Optimizing Large Language Model Inference Through CPU-GPU Hybrid Execution

Final Report

Jacob Benjamine Ioffe
Department of Computer Science
Cornell-Tech
New York, NY 10044 USA
ji97@cornell.edu

Abstract

Scaling Large Language Models (LLMs) to tens or even hundreds of thousands of tokens in their context windows pushes the limits of current inference techniques. Although prior work has primarily focused on production-scale GPU clusters, little attention has been paid to resource-constrained, desktop-grade consumer hardware—environments where cost and availability motivate alternative approaches. In these scenarios, GPU-only decoding often underutilizes GPU parallelism during the sequential decode phase and struggles with memory demands as context lengths grow.

In this work, we investigate a hybrid CPU-GPU inference paradigm that offloads LLM decoding to the CPU, aiming to enable ultra-long context inference on commodity devices. We experiment with the LLaMA 3.2 1B-parameter model, evaluating dynamic and offloaded Key-Value (KV) cache strategies, quantization approaches, and single-batch inference scenarios. Our results show that while CPU decoding is less efficient at short contexts, it achieves nearly one-third of the GPU throughput at very large contexts (e.g., 32K tokens), allowing us to double or even quadruple the maximum feasible context length compared to GPU-only decoding without running out of memory. These findings highlight that offloading decoding to the CPU offers a viable path to long-context inference in consumer-grade settings, prompting further research into advanced cache management, compression techniques, and adaptive runtime policies to bridge the efficiency gap and unlock new, memory-intensive applications of LLMs.

1 Introduction

Modern Large Language Models (LLMs) have demonstrated remarkable capabilities in understanding and reasoning over extensive textual input, often spanning tens of thousands of tokens. However, achieving efficient inference for such long contexts remains challenging on commodity hardware. Traditionally, both the *prefill* phase (which processes the entire input context in parallel) and the *decode* phase (which generates output tokens one-by-one) are executed on GPUs. While GPUs excel at parallelized batch operations, the inherently sequential decode phase can lead to underutilization of GPU resources and escalating memory demands due to the rapidly growing Key-Value (KV) cache.

These constraints are particularly salient in desktop-grade consumer environments, where GPU memory capacity may be limited and cost considerations preclude large-scale clusters. To address these issues, we explore hybrid CPU-GPU execution strategies that offload the decode phase onto the CPU. Offloading has the potential to alleviate GPU memory pressure and enable significantly larger context lengths, albeit at a potential cost to throughput and latency. Our primary aim is to characterize these trade-offs, providing insights into when and how CPU offloading can be advantageous.

2 Research Question and Hypothesis

Our core question is:

> *How does CPU decode offloading impact achievable context length, latency, and memory consumption for single-batch LLM inference on desktop-grade GPUs?*

We hypothesize that:

> *CPU decode offloading can enable doubling the context length capacity (e.g., from 16K to 32K tokens) at the expense of no more than a 50% reduction in throughput compared to GPU-only decoding.*

This hypothesis reflects a strategic trade-off: achieving significantly longer context windows while only partially sacrificing throughput, potentially opening new applications and deployment scenarios on constrained hardware.

3 Related Work

Our work on hybrid CPU-GPU execution for LLM inference builds upon several research threads in serving systems, phase-aware scheduling, and memory management. We also draw motivation from the latest developments in LLM architectures, particularly the Llama 3 family of models Dubey et al. (2024).

Recent systems research has made significant advances in optimizing LLM inference through various approaches. Key innovations include continuous batching and efficient memory management techniques for key-value caches Patel et al. (2024). The Splitwise system demonstrates that splitting computation between prompt and token generation phases can lead to substantial efficiency gains by matching computational requirements to hardware capabilities. This approach is particularly relevant given the distinct characteristics of these phases: prompt computation being compute-intensive while token generation is more memory-bound.

Our work extends these insights by exploring optimized KV cache strategies and CPU decode offloading specifically for the Llama 3.2-1B model. We focus on single-batch inference scenarios on consumer-grade GPUs, providing a detailed investigation of hybrid execution strategies for edge devices. This investigation is particularly timely given the growing interest in deploying smaller, efficient LLMs like Llama 3.2-1B in resource-constrained environments while maintaining high performance.

4 Methods

We conduct experiments using the LLaMA 3.2 1B parameter model, running on a desktop-grade environment featuring an NVIDIA RTX 6000 GPU and an Intel Xeon CPU. Our benchmarking framework isolates the prefill and decode steps, allowing fine-grained control over where computations and cache storage occur. We measure:

- **Throughput**: tokens generated per second during decoding.
- **Memory Usage**: peak GPU and CPU memory consumption at varying context lengths.
- **Latency**: mean and tail latencies for token generation.

To facilitate flexible experimentation with KV cache placement and management, we leverage the HuggingFace Cache framework. This infrastructure standardizes cache operations (updates, reorderings for beam search, etc.) across multiple backends. We consider:

1. **Dynamic Cache (GPU-only)**: The entire KV cache is maintained on the GPU, scaling dynamically with context length.

2. **Offloaded Cache (CPU-GPU)**: Segments of the KV cache are offloaded to CPU memory, with prefetching and eviction policies to mitigate data transfer overheads.
3. **Quantized Cache**: A simple min-max quantization scheme to reduce KV cache footprint, though initial attempts showed limited effectiveness.

These strategies enable us to fairly compare different device placements and compression techniques. By systematically varying context lengths and applying these cache configurations, we quantify the throughput-memory-latency trade-offs and assess the feasibility of CPU decode offloading for ultra-long context inference on commodity hardware.

5 Results

5.1 Initial KV Cache Strategy Analysis

Our initial experiments evaluate different KV cache strategies for GPU-based decoding across varying context lengths. We observe distinct performance cliffs as context length increases: from 4K to 8K tokens, throughput decreases by approximately 15%, followed by a 35% reduction from 8K to 16K tokens. The dynamic cache strategy achieves the highest throughput at moderate context lengths, reaching 21.39 tokens/s at 4K tokens with 2.64 GB peak memory usage. At 16K tokens, throughput drops to 12.54 tokens/s while memory consumption increases to 6.37 GB.

Comparing strategies at moderate context lengths reveals that while offloaded caches provide more stable performance across context lengths, they consistently operate about 20% slower than dynamic caches. Our initial experiments with quantized caches show minimal benefits, with memory usage patterns similar to unquantized approaches. This suggests that more sophisticated quantization schemes may be necessary for meaningful memory reduction.

At extreme context lengths, the performance differences between strategies become less pronounced. All approaches converge to similar throughput levels between 5.1-5.5 tokens/s at 32K tokens, with dynamic cache maintaining a slight edge at 5.49 tokens/s compared to 5.13 tokens/s for offloaded cache. While all strategies successfully reach 32K token contexts, they all suffer from substantial memory overhead, requiring over 18GB at maximum context lengths. These severe resource constraints and significant throughput degradation—a roughly 60% drop from 16K to 32K tokens—indicate fundamental limitations in GPU-only approaches, motivating our investigation of CPU decode offloading as an alternative solution.

5.2 CPU Decode Offloading and Hybrid Execution

Our exploration shows that CPU decode offloading enables processing of extremely large context lengths that exceed GPU memory limits. While GPU-only decoding encounters CUDA memory allocation errors beyond $\sim$65K tokens, CPU decoding successfully processes contexts as large as 131K tokens. This result directly supports our hypothesis that offloading the decoding step to CPU allows for ultra-long context inference—albeit at reduced throughput.

At shorter context lengths (e.g., 1K–2K tokens), CPU decoding achieves only about 10% of GPU throughput. However, as we increase the context length, CPU decoding becomes relatively more efficient. By 32K contexts, CPU decoding reaches approximately 27–31% of GPU decode speed, and at longer output lengths, this ratio improves further. Such trends indicate that fixed overheads in CPU decoding are better amortized as the context and output sizes grow.

Figure 1 (left) presents the CPU-to-GPU throughput ratio across context lengths, showing gradual improvement with scale. At 32K context, CPU decoding provides a substantial fraction of GPU speed while avoiding GPU memory exhaustion. Figure 1 (right) shows that token-level tail latencies (P99) increase predictably with context length under CPU decoding, but remain stable and manageable, indicating no catastrophic performance cliffs.

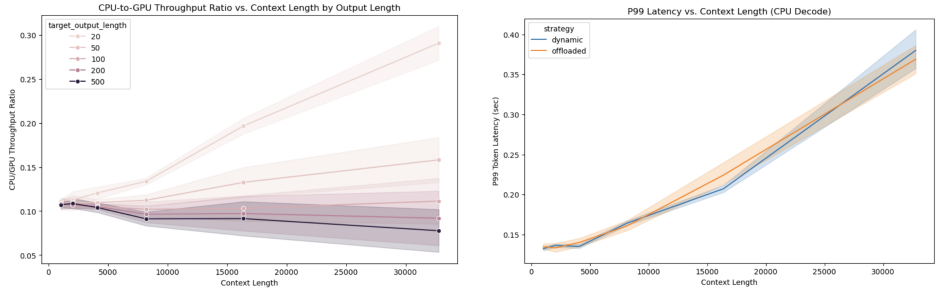

Figure 1: **Left:** CPU-to-GPU throughput ratio vs. context length for various output lengths. CPU decoding starts at about 10% of GPU throughput at small contexts but improves to nearly 30% at 32K tokens, especially at longer output sequences (e.g., 500 tokens). **Right:** P99 latency for CPU decoding as a function of context length shows predictable, gradual increases even as context lengths grow beyond 30K tokens.

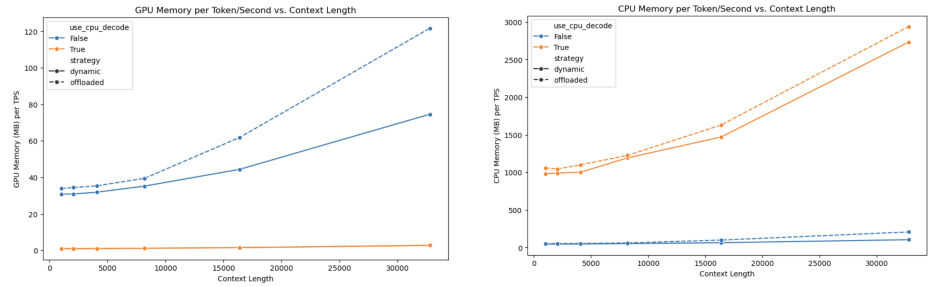

Figure 2: **Left:** GPU Memory per tokens/second vs. context length. GPU decoding (blue) becomes increasingly GPU-memory-inefficient, while CPU decoding (orange) remains flat near zero GPU memory use. **Right:** CPU Memory per tokens/second vs. context length. CPU decoding imposes a higher CPU memory cost as contexts scale, indicating a clear trade-off in resource usage.

In terms of memory trade-offs, CPU decoding dramatically reduces GPU memory usage but shifts the burden onto CPU memory. Figures 2 (right and left) illustrate these dynamics: GPU-only decoding rapidly becomes GPU-memory-inefficient with larger contexts, whereas CPU decoding keeps GPU memory usage nearly constant at the expense of consuming more CPU memory per token/second.

5.2.1 Quantitative Analysis Across Context Lengths

Table 1 presents a comprehensive comparison of memory usage and throughput across different context lengths and decoding strategies. At moderate context lengths (32K tokens), GPU-only decoding with dynamic cache achieves the highest throughput at 45.00 tokens/s while consuming 2.5 GB GPU and 3.6 GB CPU memory. The offloaded cache variant shows somewhat reduced performance at 29.06 tokens/s with comparable memory utilization. In contrast, CPU decoding strategies maintain a consistent but lower throughput of approximately 4 tokens/s while using minimal GPU memory ($\sim$8 MB), though at the cost of increased CPU memory consumption ranging from 7.9 to 8.4 GB.

The advantages of CPU decoding become particularly evident at ultra-long contexts (64K tokens), where GPU-only strategies fail due to CUDA out-of-memory errors. Both dynamic and offloaded CPU decoding strategies remain operational at this scale, though with reduced throughput of approximately 1.9 tokens/s. Notably, GPU memory usage remains minimal while CPU memory consumption increases only moderately to 8.6 GB. The offloaded cache strategy shows marginally better performance (1.92 vs 1.85 tokens/s) and slightly lower

CPU memory usage (8.60 vs 8.65 GB) compared to the dynamic cache approach at these extreme lengths.

These results illuminate a fundamental trade-off in LLM inference: while CPU decoding incurs a substantial throughput penalty, it enables processing of previously unattainable context lengths with predictable resource utilization. This capability opens new possibilities for applications requiring ultra-long context processing, despite the reduced processing speed.

Table 1: Memory and Throughput Analysis Across Context Lengths. At 32K tokens, both GPU and CPU decoding succeed, while at 64K tokens, only CPU decoding remains viable due to GPU OOM errors.

Context Length	CPU Decode	Strategy	GPU Mem (MB)	CPU Mem (MB)	Tokens/s
32K	False	Dynamic	2,517.19	3,598.43	45.00
	False	Offloaded	2,385.65	3,960.03	29.06
	True	Dynamic	8.12	7,950.41	4.15
	True	Offloaded	8.12	8,426.19	4.01
64K	True	Dynamic	8.12	8,649.08	1.85
	True	Offloaded	8.12	8,600.33	1.92
	False	Dynamic	*Failed - CUDA Out of Memory*		
	False	Offloaded	*Failed - CUDA Out of Memory*		

6 Discussion and Future Directions

Our findings highlight a fundamental trade-off in hybrid CPU-GPU inference for large language models: while CPU decoding offloading enables surpassing GPU memory constraints and achieving significantly larger context windows, it incurs non-trivial throughput penalties. For scenarios that prioritize maximal context length—such as large-scale code analysis, long-form content synthesis, or multi-document reasoning—this approach could unlock new capabilities on commodity hardware. However, practitioners must weigh these gains against the increased inference latency and reduced efficiency, particularly at shorter context lengths.

A key insight emerging from our experiments is that the relative cost of CPU decoding diminishes as context lengths grow. Although CPU decoding is initially far less efficient than its GPU counterpart, at extremely large contexts (32K and beyond), it approaches roughly 30% of the GPU throughput, reflecting better amortization of fixed overheads. This observation suggests that hybrid strategies might be most beneficial for specialized, long-context use cases rather than general-purpose inference.

Despite these encouraging results, several critical challenges and avenues for improvement remain. Foremost among them is the development of more sophisticated key-value (KV) cache management and quantization techniques. Initial attempts at KV cache quantization, including min-max approaches and experimental methods such as *quest* or *KIVI*, provided limited gains and encountered compatibility issues with LLaMA3.2's attention mechanisms and hardware configurations. Device mismatches, CUDA driver inconsistencies, and version incompatibilities with quantization toolkits (e.g., `quanto`) all contributed to significant engineering overhead. These practical obstacles underscore the complexity of integrating advanced compression and quantization pipelines into LLM inference stacks, especially given the rapid evolution of model architectures and toolchains.

Future work should focus on developing robust, hardware-agnostic KV cache compression schemes tailored to CPU offloading. Novel quantization and sparsity methods that adapt to the changing nature of the KV cache over the generation process could substantially reduce CPU memory footprint without sacrificing quality. Additionally, more granular offloading strategies such as selectively offloading only a subset of layers or token positions to the

CPU remain an intriguing possibility. Our initial attempts to offload only the final layers, rather than the entire decoding phase, encountered persistent device-mismatch issues and integration complexities. With more stable toolchains and better abstractions, such partial offloading strategies could potentially strike a more favorable balance between memory footprint and computational speed.

Another promising direction involves dynamic runtime policies that leverage predictive modeling. By actively monitoring token-level throughput, latency, and resource consumption, the system could adaptively reconfigure CPU-GPU placements and KV caching strategies on-the-fly. Such adaptive inference pipelines, inspired by recent phase-aware scheduling techniques in LLM serving systems, may achieve near-optimal resource utilization across a wide range of context lengths and workloads.

Finally, the insights gained from this initial exploration motivate a broader reconsideration of decoding algorithms and model internals. Although current generation methods are GPU-oriented, certain decoding heuristics or approximate sampling techniques might prove better suited for CPU-centric execution. By co-designing the decoding algorithm and hardware placement strategies, future systems could further mitigate the performance gap, delivering truly hybrid inference regimes that harness the complementary strengths of CPU and GPU.

In summary, while CPU decode offloading currently represents a strategic compromise offering extraordinary context capacity at a notable throughput cost, it also serves as a catalyst for innovation in LLM systems design. By addressing the engineering difficulties encountered in quantization and granularity control, and by developing adaptive, data-driven device-placement policies, future research may approach GPU-level efficiency even at ultra-long context lengths. As toolchains mature and quantization strategies improve, CPU offloading may become an even more attractive option for large-scale, memory-intensive inference scenarios.

7 Conclusion

In this work, we examined the practicality of offloading the LLM decoding phase to the CPU, enabling significantly larger context lengths than is feasible with GPU-only decoding. While this approach comes at the cost of reduced throughput, it provides a strategic trade-off for memory-constrained setups. Our findings lay the groundwork for future explorations in more advanced KV cache management, adaptive device placement, and quantization techniques that could bring CPU-based inference closer to GPU-level efficiency in handling ultra-long contexts.

References

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative llm inference using phase splitting. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pp. 118–132. IEEE, 2024.