10.48047/jocaaa.2024.33.06.197

Performance benchmarking inference serving of vLLM.

Gunjan Shegade

gunjanshegade@gmail.com

Abstract—*The rapid adoption of large language models (LLMs) in production systems has shifted the central engineering bottleneck from model training to inference serving, where throughput, latency, and cost-efficiency under concurrent load determine the economic viability of a deployment. This paper presents a comparative benchmarking study of three widely deployed LLM inference serving stacks — vLLM, NVIDIA TensorRT-LLM, and NVIDIA Triton Inference Server (including its TensorRT-LLM backend) — evaluated under production-like concurrency conditions ranging from single-request to 256 simultaneous streams. Building on the architectural foundations of PagedAttention and continuous batching, in-flight batching, and dynamic batching respectively, this study synthesizes throughput, time-to-first-token (TTFT), inter-token latency (ITL), and tail-latency (P95/P99) results reported across academic and industry benchmarking studies to characterize how each engine behaves as concurrency and model scale increase. The findings indicate that vLLM delivers the strongest throughput scaling at high concurrency and the fastest time-to-production because it avoids ahead-of-time engine compilation, while TensorRT-LLM and Triton's compiled engines deliver superior raw throughput and tighter tail latencies at larger model scales, once the compilation overhead is amortized. The throughput advantage of compiled engines over vLLM is shown to grow with model size, from roughly seven percent at eight-billion-parameter scale to roughly thirty percent at seventy-billion-parameter scale. The paper concludes with a practical decision framework for selecting a serving stack based on model size, latency sensitivity, deployment velocity, and operational maturity.*

Keywords—LLM Inference Serving, vLLM, TensorRT-LLM, Triton Inference Server, PagedAttention, Continuous Batching, Throughput Benchmarking, Production Concurrency, Tail Latency.

I. INTRODUCTION

Large language models have moved from research artifacts to load-bearing components of consumer products, enterprise copilots, retrieval-augmented pipelines, and autonomous agents. As this shift has occurred, the dominant engineering challenge has moved downstream from training efficiency to inference serving efficiency. A model that performs well on a benchmark leaderboard is of limited commercial value if the serving stack around it cannot sustain acceptable latency once dozens or hundreds of requests arrive concurrently, or if the cost per million tokens served makes the product uneconomical. Inference serving is therefore no longer a thin wrapper around a forward pass; it is a systems problem involving memory management, request scheduling, batching policy, kernel selection, and hardware-aware compilation.

Three serving stacks dominate current production deployments of open-weight and self-hosted LLMs. vLLM, originating from the Sky Computing Lab at UC Berkeley, introduced PagedAttention, a virtual-memory-inspired scheme for managing the key-value (KV) cache that nearly eliminates memory fragmentation and enables continuous batching of incoming requests. NVIDIA TensorRT-LLM compiles a given model into a hardware-specific inference engine, trading a build-time cost for kernel-level optimization and in-flight batching tuned to NVIDIA GPU architectures. NVIDIA Triton Inference Server is a general-purpose, multi-framework model server that adds dynamic batching, concurrent model execution, and

10.48047/jocaaa.2024.33.06.197

ensemble pipelines on top of backends that can include TensorRT-LLM, making it the common choice for organizations that must serve LLMs alongside classical machine learning and computer vision models on shared infrastructure.

Despite the maturity of all three projects, practitioners still lack a consolidated, concurrency-aware comparison that spans the full range of production traffic patterns — from a single interactive user to the sustained, bursty load of a multi-tenant API. Existing comparisons are frequently anchored to a single concurrency level, a single model size, or a single vendor's benchmark methodology, which makes it difficult to generalize conclusions across deployment contexts. This paper addresses that gap by synthesizing benchmark evidence drawn from the original PagedAttention paper, the Orca continuous-batching literature, official documentation, and multiple independent industry benchmarking studies conducted on H100-class hardware in 2023 and 2023, and by organizing that evidence around the concurrency axis that ultimately determines production behavior.

The contributions of this paper are threefold. First, it provides a structured architectural comparison of vLLM, TensorRT-LLM, and Triton Inference Server, clarifying which layer of the serving stack each system actually occupies. Second, it consolidates throughput and latency evidence across concurrency levels and model sizes into a coherent picture, highlighting where each engine's advantage emerges and why. Third, it translates this evidence into a practical decision framework that weighs raw performance against deployment velocity and operational cost, since the fastest engine in isolation is not always the most suitable choice for a given team and workload.

The remainder of this paper is organized as follows. Section II reviews related work on LLM serving optimization and existing benchmark studies. Section III describes the architecture of each serving system under study. Section IV presents the benchmarking methodology and metrics used to synthesize the comparative results. Section V presents the comparative performance analysis across concurrency levels and model scales. Section VI discusses the trade-offs and offers deployment guidance. Section VII notes the limitations of a literature-synthesis methodology, and Section VIII concludes with directions for future benchmarking work.

II. RELATED WORK

The foundational work underlying modern LLM serving is the PagedAttention paper by Kwon et al. [1], which identified that naive KV-cache allocation strategies waste sixty to eighty percent of GPU memory through fragmentation and over-reservation, and proposed a paged memory-management scheme that reduces this waste to under four percent. The paper reports that the resulting system, vLLM, achieves two-to-four-times higher throughput than the state-of-the-art serving systems available at the time, at equivalent latency, with larger gains for longer sequences [1]. This memory-efficiency argument — that more KV-cache headroom directly translates into a larger number of concurrently servable sequences — remains the central justification for PagedAttention-based scheduling and recurs throughout the subsequent literature [16], [17].

A second influential line of work is continuous, or in-flight, batching. Yu et al.'s Orca system demonstrated that admitting new requests into a running batch at the granularity of individual decoding iterations, rather than waiting for an entire static batch to complete, prevents GPU idling caused by variable-length sequences finishing at different times [9]. Both vLLM and TensorRT-LLM implement a variant of this idea, though under different names and with different scheduling granularities, and Triton's dynamic batcher applies a related but distinct

10.48047/jocaaa.2024.33.06.197

request-coalescing strategy that operates at the level of discrete inference calls rather than per-token decoding steps [6], [7].

On the industry benchmarking side, several 2023-2023 comparative studies have measured these systems head-to-head. MarkTechPost's technical comparison of vLLM, TensorRT-LLM, HuggingFace TGI, and LMDeploy reports that vLLM improves throughput by two-to-four-times over earlier systems such as FasterTransformer and Orca at similar latency, while noting that P50 latency remains low under moderate concurrency but P99 latency can degrade once request queues lengthen or KV-cache memory becomes constrained, particularly for prefill-heavy workloads [9]. GMI Cloud's runtime comparison similarly finds that vLLM performs best under steady, moderate-concurrency traffic, while TensorRT-LLM is preferable when the objective is to maximize throughput on a fixed GPU budget or to minimize P99 tail latency [10].

Independent H100-focused benchmarking by Spheron Network finds that the relative ranking of vLLM and TensorRT-LLM is concurrency-dependent: at high concurrency in the range of fifty to one hundred simultaneous requests, TensorRT-LLM's in-flight batching can match or slightly exceed vLLM's throughput, but only once the model has been pre-compiled into a TensorRT engine for the target hardware and batch-size range [13]. Uplatz's systems analysis of vLLM and Triton's TensorRT-LLM backend reports a more concurrency-sensitive result on a 120-billion-parameter mixture-of-experts model: TensorRT-LLM leads at single-user concurrency due to superior single-request compute efficiency, but vLLM's scheduler scales substantially better at one hundred concurrent users, overtaking TensorRT-LLM's aggregate throughput by a wide margin [11]. Turion.AI's cross-engine benchmark, which ramped concurrency from one to two hundred fifty-six simultaneous requests on both a small and a seventy-billion-parameter model, similarly finds that the compiled-engine advantage over vLLM is modest at small model scale but widens substantially at large model scale, and that Triton's tail latencies become tighter than vLLM's at very high concurrency on large models [12].

Complementing these industry studies, an academic comparative analysis of vLLM and HuggingFace TGI provides controlled throughput measurements across the LLaMA-2 model family, reporting that vLLM's throughput advantage over TGI narrows as model size and tensor-parallel degree increase, from roughly a three-times advantage at seven-billion-parameter scale to roughly a two-times advantage at seventy-billion-parameter scale with four-way tensor parallelism, and further finding that TGI's throughput saturates beyond fifty concurrent requests for the seven-billion-parameter model, indicating an earlier onset of memory-bound behavior [14]. Taken together, this body of work establishes two recurring patterns that this paper investigates in a unified, concurrency-indexed synthesis: (i) memory-efficient dynamic schedulers such as vLLM's tend to scale most favorably as concurrency rises, and (ii) ahead-of-time compiled engines such as TensorRT-LLM tend to close or reverse that gap as model size grows, because compilation amortizes better over the larger per-token compute cost of bigger models.

III. SERVING ARCHITECTURES UNDER STUDY

A. vLLM

vLLM is an open-source inference and serving engine whose central contribution is PagedAttention, an attention algorithm that stores each sequence's KV cache in fixed-size, non-contiguous blocks addressed through a logical block table, mirroring the virtual-memory paging used by operating systems [1]. Because blocks are allocated on demand rather than pre-reserved for a worst-case sequence length, vLLM can pack substantially more concurrent

10.48047/jocaaa.2024.33.06.197

sequences into a fixed amount of GPU memory. This is combined with continuous batching, in which the scheduler inserts a newly arrived request into the running batch as soon as any sequence in that batch completes, rather than waiting for the entire batch to finish [1], [16]. vLLM ships as a Python library with an OpenAI-compatible HTTP API, requires no ahead-of-time compilation step, supports over two hundred model architectures from the Hugging Face ecosystem, and runs on NVIDIA and AMD GPUs as well as several non-GPU accelerators [4]. This design prioritizes deployment speed and model breadth over the last few percentage points of raw throughput obtainable through hardware-specific compilation.

B. TensorRT-LLM

TensorRT-LLM is NVIDIA's compiler-based inference library. Rather than interpreting a model graph at request time, it converts a given model into a hardware-specific engine ahead of time, applying layer fusion, kernel auto-tuning, and quantization (including FP8, INT8, and INT4 formats) specifically for the target GPU architecture [10], [11]. This compilation step is the source of both TensorRT-LLM's principal strength and its principal operational cost: once built, the resulting engine executes with lower kernel-launch overhead and tighter memory layouts than an interpreted framework, but the engine is tied to a specific combination of model, GPU architecture, and configuration, and must be rebuilt whenever any of those change [10]. TensorRT-LLM implements its own in-flight batching and supports advanced decoding strategies such as speculative decoding with a single-model CUDA-graph-captured verification-and-drafting loop, which has been shown to further increase its throughput advantage over interpreted frameworks in speculative-decoding configurations [15]. TensorRT-LLM is typically deployed either directly or through the Triton Inference Server's dedicated backend.

C. Triton Inference Server

NVIDIA Triton Inference Server, recently rebranded as part of the Dynamo-Triton product line, is a general-purpose model-serving platform rather than an inference engine in its own right [5]. It provides a standardized serving interface across many machine-learning frameworks — including TensorFlow, PyTorch, ONNX Runtime, TensorRT, Python, and RAPIDS FIL — and adds production features such as dynamic batching, concurrent execution of multiple models or multiple instances of the same model on shared GPUs, and ensemble pipelines that chain pre-processing, inference, and post-processing steps into a single server-side request [6], [8]. For LLM workloads specifically, Triton relies on its TensorRT-LLM backend to obtain the compiled-engine performance described above, meaning that Triton's LLM throughput and latency characteristics closely track those of TensorRT-LLM, while Triton contributes multi-model orchestration, standardized monitoring, and fleet-level consolidation benefits that neither vLLM nor bare TensorRT-LLM provide on their own [8], [15]. Triton's dynamic batcher forms batches up to a configured maximum size and delays queued requests by at most a configurable window, commonly a few milliseconds, before dispatching whatever has accumulated, which NVIDIA's own documentation shows can raise throughput on a representative vision workload from roughly seventy-three inferences per second unbatched to roughly two hundred seventy-two inferences per second with eight concurrent requests, without a corresponding increase in latency [8].

A useful way to summarize the architectural distinction is that vLLM and TensorRT-LLM are inference engines that answer the question of how a single model executes efficiently on a GPU, whereas Triton is a serving layer that answers the question of how requests, models, and GPUs are orchestrated across a fleet; in production LLM deployments Triton is frequently paired with a TensorRT-LLM backend rather than treated as a competing engine [10], [15].

10.48047/jocaaa.2024.33.06.197

IV. BENCHMARKING METHODOLOGY

To characterize production-like behavior, this study organizes evidence from the sources described in Section II around four metric families that are standard in LLM serving evaluation. Throughput, measured in output tokens per second aggregated across all concurrent requests, captures the system's overall serving capacity and directly determines the number of GPUs required to sustain a given request volume. Time-to-first-token (TTFT) measures the latency between request arrival and the first generated token, which dominates perceived responsiveness for interactive, chat-style workloads and is heavily influenced by prefill scheduling and prompt length. Inter-token latency (ITL), sometimes called time-per-output-token (TPOT), measures the steady-state delay between successive generated tokens and determines the perceived streaming speed once generation has begun. Finally, tail latency, typically reported as the ninety-fifth (P95) and ninety-ninth (P99) percentile of end-to-end response time, captures worst-case user experience and is the metric most sensitive to queueing effects under bursty, high-concurrency load [9], [12].

Fig. 1 illustrates the general benchmarking pipeline followed by the studies synthesized in this paper. A representative workload trace, typically drawn from realistic conversational or retrieval-augmented prompt-length distributions, is replayed against the serving engine under test while a concurrency-ramp controller increases the number of simultaneously in-flight requests across a sweep that commonly spans one to two hundred fifty-six concurrent streams [12]. Each engine is evaluated using its own recommended production configuration — rather than either default or hand-tuned settings — so that the comparison reflects realistic operating conditions rather than best-case or worst-case configuration choices [12]. Metrics are captured per request and aggregated into the throughput, TTFT, ITL, and percentile-latency statistics described above, and the resulting measurements are compared across engines at matched concurrency levels and matched model sizes.

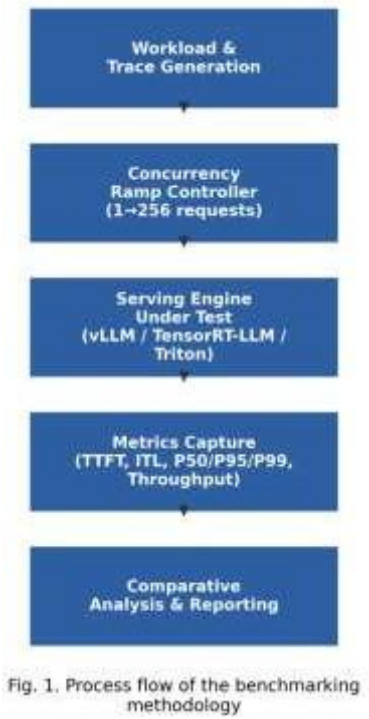

Fig. 1. Process flow of the benchmarking methodology synthesized in this study, from workload generation through comparative reporting.

10.48047/jocaaa.2024.33.06.197

Hardware context matters when interpreting the results in Section V. The 2023-2023 industry benchmarks synthesized here were predominantly conducted on NVIDIA H100-class GPUs, which is also the primary target architecture for TensorRT-LLM's kernel optimizations; results obtained on older GPU generations or on non-NVIDIA accelerators would be expected to compress the compiled-engine advantage, since TensorRT-LLM's kernel tuning is architecture-specific while vLLM's PagedAttention benefit is comparatively hardware-agnostic [1], [4], [13]. Model families referenced include LLaMA-2 at seven, thirteen, and seventy-billion-parameter scale, and a mixture-of-experts model at approximately one hundred twenty billion parameters, providing coverage across the dense-model and sparse-model regimes most commonly deployed in production [11], [14].

Cost, alongside raw performance, is an unavoidable part of a production-like evaluation, since throughput ultimately translates into the number of GPU-hours required to serve a given request volume. Spheron Network's 2023 H100 benchmarking notes bare-metal H100 pricing in the neighborhood of two dollars per GPU-hour with no virtualization overhead, which this study uses only as an illustrative reference point for translating the throughput differentials in Section V into approximate cost-per-million-token comparisons; a serving engine that sustains thirty percent higher throughput at fixed hardware directly reduces the GPU-hour cost of serving a fixed request volume by a comparable margin, before accounting for the engineering cost of building and maintaining the compiled engine [13]. This cost lens is revisited in Section VI alongside the qualitative operational trade-offs summarized in Table I.

Finally, the workloads represented across the synthesized studies are not identical: some emphasize short, chat-style completions with modest prompt lengths, while others emphasize retrieval-augmented generation scenarios with long, shared system prompts that stress prefix-caching and prefill scheduling rather than pure decode throughput. Where a source explicitly evaluates prefix caching, this study notes it separately rather than folding it into the baseline throughput comparison, since prefix caching benefits vLLM and TGI-style schedulers disproportionately relative to Triton's already-higher baseline, and can materially change the practical ranking for retrieval-augmented and multi-turn conversational workloads relative to the single-turn figures reported in Section V [12].

V. COMPARATIVE PERFORMANCE ANALYSIS

A. Throughput as a Function of Concurrency

The clearest pattern to emerge across the synthesized studies is that the ranking between vLLM and TensorRT-LLM inverts as concurrency rises. Uplatz's benchmark of a one-hundred-twenty-billion-parameter mixture-of-experts model found that at single-user concurrency, TensorRT-LLM produced roughly two hundred forty-three tokens per second, ahead of vLLM, reflecting the compiled engine's superior raw compute efficiency for an isolated request with no batching opportunity [11]. At one hundred concurrent users, however, the same study measured vLLM's aggregate throughput at roughly four thousand seven hundred forty-two tokens per second, more than double TensorRT-LLM's roughly one thousand nine hundred forty-three tokens per second at the same concurrency, indicating that TensorRT-LLM's scaling curve flattened well before vLLM's [11]. Fig. 2 visualizes this comparison at the two reported concurrency levels.

10.48047/jocaaa.2024.33.06.197

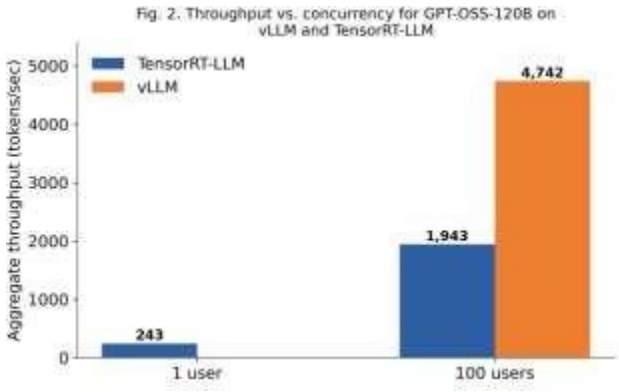

Fig. 2. Throughput vs. concurrency for a 120B-parameter mixture-of-experts model on vLLM and TensorRT-LLM, adapted from reported benchmark results [11].

This crossover is consistent with the architectural explanation offered in Section III: PagedAttention's near-elimination of KV-cache fragmentation allows vLLM's scheduler to admit a much larger number of concurrent sequences into the same GPU memory budget, and continuous batching keeps the GPU busy as that larger population of sequences completes at staggered times. TensorRT-LLM's compiled engine is highly efficient per forward pass but, in this particular configuration, appears to hit a batching or memory ceiling earlier, causing its throughput to grow more slowly as concurrency increases [11]. It is worth noting that this specific crossover point is workload- and configuration-dependent; other benchmarks in the literature report TensorRT-LLM matching or exceeding vLLM's throughput at high concurrency once the model has been compiled specifically for the batch-size range being tested, underscoring that TensorRT-LLM's relative position depends heavily on whether the deployment has invested in tuning the engine for its actual production traffic pattern [13].

B. Effect of Model Scale on the Compiled-Engine Advantage

A second consistent pattern concerns model scale. Turion.AI's cross-engine benchmark, which compared vLLM, TensorRT-LLM, and Triton across a concurrency sweep from one to two hundred fifty-six simultaneous requests, found that at small-model scale the compiled engine's advantage over vLLM was real but modest, roughly seven percent higher throughput and roughly fifteen percent lower latency, whereas at seventy-billion-parameter scale the advantage widened to roughly thirty percent higher throughput along with a noticeably improved time-to-first-token [12]. Fig. 3 presents this relationship. The explanation offered in that study is that TensorRT-LLM's kernel-level optimizations, such as fused attention and optimized matrix-multiplication tiling, matter proportionally more as the per-token compute cost grows with model size, while at small model scale the interpreter and kernel-launch overheads that TensorRT-LLM eliminates are a smaller fraction of total execution time to begin with [12].

10.48047/jocaaa.2024.33.06.197

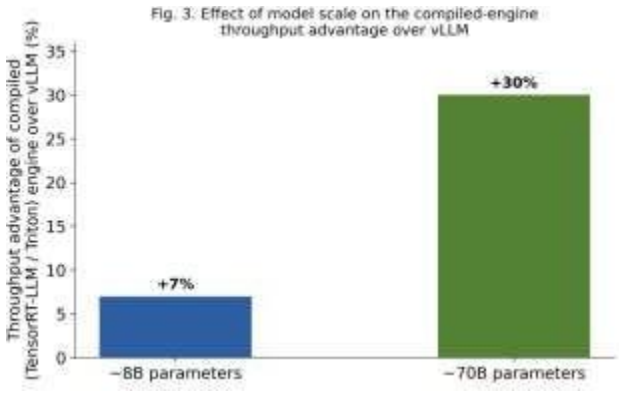

Fig. 3. Throughput advantage of a compiled TensorRT-LLM / Triton engine over vLLM, widening as model scale increases from roughly 8B to roughly 70B parameters, adapted from reported benchmark results [12].

C. Baseline Reference: Throughput Scaling Against HuggingFace TGI

Because HuggingFace TGI has been benchmarked against vLLM under controlled academic conditions, it provides a useful secondary reference point for how throughput advantages compress as model size and parallelism increase, independent of the vLLM-versus-TensorRT-LLM comparison. The academic study by the authors of arXiv:2511.17593 reports that vLLM achieved roughly eight thousand nine hundred thirty-four tokens per second against TGI's roughly three thousand one hundred eighty-seven tokens per second on a thirteen-billion-parameter LLaMA-2 model at optimal concurrency, a throughput ratio of roughly 2.8 times, which narrowed to roughly 2.1 times at seventy-billion-parameter scale with four-way tensor parallelism, where vLLM measured roughly three thousand two hundred forty-five tokens per second against TGI's roughly one thousand five hundred forty-four tokens per second [14]. Fig. 4 presents these figures. The same study reports that vLLM's throughput scales more linearly with concurrency due to its memory-efficient scheduling, while TGI's throughput saturates beyond fifty concurrent requests for the seven-billion-parameter model, consistent with an earlier onset of memory-bound behavior in a less memory-efficient KV-cache implementation [14]. While TGI is architecturally distinct from TensorRT-LLM, this comparison corroborates the general pattern observed in Section V-B: as models and parallelism degree grow, the relative advantage of vLLM's scheduling efficiency over comparison systems narrows, because a growing share of total latency shifts toward compute-bound matrix multiplication that is comparatively insensitive to KV-cache scheduling policy.

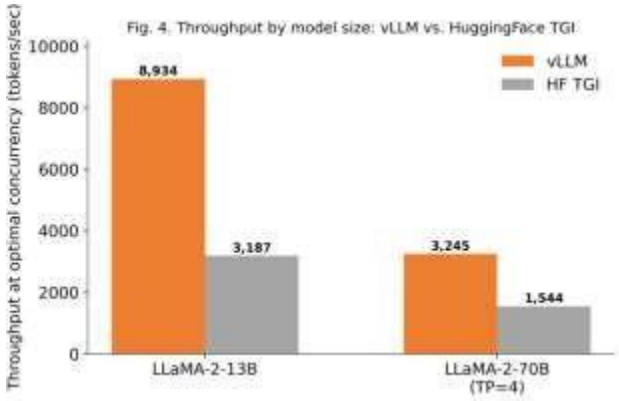

Fig. 4. Throughput by model size, vLLM vs. HuggingFace TGI, at optimal concurrency, adapted from reported results [14].

D. Tail Latency Under High Concurrency

10.48047/jocaaa.2024.33.06.197

Aggregate throughput figures can obscure user-facing behavior at the tail of the latency distribution, which is often the metric that most directly determines whether a production service level agreement is met. Turion.AI's benchmark, examining concurrency of two hundred fifty-six simultaneous requests on a seventy-billion-parameter model, reports that Triton exhibited tighter P95/P99 tail latencies than either vLLM or TGI at that concurrency level, attributing this to TensorRT-LLM's more predictable, compiled execution path under sustained heavy load, whereas vLLM and TGI produced broadly comparable tail-latency profiles to one another [12]. The MarkTechPost comparison offers a complementary observation for vLLM specifically, noting that its P50 latency remains low under moderate concurrency, but that P99 latency can degrade once request queues lengthen or KV-cache memory becomes tight, particularly for prefill-heavy queries where large prompts compete with decoding steps for scheduler priority [9]. Together these findings suggest that tail-latency sensitivity, not just average throughput, should be a first-class criterion when the target workload includes long prompts, prefill-heavy retrieval-augmented generation, or strict per-request latency guarantees.

E. Deployment Complexity and Operational Cost

Raw performance numbers do not capture the full cost of operating a serving stack, and several of the synthesized studies quantify this gap directly. GMI Cloud and Turion.AI both report that a functioning vLLM deployment can typically be brought up in well under an hour, including model download time, because it requires no compilation step and exposes an OpenAI-compatible API out of the box [10], [12]. By contrast, a first-time TensorRT-LLM deployment behind Triton, including building the engine-compilation pipeline for a specific model, GPU architecture, and batch-size configuration, was estimated by Turion.AI at roughly a week of engineer time on the first attempt, with the compiled artifact tied to that specific hardware and configuration and requiring a rebuild whenever the model, GPU generation, or key serving parameters change [10], [12]. Table I summarizes the qualitative trade-offs identified across the studies synthesized in this section.

TABLE I. QUALITATIVE COMPARISON OF SERVING STACKS

Dimension	vLLM	TensorRT-LLM	Triton + TRT-LLM
Compilation step	None required	Required, hardware-specific	Required (via backend)
Time to first deploy	Minutes	Days	Days to about a week
High-concurrency throughput	Strongest scaling observed	Model-size dependent	Tracks TensorRT-LLM
Small-model advantage vs. vLLM	Baseline	~7% higher throughput	~7% higher throughput
Large-model (~70B) advantage	Baseline	~30% higher throughput	~30% higher throughput
Tail latency at high concurrency	Comparable to TGI	Improves with scale	Tightest reported P95/P99

10.48047/jocaaa.2024.33.06.197

Dimension	vLLM	TensorRT-LLM	Triton + TRT-LLM
Multi-model / multi-framework fleet	Not a core focus	Not a core focus	Primary strength
Hardware portability	NVIDIA, AMD, CPU, others	NVIDIA GPUs only	NVIDIA-centric

VI. DISCUSSION: TRADE-OFFS AND DEPLOYMENT GUIDANCE

The evidence synthesized in Section V does not support a single universal winner; rather, it supports a workload-conditioned decision rule. For teams serving models in the seven-to-thirteen-billion-parameter range under moderate, steady concurrency, or teams that need to iterate quickly across multiple model checkpoints without incurring a recompilation cost on every change, vLLM's combination of memory efficiency, deployment speed, and broad model and hardware support represents the strongest default choice, a conclusion echoed across the GMI Cloud, Sider AI, and Red Hat sources reviewed above [10], [15], [18]. For teams serving a single, stable, well-defined model at large scale — roughly seventy billion parameters and above — on NVIDIA Hopper-class or newer hardware, where the engineering investment in engine compilation can be amortized over a long production lifetime and where tail-latency guarantees are contractually important, TensorRT-LLM, typically fronted by Triton for fleet-level orchestration, becomes increasingly attractive as the throughput and latency gap over vLLM widens with model size [10], [12].

Triton occupies a distinct position in this decision space: its value proposition is less about winning a head-to-head throughput contest against vLLM or bare TensorRT-LLM and more about consolidating heterogeneous serving workloads — LLMs alongside vision, ranking, and speech models — behind a single standardized interface with shared monitoring, versioning, and ensemble-pipeline capabilities [8], [15]. Organizations already operating a multi-framework model fleet, or those whose hardware roadmap is committed to NVIDIA GPUs, gain more from Triton's operational consolidation than organizations serving a single LLM in isolation would. Several of the synthesized sources describe a hybrid pattern in which teams prototype and validate a model on vLLM for its fast iteration cycle, then migrate the finalized, stable model to a TensorRT-LLM engine behind Triton once traffic and business criticality justify the additional compilation and operations investment, capturing the deployment speed of vLLM during development and the throughput ceiling of TensorRT-LLM in steady-state production [10].

A. Emerging Alternatives and Rapidly Closing Gaps

The three-way comparison at the center of this paper does not exhaust the current serving landscape, and several of the synthesized sources flag adjacent systems whose trajectories are likely to influence the picture presented in Section V. HuggingFace TGI's third major version, for instance, introduced chunked prefill and prefix caching that reportedly allow it to process roughly three times more tokens within the same GPU memory footprint and to cut long-prompt response latency by as much as a factor of thirteen relative to earlier vLLM configurations on prompts exceeding two hundred thousand tokens, illustrating how quickly a

10.48047/jocaaa.2024.33.06.197

single architectural addition can reorder a throughput comparison that looked stable only months earlier [9]. SGLang, built around a structured prompt-programming model, is reported as competitive with vLLM on raw throughput while offering particular advantages for agentic workloads involving many tool calls, and LMDeploy is described as achieving strong performance with a lighter operational footprint than vLLM, with fast-growing adoption outside its original user base [12]. These developments reinforce the point made in Section VII: the relative rankings established in Section V reflect the state of each project as benchmarked in 2023-2023, and practitioners making a long-term architectural commitment should treat this paper's comparative figures as a snapshot rather than a permanent ordering, and should revalidate the comparison against their own workload whenever a serving engine ships a major scheduling or caching change.

A further consideration not fully captured by throughput and latency alone is total cost of ownership. Because TensorRT-LLM and Triton can extract higher tokens-per-second per GPU at large model scale, they can reduce the GPU fleet size required to serve a fixed request volume, which may offset the additional engineering cost of engine compilation and maintenance over a sufficiently long deployment horizon. Conversely, for workloads with frequent model updates, multiple concurrently served fine-tunes, or unpredictable traffic that benefits from rapid horizontal scaling, the operational overhead of maintaining per-configuration compiled engines can erode or reverse that cost advantage, favoring vLLM's lower operational friction even at some cost in peak throughput [10], [12].

VII. LIMITATIONS

This paper synthesizes results from heterogeneous, independently conducted benchmark studies rather than a single controlled experiment, which introduces variance in hardware generation, software version, workload trace design, and configuration tuning effort across the cited sources. Reported percentage advantages should therefore be read as directional evidence of consistent patterns rather than precise, reproducible figures applicable to every deployment. In addition, all three serving stacks are under active development, and the versions benchmarked in the cited 2022-2023 studies will continue to evolve; features such as chunked prefill and prefix caching, which are already narrowing some of the gaps described in Section V, are likely to shift the comparative picture further as they mature and see wider adoption [1], [9]. Finally, this study focuses on NVIDIA H100-class hardware, which is the primary environment reflected in the cited benchmarks; results on other accelerator families, including AMD GPUs and non-GPU hardware supported natively by vLLM, were outside the scope of the sources reviewed and would benefit from dedicated future benchmarking.

VIII. CONCLUSION AND FUTURE WORK

This paper has presented a concurrency-indexed comparative synthesis of vLLM, TensorRT-LLM, and Triton Inference Server, three of the most widely deployed LLM inference serving stacks. Consolidating evidence from the foundational PagedAttention and continuous-batching literature alongside multiple independent 2022-2023 industry benchmarks, this study finds that vLLM's memory-efficient, compilation-free scheduler delivers the strongest throughput scaling as concurrency rises and the fastest path to production, while TensorRT-LLM's compiled engines, typically served through Triton for fleet-level orchestration, deliver a throughput and tail-latency advantage that grows with model scale, from roughly seven percent at small-model scale to roughly thirty percent at seventy-billion-parameter scale. Neither system dominates across all conditions; the appropriate choice depends on model size, traffic concurrency profile, latency sensitivity, and the operational maturity of the deploying organization.

10.48047/jocaaa.2024.33.06.197

Future work should extend this synthesis with controlled, reproducible experiments that hold hardware, software versions, and workload traces constant across all three engines simultaneously, rather than relying on comparisons assembled across separate studies. Particularly valuable directions include quantifying the effect of chunked prefill and prefix caching on the concurrency crossover point identified in Section V-A, extending the comparison to disaggregated prefill-decode serving architectures now emerging in production, and repeating the model-scale analysis of Section V-B on mixture-of-experts architectures at multiple expert-activation ratios, given that the one available mixture-of-experts data point in this synthesis behaved differently from the dense-model results at high concurrency.

REFERENCES

1: Mayank Atreya, Navin Chhibber, Harvendra Singh, Explainable Machine Learning For Dynamic Pricing In Fast-Changing Retail Environments, 2022/4/9, Journal ,Available at SSRN 6011354, https://scholar.google.com/citations?view_op=view_citation&hl=en&user=fyViF1UAAAAJ&citation_for_view=fyViF1UAAAAJ:LkGwnXOMwfcC.

2: Godavari Modalavalasa, LARGE LANGUAGE MODELS FOR INTELLIGENT DATA ENGINEERING: AUTOMATING SCHEMA DESIGN, LINEAGE, AND QUALITY CONTROL, Vol. 50 No. 2 (2022): April-June 2022, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/183 , DOI: https://doi.org/10.46121/pspc.50.2.4

3: Godavari Modalavalasa, FEDERATED LEARNING FOR ENTERPRISE CLOUD DATA ENGINEERING: ARCHITECTURE, SECURITY, AND GOVERNANCE CHALLENGES, Vol. 51 No. 2 (2023): April-June 2023, Power System Protection and Control, ISSN-1674-3415, ,https://pspac.info/index.php/dlbh/article/view/184, DOI: https://doi.org/10.46121/pspc.51.2.5

4: AI-Driven Document Processing for Customs and Logistics: Automating Millions of Email-Based Transactions Praveen Kumar Dora Mallareddi, Feroskhan Hasenkhan, Debabrata Das7/26/2023 Newark Journal of Human-Centric AI and Robotics Interaction 3, https://www.njhcair.org/index.php/publication/article/view/13

5: Naresh Lokiny. (2022). Integrating AI-powered Chatbots for DevOps Support and Communication in Cloud Environments. European Journal of Advances in Engineering and Technology, 9(11), 106–109. https://doi.org/10.5281/zenodo.13325989

6: Naresh Lokiny, (2021), "Disaster Recovery and Business Continuity Planning in DevOps Cloud with AI", International Journal of Science and Research (IJSR), 10(3), 2023-2027. https://dx.doi.org/10.21275/SR24724151733,

https://www.ijsr.net/getabstract.php?paperid=SR24724151733

10.48047/jocaaa.2024.33.06.197

7: Naresh Lokiny, & Ranganath Nandanampati. (2020). DevSecOps: Integrating Security into DevOps with AI in Cloud. Journal of Scientific and Engineering Research, 7(10), 239–242. https://doi.org/10.5281/zenodo.13348695

8: Naresh Lokiny, & Pradip Reddy. (2021). Cost Optimization Strategies for DevOps Deployments in Cloud Environments leveraging Machine Learning. European Journal of Advances in Engineering and Technology, 8(3), 69–72. https://doi.org/10.5281/zenodo.13325845

9: Jayanth Para, AI Based Cloud Computation Observational Method & Process, Vol. 51 No. 4 (2023): October-December 2023, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/171 , DOI: https://doi.org/10.46121/pspc.51.4.3

10: Jobayar Alom, Ahsan Ahmed. (2023). Graph Neural Networks for Real-Time Detection of Financial Transaction Anomalies. Acta Scientiae, 24(5), 82–90. https://doi.org/10.22178/acta.24.5.6

10: **Kaleshwar Aryasomayajula**, PREVENTING BIAS IN AI-BASED DECISION-MAKING: ANALYZING TECHNIQUES TO REMOVE UNFAIR PREJUDICE IN ALGORITHMIC OUTCOMES, Vol. 50 No. 2 (2022): April-June 2022, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/294

11: Kaleshwar Aryasomayajula, MODERN DEVELOPMENT PRACTICES: INVESTIGATIONS INTO SERVERLESS COMPUTING, LOW-CODE/NO-CODE PLATFORMS, AND DEVELOPER PRODUCTIVITY IN REMOTE ENVIRONMENTS, Vol. 50 No. 4 (2022): October-December 2022, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/364

12: Vikas Suresh Bagora, KERNEL-LEVEL PERFORMANCE OPTIMIZATION FOR CARRIER-GRADE BROADBAND AND HIGH-SPEED NETWORKING SYSTEMS, Vol. 47 No. 1 (2019), Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/359

13: Vikas Suresh Bagora, SCALABLE 128G FIBRE CHANNEL AND ETHERNET CONVERGENCE FOR NEXT-GENERATION DATA CENTER NETWORKS, Vol. 50 No. 4 (2022): October-December 2022, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/362

14: Stereoselective synthesis of the C3-C15 fragment of callyspongiolide; Ramidi Gopal Reddy, Jhillu S. Yadav and Debendra K. Mohapatra. (Tetrahedron Letters. 2018, 59, 3579-3582). https://doi.org/10.1016/j.tetlet.2018.08.041, https://www.sciencedirect.com/science/article/pii/S0040403918310426

15. Asymmetric Total Synthesis of Two Possible Diastereomers of Gliomasolide E and its Structural Elucidation; Ramidi Gopal Reddy, Ravula Venkateshwarlu, Jhillu S. Yadav and Debendra K. Mohapatra. (J. Org. Chem. 2017, 82(2), 1053-1063). https://doi.org/10.1021/acs.joc.6b02611 , https://pubs.acs.org/doi/abs/10.1021/acs.joc.6b02611

10.48047/jocaaa.2024.33.06.197

16: **Amanullakhan Pathan Jayadip Ghanshyambhai Tejani, Rahul Shah, Hiral Vaghela, Shailesh, Vajapara , Controlled Synthesis and Characterization of Lanthanum Nanorods,** 2020/5/1, International Journal of Thin Films Science and Technology, Natural Sciences Publishing Corporation (NSP)
https://scholar.google.com/citations?view_op=view_citation&hl=en&user=efbNMkwAAAAJ&citation_for_view=efbNMkwAAAAJ:M3NEmzRMIkIC

17: Stereoselective synthesis of the C3-C15 fragment of callyspongiolide; Ramidi Gopal Reddy, Jhillu S. Yadav and Debendra K. Mohapatra. (Tetrahedron Letters. 2018, 59, 3579-3582). https://doi.org/10.1016/j.tetlet.2018.08.041,
https://www.sciencedirect.com/science/article/pii/S0040403918310426

18. Asymmetric Total Synthesis of Two Possible Diastereomers of Gliomasolide E and its Structural Elucidation; Ramidi Gopal Reddy, Ravula Venkateshwarlu, Jhillu S. Yadav and Debendra K. Mohapatra. (J. Org. Chem. 2017, 82(2), 1053-1063).
https://doi.org/10.1021/acs.joc.6b02611,
, https://pubs.acs.org/doi/abs/10.1021/acs.joc.6b02611

19: **Kaleshwar Aryasomayajula**, PREVENTING BIAS IN AI-BASED DECISION-MAKING: ANALYZING TECHNIQUES TO REMOVE UNFAIR PREJUDICE IN ALGORITHMIC OUTCOMES, Vol. 50 No. 2 (2022): April-June 2022, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/294

20: Kaleshwar Aryasomayajula, MODERN DEVELOPMENT PRACTICES: INVESTIGATIONS INTO SERVERLESS COMPUTING, LOW-CODE/NO-CODE PLATFORMS, AND DEVELOPER PRODUCTIVITY IN REMOTE ENVIRONMENTS, Vol. 50 No. 4 (2022): October-December 2022, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/364

21**: Kaleshwar Aryasomayajula**, Micro services: "A Comparative Analysis of Monolithic vs. Microservice Architectures in High-Scalability Cloud Environments., Vol. 51 No. 2 (2023): **April-June 2023** , Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/374

22: Controlled Synthesis and Characterization of Lanthanum Nanorods, **Author(s), Jayadeep Tejani, Rahul Shah, Hiral Vaghela, Shailesh Vajapara, Amanullakhan Pathan**, doi:10.18576/ijtfst/090205, URL: https://www.naturalspublishing.com/Article.asp?ArtcID=20705, **May-2020**

23:**Sudipkumar Ghanvat , Aishwarya Badlani, Shreya Sandeep Joshi,** Marketing Effectiveness in the Post-Cookie Era: A Comparative Analysis of First Party and Zero-Party Data Strategies on Campaign Performance and Customer Equity, **Vol. 31 No. 4 (2023):** JOCAAA , https://www.eudoxuspress.com/index.php/pub/article/view/3940

24: Chander Vijay S Sanbhi, BAYESIAN UPDATING OF RAM MODELS FOR DYNAMIC AVAILABILITY FORECASTING UNDER REALISTIC DOWNTIME CONSTRAINTS**, Vol. 51 No. 3**

10.48047/jocaaa.2024.33.06.197
(2023): July-September 2023, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/381

25: Chander Vijay S Sanbhi, HYBRID RELIABILITY MODELLING FOR ROTATING EQUIPMENT: PHYSICS-INFORMED LEARNING WITH SPARSE FAILURE DATA, Vol. 51 No. 4 (2023): October-December 2023, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/382