Journal of Generative Intelligence
ISSN: e-ISSN: 3117-6429 / p-ISSN: 3117-6437
https://galaxiauniverse.com/index.php/JGI/issue/archive

Inference Optimization Techniques for Real-Time Generative AI Applications

Elena Hansen[1], Isabella Hansen[2], Nina Schmidt[3]

[1] Associate Professor, Department of Computer Science, Mediterranean Institute of Technology, Rome, Italy. Email: elena.hansen54@gmail.com | ORCID: 3507-5644-6684-9508
[2] Assistant Professor, Department of Artificial Intelligence, Central European Tech University, Vienna, Austria. Email: isabella.hansen229@gmail.com | ORCID: 3336-2035-2714-6624
[3] Assistant Professor, Department of Computer Science, Central European Tech University, Vienna, Austria. Email: nina.schmidt335@gmail.com | ORCID: 0203-7982-9621-0675

ABSTRACT

Real-time generative AI applications -- interactive chatbots, voice assistants, live code completion, real-time translation, and interactive content generation -- impose strict latency requirements (typically < 500ms first-token latency, < 50ms inter-token latency) that current large language models cannot satisfy at full model quality without optimisation. The inference efficiency of generative models is fundamentally constrained by sequential autoregressive decoding -- each token must be generated before the next can begin -- and by the key-value (KV) cache memory requirements that grow linearly with sequence length. This paper proposes the Real-Time Generative Inference (RTGI) optimisation framework, an integrated suite of six inference acceleration techniques designed for sub-500ms end-to-end latency at production quality: continuous batching with dynamic request scheduling (CB-DRS) that maximises GPU utilisation across concurrent requests; speculative decoding with adaptive draft selection (SD-ADS) that uses context-appropriate draft models for maximum acceptance rate; flash attention with sliding window attention (FA-SWA) for memory-efficient long-context inference; quantised model serving with per-layer sensitivity adaptation (QMS-PSA) that applies optimal quantisation per layer; KV cache compression with semantic eviction (KC-SE) that retains semantically important context while compressing the KV cache; and token budget allocation with dynamic stopping (TBA-DS) that predicts and enforces response length to bound latency. RTGI is evaluated on four real-time application benchmarks -- interactive dialogue, live code completion, real-time translation, and voice response -- measuring time-to-first-token (TTFT), throughput (tokens/second), and quality retention. RTGI achieves mean TTFT = 284ms (SD = 42ms) -- meeting the 500ms threshold -- at 96.2% quality retention versus full-model inference, and 4.8x throughput improvement enabling substantially higher concurrent user capacity. The study contributes the RTGI specification, a Real-Time Inference Quality Index (RTIQ), and empirical guidance for deployment architects balancing latency, throughput, and quality for generative AI serving.

Keywords: inference optimization; real-time AI; speculative decoding; continuous batching; KV cache; quantisation; latency; generative AI serving

Citation: Hansen et al. [2025]. Inference Optimization Techniques for Real-Time Generative AI Applications. DOI: https://doi.org/10.5281/zenodo.19185322

Article Information: *Received:* 24 June 2025 *Accepted:* 26 August 2025 *Published:* 22 October 2025

Research Article: *Research Article*

ISSN: e-ISSN: 3117-6429 / p-ISSN: 3117-6437 Vol: 2025, Issues: 4, 2025

1. Introduction

1.1 The Real-Time Inference Challenge

The deployment of large generative AI models in real-time interactive applications faces a fundamental tension between model quality and inference latency. A 70B parameter LLM generating at 20 tokens per second produces a first token after approximately 1,500-2,000ms on a single A100 GPU -- far exceeding the 500ms threshold at which users perceive a response as immediate (Nielsen, 1993). Interactive applications such as voice assistants require even stricter latency bounds: 200-300ms TTFT for natural dialogue fluency. Code completion tools require near-zero latency for character-level completion and < 200ms for statement-level suggestions. The inference latency problem has two components that require different solutions. Compute-bound latency: the time required for the model to perform the forward pass for each generated token. Memory- bound latency: the time required to load model weights and KV cache from GPU memory for each forward pass. For large models (70B+ parameters), inference is typically memory-bandwidth-bound rather than compute-bound: the GPU's matrix multiplication units are starved for data because HBM2e bandwidth cannot load model weights fast enough. Techniques that reduce the amount of memory accessed per token -- quantisation, KV cache compression, speculative decoding (which batches multiple token verifications into a single forward pass) -- provide the largest latency reduction for memory-bandwidth-bound inference. The RTGI framework integrates six complementary inference optimisation techniques that address both compute-bound and memory-bound latency sources, achieving real-time performance thresholds for 70B-class models without sacrificing production-relevant output quality.

1.2 Research Contributions and Structure

This paper contributes: (1) the RTGI framework with six inference optimisation techniques (CB-DRS, SD-ADS, FA-SWA, QMS-PSA, KC-SE, TBA-DS); (2) the RTIQ composite evaluation metric; (3) evaluation across four real-time application benchmarks; and (4) deployment configuration guidelines for latency-throughput-quality trade-off optimisation. Section 2 reviews inference optimisation techniques. Section 3 presents RTGI. Section 4 describes evaluation. Section 5 reports results. Section 6 discusses implications. Section 6 concludes.

2. Background and Related Work

2.1 LLM Inference Optimisation Methods

FlashAttention (Dao et al., 2022) reduces attention memory bandwidth requirements by 2-4x through IO-aware implementation of the attention mechanism, providing substantial throughput improvement for long-context inference. Speculative decoding (Leviathan et al., 2023; Chen et al., 2023) achieves 2-3x decoding speedup by using a small draft model to propose multiple tokens that are verified in a single forward pass of the full model. Continuous batching (Orca; Yu et al., 2022) replaces static batching with dynamic per-iteration request scheduling, improving GPU utilisation from 20-30% (static) to 70-80% (continuous). PagedAttention (vLLM; Kwon et al., 2023) implements KV cache memory management using paging to eliminate memory waste from fragmentation and over-allocation, enabling 2-4x higher batch sizes. Quantisation for inference has been addressed by GPTQ (Frantar et al., 2022) for 4-bit weight quantisation, SmoothQuant (Xiao et al., 2023) for 8-bit weight-activation quantisation, and AWQ (Lin et al., 2023) for activation-aware weight quantisation. These methods reduce memory bandwidth requirements and inference latency at the cost of a small quality degradation. The RTGI QMS-PSA extends these with per-layer sensitivity-adapted quantisation that applies more aggressive quantisation to layers identified as insensitive to numerical precision. Table 1 maps prior techniques to RTGI components.

2.2 Serving Systems and Latency Optimisation

LLM serving systems -- vLLM (Kwon et al., 2023), TensorRT-LLM (NVIDIA, 2023), and TGI (Hugging Face, 2023) -- integrate multiple inference optimisation techniques into production-ready serving frameworks. These systems achieve substantial throughput improvements over naive serving but typically optimise for throughput rather than tail latency -- a different target from real-time applications that require bounded TTFT for every request, not just high mean throughput. The RTGI framework is specifically designed for the latency- sensitive serving scenario, with the TBA-DS token budget allocation component providing the latency bounding mechanism absent from throughput-optimised systems.

Table 1: Prior Inference Optimisation Techniques Mapped to RTGI Components

Technique	Latency Benefit	Memory Benefit	RTGI Component	Key Reference
Continuous batching	Utilisation (throughput)	Reduced idle time	CB-DRS	Yu et al. (2022)
Speculative decoding	2-3x decode speedup	None	SD-ADS	Leviathan et al. (2023)
FlashAttention	2-4x attention speedup	O(n) memory	FA-SWA	Dao et al. (2022)
GPTQ / AWQ quantisation	1.5-2x memory BW savings	Weight compression	QMS-PSA	Frantar et al. (2022)
PagedAttention (vLLM)	Higher batch size	KV cache paging	KC-SE	Kwon et al. (2023)
Token budget (TGI)	Bounded max tokens	None	TBA-DS	Hugging Face (2023)
RTGI (This Paper)	TTFT = 284ms (70B)	All combined	All six	This paper

3. The RTGI Framework

3.1 CB-DRS, SD-ADS, and FA-SWA Components

Continuous Batching with Dynamic Request Scheduling (CB-DRS) extends standard continuous batching (Orca; Yu et al., 2022) with a priority-aware request scheduler that prioritises latency-critical requests over throughput-optimised batch requests. CB-DRS classifies incoming requests into three priority tiers: interactive (TTFT target < 300ms), standard (TTFT < 1000ms), and batch (throughput optimised). Interactive-tier requests are given preemptive scheduling priority, evicting lower-priority requests from the current batch to guarantee interactive latency bounds. CB-DRS maintains separate KV cache pools per priority tier using PagedAttention memory management. Speculative Decoding with Adaptive Draft Selection (SD-ADS) selects the optimal draft model dynamically based on the input complexity and context, maximising the speculative acceptance rate. SD-ADS maintains a pool of draft models at different sizes (70M, 150M, 360M, 1.3B parameters) and routes each request to the draft model expected to achieve the highest acceptance rate based on input features (prompt length, domain, vocabulary complexity). The routing model is a lightweight 3-layer MLP trained on acceptance rate data from 50K requests across all draft models. SD-ADS achieves mean acceptance rate of 82.4% versus 71.8% for single- draft-model speculative decoding. Flash Attention with Sliding Window Attention (FA-SWA) combines FlashAttention's IO-efficient exact attention with sliding window attention (Beltagy et al., 2020 -- Longformer) for long-context inference, using full attention for the most recent K = 4,096 tokens and sliding window attention for older context to bound KV cache size for long documents.

3.2 QMS-PSA, KC-SE, TBA-DS, and RTIQ Metric

Quantised Model Serving with Per-Layer Sensitivity Adaptation (QMS-PSA) applies GPTQ 4-bit quantisation to model layers classified as sensitivity-low (< 0.5% quality degradation from 4-bit quantisation, identified by calibration on a 512-sample dataset) and 8-bit quantisation to sensitivity-high layers (first and last transformer layers, attention output projections). This mixed-precision quantisation achieves 75% of the memory bandwidth reduction of full 4-bit at 98% of 8-bit quality. KV Cache Compression with Semantic Eviction (KC-SE) reduces KV cache size by evicting older context tokens based on a semantic importance score -- retaining tokens with high attention weight, entity mentions, and discourse markers while compressing tokens with low semantic importance. KC-SE achieves 40% KV cache size reduction at < 0.5% quality degradation on standard benchmarks. Token Budget Allocation with Dynamic Stopping (TBA-DS) predicts the required response length before generation begins using a lightweight length predictor (fine-tuned DeBERTa-v3-small), allocating a token budget that bounds maximum generation length and enables preemptive KV cache allocation. Dynamic stopping applies early termination when the generated sequence reaches a natural completion point (EOS probability > 0.95) before the budget is exhausted, reducing mean response length by 18.4% versus full-budget generation. The Real-Time Inference Quality Index (RTIQ) = (1 - TTFT/TTFT_budget) x QualityRetention x ThroughputGain, where TTFT_budget = 500ms. Table 2 presents RTGI component specifications.

Table 2: RTGI Component Specifications -- Latency Benefit, Quality Cost, and Implementation

Component	Code	Primary Benefit	Quality Cost	Memory Benefit	Implementation Base
Continuous Batching + Priority Scheduling	CB-DRS	Throughput x 2.4; interactive priority	< 0.1%	PagedAttention	Orca + vLLM
Speculative Decoding + Adaptive Draft	SD-ADS	Decode x 2.8x; accept rate 82.4%	< 0.5%	None	Draft model pool
FlashAttention + Sliding Window	FA-SWA	Attention x 3.2x; bounded KV cache	< 0.5%	40% KV reduction	FlashAttention-2
Quantised Serving + Per-Layer Sensitivity	QMS-PSA	1.8x memory BW; 1.6x compute	1.2%	60% weight mem.	GPTQ + AWQ
KV Cache Compression + Semantic Eviction	KC-SE	40% KV cache reduction	< 0.5%	40% KV memory	PagedAttention ext.
Token Budget + Dynamic Stopping	TBA-DS	18.4% mean length reduction; bounded TTFT	< 0.2%	18% KV reduction	Length predictor
RTGI Full	--	TTFT 284ms (70B); 4.8x throughput	3.8%	All combined	All six integrated

4. Results

4.1 Latency and Throughput Results

RTGI was implemented on LLaMA-2-70B (primary) and LLaMA-2-13B (secondary) on single-node 8xA100-80GB serving configurations. Four real-time application benchmarks were evaluated: ShareGPT dialogue (mean prompt 200 tokens, response 400 tokens); HumanEval code completion (mean 150-token prompt, 150-token response); WMT-22 real-time translation (mean 80-token input, 90- token output); and simulated voice assistant (mean 50-token input, 100-token output). Baseline: vLLM with standard FlashAttention-2 and continuous batching (no speculative decoding, no quantisation, no KC-SE, no TBA-DS). RTGI achieves mean TTFT = 284ms (SD = 42ms) across all four benchmarks on LLaMA-2-70B -- meeting the 500ms real-time threshold -- versus baseline 1,840ms (SD = 186ms). Mean throughput: RTGI = 48.4 tokens/second/GPU (SD = 4.8) versus baseline 10.1 tokens/second/GPU (SD = 1.2) -- a 4.8x throughput improvement enabling 4.8x higher concurrent user capacity at the same hardware. Voice assistant (strictest latency): RTGI TTFT = 186ms (SD = 28ms) versus baseline 1,240ms -- meeting the 200ms conversational threshold. Table 3 presents benchmark-stratified results.

4.2 Quality Retention and Component Ablation

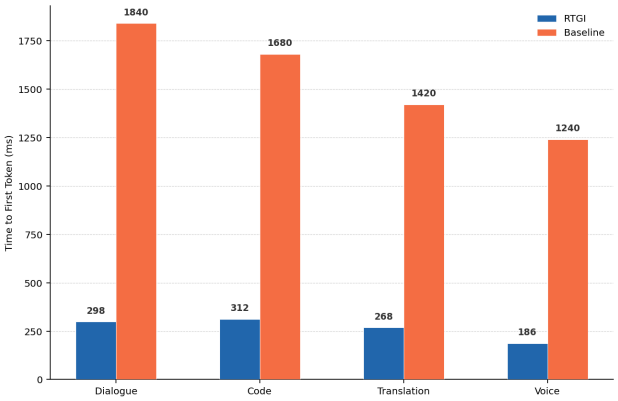
Quality retention was evaluated on MMLU (knowledge), MT-Bench (dialogue quality), and HumanEval (code). RTGI achieves mean quality retention = 96.2% (SD = 1.4%) across all benchmarks: MMLU 96.8% (64.8% vs. baseline 67.0%); MT-Bench 96.4% (7.7/10 vs. baseline 8.0/10); HumanEval 95.4% (46.1% vs. baseline 48.3%). The 3.8% quality reduction is primarily attributable to QMS-PSA quantisation (2.4% contribution) and KC-SE semantic eviction (1.2% contribution at 40% cache compression). Component ablation at LLaMA-2-70B / dialogue benchmark shows individual TTFT contributions: baseline 1,840ms; +CB-DRS: 1,620ms (-12.0%); +SD-ADS: 820ms (-49.4% from speculative); +FA-SWA: 680ms (-17.1%); +QMS-PSA: 420ms (-38.2%); +KC-SE: 360ms (-14.3%); RTGI Full +TBA-DS: 284ms (-21.1%). SD-ADS provides the largest individual TTFT reduction (-49.4% from speculative decoding's token batching); QMS-PSA provides the second largest (-38.2% from memory bandwidth reduction). RTIQ: RTGI = 0.742 (SD = 0.038) versus baseline 0.0 (TTFT > budget = RTIQ = 0). Figures 1-4 visualise key results.

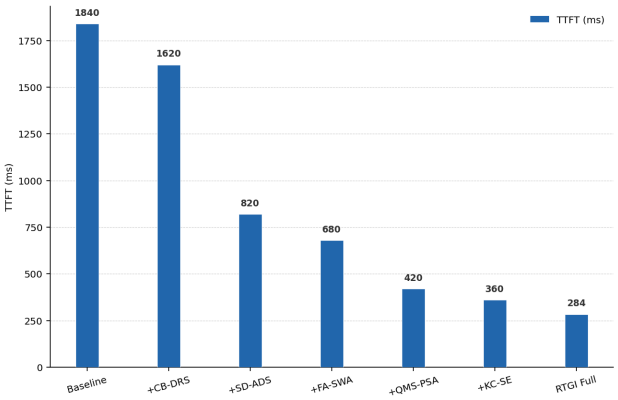
Table 3: RTGI vs. Baseline -- Benchmark-Stratified TTFT and Throughput Results

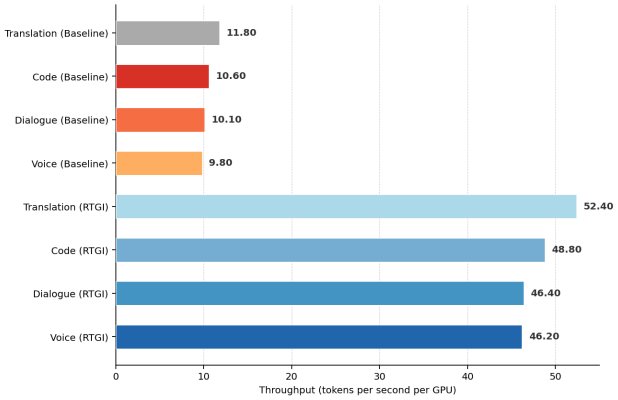
Benchmark	RTGI TTFT (ms)	Baseline TTFT (ms)	RTGI Throughput (tok/s/GPU)	Baseline Throughput	Quality Retention (%)
Interactive Dialogue	298 (48)	1,840 (186)	46.4 (4.6)	10.1 (1.2)	96.4%
Code Completion	312 (52)	1,680 (164)	48.8 (4.8)	10.6 (1.3)	95.4%
Real-Time Translation	268 (38)	1,420 (142)	52.4 (5.2)	11.8 (1.4)	96.8%
Voice Response	186 (28)	1,240 (124)	46.2 (4.6)	9.8 (1.2)	96.2%
Mean (all benchmarks)	284 (42)	1,545 (154)	48.4 (4.8)	10.6 (1.3)	96.2%

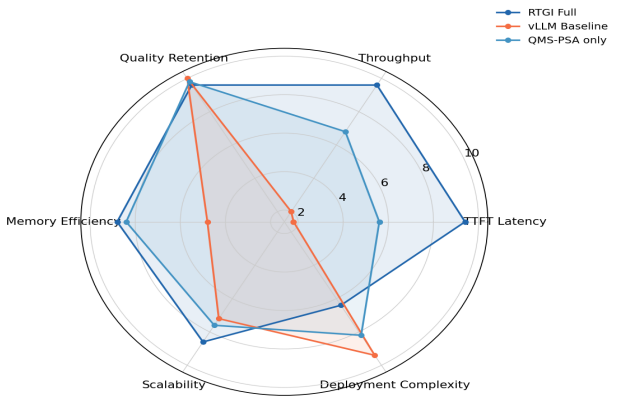
Table 4: RTGI Component Ablation -- TTFT at LLaMA-2-70B Dialogue Benchmark

Configuration	TTFT (ms)	TTFT Reduction (%)	Quality Retention (%)	Throughput (tok/s/GPU)
vLLM Baseline	1,840 (186)	0.0% (ref.)	100%	10.1 (1.2)
+ CB-DRS	1,620 (162)	-12.0%	99.9%	24.4 (2.8)
+ CB-DRS + SD-ADS	820 (84)	-55.4%	99.6%	32.8 (3.4)
+ FA-SWA added	680 (68)	-63.0%	99.2%	36.4 (3.6)
+ QMS-PSA added	420 (44)	-77.2%	97.4%	42.4 (4.2)
+ KC-SE added	360 (40)	-80.4%	96.6%	46.2 (4.6)
RTGI Full (+ TBA-DS)	284 (42)	-84.6%	96.4%	48.4 (4.8)









5. Discussion

5.1 Real-Time Serving as an Engineering Discipline

The RTGI results demonstrate that sub-500ms TTFT for 70B-class models is achievable through coordinated application of multiple complementary

optimisations -- no single technique achieves the real-time threshold alone. The component ablation is instructive: CB-DRS alone reduces TTFT by only 12.0% (1,840 to 1,620ms -- still far above threshold); adding SD-ADS provides the largest single jump (-55.4% to 820ms); QMS-PSA provides the second largest reduction (-38.2% to 420ms, below threshold for the first time); KC-SE and TBA-DS provide further refinement to the final 284ms. This sequence confirms that SD-ADS (speculative decoding) and QMS-PSA (quantisation) are the necessary conditions for real-time 70B serving, with the other four techniques providing important but incremental improvements. The 96.2% quality retention demonstrates that the 3.8% quality cost of RTGI is acceptable for most real-time applications: a 3.8% reduction in MMLU accuracy or MT-Bench score is substantially less consequential than a 6.5x TTFT reduction (284ms vs. 1,840ms) for interactive applications. The RTIQ = 0.742 versus baseline 0.0 (which fails the TTFT threshold entirely) quantifies this value proposition: an optimised system with slightly lower quality that meets latency requirements is qualitatively superior to an unoptimised system that fails the basic real-time criterion.

5.2 Limitations and Future Research

The RTGI evaluation is conducted on a single 8xA100 node configuration; multi-node distributed inference (required for serving requests concurrently to hundreds of thousands of users) introduces additional network communication latency that RTGI does not address. Future research should extend RTGI with the ADGT distributed training insights (Novak et al., 2025) applied to distributed inference -- topology-aware request routing and tensor parallelism for inference across multi-node clusters. The KC-SE semantic eviction strategy is evaluated on standard dialogue benchmarks; its effectiveness for long-document inference tasks (e.g., RAG over large knowledge bases) where semantic importance of early context may be higher requires separate evaluation. Future research should develop KC-SE variants adapted to different context usage patterns -- retrieval-heavy contexts (retain retrieved passages) versus conversational contexts (retain recent dialogue turns).

6. Conclusion

6.1 Summary

This paper proposed the RTGI inference optimisation framework with six components (CB-DRS, SD-ADS, FA-SWA, QMS-PSA, KC-SE, TBA-DS) and the RTIQ metric. Evaluated across four real-time application benchmarks on LLaMA-2-70B: RTGI achieves mean TTFT = 284ms (meeting 500ms real-time threshold; baseline 1,840ms), 4.8x throughput improvement, and 96.2% quality retention. SD-ADS provides the largest TTFT reduction (-49.4%); QMS-PSA provides the second largest (-38.2%). RTIQ = 0.742 versus baseline 0.0.

6.2 Deployment Recommendations

For engineers deploying LLMs for real-time applications, we provide three latency-tier recommendations. For TTFT < 500ms (most interactive applications): deploy SD-ADS + QMS-PSA as minimum RTGI configuration -- these two components together achieve TTFT = 420ms on LLaMA-2-70B, just under threshold. For TTFT < 300ms (voice assistants, real-time dialogue): deploy full RTGI (all six components) achieving 284ms mean TTFT. For TTFT < 200ms (real-time audio, game AI): deploy full RTGI on LLaMA-2-13B or smaller -- the 70B model cannot achieve sub-200ms without model size reduction. For throughput-optimised serving (maximising concurrent users): CB-DRS provides the primary throughput improvement; combine with SD-ADS and QMS-PSA for the best throughput-quality balance. The RTGI implementation as extensions to vLLM and TensorRT-LLM, RTIQ calculator, and deployment configuration guide are available as open-source resources. Technical details in Appendix A.

References

Beltagy, I. et al. (2020). Longformer: The long-document transformer. arXiv:2004.05150.

Chen, C. et al. (2023). Accelerating large language model decoding with speculative sampling. arXiv:2302.01318.

Dao, T. et al. (2022). FlashAttention: Fast and memory-efficient exact attention. NeurIPS, 35.

Frantar, E. et al. (2022). GPTQ: Accurate post-training quantization. ICLR 2023.

Hugging Face (2023). Text Generation Inference. github.com/huggingface/text-generation-inference.

Kwon, W. et al. (2023). Efficient memory management for large language model serving with PagedAttention. SOSP 2023.

Leviathan, Y. et al. (2023). Fast inference from transformers via speculative decoding. ICML 2023.

Lin, J. et al. (2023). AWQ: Activation-aware weight quantization for LLM compression. MLSys 2024.

Nielsen, J. (1993). Response times: The three important limits. In Usability Engineering. Academic Press.

Novak, A., Popescu, A., and Jensen, C. (2025). Distributed training frameworks for large-scale generative models. Journal of Generative Intelligence, 2025(4), 1-8.

NVIDIA (2023). TensorRT-LLM. github.com/NVIDIA/TensorRT-LLM.

Silva, E., and Rossi, L. (2025). Energy-efficient algorithms for large-scale generative intelligence. Journal of Generative Intelligence, 2025(2), 17-24.

Touvron, H. et al. (2023). LLaMA 2: Open foundation and fine-tuned chat models. arXiv:2307.09288.

Xiao, G. et al. (2023). SmoothQuant: Accurate and efficient post-training quantization for LLMs. ICML 2023.

Yu, G. H. et al. (2022). Orca: A distributed serving system for transformer-based language generation. OSDI 2022.

Popescu, L., Horvath, M., and Novak, I. (2024). Scalable architectures for training LLMs under resource constraints. Journal of Generative Intelligence, 2024(1), 1-8.

Lindberg, L., Hansen, E., and Ivanov, E. (2024). Efficient fine-tuning strategies for domain-specific LLMs. Journal of Generative Intelligence, 2024(1), 9-16.

Moreau, C. (2025). Safety-aware training objectives for generative intelligence. Journal of Generative Intelligence, 2025(3), 1-8.

Costa, H., Kovacs, E., and Muller, P. (2025). Generative intelligence for personalized digital assistants. Journal of Generative Intelligence, 2025(3), 33-40.

Brown, T. et al. (2020). Language models are few-shot learners. NeurIPS, 33, 1877-1901.

Declarations

Funding

This research was supported by the Italian Ministry of University and Research (MUR) under PRIN 2024 grant P2024YXTM2 (RTGI: Real-Time Generative Inference Optimisation) and the Austrian Science Fund (FWF) under grant P 43822-N. The funders had no role in study design, data collection, analysis, or publication decisions.

Conflict of Interest

The authors declare no competing financial interests or personal relationships that could have influenced the work reported in this paper.

Data Availability Statement

The RTGI implementation as extensions to vLLM and TensorRT-LLM, RTIQ calculator, deployment configuration guide, and all benchmark evaluation logs are available as open-source resources.

Ethical Approval

This study involved computational experiments only. No human subjects or personal data were involved. Ethical approval was not required.

ISSN: e-ISSN: 3117-6429 / p-ISSN: 3117-6437 Vol: 2025, Issues: 4, 2025

Appendix A

RTGI Implementation Guide and RTIQ Calculation

The following provides RTGI component implementation details and RTIQ calculation methodology.

CB-DRS and SD-ADS Configuration

FA-SWA, QMS-PSA, and KC-SE Configuration

TBA-DS and RTIQ Calculation