Journal of Global Research in Electronics and Communication
Volume 2, No. 05, May 2026
Available Online at: www.jgrec.info
REVIEWPAPER

Efficient LLM Serving Systems A Survey of Model Placement, Batching, and Resource Optimization Techniques

Prachi Rajput
Department of Computer Science and Engineering, ITM SLS Baroda University, Vadodara
prachi9497@gmail.com

***Abstract*—Large Language Models (LLMs) have become a fundamental component of modern artificial intelligence applications due to their remarkable capabilities in natural language understanding, generation, reasoning, and decision support. However, the increasing scale and complexity of these models have introduced significant challenges related to inference efficiency, memory consumption, latency, and resource utilization. This paper presents a comprehensive survey of efficient LLM serving systems, focusing on model placement strategies, batching techniques, and resource optimization approaches. The study reviews key deployment methods, including distributed serving, model parallelism, edge–cloud architectures, dynamic and continuous batching, KV-cache management, quantization, and intelligent scheduling. In addition, major challenges associated with scalability, energy consumption, load balancing, and memory management are examined. The analysis of recent literature demonstrates that the integration of system-level and algorithmic optimizations can substantially improve throughput, reduce latency, enhance resource utilization, and lower operational costs. The findings highlight that efficient memory management, adaptive scheduling, and resource-aware deployment strategies are essential for building scalable, reliable, and high-performance infrastructures for large-scale LLM serving**

Keywords*—Large Language Models (LLMs), Continuous Batching, KV-Cache Optimization, Model Parallelism, Quantization, Resource Scheduling.

I. INTRODUCTION

Large Language Models (LLMs) have caused a major shift in the artificial intelligence domain due to their wide-ranging abilities in natural language understanding, content generation, reasoning and code synthesis, as well as their uses in conversational applications [1]. LLMs rapidly developing & also finding uses in many different areas of our daily lives such as healthcare, education, finance and software engineering [2][3]; however, they continue to face challenges associated with deploying them for real-time or interactive AI services because require extensive compute and memory resources during inference. As the demand for interactive AI services grows, there is an increasing need to efficiently deploy and serve LLMs [4].

Unlike how LLMs are trained, LLMs must serve user-generated content in real-time while keeping actual latency low and throughput high. An instance of this is during autoregressive i.e., one word at a time, non-sequential text generation where the dynamic generation of Key-Value (KV) caches grows as the sequence continues to increase thus becoming the primary consumer of GPU memory [5]. Utilizing memory in a fragmented manner with redundant data stored can cause significant limitations to batch sizes and therefore have negatively impacted serving efficiency. Recent studies have demonstrated that using memory-aware frameworks and managing KV caches more effectively can significantly increase throughput while maintaining acceptable response latencies.

Various optimization approaches have been proposed to address challenges in model placement, quantization, and distributed inference. Recent studies emphasize that achieving efficient Large Language Model (LLM) deployment requires the integration of both algorithmic and system-level optimizations to balance latency, throughput, and resource utilization effectively. This paper presents a comprehensive survey of efficient LLM serving systems, covering model placement strategies, batching techniques, resource management approaches, and key deployment challenges. Furthermore, it provides a comparative analysis of existing solutions and identifies promising future directions for developing scalable, reliable, and high-performance infrastructures for LLM deployment.

A. Structure of the paper:

Section II presents Large Language Models Serving Systems. Section III discusses Model Placement Strategies. Section IV covers Batching Techniques for LLM Serving. Section V explains Resource Optimization Techniques. Section VI highlights challenges. Section VII reviews related literature. Section VIII concludes and outlines future work.

II. LARGE LANGUAGE MODELS SERVING SYSTEMS

LLMs create text from the autoregressive inference process whereby tokens are generated one after the other and are based on prior outputs from the same model[6]. The process of inference differs from that of training and therefore requires the persistence of intermediate states such as the Key-Value (KV) cache, which increases as the sequence length increases and leads to substantial amounts of memory consumption. Because modern LLMs are composed of billions of parameters, efficient inference represents an ever incandescently growing challenge, due to the skyrocketing computational and memory requirement necessary to operate LLMs[7]. Recent serving systems have focused on improving inference efficiency through optimized memory management, batching mechanisms, and resource allocation strategies. These techniques aim to increase throughput while maintaining acceptable latency, enabling large-scale

deployment of LLM-powered applications in production environments.

A. Serving Architecture

A typical LLM serving architecture consists of several components, including request scheduling, batching, model execution, and memory management [8]. When a user submits a request, it is first processed by the scheduler, which decides how and when the request should be executed. The request is then grouped with others into batches before being sent to GPUs for inference. The overall efficiency of the serving system depends on how effectively these components work together.

As the size of language models continues to grow, traditional serving architectures face challenges related to memory limitations and resource utilization. To address these issues, recent systems have introduced distributed and disaggregated architectures that separate computation and memory management tasks. These designs improve scalability and help organizations serve larger numbers of users efficiently [9].

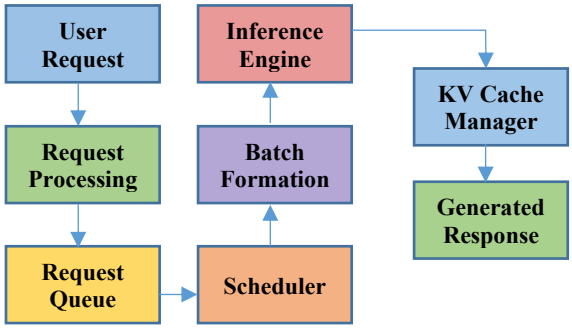

Fig. 1. General Architecture of an LLM Serving System

Figure 1 illustrates the general workflow of an LLM serving system. User requests are processed, queued, and scheduled before being grouped into batches for efficient execution by the inference engine. The KV Cache Manager stores intermediate states during generation, helping improve memory efficiency and reduce latency. Finally, the generated response is returned to the user.

B. Performance Metrics

To assess the performance of a large language model (LLM) service, metrics like throughput, latency, time to first token (TTFT), and GPU utilization are used. Throughput is defined as the quantity of work performed over a period of time, and latency is defined as how long it takes the user to receive a response after submit a request [10]. In production deployments, meeting only a high level of throughput is not enough. Users also desire rapid response times, as well as reliable performance. Modern LLM-serving systems should focus on achieving an appropriate balance between throughput, latency, and resource utilization to provide an optimal user experience.

III. Model Placement Strategies

Model placement strategies are fundamental to the efficient deployment of large language models, as they determine how computational workloads and model components are allocated across available resources. Effective placement approaches help optimize resource utilization, scalability, latency, and overall inference performance in diverse computing environments.

A. Single-Node and Distributed Deployment

The use of a single-node deployment allows smaller language models to run on a single GPU server, where the model fits completely in the memory of the device and is easy to manage since there is no need for communication overhead between devices. however, as models become larger, this is no longer feasible. To address this situation, the distributed deployment spreads the processing of a model over many GPUs instead of requiring one GPU to process a model entirely [11]. While this method does allow for scalability, it also adds additional costs associated with synchronizing and communicating between the devices; therefore, these must be monitored closely to ensure that inference performance for the model is not adversely affected.

B. Model Parallelism

Model parallelism is a widely used technique for serving large language models that cannot fit into a single device. Instead of storing the entire model on one GPU, model parameters are distributed across multiple devices, allowing larger models to be executed efficiently [12]. Different forms of model parallelism, including tensor parallelism and pipeline parallelism, have been developed to reduce memory pressure and improve scalability. These techniques play a key role in enabling the deployment of today's largest language models .

C. Edge-cloud Placement

The edge-cloud placement of processing has shown to be a beneficial method for performance critical real-time applications where latency is unacceptable or needs to happen in near real-time conditions due to edge devices being used to handle low weight processing while high weight processing remains at the cloud data centres. The addition of both resources lead organizations to achieve performance gains and minimal delays, maintaining the strength required in running large scale language models [13]. This can be of huge importance when undertaking mobile and Internet-of-Things (IoT) applications [14].

IV. Batching Techniques For LLM Serving

Efficient request scheduling plays a crucial role in improving the performance of large language model inference systems. By organizing multiple user requests for simultaneous processing, these approaches enhance hardware utilization, increase throughput, reduce response delays, and support scalable deployment under varying workload conditions.

A. Static Batching

Static batching groups multiple requests into fixed-size batches before execution. Processing requests together improves GPU utilization and increases throughput compared with executing requests individually [15]. Despite its simplicity, static batching may increase waiting time because requests often need to wait until a batch becomes full. This limitation makes it less suitable for highly dynamic workloads.

B. Dynamic Batching

Dynamic batching improves the efficiency of using resources when compared with static batched requests through dynamic batching by creating batch sizes that adjust and are based on existing workloads, such that an arrival pattern is used versus an established batch size for grouping requests .

Therefore, dynamic batching provides greater efficiencies of resource use, resulting in lower latency and greater throughput performance, and is therefore used frequently in production level LLM serve systems.

C. Continuous Batching

Continuous batching enables new requests to join an active batch during execution rather than waiting for the entire batch to complete. This keeps GPUs occupied more consistently and reduces idle time, because autoregressive generation produces tokens sequentially, continuous batching has become one of the most effective approaches for improving resource utilization in modern LLM serving environments [16].

V. Resource Optimization Techniques

As large language models continue to grow in size and complexity, efficient utilization of computational resources becomes essential for maintaining scalable and cost-effective inference. Various optimization approaches focus on reducing memory overhead, improving hardware efficiency, and enhancing throughput while preserving model performance and service quality.

A. Memory Management

Memory fragmentation is one of the major difficulties for LLMs that rely on KV caches due to the unpredictable amount of cache space used as the number of tokens increases. The cache size of each of the different models and the random number of tokens produced from each model means that a significant amount of the available amount of GPU memory is unusable [17]. Demand-paging solutions allow for physical memory to be allocated to cache blocks as tokens are generated vs reserving physical memory at the beginning of model execution. Distributed allocation of cache blocks to virtual addresses based on demand via an existing GPU driver support for physical memory allocation and thus avoids the costs associated with kernel rewrites and the other complexities involved with earlier implementations of block remapping.

B. KV-Cache Optimization

Reusing pre-computed KV caches across requests that share the same context can eliminate substantial redundant computation, but transmitting large cache tensors over a network introduces delays that offset the benefit [18]. This addresses by encoding KV caches into compact bitstreams that exploit the distributional structure of cached tensors, reducing transmission size significantly while adapting compression aggressiveness to available bandwidth. This makes long context reuse genuinely practical in multi-user deployments.

C. Quantization and Scheduling

The decrease of the number of bits used to represent neural network weights decreases the amount of memory used and allows for quicker inference times. however, naive quantization lead to significant accuracy loss due to the relatively small number of weights that experience large activation magnitudes are significantly more sensitive to precision loss than lower-magnitude weights. Adaptive weight quantization (AWQ) identifies which weights are statistically more likely to be affected by loss of precision based on activation distribution statistics; thus, when are compressed to a lower-precision bit width, use a mathematically equivalent per-weight scaling transformation. In conjunction with utilizing scheduling strategies that group model inputs based on their expected sequence length and/or model architecture affinity, using quantization as part of an overall optimization stack that considers memory, throughput and latency as inter-related objectives becomes possible.

VI. Challenges In Efficient LLM Serving Systems

Despite significant advancements in large language model serving technologies, several challenges continue to affect their efficiency, scalability, and practical deployment. Addressing these issues is essential for developing robust, cost-effective, and sustainable LLM serving systems capable of supporting large-scale real-world applications, and a variety of challenges for efficient LLM serving as follows:

A. Scalability and Resource Constraints

- A significant part of LLMs' efficiency lies in the strength of the GPU memory and computing power are fed.
- High expenses are incurred in deploying and serving these models, which could be mostly challenging for small firms.
- Model compression and resource optimization are just some of the ways to bring down the price.

B. Latency Throughput Trade-offs

- A system's throughput can be Greatly improved by processing a bigger number of requests simultaneously.
- Usually, larger batches make users wait longer and response latency goes up.
- Achieving the right balance between a quick response and the highest level of efficiency.

C. Efficient KV-Cache and Memory Management

- Longer input sequences make the KV-cache bigger, thereby leading to increased memory usage.
- Also, if cache management is poorly done, it may end up in the precomputation of the same thing and further slowdown of performance.
- There is a strong need for sound caching strategies to correctly handle huge workloads [19].

D. Load Balancing and Dynamic Scheduling

- The distribution of requests received to the point at which are the available resources must be made even to maintain the balance.
- And, achieving better performance is likely when cache locality is held. So, the designing of scheduling mechanisms that meeting both these goals is a challenging problem.

E. Distributed and Edge-Cloud Serving Challenges

- Distributed serving depends on communication among different servers at regular intervals, which adds overhead.
- Edge devices typically come with very limited memory, power, and processing capabilities. It is always a struggle to be able to provide reliable performance on hardware that is diverse.

F. Energy Efficiency and Sustainability

- Servicing LLMs at a large scale calls for a lot of energy, and this also means that the operational costs go up.

- The environmental impact of the AI services, which, by their very nature, are energy-intensive. That means, improving energy efficiency is among the top priorities when it comes to the development of next-generation LLM serving systems.

VII. Literature Review

This section reviews recent studies on Large Language Model (LLM) serving systems and optimization techniques, while Table I summarizes their research focus, approaches, key findings, challenges, and future directions. Collectively, these studies highlight advancements in efficient, scalable, and resource-aware LLM deployment frameworks.

J. Jiang et al. (2026) Large language model (LLM) serving systems are memory-intensive and costly, with the key-value (KV) cache playing a vital role in low-latency and high-throughput inference. This survey reviews system-aware KV infrastructure (sKis) from a system behavior perspective, focusing on execution and scheduling, placement and migration, and representation and retention, while highlighting co-design opportunities and future research directions [20].

J. Delavande, R. Pierrard, and S. Luccioni (2026) Large Language Models (LLMs) are increasingly deployed in production, making inference a major contributor to computational and energy costs. This study examines how system-level factors such as quantization, batching, and request scheduling affect energy consumption and latency, providing an empirical evaluation of LLM inference performance on NVIDIA H100 GPUs under different serving configurations [21].

V. S. P. Bharathula (2025) Examines four key optimization areas: hardware acceleration, serving architecture design, model compression, and dynamic scaling strategies. It highlights improvements in memory efficiency, throughput, latency, and cost-effectiveness through techniques such as memory management, adaptive batching, model parallelism, quantization, pruning, and intelligent caching, while also discussing future directions in hardware-software co-design and compiler optimization [22].

G. Bai et al. (2024) a wide range of techniques for improving the resource efficiency of LLMs. It categorizes methods based on optimization objectives, including computational, memory, energy, financial, and network resources, across different stages of the LLM lifecycle. The survey also analyzes the relationships between resource types and optimization techniques and presents standardized evaluation metrics and datasets for fair and consistent comparison [23].

Z. Wan et al. (2024) LLMs have demonstrated remarkable capabilities in natural language understanding and generation, but require substantial computational resources. This survey provides a systematic review of efficient LLM research, organizing existing methods into a taxonomy of three interconnected categories: model-centric, data-centric, and framework-centric approaches for improving efficiency [24].

W. X. Zhao et al. (2023) LLMs, containing tens or hundreds of billions of parameters, have gained significant attention due to their advanced capabilities and widespread adoption. This survey reviews recent advances in LLMs, covering their background, key developments, and mainstream techniques, with a focus on four major aspects: pre-training, adaptation tuning, utilization, and capacity evaluation [25].

TABLE I. Summary of Studies on LLM Serving System and Techniques

Authors	Study On	Approach	Key Findings	Challenges / Limitations	Future Directions
J. Jiang et al. (2026)	KV Cache Management in LLM Serving	System-aware KV infrastructure (sKis) analysis	Improved understanding of execution, scheduling, placement, and retention mechanisms for efficient inference	High memory overhead and complex cache management in large-scale deployments	Adaptive KV cache optimization and system-model co-design
J. Delavande et al. (2026)	Energy-Efficient LLM Inference	Evaluation of quantization, batching, and scheduling on NVIDIA H100 GPUs	Serving configurations significantly affect latency and energy consumption	Trade-off between performance and energy efficiency remains unresolved	Development of energy-aware serving frameworks
V. S. P. Bharathula (2025)	LLM Serving Optimization	Hardware acceleration, model compression, and dynamic scaling	Enhanced throughput, memory efficiency, and cost-effectiveness	Scalability and resource management challenges in heterogeneous environments	Hardware-software co-design and advanced compiler optimizations
G. Bai et al. (2024)	Resource-Efficient LLMs	Computational, memory, energy, financial, and network optimization techniques	Comprehensive taxonomy and evaluation metrics for resource optimization	Lack of unified optimization across multiple resource dimensions	Multi-objective optimization frameworks and standardized benchmarks
Z. Wan et al. (2024)	Efficient LLM Frameworks	Model-centric, data-centric, and framework-centric efficiency methods	Structured taxonomy for improving LLM efficiency	Limited integration among different optimization categories	Unified frameworks combining multiple efficiency strategies
W. X. Zhao et al. (2023)	LLM Development and Capabilities	Review of pre-training, adaptation, utilization, and evaluation techniques	Comprehensive overview of LLM evolution and capabilities	Limited focus on deployment efficiency and serving infrastructure	Research on scalable and efficient deployment mechanisms

VIII. Conclusion And Future Work

Large Language Models (LLMs) have emerged as a transformative technology that powers a wide range of intelligent applications across various domains. However, their growing size and computational requirements have made efficient serving a critical research challenge. This study examined the key techniques and challenges associated with efficient LLM serving systems, with particular emphasis on model placement strategies, batching mechanisms, and resource optimization approaches. The analysis showed that efficient deployment requires the integration of both system-level and algorithmic optimizations to balance latency, throughput, memory utilization, and operational cost. The reviewed literature highlighted the effectiveness of distributed deployment, model parallelism, continuous batching, KV-cache optimization, quantization, and intelligent scheduling in improving inference performance. Overall, the findings indicate that advanced memory management and resource-aware serving strategies are essential for developing scalable, reliable, and high-performance infrastructures capable of supporting the increasing demand for LLM-based applications.

Future research should focus on developing energy-efficient and resource-aware LLM serving frameworks that jointly optimize latency, throughput, and sustainability. Further investigation into adaptive KV-cache management, intelligent scheduling, edge-cloud deployment, and hardware-software co-design is required to support scalable, cost-effective, and environmentally sustainable large-scale LLM serving systems.

References

[1] X. Miao *et al.*, "Towards Efficient Generative Large Language Model Serving : A Survey from Algorithms to Systems," *ACM Comput. Surv.*, vol. 1, no. 1, pp. 1–35, January, 2025, doi: 10.1145/3754448.

[2] P. Kumar, "Leveraging Generative AI for Automated Data Standardization and Interoperability in Healthcare," in *2025 4th International Conference on Applied Artificial Intelligence and Computing (ICAAIC)*, IEEE, Dec. 2025, pp. 99–104. doi: 10.1109/ICAAIC64647.2025.11330217.

[3] J. W. Sajja and A. Nerella, "Enterprise Finance Reimagined: Harnessing ERP and Data Innovation for Next-Generation Value Creation," *Comput. Fraud Secur.*, vol. 2024, no. 4, p. 10, Apr. 2024, doi: 10.52710/cfs.743.

[4] N. K. Narendra Kumar Reddy Choppa, "Contextual Frameworks for Agentic AI: Engineering Adaptive Memory and Retrieval Mechanisms," *Comput. Fraud Secur.*, vol. 2024, no. 11, pp. 395–406, Nov. 2024, doi: 10.52710/cfs.747.

[5] W. Kwon *et al.*, "Efficient Memory Management for Large Language Model Serving with PagedAttention," pp. 611–626, 2023, doi: 10.1145/3600006.3613165.

[6] T. P. Patel, "Adaptive Token Routing for Heterogeneous LLM Inference in Edge-Cloud Continuum," in *SoutheastCon 2026*, 2026, pp. 1–7. doi: 10.1109/SoutheastCon63549.2026.11476596.

[7] Z. Zhou *et al.*, "A Survey on Efficient Inference for Large Language Models," 2024. doi: 10.48550/arXiv.2404.14294.

[8] B. Krishnan, S. Perla, S. Maddela, and R. Lingam, "Adaptive Multi-Cloud Infrastructure for CRM Analytics: Real-Time ML and Data Sync with LLMs," in *2025 IEEE 3rd Global Conference on Wireless Computing and Networking (GCWCN)*, IEEE, Nov. 2025, pp. 1–8. doi: 10.1109/GCWCN66157.2025.11448404.

[9] P. Patel *et al.*, "Splitwise: Efficient Generative LLM Inference Using Phase Splitting," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, 2024, pp. 118–132. doi: 10.1109/ISCA59077.2024.00019.

[10] Z. R. K. Rostam and S. Szénási, "Achieving Peak Performance for Large Language Models : A Systematic Review," vol. 3, no. July, pp. 96017–96050, 2024, doi: 10.1109/ACCESS.2024.3424945.

[11] R. Y. Aminabadi *et al.*, "DeepSpeed- Inference: Enabling Efficient Inference of Transformer Models at Unprecedented Scale," in *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2022, pp. 1–15. doi: 10.1109/SC41404.2022.00051.

[12] D. Narayanan *et al.*, "Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM," 2021, doi: 10.48550/arXiv.2104.04473.

[13] T. P. Patel and P. Agarwal, "DeepServe: Hierarchical Model Placement and Dynamic Batching for Cost-Efficient Multi-Tenant LLM Inference at Scale," in *SoutheastCon 2026*, Huntsville, AL, USA: IEEE, 2026, pp. 1–6, April. doi: 10.1109/SoutheastCon63549.2026.11476566.

[14] A. Sharshar, L. U. Khan, W. Ullah, and M. Guizani, "Vision-Language Models for Edge Networks: A Comprehensive Survey," *IEEE Internet Things J.*, vol. 12, no. 16, pp. 32701–32724, 2025, doi: 10.1109/JIOT.2025.3579032.

[15] G. Li *et al.*, "Easy and Efficient Transformer : Scalable Inference Solution For Large NLP Model," pp. 62–68, 2022, doi: 10.48550/arXiv.2104.12470.

[16] Y. He, Y. Lu, and G. Alonso, "Deferred Continuous Batching in Resource-Efficient Large Language Model Serving," in *Proceedings of the 4th Workshop on Machine Learning and Systems*, New York, NY, USA: ACM, Apr. 2024, pp. 98–106. doi: 10.1145/3642970.3655835.

[17] R. Prabhu, *vAttention : Dynamic Memory Management for Serving LLMs without PagedAttention*, vol. 1, no. 1. Association for Computing Machinery, 2025. doi: 10.1145/3669940.3707256.

[18] Y. Liu *et al.*, "CacheGen: KV Cache Compression and Streaming for Fast Large Language Model Serving," in *Proceedings of the ACM SIGCOMM 2024 Conference*, New York, NY, USA: ACM, Aug. 2024, pp. 38–56. doi: 10.1145/3651890.3672274.

[19] M. Parikh, A. A. Soni, S. M. Shah, and A. R. Jha, "Big Data Workload Profiling for Energy-Aware Cloud Resource Management," in *2026 International Conference on Data Analytics for Sustainability and Engineering Technology (DASET 2026)), Track: Big Data and Machine Learning Applications*, IEEE, Ed., arXiv preprint arXiv, 2026, pp. 01–07, januray. doi: 10.48550/arXiv.2601.11935.

[20] J. Jiang, P. Yang, R. Zhang, and F. Liu, "Towards Efficient Large Language Model Serving: A Survey on System-Aware KV Cache Optimization," Jan. 05, 2026. doi: 10.36227/techrxiv.176046306.66521015/v3.

[21] J. Delavande, R. Pierrard, and S. Luccioni, "Understanding Efficiency: Quantization, Batching, and Serving Strategies in LLM Energy Use," 2026, *arXiv*. doi: https://doi.org/10.48550/arXiv.2601.22362.

[22] Venkata Siva Prasad Bharathula, "LLM Serving Optimization Techniques: A Comprehensive Analysis," *J. Comput. Sci. Technol. Stud.*, vol. 7, no. 5, pp. 174–181, May 2025, doi: 10.32996/jcsts.2025.7.5.23.

[23] G. Bai *et al.*, "Beyond Efficiency: A Systematic Survey of Resource-Efficient Large Language Models," 2024, *arXiv*. doi: 10.48550/arXiv.2401.00625.

[24] Z. Wan *et al.*, "Efficient Large Language Models: A Survey," 2024. doi: 10.48550/arXiv.2312.03863.

[25] W. X. Zhao *et al.*, "A Survey of Large Language Models," *Front. Comput. Sci.*, vol. 20, no. 12, p. 2012627, Dec. 2026, doi: 10.1007/s11704-026-60308-3.

