Software, Not Silicon: Why Better Algorithms Beat Bigger GPUs

Sanchit Ahuja and Harshit Garg

It has been nearly three years since the first version of ChatGPT became publicly available. In that time, both LLM research and real world usage have expanded at an exponential pace. We now have not only humans interacting with these models but also complex agentic systems where multiple models coordinate with one another. Yet, despite this rapid acceleration in demand and capability, the growth of AI hardware has not kept pace. This widening gap sets the stage for a deeper argument: relying on GPU memory alone is no longer a sustainable path for scaling LLM inference.

But if not GPU memory, then what?

The first version of vLLM [Kwon et al., 2023], an LLM inference engine, was released in June 2023 with the promise of being a unified solution for efficient model deployment. In the two years since, vLLM has evolved rapidly, not just in its support for a wide range of model architectures, but also through significant kernel-level improvements, optimization techniques, and broader system enhancements.

Inspired by previous work on tracing the evolution of complex software systems, such as the study of Linux evolution across generations [Ren et al., 2019], we set out to examine how vLLM itself has changed and matured during this period.

Experimental Setup

The vLLM framework has not yet issued a major release, so we focused our study on benchmarking its minor versions from 0.1.0 through 0.11.0. Because each version maintains backward compatibility, we selected a model supported since v0.1.0, `stabilityai/stablelm-tuned-alpha-7b` [Taori et al., 2023, Anand et al., 2023, Chiang et al., 2023]. For evaluation, we used the ShareGPT dataset[1]. All experiments were run on a single A100 40GB GPU with CUDA 12.8. We tracked five metrics in total: throughput (requests per second), the total time taken by the engine to complete inference, the average end-to-end latency per request, and the average latency per generated token and per output token.

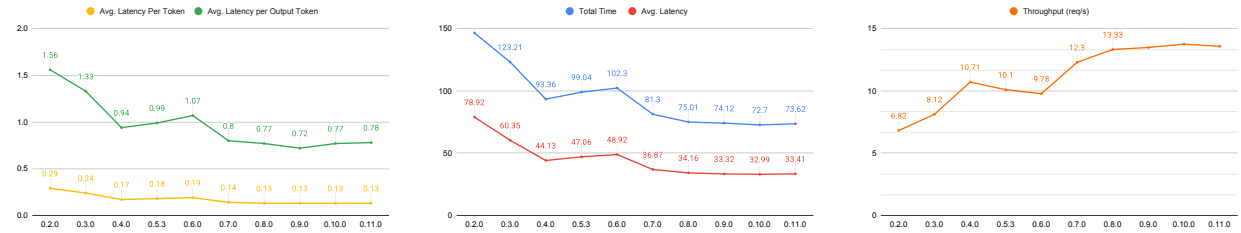
Results

Even with the same GPU, model, and dataset held constant across all versions, we observe substantial improvements in every major vLLM metric we tracked. Average latency per token dropped from 0.29s in v0.2.0 to 0.13s, and average latency per output token fell from 1.56s

[1] `https://huggingface.co/datasets/anon8231489123/ShareGPT_Vicuna_unfiltered`

to 0.78s, nearly a 2× speedup. Throughput showed a similar leap, rising from 6.82 req/s in v0.2.0 to 13.58 req/s as shown in Figure 1.

These gains highlight an important point: despite no increase in GPU memory, vLLM achieved almost 2× better inference performance purely through software evolution.



(a) Latency per token (b) Overall latency (c) Throughput performance

Figure 1: Performance metrics across vLLM versions 0.2.0 to 0.11.0. We skipped the 0.1.0 because the changes from the 1st version to the 2nd version were too drastic (in a good way!) to make a nice analysis.

What is vLLM doing to achieve this kind of performance?

To look into this, we decided to manually do a changelog study across various minor version releases of vLLM. Our changelog analysis revealed that vLLM's 2× performance improvement stems from fundamental algorithmic innovations in memory management, scheduling, and kernel design. However, the path to realizing these gains was non-linear: performance actually regressed from v0.4.0 through v0.6.x before recovering in v0.7.0. This trajectory illustrates both the power of algorithmic innovation and the engineering challenge of delivering these improvements in production systems.

The Implementation Challenge: Why Performance Regressed

Despite these algorithmic advances, performance decreased from v0.4.0 to v0.6.x. This regression reveals a critical insight: *algorithmic improvements require careful system integration to deliver real-world gains.*

We claim that this is due to several iterative improvements applied in an incoherent manner. Version 0.4.0's prefix caching illustrates this challenge. The algorithm itself is sound; reuse of shared computations is theoretically optimal. However, our ShareGPT dataset lacks the prompt repetition this algorithm targets, it mostly contains diverse user prompts without a common system prompt. The result: overhead from tracking, storing, and searching for matches without corresponding benefits. Subsequent versions compounded this problem by adding more algorithmic improvements (speculative decoding, pipeline parallelism) that each required specific conditions to provide value.

Architectural Reset: Unleashing Algorithmic Potential

Version 0.7.0's V1 engine didn't introduce new algorithms; instead, it created an architecture where existing algorithms could maximize their potential. We believe it is due to a few key principles: **simplified integration**, **compiler-driven optimization** (torch.compile is enabled by default), **workload-aware activation** (they support prefix-cache aware scheduling now), and **clean abstractions** (better isolation between algorithmic innovations prevents interference).

Post-reset, the same algorithmic innovations that previously caused regression now drive the 2× performance improvement. The algorithms didn't change, their implementation did.

Lessons for Software-Driven Scaling

This evolution demonstrates three critical points about algorithmic innovation in systems:

First, algorithmic improvements provide multiplicative gains–memory management, scheduling, and kernel optimizations compound to deliver 2× improvement without hardware changes.

Second, realizing algorithmic gains requires system-level thinking. The most elegant algorithm can degrade performance if poorly integrated. This explains why academic algorithmic improvements often fail to translate to production systems.

Third, software's ability to architecturally reset–impossible with hardware–enables recovery from accumulated complexity while preserving algorithmic advances. The V1 engine kept the algorithmic innovations while discarding the implementation debt.

The implication is clear: sustainable performance scaling requires not only algorithmic research, but also the systems engineering to deliver these algorithms effectively. The 2× improvement comes from algorithms; achieving it required architectural discipline.

Limitations

Measurement Scope

Our analysis uses a 7B parameter model from 2023 on a single A100 GPU, which may not represent modern production deployments. Larger models (70B+) might already incorporate optimizations that reduce the relative impact of framework improvements. Multi-GPU setups introduce distributed communication overheads that could dominate performance. Different hardware architectures (H100, TPUs) have distinct optimization profiles that may not benefit equally from these software improvements.

Optimization Specificity

Our changelog analysis reveals concerning patterns about generalizability. Many optimizations are hardware-specific (CUDA graphs for H200, x86-specific paths), limiting portability. Others are model-specific (MoE kernels, encoder-decoder improvements), raising questions about

whether these gains transfer to different architectures. This specificity suggests that achieving consistent improvements across diverse deployments requires significant engineering effort.

Implications for the Field

The Economics of Scale

The cost dynamics favor software optimization over hardware scaling. The H100 costs approximately 3× more than the A100 for roughly 2× the memory, while these software optimizations achieved 2× improvement at engineering cost amortized across thousands of deployments. Additionally, software optimizations reduce power consumption without new silicon, improving both operational costs and environmental impact.

Sustainable Scaling Path

The traditional approach of waiting for next-generation hardware cannot meet the exponential growth in LLM demand. A sustainable path requires treating algorithmic innovation as a first-class citizen alongside silicon advancement. Organizations should prioritize hiring and retaining systems engineers who understand both ML and low-level optimization.

Future Work

Comprehensive Benchmarking

Our analysis can be extended to modern models like Llama 3.1 70B and Mixtral to validate whether optimizations scale to larger architectures. Multi-GPU scaling efficiency across versions needs investigation to understand distributed performance. Cross-framework comparisons with TensorRT-LLM and Text Generation Inference would reveal which optimizations are fundamental versus implementation-specific.

Optimization Attribution

Future work should isolate individual optimization contributions through ablation studies. Understanding which optimizations make up or interfere would guide development priorities. Determining the theoretical limits of software optimization would help set realistic expectations for future improvements.

Predictive Modeling

Developing performance models that predict the impact of optimization before implementation could accelerate development. Identifying "optimization opportunities" in new model architectures would enable proactive rather than reactive optimization. Such models could guide resource allocation between hardware and software investments.

Conclusion

Our analysis of vLLM's evolution from v0.2.0 to v0.11.0 reveals that software innovation alone achieved nearly 2× performance improvement, matching what typically requires a hardware generation upgrade. This validates our hypothesis that traditional compute and memory metrics are insufficient indicators for inference optimization potential.

However, the increasing complexity of optimizations, from simple caching strategies to complete engine rewrites, suggests that we may be approaching the limits of "easy" software gains. The path forward isn't choosing between hardware and software, but recognizing that sustainable AI scaling requires algorithmic innovation as a first-class citizen alongside silicon advancement.

As the gap between AI capability and hardware growth widens, the question isn't whether we need bigger GPUs, but whether we're exhausting algorithmic possibilities before reaching for more memory. The evidence suggests we're not, but capturing these gains requires systematic investment in software engineering, not just model research.

The AI community stands at a crossroads: continue the expensive march toward ever-larger hardware, or invest in the software innovation that our analysis shows can deliver comparable gains at a fraction of the cost. The data suggest that the choice should be clear.

References

Y. Anand, Z. Nussbaum, B. Duderstadt, B. Schmidt, and A. Mulyar. Gpt4all: Training an assistant-style chatbot with large scale data distillation from gpt-3.5-turbo. `https://github.com/nomic-ai/gpt4all`, 2023.

W.-L. Chiang, Z. Li, Z. Lin, Y. Sheng, Z. Wu, H. Zhang, L. Zheng, S. Zhuang, Y. Zhuang, J. E. Gonzalez, I. Stoica, and E. P. Xing. Vicuna: An open-source chatbot impressing gpt-4 with 90URL `https://vicuna.lmsys.org`.

W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, page 611–626, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400702297. doi: 10.1145/3600006.3613165. URL `https://doi.org/10.1145/3600006.3613165`.

X. J. Ren, K. Rodrigues, L. Chen, C. Vega, M. Stumm, and D. Yuan. An analysis of performance evolution of linux's core operations. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, SOSP '19, page 554–569, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450368735. doi: 10.1145/3341301.3359640. URL `https://doi.org/10.1145/3341301.3359640`.

R. Taori, I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, P. Liang, and T. B. Hashimoto. Stanford alpaca: An instruction-following llama model. `https://github.com/tatsu-lab/stanford_alpaca`, 2023.