-trained, the models must be deployed to handle requests with varying prompt lengths from numerous concurrent users. With the fast growth of parameter sizes, it becomes increasingly challenging to deploy these models, especially in environments with limited memory, since the memory demands of the massive number of parameters in modern LLMs (e.g., hundreds of gigabytes of memory) can easily surpass the memory capacity of modern GPUs like NVIDIA A100. To mitigate the memory limitation problem, a common approach is to offload part of the model parameters to host memory and then fetch them back to GPU memory on demand during inference. For example, the widely used inference engine vLLM proposes a naive offloading mechanism that offloads layers starting from block 0 and fetches the blocks synchronously right before performing the computation of that layer. These inefficient data and synchronous data transfers between the GPU and host memory can introduce bottlenecks, severely degrading inference performance.

*Conference'17, July 2017, Washington, DC, USA*
2025. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

**Contribution:** To reduce the latency caused by data movement, we propose a novel approach that: (1) selectively offloads transformer blocks based on a configurable offloading interval, easy to control of the offload granularity; (2) can asynchronously prefetch the offloaded transformer blocks with a separate CUDA stream to overlap the I/O and computation during inference.

## 2 OUR APPROACH

Fig. 1 compares the difference between vLLM's naive offloading approach and our asynchronous prefetching and offloading approach. vLLM's built-in model offload approach progressively offloads transformer blocks starting from block 0 to block N, and stops offloading until reaching the pre-configured maximum offload memory size. In a model forward pass, the offloaded blocks are fetched on demand synchronously right before executing the forward pass of that block, which stalls and delays the computation due to the long data movement cost.

Instead of simply offloading the transformer blocks from block 0 to block N. We provide a configurable parameter, offloading interval k, and will selectively offload blocks based on the offloading interval. Here, an offloading interval of k means that for every k transformer blocks, the state of the last transformer block is offloaded to the Host memory. In addition, we prefetch the offloaded transformer blocks asynchronously in advance using a separate CUDA stream to overlap the I/O and computation. With these two optimizations, we can accelerate the inference throughput when deploying LLMs in a GPU memory-constrained environment.

## 3 EVALUATIONS

### 3.1 Experimental Setup

All the experiments on the ALCF's Polaris platform. Each node has 4×A100 GPUs with 40 GB HBM2 on each, and 1×512 GB DDR4 RAM. We use the Llama-3.1-8B model with vLLM 0.7.0 for inference on LongBench's multi-news dataset (200 requests, avg. prompt length: 2,113, max output: 512).

### 3.2 Results: Inference Serving Performance and Memory Analysis

**Throughput:** Fig. 2 and Fig. 3 presents the inference throughput when serving 200 requests with the Llama-3.1-8B model with different model weights offloading methods. We observe that our approach can achieve at least 1.1x higher throughput compared with vLLM's naive offload approach, confirming that efficient data movement can accelerate the inference performance.

**Latency:** As Fig. 2 and Fig. 4 show, our approach can also get lower TTFT (i.e., time to first token) and TPOT (i.e., time per output token) compared with vLLM's vLLM's naive offload approach.

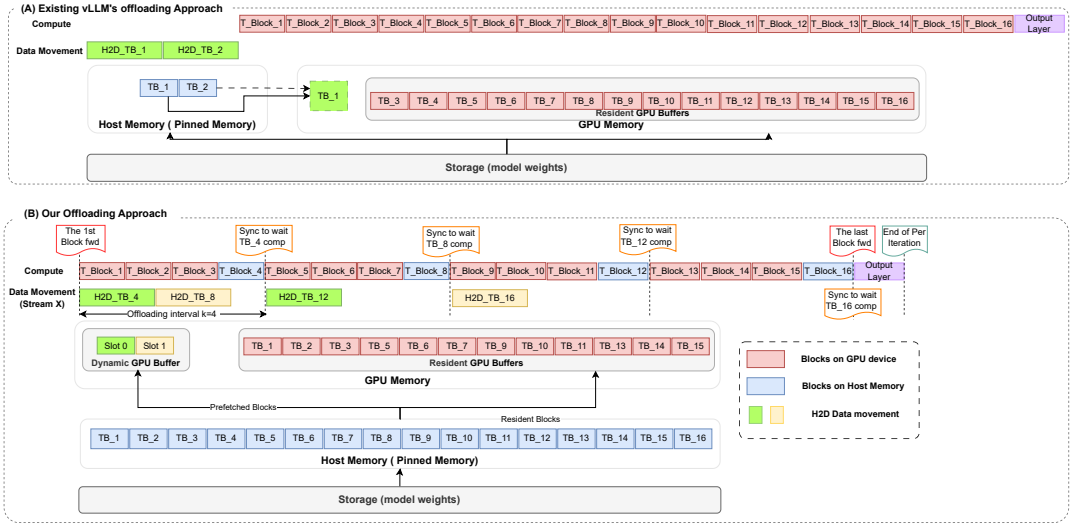


**Figure 1: An overview of vLLM's built-in offloading approach and our offloading approach. (a) vLLM's naive offloading approach: offload transformer blocks from block 0 to block N and synchronously fetch the offloaded blocks; (b) Our approach: selectively offload blocks based on offloading interval k and asynchronously prefetch the required blocks**

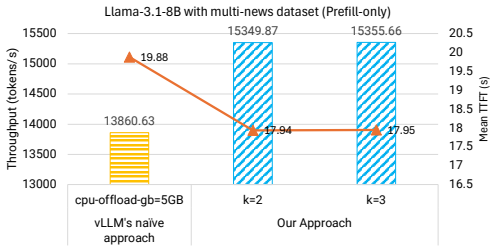


**Figure 2: Throughput and Latency of prefill-only: vLLM's naive method vs. our approach (gpu_limit=0.8, reqs=200)**

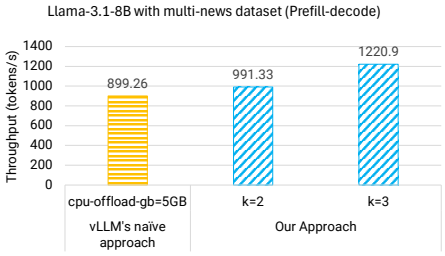


**Figure 3: Throughput of prefill-decode: vLLM's naive method vs. our approach (gpu_limit=0.8, reqs=200)**

**Memory Analysis:** Fig 5 demonstrates the memory usage of model weights and the maximum free available GPU can be preserved for KV cache when using different offloading methods: without offloading, vLLM's built-in offloading, and our offloading approach. The results show that our approach will use less memory when setting the offloading interval k = 2.

## 4 CONCLUSIONS

To summarize, we propose a method to selectively offload the transformer blocks based on the offloading interval k, and asynchronously prefetch the blocks to overlap I/O and computation. The results show that we can achieve higher inference throughput while saving more GPU memory compared with the existing approach.

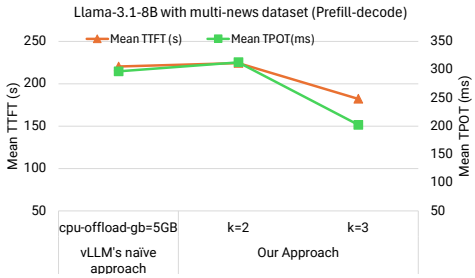


**Figure 4: Latency of prefill-decode: vLLM's naive method vs. our approach (gpu_limit=0.8, reqs=200)**

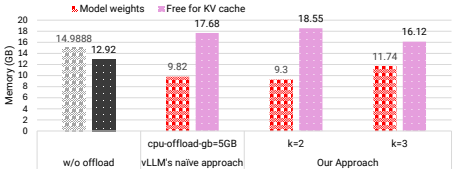


**Figure 5: Memory Analysis: Model Weights vs. Maximum Free Memory for KV Cache with different offload methods**

## ACKNOWLEDGMENT

This material is based upon work supported in part by the National Science Foundation (NSF), Division of Computer and Network Systems (CISE/CNS), under Grant CCRI-CISE 2346504.

## REFERENCES

[1] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783* (2024).
[2] Anis Koubaa. 2023. Gpt-4 vs. gpt-3. *Authorea Preprints* (2023).
[3] Raja Vavekanand and Kira Sam. 2024. Llama 3.1: An in-depth analysis of the next-generation large language model.