A Survey on Efficient Serving of Large Language Models

Yavuz Ferhatosmanoglu
yff23@cam.ac.uk
Word Count: 2000

Abstract

This survey critically reviews systems-level techniques for efficient serving of large language models (LLMs), presents a unified taxonomy of optimisations, evaluates representative systems, and analyses key design trade-offs. We conclude with promising future research directions for efficient LLM serving.

1 Introduction

Large language models (LLMs) have achieved immense success across tasks such as question answering, text generation, and summarisation [14, 17, 32], with model families such as GPT, Llama, and DeepSeek reaching state-of-the-art results [2, 12, 18]. However, their scale and complexity introduce substantial inference challenges: Transformer-based architectures [26] with massive parameter counts incur high computational cost, significant memory pressure, long-tail latency, and considerable energy consumption, limiting the accessibility and practicality of real-world deployment of LLMs.

Consequently, efficient LLM serving has become a central research topic spanning machine learning, distributed systems, and computer architecture. A wide range of optimisation opportunities has emerged to meet the demands of large-scale model deployment. Yet, the rapid pace of progress makes it challenging to maintain a coherent understanding of the design space and the trade-offs between competing approaches.

This survey addresses this gap by providing a structured and critical review of efficient LLM inference methods. We introduce a taxonomy that organises existing work, use it to analytically evaluate leading frameworks, and highlight the key design decisions, trade-offs, and open challenges that shape LLM serving.

2 Background

Modern LLMs are built on the Transformer architecture [26], which alternates self-attention and feed-forward layers. In self-attention, the model weighs previous tokens to predict the next one, using per-token Query (Q), Key (K), and Value (V) vectors computed through learned transformations. This generation process is *auto-regressive*: each generated token is appended to the sequence, requiring a full forward pass per token.

Inference phases. LLM inference consists of two phases (illustrated in Figure 1):

(1) **Prefill.** The input prompt is processed in parallel to compute KV vectors. This stage is typically *compute-bound* and fully utilises GPU compute units.
(2) **Decode.** Tokens are generated auto-regressively, making this stage *memory-bandwidth-bound*, as model parameters must be repeatedly loaded despite minimal computation per token.

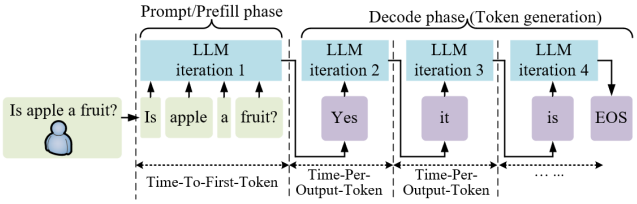

Figure 1: The LLM inference process (from [36]).

KV caching. To avoid recomputing attention over the full sequence during decoding, serving systems cache KV vectors. While this eliminates redundant computation, the cache grows linearly with sequence length and can become the primary memory bottleneck, restricting batch sizes and limiting throughput.

Evaluation metrics. Serving systems are commonly assessed using:

(1) **Time to First Token:** Latency before the first output token, largely determined by prefill performance.
(2) **Time per Output Token:** Average delay between consecutive generated tokens, reflecting decode performance.
(3) **Latency:** Total end-to-end time to generate a complete response.
(4) **Throughput:** Number of tokens or requests served per second across all users.
(5) **Resource utilisation:** Efficiency of GPU, CPU, memory, and bandwidth usage during inference.
(6) **Cost and energy efficiency:** Monetary or energy cost per generated token or request.

3 Taxonomy of LLM Serving Optimisations

This section presents a comprehensive taxonomy of LLM serving optimisations, highlighting their design motivations and critically evaluating their impact and limitations. Figure 2 provides an overview of the taxonomy.

3.1 Memory and State Management

Memory pressure is a central challenge in LLM inference, driven by both the dynamic growth of the KV cache and the static footprint of model weights. Recent work addresses these constraints through improved memory utilisation, reduced per-token storage, and caching strategies.

Memory virtualisation. Naively pre-allocating contiguous memory for KV caches can lead to fragmentation and wasted space, as variable sequence lengths in workloads cause dynamically sized KV caches. Inspired by virtual memory, vLLM [7] introduces *paged attention*, dividing the KV cache into fixed-size, non-contiguous pages mapped through a page table. While this enables larger batch sizes and avoids costly compaction [27], it introduces challenges: attention kernels must operate over scattered memory regions,

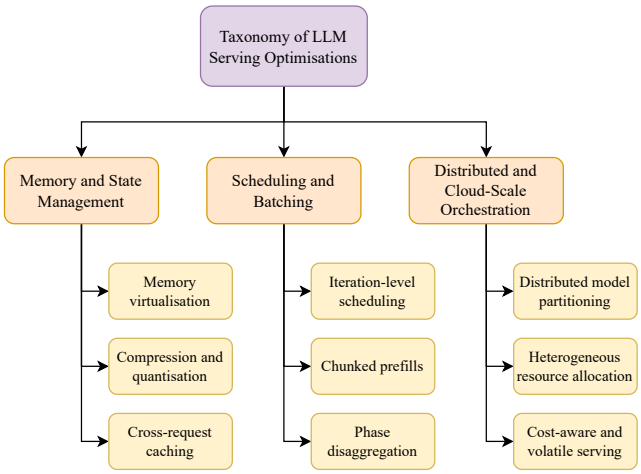

Figure 2: Overview of our taxonomy of LLM serving optimisations.

necessitating bespoke CUDA kernel implementations and complicating support for new attention variants. vAttention [21] addresses these by leveraging CUDA's virtual memory mechanisms, simplifying kernel design for paged layouts. Similarly, SGLang [33] adopts a structured KV-cache layout that improves memory locality and supports kernels implemented in higher-level systems such as Triton [25], reducing reliance on hand-written CUDA. These developments represent ongoing efforts to retain paged memory flexibility while mitigating kernel-level complexity.

Compression and quantisation. KV caches have a large memory footprint, motivating quantisation techniques [4, 6, 37]. KIVI [37] applies tuning-free 2-bit quantisation of KV tensors, and has been adopted in ThunderServe [5] to reduce communication overhead in distributed systems. While KIVI targets dynamic memory, static memory bottlenecks are addressed through model-level quantisation, representing weights in reduced precision (e.g., FP16, INT8) using Quantisation-Aware Training (QAT) [29] or Post-Training Quantisation (PTQ) [30]. QAT tends to retain accuracy better, while PTQ is cheaper and faster. Although quantisation reduces memory and increases throughput, aggressive schemes can degrade accuracy or cause numerical instability, highlighting the need for workload-aware strategies.

Cross-request caching. Beyond per-request KV caching, several approaches aim to reuse computation across requests. For instance, SGLang [33] uses an LRU cache of prefill KV blocks in a radix tree, enabling efficient detection of shared prefixes and eliminating redundant prompt processing. Similarly, *Prompt Cache* [3] segments prompts into reusable cross-request components. End-to-end semantic caches, such as GPTCache [22], store prompts with generated outputs, shifting focus from KV reuse to request-level caching. With the rise of multi-step agentic workflows [20], such cross-request caching mechanisms become increasingly important, complementing low-level memory optimisations and offering substantial system-level efficiency gains.

3.2 Scheduling and Batching

Maximising GPU utilisation requires minimising idle time. Early systems used static batching, processing a batch until the longest sequence is completed, necessitating excessive padding and causing resource under-utilisation due to *stragglers*. Scheduling strategies have since evolved towards finer-grained iteration control and architectural disaggregation.

Iteration-level scheduling. To address the inefficiencies of static batching, Orca [31] introduces iteration-level scheduling, also called *continuous batching*, treating each decoding step as a scheduling unit. Instead of waiting for full batch completion, finished slots are immediately reassigned to new requests, improving GPU utilisation and eliminating stragglers. Unlike sequence-length-aware batching [34], which statically groups requests with similar predicted output lengths, continuous batching dynamically interleaves requests regardless of their lengths, offering finer-grained GPU control. However, this flexibility can introduce *prefill–decode interference*: admitting a new request triggers a compute-intensive prefill, which can stall ongoing decode requests, increasing tail-latency and jitter.

Chunked prefills. Sarathi-Serve [1] mitigates prefill–decode interference through chunked prefill scheduling. Instead of processing a long prompt as a single operation, the prefill phase is divided into chunks that interleave with decode steps. This fine-grained time-sharing scheduler prevents resource monopolisation, reduces latency spikes, and ensures stable, consistent token delivery. Such behaviour is particularly important for interactive applications like chatbots, where predictable generation matters more than peak throughput.

Phase disaggregation. Despite improvements in scheduling logic, a fundamental hardware tension remains: prefilling is *compute-bound*, saturating tensor cores, while decoding is *memory-bandwidth bound*, saturating high-bandwidth memory. Co-locating both on the same hardware inevitably causes interference. Systems such as Splitwise [19] and DistServe [35] address this by disaggregating the phases across separate GPUs or nodes, enabling independent optimisation of parallelism and resource allocation for each phase. This improves SLA compliance and *goodput* for long-context workloads, but adds complexity: the KV cache must be transferred between prefill and decode nodes, making performance sensitive to network bandwidth and latency.

3.3 Distributed and Cloud-Scale Orchestration

As LLMs scale to hundreds of billions of parameters, their memory and compute demands often exceed a single GPU. While *data parallelism* is commonly employed, efficient serving increasingly relies on *model parallelism* and coordinated orchestration across multiple GPUs, especially in heterogeneous and volatile cloud environments.

Distributed model partitioning. Serving massive LLMs requires parallelism along different dimensions. *Tensor parallelism* [23] splits computations within layers across devices, offering high throughput but requiring high-bandwidth interconnects [8] due to frequent communication. *Pipeline parallelism* [16] partitions layers sequentially, reducing communication but potentially introducing idle bubbles waiting for data. More recently, *sequence parallelism* [9] has emerged to address long-context workloads by splitting the

Table 1: Comparison of LLM Serving Systems.

System	Primary Issue Addressed	Core Innovation	Memory Virtualisation	Scheduling Strategy	Hardware Scope
Orca [31]	Straggler-induced underutilisation	Iteration-level scheduling	Contiguous KV cache	Iteration-level	Homogeneous cluster
vLLM [7]	Memory fragmentation	Paged attention	Paged	Iteration-level	Homogeneous cluster
SGLang [33]	Redundant prompt prefix computation	Radix tree-based caching	Paged (with radix tree)	Iteration-level (with prefix caching)	Homogeneous cluster
Sarathi-Serve [1]	Latency spikes	Chunked prefills	Paged	Iteration-level (with chunking)	Homogeneous cluster
DistServe [35]	Compute and IO-bound interference	Disaggregated serving	Paged	Disaggregated iteration-level	Homogeneous cluster
Helix [11]	Heterogeneous cluster utilisation	Max-flow routing	Paged	Network-aware iteration-level	Heterogeneous (cloud)
SpotServe [13]	Spot instance preemptions	Token-level checkpointing	Paged	Preemption-aware	Heterogeneous (cloud)
ThunderServe [5]	Cloud instability and cost	Dynamic cost-based scheduling	Paged	Cloud-aware, disaggregated	Heterogeneous (cloud)

input sequence across devices, with each processing a sequence segment and synchronising intermediate attention outputs using *all-gather*. While these strategies enable large-model inference, they often require homogeneous, high-speed clusters, constraining high-performance serving to costly, enterprise-grade data centres.

Heterogeneous resource allocation. Real-world deployments are rarely uniform, with GPUs differing in compute power, memory, and interconnect speed. Efficient serving must account for this heterogeneity rather than assume a uniform cluster. An example is Helix [11], which models GPUs and network links as a weighted graph and solves a max-flow problem to optimise model placement, balancing compute load and communication across diverse resources. Similarly, ThunderServe [5] formulates the scheduling problem as a two-level hierarchical optimisation problem to produce a deployment plan. Such heterogeneity-aware strategies can significantly improve resource utilisation and performance, democratising LLM serving by enabling it on older or consumer-grade hardware.

Cost-aware and volatile serving. Although cloud deployments offer cost savings, they pose challenges for LLM serving due to variable node availability, preemptions, and diverse nodes. ThunderServe [5] addresses these with a lightweight rescheduler that dynamically adapts to workload and node conditions. Complementing this, SpotServe [13] targets *spot instances* with token-level stateful recovery, allowing interrupted requests to resume on other nodes without recomputation. While these systems can cut price-per-token by 50% [5, 13], the overhead of monitoring, migration, and recovery makes them better suited for batch or non-critical workloads than for real-time applications.

4 Comparative Analysis of Serving Systems

In this section, we critically evaluate recent LLM serving systems, building on our taxonomy. Table 1 provides a comprehensive analysis of these systems.

A clear trend is the standardisation of mechanisms such as continuous batching and paged KV management, introduced by Orca's iteration-level scheduling and vLLM's PagedAttention, which have resolved core utilisation and memory-fragmentation bottlenecks. With these layers now largely mature, memory management and local scheduling have become primitives rather than differentiators, and so primary research opportunities lie in higher-level orchestration and workload-aware optimisations.

Divergence across systems arises in balancing throughput and latency. Throughput-centric engines like Orca and early vLLM maximise GPU utilisation but exhibit latency jitter when compute-heavy prefills interfere with decodes. Sarathi-Serve mitigates this by chunking prefills to stabilise latency, while SGLang goes further by avoiding computation entirely through prefix caching. These mark a transition from hardware-driven optimisations to computation-avoidant strategies, which are increasingly important for retrieval-augmented and agentic workloads with high context reuse.

These scheduling and workload-level advancements highlight a broader tension in system design: optimising for idealised data-centre conditions versus volatile cloud deployments. DistServe embodies the former, delivering strong SLA compliance through physical phase disaggregation, but depending on fast interconnects. In contrast, Helix, ThunderServe, and SpotServe accommodate heterogeneous GPUs, fluctuating bandwidth, and frequent preemptions, trading peak performance for robustness and cost efficiency. This contrast exposes a broader limitation of today's landscape:

while each system excels under specific assumptions, their optimisations rarely compose cleanly, underscoring the need for adaptable inference engines that dynamically adjust to match workload characteristics and cloud conditions.

5 Future Directions

The rapid evolution of LLM applications presents new frontiers for system design. We identify three critical directions for future research.

Agent-native orchestration. The shift from chatbots to autonomous agents introduces non-linear workflows involving loops, branching, and tool usage [28]. While traditional schedulers optimise for independent requests, agentic workloads call for dependency-aware scheduling, where related reasoning steps should be co-scheduled to minimise end-to-end latency and keep multi-step *thought processes* responsive for users.

Serving for large context. As context windows reach millions of tokens, the quadratic cost of attention and the linear KV cache growth remain bottlenecks, even with paged memory. Techniques such as ring attention [10] and sparse attention [24] should be tightly integrated into future inference systems to support "infinite" context streams without exceeding memory capacity.

Hardware-software co-design. Although GPUs currently dominate LLM inference, specialised hardware (e.g., LPUs [15]) should be developed to better match the dataflow of LLMs. Efficient serving on these platforms will require automating compiler stacks and inference engines to generate optimal kernels across diverse accelerators. Co-design strategies could embed KV caches closer to processing units and tailor chip layouts for sparse or long-context attention, allowing inference engines to fully exploit next-generation hardware.

6 Conclusion

This survey has critically examined systems-level techniques for efficient LLM serving through a unified taxonomy spanning memory management, fine-grained scheduling, and infrastructure-aware architectures. Using this framework, we analytically compared representative systems and analysed trade-offs shaping throughput, latency, and cost.

As LLMs move towards richer agentic workflows, longer contexts, and heterogeneous deployments, the focus shifts from isolated optimisations to coordinated orchestration. Delivering efficient inference in this evolving landscape will require adaptable serving strategies supported by holistic hardware-software co-design.

References

[1] Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, Alexey Tumanov, and Ramachandran Ramjee. 2024. Taming throughput-latency tradeoff in LLM inference with sarathi-serve. In *Proceedings of the 18th USENIX Conference on Operating Systems Design and Implementation* (Santa Clara, CA, USA) *(OSDI'24)*. USENIX Association, USA, Article 7, 18 pages.

[2] DeepSeek-AI. 2025. DeepSeek-V3 Technical Report. arXiv:2412.19437 [cs.CL] https://arxiv.org/abs/2412.19437

[3] In Gim, Guojun Chen, Seung seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. 2024. Prompt Cache: Modular Attention Reuse for Low-Latency Inference. arXiv:2311.04934 [cs.CL] https://arxiv.org/abs/2311.04934

[4] Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. 2025. KVQuant: Towards 10 Million Context Length LLM Inference with KV Cache Quantization. arXiv:2401.18079 [cs.LG] https://arxiv.org/abs/2401.18079

[5] Youhe Jiang, Fangcheng Fu, Xiaozhe Yao, Taiyi Wang, Bin Cui, Ana Klimovic, and Eiko Yoneki. 2025. ThunderServe: High-performance and Cost-efficient LLM Serving in Cloud Environments. arXiv:2502.09334 [cs.DC] https://arxiv.org/abs/2502.09334

[6] Hao Kang, Qingru Zhang, Souvik Kundu, Geonhwa Jeong, Zaoxing Liu, Tushar Krishna, and Tuo Zhao. 2024. GEAR: An Efficient KV Cache Compression Recipe for Near-Lossless Generative Inference of LLM. arXiv:2403.05527 [cs.LG] https://arxiv.org/abs/2403.05527

[7] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles* (Koblenz, Germany) *(SOSP '23)*. Association for Computing Machinery, New York, NY, USA, 611–626. doi:10.1145/3600006.3613165

[8] Ang Li, Shuaiwen Leon Song, Jieyang Chen, Jiajia Li, Xu Liu, Nathan R. Tallent, and Kevin J. Barker. 2020. Evaluating Modern GPU Interconnect: PCIe, NVLink, NV-SLI, NVSwitch and GPUDirect. *IEEE Transactions on Parallel and Distributed Systems* 31, 1 (Jan. 2020), 94–110. doi:10.1109/tpds.2019.2928289

[9] Shenggui Li, Fuzhao Xue, Chaitanya Baranwal, Yongbin Li, and Yang You. 2022. Sequence Parallelism: Long Sequence Training from System Perspective. arXiv:2105.13120 [cs.LG] https://arxiv.org/abs/2105.13120

[10] Hao Liu, Matei Zaharia, and Pieter Abbeel. 2023. Ring Attention with Blockwise Transformers for Near-Infinite Context. arXiv:2310.01889 [cs.CL] https://arxiv.org/abs/2310.01889

[11] Yixuan Mei, Yonghao Zhuang, Xupeng Miao, Juncheng Yang, Zhihao Jia, and Rashmi Vinayak. 2025. Helix: Serving Large Language Models over Heterogeneous GPUs and Network via Max-Flow. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1* (Rotterdam, Netherlands) *(ASPLOS '25)*. Association for Computing Machinery, New York, NY, USA, 586–602. doi:10.1145/3669940.3707215

[12] Meta. 2024. The Llama 3 Herd of Models. arXiv:2407.21783 [cs.AI] https://arxiv.org/abs/2407.21783

[13] Xupeng Miao, Chunan Shi, Jiangfei Duan, Xiaoli Xi, Dahua Lin, Bin Cui, and Zhihao Jia. 2023. SpotServe: Serving Generative Large Language Models on Preemptible Instances. arXiv:2311.15566 [cs.DC] https://arxiv.org/abs/2311.15566

[14] Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. 2025. Large Language Models: A Survey. arXiv:2402.06196 [cs.CL] https://arxiv.org/abs/2402.06196

[15] Seungjae Moon, Jung-Hoon Kim, Junsoo Kim, Seongmin Hong, Junseo Cha, Minsu Kim, Sukbin Lim, Gyubin Choi, Dongjin Seo, Jongho Kim, Hunjong Lee, Hyunjun Park, Ryeowook Ko, Soongyu Choi, Jongse Park, Jinwon Lee, and Joo-Young Kim. 2024. LPU: A Latency-Optimized and Highly Scalable Processor for Large Language Model Inference. arXiv:2408.07326 [cs.AR] https://arxiv.org/abs/2408.07326

[16] Deepak Narayanan, Amar Phanishayee, Kaiyu Shi, Xie Chen, and Matei Zaharia. 2021. Memory-Efficient Pipeline-Parallel DNN Training. arXiv:2006.09503 [cs.LG] https://arxiv.org/abs/2006.09503

[17] Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, and Ajmal Mian. 2025. A Comprehensive Overview of Large Language Models. *ACM Trans. Intell. Syst. Technol.* 16, 5, Article 106 (Aug. 2025), 72 pages. doi:10.1145/3744746

[18] OpenAI. 2025. GPT-5 System Card. https://cdn.openai.com/gpt-5-system-card.pdf

[19] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. 2024. Splitwise: Efficient generative LLM inference using phase splitting. arXiv:2311.18677 [cs.AR] https://arxiv.org/abs/2311.18677

[20] Aske Plaat, Max van Duijn, Niki van Stein, Mike Preuss, Peter van der Putten, and Kees Joost Batenburg. 2025. Agentic Large Language Models, a survey. arXiv:2503.23037 [cs.AI] https://arxiv.org/abs/2503.23037

[21] Ramya Prabhu, Ajay Nayak, Jayashree Mohan, Ramachandran Ramjee, and Ashish Panwar. 2025. vAttention: Dynamic Memory Management for Serving LLMs without PagedAttention. arXiv:2405.04437 [cs.LG] https://arxiv.org/abs/2405.04437

[22] Sajal Regmi and Chetan Phakami Pun. 2024. GPT Semantic Cache: Reducing LLM Costs and Latency via Semantic Embedding Caching. arXiv:2411.05276 [cs.LG] https://arxiv.org/abs/2411.05276

[23] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2020. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. arXiv:1909.08053 [cs.CL] https://arxiv.org/abs/1909.08053

[24] Yutao Sun, Zhenyu Li, Yike Zhang, Tengyu Pan, Bowen Dong, Yuyi Guo, and Jianyong Wang. 2025. Efficient Attention Mechanisms for Large Language Models: A Survey. arXiv:2507.19595 [cs.CL] https://arxiv.org/abs/2507.19595

[25] Philippe Tillet, H. T. Kung, and David Cox. 2019. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the*

3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages (Phoenix, AZ, USA) *(MAPL 2019)*. Association for Computing Machinery, New York, NY, USA, 10–19. doi:10.1145/3315508.3329973

[26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 6000–6010.

[27] Jing Wang, Youyou Lu, Qing Wang, Minhui Xie, Keji Huang, and Jiwu Shu. 2022. Pacman: An Efficient Compaction Approach for Log-Structured Key-Value Store on Persistent Memory. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. USENIX Association, Carlsbad, CA, 773–788. https://www.usenix.org/conference/atc22/presentation/wang-jing

[28] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science* 18, 6 (March 2024). doi:10.1007/s11704-024-40231-1

[29] Jiawei Yang, Zhongbo Li, Zeqin Feng, and Yongqiang Xie. 2025. A Survey On Neural Network Quantization. In *Proceedings of the 2025 6th International Conference on Computer Information and Big Data Applications (CIBDA '25)*. Association for Computing Machinery, New York, NY, USA, 384–394. doi:10.1145/3746709.3746773

[30] Zhewei Yao, Xiaoxia Wu, Cheng Li, Stephen Youn, and Yuxiong He. 2023. ZeroQuant-V2: Exploring Post-training Quantization in LLMs from Comprehensive Study to Low Rank Compensation. arXiv:2303.08302 [cs.LG] https://arxiv.org/abs/2303.08302

[31] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 2022. Orca: A Distributed Serving System for Transformer-Based Generative Models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, Carlsbad, CA, 521–538. https://www.usenix.org/conference/osdi22/presentation/yu

[32] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. 2025. A Survey of Large Language Models. arXiv:2303.18223 [cs.CL] https://arxiv.org/abs/2303.18223

[33] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. 2024. SGLang: efficient execution of structured language model programs. In *Proceedings of the 38th International Conference on Neural Information Processing Systems* (Vancouver, BC, Canada) *(NIPS '24)*. Curran Associates Inc., Red Hook, NY, USA, Article 2000, 27 pages.

[34] Zangwei Zheng, Xiaozhe Ren, Fuzhao Xue, Yang Luo, Xin Jiang, and Yang You. 2023. Response length perception and sequence scheduling: an LLM-empowered LLM inference pipeline. In *Proceedings of the 37th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA) *(NIPS '23)*. Curran Associates Inc., Red Hook, NY, USA, Article 2859, 14 pages.

[35] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. 2024. DistServe: disaggregating prefill and decoding for goodput-optimized large language model serving. In *Proceedings of the 18th USENIX Conference on Operating Systems Design and Implementation* (Santa Clara, CA, USA) *(OSDI'24)*. USENIX Association, USA, Article 11, 18 pages.

[36] Hao Zhou, Chengming Hu, Dun Yuan, Ye Yuan, Di Wu, Xue Liu, Zhu Han, and Charlie Zhang. 2025. Generative AI as a Service in 6G Edge-Cloud: Generation Task Offloading by In-context Learning. arXiv:2408.02549 [eess.SY] https://arxiv.org/abs/2408.02549

[37] Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2023. KIVI : Plug-and-play 2bit KV Cache Quantization with Streaming Asymmetric Quantization. (2023). doi:10.13140/RG.2.2.28167.37282