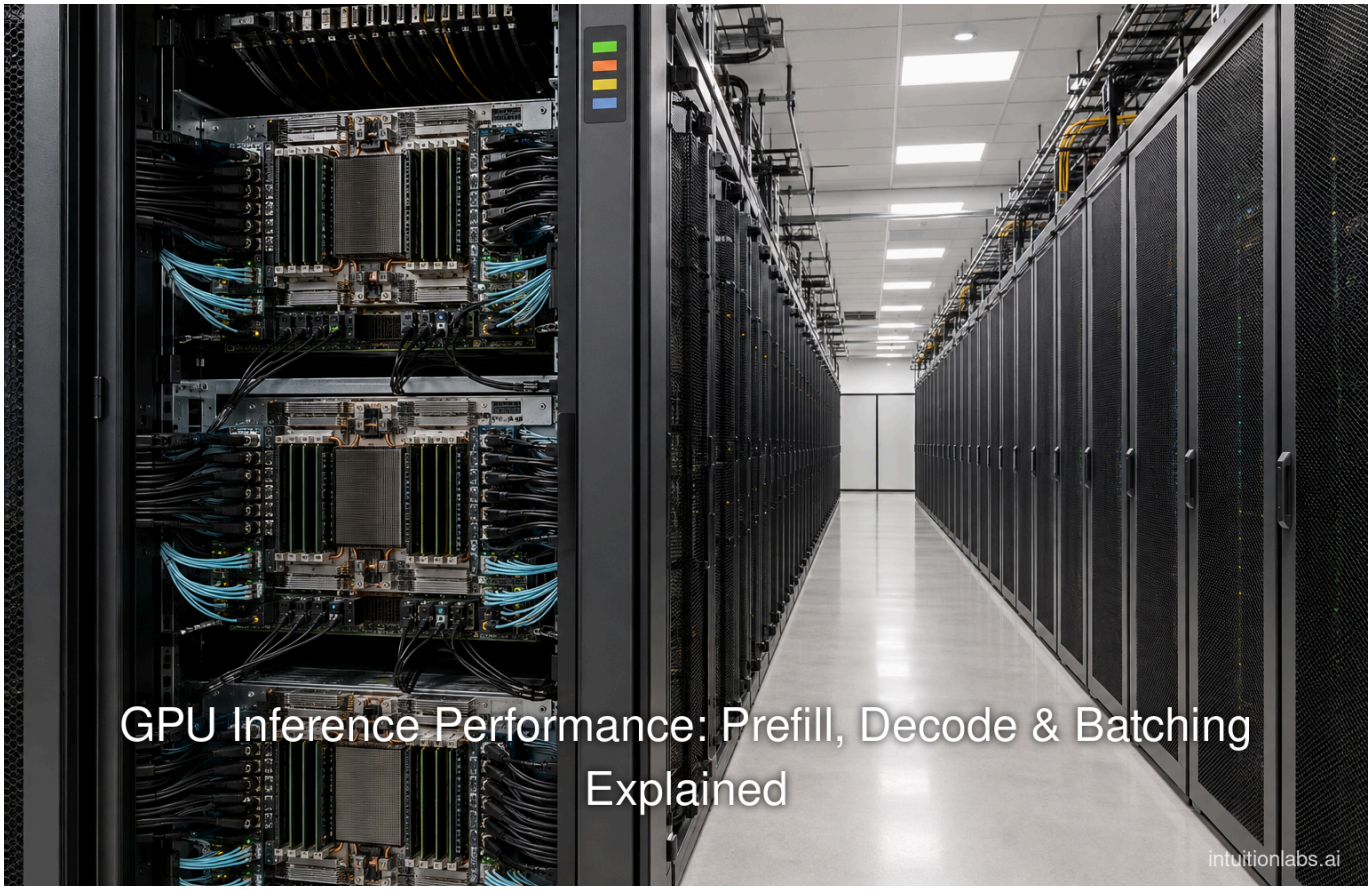
GPU Inference Performance: Prefill, Decode & Batching Explained

IntuitionLabs.ai · Adrien Laurent · 2026-09-05 · gpu inference performance · prefill decode · continuous batching · memory bandwidth · kv cache · llm inference optimization · mlperf inference · gpu utilization



Executive Summary

Large language model (LLM) inference on a graphics processing unit (GPU) splits into two phases with opposite performance bottlenecks, and conflating them is the most common error in reading a GPU benchmark or specification sheet. **Prefill**, which processes an entire input prompt at once, is compute-bound: NVIDIA's TensorRT-LLM documentation describes it as "computationally demanding" because it can saturate a GPU's parallel tensor cores (developer.nvidia.com). **Decode**, which generates output tokens one at a time, is memory-bandwidth-bound: a Hugging Face engineering analysis notes that for the decode step, "the speed is typically limited by the GPU memory bandwidth" (huggingface.co), because every step must reread the model's weights and its growing key-value (KV) cache from high-bandwidth memory (HBM). This distinction is formalized industry-wide through two latency metrics, time to first token (TTFT) for prefill and time per output token (TPOT) for decode (arxiv.org).

End-to-end serving throughput depends on the balance of compute-bound prefill and memory-bound decode work, so neither GPU memory bandwidth nor peak FLOPs alone predicts it; memory bandwidth is particularly important for decode-heavy workloads. (vLLM documentation) As of September 2026, published vendor figures show bandwidth rising from 2,039 GB/s on the NVIDIA A100 (2020) (nvidia.com) to 4.8 TB/s on the H200 (nvidia.com) and roughly 5.3 to 6 TB/s on AMD's MI300X and MI325X accelerators (amd.com). The KV cache that decode must reread scales linearly with batch size, sequence length, precision, layers, KV-head count, and head size: for a uniform-length batch, its size in bytes is batch_size × sequence_length × 2 × num_layers × num_kv_heads × head_size × bytes_per_element. NVIDIA documents K and V cache inputs with separate `numKvHeads` and `headSize` dimensions, so the hidden-size form is the multi-head-attention (MHA) special case. (NVIDIA TensorRT documentation) This is why the vLLM project's PagedAttention technique, cutting memory waste from the 60 to 80% reported in prior systems to near-zero (vllm.ai), materially raises achievable batch size and throughput.

Batching strategy is the other major throughput lever. Continuous (iteration-level) batching, introduced in the Orca paper, reported a 36.9x throughput gain over static batching at matched latency (usenix.org), and the Anyscale/vLLM team measured a 23x gain in its own benchmark (anyscale.com). Newer techniques, chunked prefill and prefill/decode disaggregation, target the resulting throughput-versus-latency tradeoff directly, letting research systems serve substantially more requests under the same latency budget by separating the two phases onto dedicated hardware. As historical examples, the MLPerf Inference v4.1 and v5.0 rounds show NVIDIA's Blackwell-generation B200 GPU delivering up to 4x higher per-GPU tokens per second than H100 on Llama 2 70B (developer.nvidia.com), cloud provider CoreWeave reporting its own H200 instances reaching "33,000 TPS on the Llama 2 70B model, marking a 40% improvement in throughput compared to NVIDIA H100 instances" (coreweave.com), and AMD's MI300X making its MLPerf Inference debut in the v4.1 round; its 192 GB of memory let a full 70-billion-parameter model run on a single GPU (amd.com) (MLCommons).

Software techniques compound these hardware gains: quantization methods report speedups from roughly 1.56x (SmoothQuant, INT8) (arxiv.org) to 2.3x (NVIDIA TensorRT-LLM FP8 quantization on H100) (nvidia.github.io), and speculative decoding reports 2 to 3x acceleration with identical output quality (arxiv.org). No single number from any of these sources is comparable to another without matching model, precision, batch size, and concurrency, a discipline this report applies throughout and one increasingly built directly into benchmarks such as MLPerf Inference's latency-constrained interactive scenarios.

Introduction and Background

Vendor GPU (graphics processing unit) datasheets report peak floating-point throughput and memory bandwidth as if they were interchangeable measures of speed. For large language model (LLM) inference, they are not. A single inference request passes through two phases with opposite performance characteristics: **prefill**, which processes the entire input prompt at once, and **decode**, which generates output tokens one at a time. NVIDIA's own TensorRT-LLM engineering documentation states plainly that prefill "is computationally demanding and can effectively use a GPU's vast parallel compute resources" (developer.nvidia.com), while the decode step that follows "is less computationally intensive" (developer.nvidia.com) because it reuses cached state rather than recomputing it.

This asymmetry is why a benchmark number like "tokens per second" is meaningless without also stating the batch size, context length, numeric precision, and concurrency level under which it was measured. The vLLM serving project's own optimization documentation frames the entire batching problem in these terms, noting that a

scheduling feature exists specifically for "better balancing compute-bound (prefill) and memory-bound (decode) operations" (docs.vllm.ai). Understanding why the two phases behave differently, and how serving systems exploit that difference, is a prerequisite for reading any GPU inference benchmark or specification sheet correctly.

This report is a companion methodology piece to a prior technical reference on data-center GPU hardware, IntuitionLabs' NVIDIA data-center GPU specifications report, which catalogs raw specifications; this report explains how those specifications translate into observed inference performance. As of September 2026, the underlying hardware, software, and benchmark landscape changes on a timescale of months, so every figure below is dated and sourced to a document a reader can independently re-check.

The Two-Phase Structure of LLM Inference: Prefill and Decode

Every request to an autoregressive LLM is served in two distinct stages. **Prefill** (also called the "prompt processing" or "context" phase) runs the model once over every token of the input prompt simultaneously, computing the attention **key-value (KV) cache** entries for each input token and producing the first output token. **Decode** then runs the model repeatedly, once per output token, each pass consuming only the single most recently generated token as new input while reading the entire accumulated KV cache from memory. NVIDIA's documentation describes prefill as processing "all input tokens to compute the KV cache, which is then used to generate the first token of the output" (developer.nvidia.com).

The industry has standardized on two latency metrics that map directly onto these phases. **Time to first token (TTFT)** measures prefill latency: Anyscale's serving-benchmark documentation defines it as "the time elapsed between submitting a prompt and receiving the first token of the model's response" (docs.anyscale.com), and notes that this step "is compute-intensive and directly determines how quickly the model can begin generating the first token" (docs.anyscale.com). **Time per output token (TPOT)** measures decode latency per generated token. The academic DistServe paper formalizes both terms as "time to first token (TTFT) for the prefill phase and time per output token (TPOT) of each request for the decoding phase" (arxiv.org).

Why Prefill Is Compute-Bound

Prefill's defining property is parallelism: every input token's representation can be computed simultaneously because none of them depends on a token the model has not yet generated. This lets prefill saturate a GPU's matrix-multiplication units, making it **compute-bound**, meaning its runtime is limited by how many floating-point operations (FLOPs) the GPU's tensor cores can execute per second, not by how quickly data can be fetched from memory. A widely cited engineering analysis published on arXiv formalizes this with a **roofline model**, an analytical framework borrowed from HPC performance engineering that classifies any computation as compute-bound or memory-bound based on its **arithmetic intensity** (operations performed per byte of data moved from memory). That analysis concludes "during the prefill stage, the majority of computations are compute-bound, leading to high performance" (arxiv.org).

Why Decode Is Memory-Bandwidth-Bound

Decode inverts this picture. Because each decode step generates exactly one new token, the GPU must still read the model's entire weight set and the growing KV cache from memory for every single step, but the amount of new computation per step is tiny. The same arXiv roofline analysis explains the mechanism: "the memory loading is

innately sequential and cannot be easily parallelized" as a model autoregressively decodes (arxiv.org). A Hugging Face engineering blog from consultancy TNG Technology makes the same point in operational terms: for the decode step, "the speed is typically limited by the GPU memory bandwidth" rather than by compute (huggingface.co). Microsoft's Splitwise paper, published on arXiv and later at ISCA 2024, independently reaches the same conclusion at cluster scale, characterizing inference as splitting into "a compute-intensive prompt computation, and a memory-intensive token generation," and reporting that, even with state-of-the-art batching, the token generation phase underutilizes compute resources (arxiv.org). A 2025 hardware-architecture paper on GPU design for inference restates the same premise as an established fact of the field: LLM inference decomposes into "a compute-bound prefill phase followed by a memory-bound decode phase" (arxiv.org).

The practical consequence is a direct tradeoff, which NVIDIA's own chunked-prefill documentation quantifies for its TensorRT-LLM serving framework: increasing the chunk size used to process a prefill request lowers TTFT, but the same change "increases the time taken to complete the decode phase of ongoing requests," worsening TPOT and reducing overall "output tokens per second (TPS)" (developer.nvidia.com). A Hugging Face/TNG measurement makes this concrete with a stated test configuration: "Numbers were measured for vLLM with Llama-3.1-8B on an H100 GPU, at 3000/1500 input tokens and 100 output tokens" (huggingface.co), illustrating why any throughput number is only interpretable alongside its input length, output length, model, GPU, and batch size.

Why Memory Bandwidth Governs the Decode Phase: KV Cache Mechanics

The KV cache is the data structure that makes decode possible without recomputing the whole prompt at every step. The vLLM project's engineering blog explains that after prefill computes each input token's attention key and value tensors, "these tensors are kept in GPU memory to generate next tokens" (vllm.ai). Every decode step must read this entire cache back from GPU memory, and the cache grows by one token's worth of data with every step, which is precisely why decode's bottleneck is the rate at which bytes move off HBM (high-bandwidth memory), not the GPU's arithmetic throughput.

Sizing the KV Cache

For a uniform-length batch, general KV-cache size in bytes is batch_size × sequence_length × 2 × num_layers × num_kv_heads × head_size × bytes_per_element; the factor of 2 stores both keys and values. NVIDIA documents K and V cache inputs with separate `numKvHeads` and `headSize` dimensions, and supports MHA, grouped-query attention (GQA), and multi-query attention (MQA); only for MHA does num_kv_heads × head_size equal hidden_size. (NVIDIA TensorRT documentation) NVIDIA works a concrete example for a 7-billion-parameter Llama 2 model at 16-bit floating point (FP16) precision and batch size 1, computing that "the size of the KV cache will be 1 * 4096 * 2 * 32 * 4096 * 2 bytes, which is ~2 GB" (developer.nvidia.com). Because this size scales linearly with both batch size and sequence length, long-context, high-concurrency serving can consume enormous amounts of GPU memory: NVIDIA cites an example in which a 128,000-token context window for a single user "consumes about 40 GB of memory with Llama 3 70B, and this scales linearly with the number of users" (developer.nvidia.com).

Memory Waste and PagedAttention

Before late 2023, most serving systems allocated KV cache memory in large contiguous blocks sized for a request's maximum possible length, which wasted enormous amounts of memory whenever a request finished early. The vLLM team's own account of this problem states that prior serving systems waste 60 to 80 percent "of memory due to fragmentation and over-reservation" (vllm.ai). Their fix, **PagedAttention**, borrows the paging concept from operating-system virtual memory and "partitions the KV cache of each sequence into blocks, each block containing the keys and values for a fixed number of tokens" (vllm.ai), so blocks need not be contiguous in memory. The peer-reviewed PagedAttention paper reports this achieves "near-zero waste in KV cache memory" (arxiv.org) while enabling flexible memory sharing across requests, which directly raises the effective batch size a GPU can hold and therefore its achievable throughput.

GPU Memory Bandwidth: The Hardware Ceiling

Because decode throughput is bounded by memory bandwidth, the raw HBM bandwidth figure on a GPU's datasheet is one of the single most decision-relevant specifications for inference workloads, more so than peak FLOPs. *Table 1* below summarizes officially published memory bandwidth and capacity figures for the datacenter GPUs most commonly used for LLM inference as of September 2026.

Table 1: Datacenter GPU HBM Bandwidth and Capacity (as reported on official vendor pages)

GPU	Memory Capacity	HBM Bandwidth	Source (accessed 2026-09-05)
NVIDIA A100 (SXM, 80GB)	80 GB HBM2e	2,039 GB/s (nvidia.com), described by NVIDIA as debuting "the world's fastest memory bandwidth at over 2 terabytes per second (TB/s)" at launch (nvidia.com)	nvidia.com A100 datasheet and product page
NVIDIA L40S	48 GB GDDR6	864 GB/s (nvidia.com)	nvidia.com L40S product page
NVIDIA H100 (PCIe, 80GB)	80 GB HBM2e	2,000 GB/s peak (nvidia.com)	nvidia.com H100 PCIe datasheet
NVIDIA H100 (SXM)	80 GB HBM3	3.35 TB/s (NVIDIA HGX reference architecture)	NVIDIA HGX reference architecture
NVIDIA H100 NVL (dual-GPU)	up to 188 GB combined	3,938 GB/s peak (nvidia.com)	nvidia.com H100 NVL datasheet
NVIDIA H200	141 GB HBM3e	4.8 TB/s, which NVIDIA states is "1.4X more memory bandwidth" than H100 (nvidia.com) (nvidia.com)	nvidia.com H200 product page
NVIDIA DGX B200 (8-GPU system)	1,440 GB total	64 TB/s aggregate HBM3e bandwidth across 8 GPUs (nvidia.com), roughly 180 GB and 8 TB/s per GPU	nvidia.com DGX B200 product page
NVIDIA GB200 NVL72 (72-GPU rack)	up to 13.5 TB HBM3e rack-wide	up to 8 TB/s HBM3e per B200 GPU (NVIDIA HGX reference architecture); its separate NVLink communication speed is 1.8 TB/s per GPU (NVIDIA GB200 tuning guide)	NVIDIA documentation

GPU	Memory Capacity	HBM Bandwidth	Source (accessed 2026-09-05)
AMD Instinct MI300X	192 GB HBM3	"total peak memory bandwidth of 5.325 TB/s" (amd.com)	amd.com MI300X product page
AMD Instinct MI325X	256 GB HBM3e	"6 TB/s GPU peak theoretical memory bandwidth performance" (amd.com)	amd.com MI325X product page

Table 1 makes the practical point of this section visible: memory bandwidth has grown roughly threefold from the A100 generation (2,039 GB/s) to the Blackwell generation (roughly 8 TB/s per GPU). AMD's MI300X and MI325X are multi-chiplet CDNA 3 accelerators with larger HBM capacity (192 GB and 256 GB respectively) than NVIDIA's comparable Hopper-generation parts, and their capacity can fit larger models or longer KV caches on a single GPU without cross-GPU communication overhead, a point examined further in the Data Analysis section below.

Batching Strategies: From Static to Continuous Batching

A single request rarely saturates a GPU's memory bandwidth, since the bottleneck is reading model weights rather than compute; batching multiple concurrent requests together amortizes that one weight-read across many requests' outputs. How a serving system forms and manages those batches is one of the largest levers on achieved throughput.

Static Batching's Limitation

Early LLM serving systems used **static (request-level) batching**: a fixed group of requests is processed together, and the entire batch does not return any results until every request in it has finished generating. The Orca paper, presented at OSDI 2022 and one of the first academic treatments of LLM serving efficiency, identifies the core problem with this approach: "requests that have finished earlier than other requests in a batch cannot return to the client, while newly arrived requests have to wait until the current batch completely finishes" (paraphrased from the original), and proposes **iteration-level scheduling**, "a new scheduling mechanism that schedules execution at the granularity of iteration" rather than of whole requests (usenix.org). Orca reports this delivers a "36.9x throughput improvement at the same level of latency" against NVIDIA's FasterTransformer baseline on a GPT-3 175B-parameter model (usenix.org).

Continuous and In-Flight Batching in Production Systems

The technique Orca introduced is now widely known as **continuous batching**. Anyscale's engineering blog, co-authored with the vLLM project, defines it as being "also known as dynamic batching, or batching with iteration-level scheduling" (anyscale.com), and reports a specific measured result: "By leveraging vLLM, users can achieve 23x LLM inference throughput while reducing p50 latency" (anyscale.com) versus naive batching, under a stated test configuration: "we benchmark throughput and latency on a single NVIDIA A100 GPU provided by Anyscale," running Meta's OPT-13B model (anyscale.com).

NVIDIA's TensorRT-LLM implements the same idea under the name **in-flight batching**, described in NVIDIA's developer blog as a scheduling technique in which, "rather than waiting for the whole batch to finish before moving on to the next set of requests, the TensorRT-LLM runtime immediately evicts finished sequences" from the active

batch and admits new ones, an approach built for real-world variable-length request streams (developer.nvidia.com).

Chunked Prefill and Prefill/Decode Disaggregation

Continuous batching alone still allows a large incoming prefill request to stall the in-progress decode steps of other requests sharing the same GPU, since prefill's compute-bound burst competes for the same tensor cores as everyone else's decode. Two related lines of research address this. **Chunked prefill**, implemented in both NVIDIA TensorRT-LLM and the academic Sarathi-Serve system, splits a large prefill computation into smaller pieces interleaved with ongoing decode steps; the Sarathi-Serve paper describes "chunked-prefills which splits a prefill request into near equal sized chunks and creates stall-free schedules," reporting that this lets it reach roughly 2.6 times the serving capacity of vLLM for Mistral-7B on a single A100 GPU under tail-latency service-level constraints (arxiv.org). An earlier academic system, Sarathi, reports that on a LLaMA-13B model running on an A6000 GPU, its scheduling approach "improves decode throughput by up to 10x, and accelerates end-to-end throughput by up to 1.33x" versus a naive baseline (arxiv.org).

A second approach, **prefill/decode disaggregation**, runs prefill and decode on physically separate GPU pools instead of interleaving them on the same hardware. The DistServe paper argues this is necessary because "LLM applications often emphasize individual latency for each phase" and colocating the two phases forces a compromise between TTFT and TPOT that neither phase needs to make on its own; DistServe reports its disaggregated design "can serve 7.4x more requests or 12.6x tighter SLO, compared to state-of-the-art systems, while staying within latency constraints for greater than 90% of requests" (arxiv.org). Microsoft's Splitwise reaches a parallel conclusion using phase-specialized machine pools, reporting "we can achieve 1.4x higher throughput at 20% lower cost than current designs" when prompt computation and token generation run on separate, differently provisioned hardware (arxiv.org).

Together, these results describe a direct **throughput-versus-latency tradeoff**: larger batches and larger prefill chunks raise aggregate tokens-per-second throughput across all concurrent requests, but do so by adding milliseconds to each individual request's TTFT or TPOT. Production serving systems tune batch size, chunk size, and (where supported) disaggregation specifically to hit a target latency service-level objective (SLO) at the highest throughput that objective allows, rather than chasing maximum throughput unconditionally.

Techniques to Raise GPU Utilization for LLM Serving

Because decode is memory-bandwidth-bound, most techniques that improve decode-phase performance work by reducing the number of bytes that must move across the memory bus per useful unit of output, rather than by adding more compute.

Quantization

Quantization reduces the numeric precision used to store model weights, activations, and/or the KV cache, shrinking the volume of data read per step. NVIDIA's TensorRT-LLM documentation reports that "quantization with FP8 and a batch size of 16 achieves a notable 2.3x inference speedup compared to FP16 on a H100" for a LLaMA-2-7B model (nvidia.github.io), and separately that quantizing the KV cache itself to FP8 "enables you to run 2-3x larger batch size on H100 machine for models like GPT-J," which in turn yields a further throughput benefit

(nvidia.github.io). Academic quantization methods report similar or larger gains from more aggressive precision reduction: **GPTQ**, a one-shot weight-quantization method, reports "end-to-end inference speedups over FP16, of around 3.25x when using high-end GPUs (NVIDIA A100) and 4.5x when using more cost-effective ones (NVIDIA A6000)" at 3 to 4-bit weight precision (arxiv.org). **AWQ** (activation-aware weight quantization) reports that its TinyChat inference implementation "offers more than 3x speedup over the Huggingface FP16 implementation" on desktop and mobile GPUs at 4-bit weight precision (arxiv.org). **SmoothQuant**, an 8-bit weight-and-activation (INT8, "W8A8") method, reports "up to 1.56x speedup and 2x memory reduction for LLMs with negligible loss in accuracy" (arxiv.org). Because these are vendor and academic self-reported benchmarks on specific model and hardware combinations, they should be read as evidence that quantization helps, not as a universal multiplier that transfers unchanged to a different model, GPU, or batch size.

Parallelism Across GPUs

When a model is too large to fit, or its KV cache demands exceed, a single GPU's memory, inference workloads are split across multiple GPUs using **tensor parallelism** or **pipeline parallelism**. NVIDIA's TensorRT-LLM architecture documentation summarizes the tradeoff: "Tensor Parallelism usually leads to more balanced executions but requires more memory bandwidth between the GPUs," whereas pipeline parallelism reduces that inter-GPU bandwidth requirement at some risk of load-balancing inefficiency (nvidia.github.io). This is why high-bandwidth GPU-to-GPU interconnects, such as the 1.8 TB/s NVLink links in NVIDIA's GB200 NVL72 rack architecture noted in Table 1, matter specifically for multi-GPU inference of very large models, not only for training.

Speculative Decoding

Speculative decoding attacks the memory-bandwidth bottleneck from a different angle: rather than reducing bytes read per step, it reduces the number of sequential decode steps needed. A small, fast "draft" model proposes several candidate tokens at once, and the large target model verifies them in a single parallel pass, accepting correct predictions and discarding wrong ones. The original speculative decoding paper (Leviathan et al.) reports "a 2X-3X acceleration compared to the standard T5X implementation, with identical outputs" on a T5-XXL model, with no retraining or output distribution change required (arxiv.org).

Measuring Utilization: FLOPs Utilization and Bandwidth Utilization

Two metrics are commonly used to express how close an observed inference run comes to a GPU's theoretical ceiling. **Model FLOPs utilization (MFU)**, a term popularized by Google's PaLM paper, is defined as "observed throughput relative to theoretical max throughput"; PaLM reports achieving "46.2% in model FLOPs utilization" when training a 540-billion-parameter model across 6,144 TPU v4 chips (ar5iv.labs.arxiv.org), a figure widely cited as a reference point for "good" large-scale utilization even though it describes training, not inference. On the inference and profiling side, NVIDIA's Nsight Compute documentation defines the analogous hardware concept for both compute and memory units: "the throughput reports the achieved percentage of utilization with respect to the theoretical maximum" for each hardware unit measured (docs.nvidia.com), which is the metric an engineer would examine to determine whether a specific inference kernel is compute-bound or memory-bandwidth-bound in practice, rather than assuming it from the roofline model alone.

Independently, FlashAttention, an IO-aware exact-attention algorithm, reduces the actual number of memory reads and writes the attention computation performs regardless of quantization or batching choices. The original FlashAttention paper reports "runtime speedup (2-4x)" versus prior optimized attention implementations by avoiding materializing large intermediate matrices in slow memory, though the authors note it still reached only 25 to 40% of theoretical peak FLOPs per second. Its successor, FlashAttention-2, restructures the GPU work partitioning to close that gap, reporting results that "yield around 2x speedup compared to FlashAttention, reaching 50-73% of the theoretical" peak, corresponding to roughly 225 TFLOPs per second per A100 GPU (72% model FLOPs utilization) in the paper's own measurements (arxiv.org).

Data Analysis and Evidence

Benchmark numbers for LLM inference are only comparable when the model, GPU, batch size, context length, precision, and concurrency level are all held constant or explicitly stated, which the following figures attempt to preserve.

Historical MLPerf Inference Examples (v4.1 and v5.0)

MLPerf Inference, run by the MLCommons consortium, is the closest the industry has to an independently audited, cross-vendor LLM inference benchmark, though submissions are still self-run by vendors under published rules rather than run by an independent third party. In the MLPerf Inference v4.1 round, NVIDIA reported that its Blackwell-generation B200 GPU "delivered up to 4x higher tokens per second per GPU compared to the H100 GPU" on the Llama 2 70B benchmark (developer.nvidia.com), with a single submitted B200 GPU reaching 10,756 tokens per second in the Server scenario and 11,264 tokens per second in the Offline scenario (developer.nvidia.com). Separately, NVIDIA ran the Llama 2 70B benchmark from MLPerf Inference v4.1 on GB200 NVL72, achieving an unverified result of 869,203 tokens/second (developer.nvidia.com). In the MLPerf Inference v5.0 round, NVIDIA reported that GB200 NVL72 delivered up to 30x system-level throughput on the Llama 3.1 405B benchmark (developer.nvidia.com). Cloud provider CoreWeave's own MLPerf v5.0 submission reported that its "NVIDIA H200 GPU instances also delivered 33,000 TPS (tokens per second) on the Llama 2 70B model, marking a 40% improvement in throughput compared to NVIDIA H100 instances" (coreweave.com).

AMD entered MLPerf Inference for the first time in the v4.1 round; AMD's own account states "MI300X delivered impressive performance in its inaugural MLPerf submission" using an 8-GPU Supermicro system (amd.com), with one submission specifically demonstrating that MI300X's memory capacity, not just its bandwidth, changes deployment topology: "this entry highlighted the vast 192 GB memory of AMD Instinct MI300X, enabling a single GPU to efficiently run the entire LLaMA2-70B model," avoiding the network overhead that model-splitting across multiple smaller-memory GPUs requires (amd.com). AMD's subsequent MI350-series announcement makes a cost-adjusted claim rather than a raw-throughput one, stating its newer MI355X "delivers up to 40% better tokens-per-dollar than B200" on Llama 3.1 405B inference at FP4 precision, per AMD's own internal testing footnotes dated June 2025 (amd.com); this is a vendor-published, cost-normalized claim rather than an MLPerf-audited figure and should be read with that distinction in mind.

Table 2 below summarizes representative MLPerf-linked and vendor-reported throughput results alongside the exact stated benchmark and configuration, since these numbers are not otherwise comparable to one another.

Table 2: Historical Representative LLM Inference Benchmark Results (as reported by submitters, MLPerf Inference rounds v4.1 and v5.0)

For current coverage, MLCommons reports that five of the eleven datacenter tests in MLPerf Inference v6.0 are new or updated (MLCommons).

System	Benchmark / Model	Reported Result	Source
1x NVIDIA B200 GPU	MLPerf Inference v4.1, Llama 2 70B	10,756 tokens/s (Server), 11,264 tokens/s (Offline); 4x/3.7x increase over H100 per-GPU	NVIDIA, MLPerf entry 4.1-0074 (developer.nvidia.com)
NVIDIA GB200 NVL72	Unverified MLPerf Inference v4.1 Llama 2 70B run	869,203 tokens/s, unverified	NVIDIA (developer.nvidia.com)
CoreWeave 8x NVIDIA H200	MLPerf Inference v5.0, Llama 2 70B	33,000 tokens/s, a stated 40% gain over H100 instances	CoreWeave (coreweave.com)
8x AMD Instinct MI300X	MLPerf Inference v4.1, LLaMA2-70B	AMD's first MLPerf submission; single-GPU entry ran the full 70B model on one 192GB MI300X	AMD (amd.com)
AMD Instinct MI355X (vendor-reported)	Llama 3.1 405B inference, FP4	"Up to 40% better tokens-per-dollar than B200" (cost-normalized, not raw throughput)	AMD internal testing (amd.com)
NVIDIA H100 vs A100 (TensorRT-LLM)	Vendor benchmark, unspecified model	H100 FP8 up to 4.6x max throughput, 4.4x faster first-token latency than A100	NVIDIA (nvidia.github.io)

Table 2 underscores that vendor-reported inference figures are not directly interchangeable: some are audited MLPerf submissions with published rules, some are marked explicitly "unverified" even by the submitter, and some (AMD's tokens-per-dollar claim) normalize for cost rather than reporting raw speed. MLCommons itself notes that the v5.0 round introduced new latency-constrained "interactive" scenarios specifically to reflect production chat-style usage, with a Llama 3.1 405B interactive scenario requiring a "99th percentile TTFT of 6 seconds" (mlcommons.org) alongside a 175-millisecond TPOT limit, formalizing the TTFT/TPOT distinction discussed earlier as an actual benchmark constraint rather than only a conceptual one.

Cost Context

Inference throughput ultimately matters to buyers as part of inference economics. Together AI's public pricing page lists its hosted Llama 3.1 70B model (128K context, FP8 precision) at an "input price $0.88 / 1M tokens" (together.ai), while AWS publishes Capacity Blocks rates for p5.48xlarge by region (aws.amazon.com). These are input-token and infrastructure-hour price examples, respectively, rather than output-token costs; AWS says a Capacity Block is charged at the prevailing rate at the time of purchase.

Case Studies and Real-World Examples

AMD MI300X: Single-GPU Deployment of a 70-Billion-Parameter Model

AMD's MLPerf Inference v4.1 submission is a documented, neutral example of how memory capacity, not only bandwidth, changes system architecture. Rather than splitting a 70-billion-parameter Llama 2 model across two or more GPUs, as its 80GB-class competitors require, AMD ran the full model on a single 192 GB MI300X accelerator, which AMD states avoided the inter-GPU communication overhead that model-parallel splitting introduces (amd.com). This illustrates a genuine architectural tradeoff documented in Table 1: a GPU with larger onboard memory can simplify a deployment's topology even when its raw bandwidth-per-byte figure is comparable to a competing part.

NVIDIA GB200 NVL72: Rack-Scale Disaggregation in Practice

NVIDIA's GB200 NVL72 rack, evaluated in the MLPerf Inference v5.0 round, is a real production example of the interconnect-bandwidth argument made in the parallelism section above: its 1.8 TB/s per-GPU NVLink fabric is what makes a 72-GPU rack behave, for scheduling purposes, closer to one very large accelerator than to 72 independent ones, which NVIDIA credits for the reported 30x throughput increase on the newly added Llama 3.1 405B benchmark relative to the comparison system used in that MLPerf round (developer.nvidia.com).

Implications and Future Directions

The trend across every category examined in this report points in one direction: hardware and software optimization effort for LLM inference is concentrating on memory bandwidth and memory capacity, not raw compute, because decode is the phase most requests spend the most time in in production chat and agentic workloads. The generational trajectory in Table 1, from roughly 2 TB/s (A100, 2020) to roughly 5 to 8 TB/s (MI300X, H200, Blackwell-class parts, 2024 to 2026), is the physical expression of that priority, and vendors' own quantization, KV-cache-compression, and disaggregation techniques described above are software strategies aimed at the same bottleneck.

For organizations evaluating GPU inference infrastructure, the practical implication is that no single specification or benchmark number is sufficient to compare options; a defensible comparison requires model, precision, batch size, context length, and concurrency to be matched across the systems being compared, precisely the parameters MLCommons has progressively added to MLPerf Inference's interactive scenarios.

Looking forward, three trends are likely to continue shaping the prefill/decode distinction discussed throughout this report. First, context windows continue to grow, which by the KV-cache formula in this report's memory bandwidth section increases memory pressure linearly with sequence length, making techniques like PagedAttention and KV-cache quantization more, not less, important over time. Second, prefill/decode disaggregation, demonstrated in research systems like DistServe and Splitwise, is moving from academic prototypes toward production serving stacks, since it lets operators provision GPU pools independently tuned to each phase's actual bottleneck. Third, speculative decoding and related techniques that reduce the number of sequential memory-bound steps, rather than the bytes read per step, represent a structurally different lever than quantization or hardware bandwidth growth, and the two approaches are increasingly combined rather than treated as alternatives.

Frequently Asked Questions (FAQs)

What is the difference between the prefill and decode phases of LLM inference?

Prefill processes an entire input prompt in parallel to produce the first output token and populate the KV cache; decode then generates each subsequent output token one at a time, reading the full KV cache and model weights from memory at every step. vLLM's own scheduling documentation frames this operationally as a technique for "locating compute-bound (prefill) and memory-bound (decode) requests to the same batch" (docs.vllm.ai).

Why is decode memory-bandwidth bound while prefill is compute bound?

Prefill can process every input token in parallel, saturating the GPU's compute units, while decode must sequentially read the growing KV cache and full weight set from memory for each single new token, so its runtime tracks the GPU's memory bandwidth rather than its FLOPs, consistent with engineering measurements showing decode speed is "typically limited by the GPU memory bandwidth" (huggingface.co).

What is continuous batching in GPU inference?

Continuous batching (also called dynamic batching or iteration-level scheduling) lets a serving system admit new requests and return finished ones at every generation step, rather than waiting for an entire fixed batch to complete, an approach the Anyscale/vLLM benchmark measured at up to 23x higher throughput than static batching under its stated test conditions (anyscale.com).

How is KV cache memory bandwidth calculated?

For a uniform-length batch, KV-cache size in bytes is batch_size × sequence_length × 2 × num_layers × num_kv_heads × head_size × bytes_per_element. NVIDIA documents K and V cache inputs with separate `numKvHeads` and `headSize` dimensions, so hidden_size is appropriate only for the MHA special case; the bandwidth required per second also scales with how often that cache is read. (NVIDIA TensorRT documentation)

How can GPU utilization be optimized for LLM serving?

The documented techniques with published, sourced results include quantization (reported speedups of roughly 1.5x to 2.3x depending on method and precision) (nvidia.github.io), continuous or in-flight batching, chunked prefill, tensor or pipeline parallelism across GPUs, speculative decoding, and IO-aware attention kernels such as FlashAttention-2 (arxiv.org).

What is the throughput versus latency tradeoff in LLM inference?

Increasing batch size or prefill chunk size raises total tokens-per-second throughput across all concurrent requests but adds latency (measured as TTFT or TPOT) to individual requests, a tradeoff visible directly in vendor benchmarks, where NVIDIA reports H100 (FP8) achieving "up to 4.6x max throughput and 4.4x faster 1st token latency than A100" only under specific batch-size and precision settings (nvidia.github.io), and which MLPerf Inference now benchmarks explicitly through latency-constrained interactive scenarios (mlcommons.org).

Conclusion

GPU inference performance for large language models cannot be reduced to a single number on a specification sheet. Prefill and decode are governed by different physical bottlenecks, compute throughput for the former and memory bandwidth for the latter, and every serving-system technique surveyed in this report, from continuous batching to chunked prefill to KV-cache quantization to prefill/decode disaggregation, exists specifically to manage that asymmetry. The memory bandwidth figures in Table 1 and the benchmark results in Table 2 are only meaningful when read together with the batch size, context length, precision, and concurrency under which they were measured, a discipline MLCommons has progressively formalized into MLPerf Inference's benchmark rules and one this report has tried to model throughout. Readers evaluating GPU inference options, whether choosing between NVIDIA and AMD accelerators, between cloud and on-premises deployment, or between competing serving frameworks, should treat any single throughput or latency claim as incomplete until its measurement conditions are stated, and should expect the underlying hardware and software landscape summarized here to continue shifting on a timescale of months rather than years.
