International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 10 | Oct - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

Mobile Edge Intelligence for LLM Inference: Offloading Mechanisms, Resource Optimization, and Open Challenges

B. RAMESH KUMAR
Research Scholar & Assistant Professor, Department of Computer Science with Data Analytics
KPR College of Arts Science and Research, Coimbatore
rameshhit2017@gmail.com

Dr.V VINODHINI
Associate Professor, Department of Information Technology, KPR College Of Arts Science And Research, Coimbatore.
vvprof133@gmail.com

Abstract
Large language models (LLMs) have transformed natural language understanding and generation, but their billions of parameters, large memory footprint, and autoregressive decoding make them difficult to run on resource-constrained mobile devices. Mobile edge computing (MEC) offers a middle path: it places compute, storage, and caching resources at the network edge, close to users, so that LLM inference can be partially or fully offloaded from devices without the latency, bandwidth, and privacy costs of distant cloud data centres [1], [2]. This paper presents a comprehensive review of LLM inference offloading and resource optimization in MEC environments. We first introduce the architectural foundations of MEC and the computational characteristics of transformer-based LLM inference, including the distinct prefill and decode phases. We then propose a taxonomy of offloading strategies covering full offloading, partial and split inference, multi-node collaborative inference, and hybrid device–edge–cloud routing. Next, we review resource optimization techniques across five dimensions: model compression, memory and key–value (KV) cache management, computation scheduling, communication efficiency, and energy management. We also summarize enabling frameworks, representative applications, and evaluation metrics. Finally, we identify open research challenges, including dynamic offloading under wireless uncertainty, privacy-preserving split inference, heterogeneous hardware support, mixture-of-experts deployment, standardized benchmarking, and sustainable edge AI. The review aims to serve as a structured reference for researchers and practitioners designing efficient, trustworthy edge-native LLM services for 6G networks.
Keywords: mobile edge computing, large language models, inference offloading, split inference, resource allocation, model compression, KV cache, edge intelligence, 6G

1. Introduction

1.1 Background and Motivation

Edge computing emerged as a response to the limits of the centralized cloud. As billions of connected devices generate data at the network periphery, moving every computation to remote data centres introduces unacceptable latency, backhaul congestion, and privacy exposure [1], [3]. Mobile edge computing (MEC), standardized by ETSI as multi-access edge computing, deploys servers at base stations and access points so that mobile users can offload computation-intensive tasks within one wireless hop [2], [4].

In parallel, the transformer architecture [5] has enabled large language models (LLMs) with remarkable few-shot and generative capabilities [6]. These models now underpin conversational assistants, code generation, translation, and multimodal agents. However, LLM inference is extremely resource-hungry. A model with tens of billions of parameters requires tens of gigabytes of memory in half precision, and each generated token requires a full forward pass through the network. Mobile devices, constrained by memory, battery, and thermal limits, cannot host such models efficiently, while cloud-only deployment suffers from network latency, service cost, and data privacy concerns.

The convergence of these two trends has given rise to *edge intelligence*: the idea of pushing AI inference to the network edge [7]. For LLMs specifically, recent visions argue that 6G networks should host LLMs at the edge to deliver low-latency, privacy-aware generative services [8], [9]. Achieving this requires answering three core questions: *where* each part of the inference should run (offloading), *how* scarce edge resources should be allocated (resource optimization), and *what* problems remain unsolved (open challenges).

1.2 Contributions

This review makes the following contributions:

1. It consolidates the architectural and computational foundations of MEC and LLM inference, highlighting why LLM workloads differ from conventional deep neural network (DNN) offloading.
2. It proposes a taxonomy of LLM inference offloading strategies and compares representative approaches in structured tables.
3. It reviews resource optimization techniques across model, memory, computation, communication, and energy dimensions, and presents a general joint optimization formulation.
4. It summarizes enabling frameworks, application domains, and evaluation metrics used in the literature.
5. It identifies open research challenges and outlines future research directions for edge-native LLM services.

1.3 Paper Organization

Section 2 describes the review methodology and positions this work against existing surveys. Section 3 presents background on MEC and LLM inference. Section 4 reviews inference offloading strategies. Section 5 covers resource optimization. Section 6 discusses frameworks, applications, and metrics. Section 7 outlines open challenges and future directions, and Section 8 concludes the paper.

2. Review Methodology

2.1 Search Strategy

This review follows a structured literature review approach. Literature was retrieved from IEEE Xplore, ACM Digital Library, SpringerLink, ScienceDirect, and arXiv. Search strings combined edge-related terms ("mobile edge computing", "edge intelligence", "multi-access edge computing") with model-related terms ("large language model", "LLM inference", "transformer", "generative AI") and technique-related terms ("offloading", "split inference", "resource allocation", "quantization", "KV cache"). Backward and forward snowballing from key surveys [2], [7], [9] was used to identify additional works.

2.2 Inclusion and Exclusion Criteria

Studies were included if they (a) addressed inference rather than only training, (b) targeted edge, mobile, or resource-constrained settings, or proposed techniques directly applicable to them, and (c) reported a system design, algorithm, or empirical evaluation. Foundational works on MEC and transformers were included to provide context. Studies focusing exclusively on cloud-scale training, or lacking technical detail, were excluded. The final corpus analysed in this paper comprises the 25 works listed in the References.

2.3 Comparison with Existing Surveys

Table 1 positions this review against related surveys. Earlier MEC surveys focus on generic task offloading, while edge intelligence surveys predate the LLM era. Recent generative AI surveys address the edge but give limited attention to LLM-specific resource optimization such as KV cache management.

Table 1. Comparison of this review with related surveys

Survey	Year	Focus	MEC offloading	LLM-specific	Resource optimization	Open challenges
Mao et al. [2]	2017	MEC from a communication perspective	Yes	No	Communication and computation	Yes
Mach & Becvar [4]	2017	MEC architecture and offloading	Yes	No	Offloading decisions	Yes
Zhou et al. [7]	2019	Edge intelligence for DNNs	Partial	No	Model and system level	Yes
Lin et al. [8]	2025	LLMs at the 6G edge (vision)	Partial	Yes	Partial	Yes

Survey	Year	Focus	MEC offloading	LLM-specific	Resource optimization	Open challenges
Xu et al. [9]	2024	Edge–cloud AIGC services	Partial	Partial	Partial	Yes
This review	2026	LLM inference offloading in MEC	Yes	Yes	Model, memory, compute, communication, energy	Yes

3. Background

3.1 Mobile Edge Computing Architecture

MEC extends cloud capabilities to the radio access network. Instead of routing every request through the core network to a remote data centre, MEC servers co-located with base stations or access points process requests locally [2], [4]. This reduces end-to-end latency, relieves backhaul traffic, and keeps sensitive data within the operator's edge domain [1], [3]. Figure 1 shows the three-tier device–edge–cloud architecture that most LLM offloading studies assume.

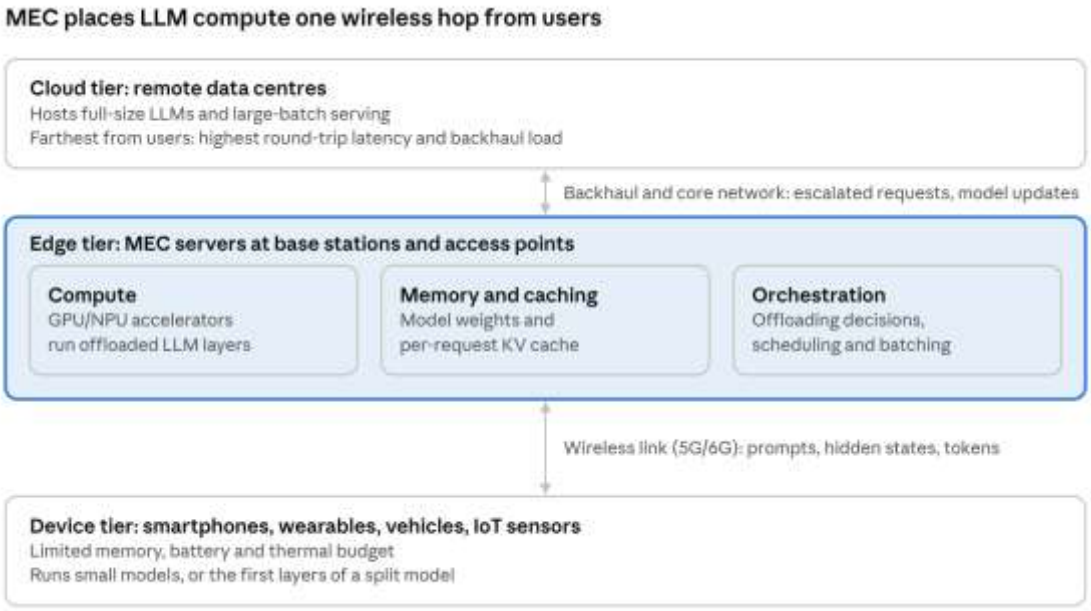


Figure 1. Three-tier MEC architecture for LLM inference

Figure 1. Three-tier device–edge–cloud architecture. The edge tier, highlighted, is the focus of this review; it hosts accelerators, model and KV cache memory, and the orchestration logic that decides where each inference step runs.

Classic MEC research models offloading as a trade-off between local execution cost and transmission plus remote execution cost, subject to wireless channel conditions, server capacity, and device energy [2], [4]. Edge intelligence extends this model to DNN inference, where a network can be partitioned at layer boundaries [7], [10], [11]. LLMs inherit these ideas but introduce new characteristics that change the trade-offs, as described next.

3.2 Fundamentals of Large Language Models

Modern LLMs are decoder-only transformers built from stacked blocks of multi-head self-attention and feed-forward layers [5]. Scaling parameters, data, and compute yields emergent abilities such as in-context learning [6]. Model sizes range from about one billion parameters for on-device models to hundreds of billions for frontier models. Mixture-of-experts (MoE) designs activate only a subset of parameters per token, reducing computation but not total memory [12].

3.3 Characteristics of LLM Inference

LLM inference proceeds in two phases with very different resource profiles, summarized in Table 2. In the **prefill** phase, the entire input prompt is processed in parallel, which is compute-bound. In the **decode** phase, tokens are generated one at a time, each step reading all model weights and the accumulated key–value (KV) cache, which is memory-bandwidth-bound [13], [14]. The KV cache grows linearly with sequence length and batch size:

$$M_{KV} = 2 \cdot L \cdot h_{kv} \cdot d_{head} \cdot S \cdot B \cdot b$$

where the factor 2 accounts for keys and values, L is the number of layers, h_kv the number of key–value heads (equal to the number of attention heads in standard multi-head attention, and fewer under grouped-query attention), d_head the head dimension, S the sequence length, B the batch size, and b the bytes per element. For long contexts, the KV cache can rival or exceed the size of the model weights, making memory management a first-class concern at the edge [13].

Table 2. Resource profile of LLM inference phases and implications for MEC

Phase	Parallelism	Bottleneck	Main latency metric	Implication for MEC offloading
Prefill	All prompt tokens at once	Compute (FLOPs)	Time to first token (TTFT)	Benefits most from powerful edge accelerators; long prompts favour offloading
Decode	One token per step, sequential	Memory bandwidth and KV cache capacity	Time per output token (TPOT)	Repeated per-token round trips make wireless latency critical; favours keeping decode close to the user
KV cache	Grows with context and batch	Memory capacity	Throughput, maximum concurrency	Limits how many users one edge server can serve concurrently

Three properties distinguish LLM offloading from conventional DNN offloading. First, **autoregressive generation** means that a split model may exchange intermediate activations once per generated token, multiplying communication cost. Second, **stateful execution** through the KV cache ties a request to the server that holds its cache, complicating migration during user mobility. Third, **extreme model size** means that even edge servers may not hold a full model, motivating multi-server collaboration [15], [16].

4. LLM Inference Offloading in MEC

Inference offloading determines where each part of an LLM request is executed across the device, edge, and cloud tiers. Figure 2 compares four strategies by where the model layers run and what crosses the wireless link during generation; a shared decision layer selects among them.

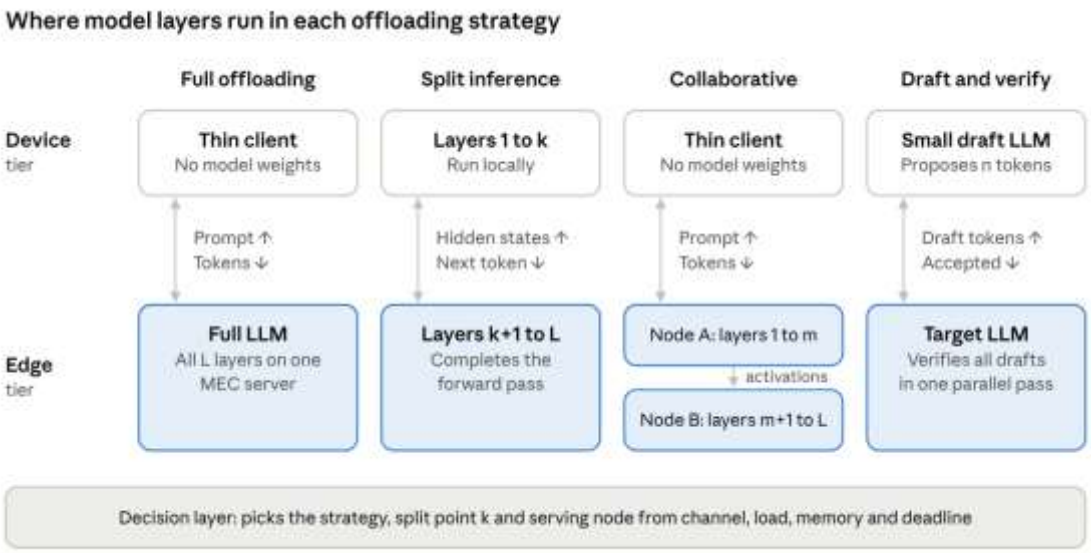


Figure 2. Taxonomy of LLM inference offloading strategies

Figure 2. Placement of model layers between the device and edge tiers in four offloading strategies. L is the number of transformer layers, k and m are split points, and n is the number of draft tokens per round. Labels between tiers show what is transmitted uplink (↑) and downlink (↓).

4.1 Full Offloading

In full offloading, the device transmits the prompt to an edge server that hosts the complete model and streams generated tokens back. This is the simplest approach and is the default assumption in edge-hosted generative AI service architectures [8], [9]. Its advantages are minimal device load and access to larger models. Its drawbacks are dependence on wireless link quality, exposure of raw user prompts to the edge operator, and contention when many users share one server. Research in this family therefore focuses on server selection, request scheduling, and admission control rather than on model partitioning.

4.2 Partial Offloading and Split Inference

Split inference partitions the model at a layer boundary: the device executes the first layers and sends intermediate activations to the edge, which completes the forward pass. Neurosurgeon pioneered automatic layer-level partitioning for DNNs by predicting per-layer latency and energy [10]. Edgent combined partitioning with early exit to meet latency deadlines under changing bandwidth [11]. For LLMs, the uniform width of transformer layers means that activation size does not shrink with depth, unlike convolutional networks. Split points are therefore chosen mainly by device compute and memory capacity, and by privacy considerations, since raw tokens never leave the device [8]. The key challenge is that autoregressive decoding repeats the device–edge exchange for every generated token, so per-hop latency accumulates over long outputs.

4.3 Collaborative and Distributed Inference

When a model exceeds the memory of any single node, it can be sharded across several devices or edge servers. Petals demonstrated that large models can be served collaboratively over the internet by distributing transformer blocks across volunteer machines, with fault tolerance for unreliable nodes [15]. EdgeShard applied a similar idea to collaborative edge computing, jointly optimizing device selection and model partitioning across heterogeneous edge devices and servers to minimize latency or maximize throughput [16]. Pipeline parallelism assigns consecutive layers to different nodes, while tensor parallelism splits individual layers; the latter requires high-bandwidth links that are rarely available between edge nodes, so pipeline approaches dominate in MEC.

4.4 Hybrid Device–Edge–Cloud Routing

Hybrid approaches keep a small model on the device or edge and invoke a larger model only when needed. FrugalGPT showed that cascading queries through models of increasing cost can match the quality of the strongest model at much lower cost [17]. Hybrid LLM trained a router that predicts query difficulty and sends easy queries to a small model and hard queries to a large one [18]. Speculative decoding offers a complementary pattern: a small draft model proposes several tokens that a large target model verifies in parallel, preserving the target model's output distribution while reducing latency [19]. In MEC, the draft model can run on the device and the verifier at the edge, reducing the number of sequential round trips.

4.5 Offloading Decision-Making Methods

Offloading decisions are typically formulated as optimization problems that minimize a weighted sum of latency and energy subject to accuracy, memory, and deadline constraints. Because such problems are usually mixed-integer and non-convex, the literature relies on four classes of solvers: (a) convex relaxation and decomposition methods; (b) heuristics and greedy algorithms for real-time decisions; (c) game-theoretic methods for multi-user competition over shared servers; and (d) deep reinforcement learning (DRL), which learns decision policies online under time-varying channels [20]. DRL-based offloading was shown to achieve near-optimal decisions with computation time low enough for fast-fading channels [20], which makes it attractive for dynamic LLM workloads.

Table 3. Comparison of representative offloading approaches

Work	Strategy	Target model	Decision method	Key idea	Main limitation
Neurosurgeon [10]	Split inference	DNNs	Per-layer performance prediction	Automatic layer-level partitioning between mobile and cloud	Not designed for autoregressive generation
Edgent [11]	Split with early exit	DNNs	Latency-aware optimization	Joint partitioning and early exit under bandwidth changes	Early exit hard to apply to token generation
Petals [15]	Collaborative	LLMs	Client-side selection of a server chain	Transformer blocks distributed across volunteer nodes	Internet-scale latency; trust in peers
EdgeShard [16]	Collaborative	LLMs	Dynamic programming	Joint device selection and model partitioning across edge devices and cloud servers	Assumes stable links between nodes
FrugalGPT [17]	Cascade routing	LLMs	Learned scoring	Query cascades across models of increasing cost	Designed for cloud APIs
Hybrid LLM [18]	Router-based hybrid	LLMs	Learned router	Difficulty-aware routing between small and large models	Router adds overhead and needs training data
Speculative decoding [19]	Draft–verify	LLMs	Algorithmic	Small draft model with parallel verification	Gains depend on draft acceptance rate

Work	Strategy	Target model	Decision method	Key idea	Main limitation
DROO [20]	Binary offloading	Generic tasks	Deep reinforcement learning	Online decisions in wireless-powered MEC	Not LLM-specific

5. Resource Optimization for Edge LLM Inference

Even with effective offloading, edge servers and devices have limited compute, memory, bandwidth, and energy. This section reviews optimization techniques along five dimensions and then presents a joint optimization formulation. Table 4 summarizes the techniques.

5.1 Model Compression

Quantization reduces the bit-width of weights and activations. LLM.int8() showed that 8-bit matrix multiplication can preserve accuracy for large models by isolating outlier feature dimensions and processing them in higher precision [21]. GPTQ introduced one-shot post-training quantization to 3–4 bits using approximate second-order information [22]. AWQ observed that a small fraction of salient weights, identified from activation magnitudes, dominates quantization error, and protected them through per-channel scaling, enabling efficient 4-bit on-device inference [23]. **Knowledge distillation** transfers the behaviour of a large teacher model into a smaller student [24], producing compact models suitable for device-side drafting or routing (Section 4.4). Compression directly reduces the memory needed at the edge and the volume of weights read per decoded token.

5.2 Memory and KV Cache Management

Because decode is memory-bound, efficient memory management often matters more than raw compute. vLLM introduced PagedAttention, which stores the KV cache in non-contiguous fixed-size blocks, analogous to virtual memory paging; this nearly eliminates fragmentation and enables higher batch sizes and throughput [13]. FlexGen showed that a large model can run on a single commodity GPU by offloading weights, activations, and KV cache across GPU, CPU, and disk memory, using a linear-programming search to find an efficient offloading policy [14]. LLM in a Flash stores model parameters in flash storage and loads them on demand into DRAM, using windowing and row–column bundling to minimize data transfer on memory-constrained devices [25]. For MoE models, EdgeMoE keeps non-expert weights in memory and loads only the experts needed, with expert-wise bit-width adaptation and preloading to hide I/O latency [12].

5.3 Computation Scheduling and Batching

Edge servers must multiplex many users on few accelerators. Batching amortizes weight reads across requests and raises throughput, but larger batches increase per-request latency and KV cache pressure [13]. Scheduling policies must therefore balance throughput against time-to-first-token and time-per-output-token targets. The distinct prefill and decode profiles (Table 2) motivate scheduling them separately so that compute-heavy prefills do not stall latency-sensitive decode steps. In MEC, scheduling also interacts with offloading: the scheduler must decide which requests to admit locally, which to forward to neighbouring edge nodes [16], and which to escalate to the cloud [9].

5.4 Communication Optimization

Communication cost dominates when the wireless link is weak or when split inference exchanges activations per token. Techniques include compressing or quantizing intermediate activations before transmission, choosing split points that minimize transferred data [10], [11], and jointly allocating bandwidth and transmit power across users [2]. Speculative decoding reduces the number of sequential device–edge exchanges by verifying several draft tokens in one round [19]. Semantic and task-oriented communication, which transmits task-relevant features rather than raw bits, has been proposed as a 6G enabler for edge LLMs [8].

5.5 Energy Management

For battery-powered devices, the energy cost of local computation must be weighed against the energy of wireless transmission. Classic MEC work models device energy as a function of CPU frequency and transmit power, enabling dynamic voltage and frequency scaling (DVFS) and power control [2], [4]. Wireless-powered MEC further couples energy harvesting with offloading decisions [20]. For LLMs, decoding repeatedly reads model weights from memory, so memory traffic is a major contributor to on-device energy; compression [22], [23] and flash-aware loading [25] therefore also act as energy optimizations. At the server side, the energy footprint of edge accelerators is a growing sustainability concern (Section 7.8).

5.6 Joint Optimization Formulation

Most studies combine these dimensions into a joint problem. For N users, let x_n denote the offloading or split decision of user n, b_n its allocated bandwidth, and f_n its allocated edge compute. A general formulation minimizes a weighted sum of latency and energy:

$$\min_{\{x_n, b_n, f_n\}} \sum_{n=1}^{N} \left[\alpha\, T_n(x_n, b_n, f_n) + (1-\alpha)\, E_n(x_n, b_n)\right]$$

subject to a total bandwidth budget (sum of b_n at most B), a total edge compute budget (sum of f_n at most F), an edge memory constraint covering model weights and all active KV caches, a per-user deadline (T_n at most the deadline of user n), and an accuracy constraint when compressed or smaller models are used. Here T_n combines local computation, transmission, edge queuing, and edge computation time over all generated tokens, and alpha in [0, 1] sets the latency–energy trade-off. Because x_n is discrete and the objective is non-convex, solutions rely on decomposition, heuristics, or DRL (Section 4.5) [20].

Table 4. Summary of resource optimization techniques for edge LLM inference

Dimension	Technique	Representative work	Resource saved	Trade-off
Model	8-bit quantization with outlier handling	LLM.int8() [21]	Memory	Mixed-precision overhead
Model	Post-training 3–4-bit quantization	GPTQ [22]	Memory, bandwidth	Accuracy loss at very low bit-widths
Model	Activation-aware weight quantization	AWQ [23]	Memory, energy	Requires calibration data
Model	Knowledge distillation	Hinton et al. [24]	Compute, memory	Training cost; capability gap
Memory	Paged KV cache management	vLLM [13]	Memory, throughput	Kernel complexity
Memory	GPU–CPU–disk offloading	FlexGen [14]	Memory	Higher latency; suited to batch jobs
Memory	Flash-based parameter loading	LLM in a Flash [25]	DRAM	Depends on storage bandwidth
Memory	Expert-wise loading for MoE	EdgeMoE [12]	Memory	I/O latency when predictions miss
Computation	Collaborative sharding and scheduling	EdgeShard [16]	Per-node compute	Inter-node communication
Communication	Draft–verify decoding	Speculative decoding [19]	Round trips, latency	Wasted work on rejected drafts
Energy	Online offloading with DRL	DROO [20]	Device energy	Training and generalization of policy

6. Enabling Frameworks, Applications, and Evaluation Metrics

6.1 Enabling Frameworks and Systems

Several systems described in the literature provide building blocks for edge LLM deployment. Table 5 groups them by the layer of the stack they address.

Table 5. Enabling frameworks and systems for edge LLM inference

System	Layer	Deployment setting	Main contribution
vLLM [13]	Serving engine	Edge or cloud GPU servers	PagedAttention for high-throughput, memory-efficient serving
FlexGen [14]	Serving engine	Single GPU with CPU and disk	Throughput-oriented offloading across the memory hierarchy
Petals [15]	Distributed runtime	Geo-distributed nodes	Collaborative inference over shared transformer blocks
EdgeShard [16]	Distributed runtime	Edge devices and cloud servers	Latency- or throughput-optimal model partitioning
LLM in a Flash [25]	On-device runtime	Memory-constrained devices	Flash-based on-demand parameter loading
EdgeMoE [12]	On-device runtime	Mobile devices	Memory-efficient MoE inference
GPTQ [22], AWQ [23]	Model preparation	Any	Low-bit quantized model generation

6.2 Representative Applications

Edge-hosted LLMs are expected to support a broad range of mobile services [8], [9]. Table 6 lists representative domains and their dominant requirements.

Table 6. Application domains and their requirements

Domain	Example service	Latency sensitivity	Privacy sensitivity	Suitable offloading strategy
Personal assistants	Voice and chat assistants on smartphones	High	High	Split inference or hybrid routing
Healthcare	Clinical note summarization, patient triage	Medium	Very high	On-device or split inference
Intelligent transportation	In-vehicle assistants, scene description	Very high	Medium	Edge offloading with mobility-aware migration
Industrial IoT	Maintenance assistants, log analysis	Medium	High	Private edge deployment
Extended reality	Real-time conversational avatars	Very high	Medium	Edge offloading with speculative decoding
Smart education	Tutoring and translation	Medium	Medium	Hybrid device–edge routing

6.3 Evaluation Metrics

Studies evaluate edge LLM systems along latency, throughput, resource, energy, and quality dimensions. Using a consistent set of metrics is essential for comparing results across papers (Section 7.7).

Table 7. Common evaluation metrics

Category	Metric	Definition
Latency	Time to first token (TTFT)	Time from request submission to the first generated token
Latency	Time per output token (TPOT)	Average interval between successive generated tokens
Latency	End-to-end latency	Total time to complete a response, including transmission and queuing
Throughput	Tokens per second	Generated tokens per second per server or per user
Resource	Peak memory	Maximum memory used by weights, activations, and KV cache
Communication	Data transmitted	Bytes exchanged between device, edge, and cloud per request
Energy	Energy per token	Device or server energy consumed per generated token
Quality	Perplexity, task accuracy	Model quality after compression, splitting, or routing
Service	SLO attainment	Share of requests meeting their latency targets

7. Open Research Challenges and Future Directions

Despite rapid progress, edge LLM inference remains an immature field. Table 8 summarizes eight open challenges, each discussed below with a suggested research direction.

7.1 Dynamic Offloading under Wireless and Workload Uncertainty

Existing offloading methods largely assume stable bandwidth and known request lengths. In practice, wireless channels fluctuate, and output length is unknown until generation ends, so the total cost of an offloading decision cannot be predicted in advance. Future work should develop token-level adaptive offloading that can revise the split point or serving node during generation, and output-length prediction to support proactive scheduling. DRL [20] is promising but must generalize across models and traffic patterns.

7.2 Mobility and KV Cache Migration

The KV cache binds a request to the server that holds it. When a user moves between cells, the system must migrate the cache, recompute it, or tolerate a longer path to the original server. Cache sizes grow with context length (Section 3.3), so naive migration can be costly. Research is needed on mobility-aware placement, incremental cache transfer, and cache compression tailored to handover.

7.3 Privacy and Security in Split and Collaborative Inference

Split inference keeps raw tokens on the device, but intermediate activations may still leak information about the input. Collaborative inference over third-party nodes, as in Petals [15], raises trust and integrity concerns. Future work should quantify leakage from transformer activations and develop lightweight defences, such as activation perturbation, secure aggregation, and trusted execution environments, that fit edge latency budgets.

7.4 Heterogeneous Hardware and Model Portability

Edge nodes combine GPUs, NPUs, and CPUs of different generations, and devices vary widely in memory and accelerator support. Techniques such as quantization [22], [23] and collaborative sharding [16] are often evaluated on uniform hardware. Hardware-aware partitioning, compilers that target diverse edge accelerators, and portable model formats remain open problems.

7.5 Efficient Mixture-of-Experts and Long-Context Inference at the Edge

MoE models reduce computation per token but keep total memory high [12]. Distributing experts across edge nodes introduces routing-dependent communication. Long-context inference similarly amplifies KV cache pressure [13].

Promising directions include expert placement across edge servers, expert caching based on predicted routing, and KV cache eviction or compression policies evaluated on edge workloads.

7.6 Joint Communication–Computation–Model Co-Design for 6G

Most studies optimize model compression, offloading, and radio resource allocation separately. 6G visions call for integrated designs in which the network itself is LLM-aware [8], [9]. Research directions include semantic communication of intermediate representations, cross-layer scheduling that exposes model state to the radio scheduler, and joint optimization frameworks that extend the formulation in Section 5.6 with realistic token-level latency models.

7.7 Standardized Benchmarks and Reproducible Evaluation

Papers differ in models, hardware, network settings, and metrics, which makes fair comparison difficult. The field needs open benchmarks that combine representative LLM workloads, realistic wireless traces, and heterogeneous edge hardware, together with agreed metrics such as those in Table 7. Reproducible simulators that model both network and inference behaviour would accelerate progress.

7.8 Energy Efficiency and Sustainable Edge AI

Dense deployment of edge accelerators increases the energy footprint of mobile networks. Device batteries also limit how much local inference is practical. Future research should consider carbon-aware scheduling, energy-proportional edge hardware, and joint optimization of accuracy, latency, and energy under renewable-energy constraints, building on energy-aware offloading in MEC [2], [20].

Table 8. Summary of open challenges and future directions

Challenge	Why it matters	Future direction
Dynamic offloading	Channel and output length are unknown in advance	Token-level adaptive offloading; output-length prediction
Mobility and KV cache	Cache ties requests to one server	Mobility-aware placement; incremental cache migration
Privacy and security	Activations can leak inputs; untrusted nodes	Leakage analysis; lightweight defences; trusted execution
Hardware heterogeneity	Diverse accelerators and memory sizes	Hardware-aware partitioning; portable compilers
MoE and long context	Large memory and routing-dependent traffic	Expert placement and caching; KV cache compression
Cross-layer co-design	Siloed optimization of model and network	LLM-aware 6G scheduling; semantic communication
Benchmarking	Results are hard to compare	Open benchmarks with wireless traces and common metrics
Sustainability	Growing energy footprint	Carbon-aware scheduling; energy-proportional hardware

8. Conclusion

This review examined how mobile edge computing can make large language model inference practical for mobile users. We showed that LLM inference differs from conventional DNN offloading because of its autoregressive decoding, stateful KV cache, and extreme model size [5], [13]. We organized offloading strategies into full offloading, split inference, collaborative inference, and hybrid draft–verify or routing schemes [8]–[11], [15]–[20], and reviewed resource optimization across model compression [21]–[24], memory management [12]–[14], [25], computation scheduling, communication, and energy. We also summarized enabling systems, applications, and evaluation metrics.

The analysis suggests that no single technique is sufficient. Efficient edge LLM services will combine compressed models, memory-efficient serving, adaptive offloading, and network-aware scheduling within a joint optimization framework. Key open problems include token-level adaptive offloading, KV cache mobility, privacy of intermediate activations, hardware heterogeneity, MoE deployment, cross-layer 6G co-design, standardized benchmarking, and sustainability. Addressing these challenges will be central to realizing the vision of edge-native generative AI in future mobile networks [8], [9].

References

References follow APA 7th edition format and are numbered in order of first citation to match the in-text markers.

[1] Satyanarayanan, M. (2017). The emergence of edge computing. *Computer, 50*(1), 30–39. https://doi.org/10.1109/MC.2017.9

[2] Mao, Y., You, C., Zhang, J., Huang, K., & Letaief, K. B. (2017). A survey on mobile edge computing: The communication perspective. *IEEE Communications Surveys & Tutorials, 19*(4), 2322–2358. https://doi.org/10.1109/COMST.2017.2745201

[3] Shi, W., Cao, J., Zhang, Q., Li, Y., & Xu, L. (2016). Edge computing: Vision and challenges. *IEEE Internet of Things Journal, 3*(5), 637–646. https://doi.org/10.1109/JIOT.2016.2579198

[4] Mach, P., & Becvar, Z. (2017). Mobile edge computing: A survey on architecture and computation offloading. *IEEE Communications Surveys & Tutorials, 19*(3), 1628–1656. https://doi.org/10.1109/COMST.2017.2682318

[5] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems* (Vol. 30). Curran Associates.

[6] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., . . . Amodei, D. (2020). Language models are few-shot learners. In *Advances in Neural Information Processing Systems* (Vol. 33, pp. 1877–1901). Curran Associates.

[7] Zhou, Z., Chen, X., Li, E., Zeng, L., Luo, K., & Zhang, J. (2019). Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proceedings of the IEEE, 107*(8), 1738–1762. https://doi.org/10.1109/JPROC.2019.2918951

[8] Lin, Z., Qu, G., Chen, Q., Chen, X., Chen, Z., & Huang, K. (2025). Pushing large language models to the 6G edge: Vision, challenges, and opportunities. *IEEE Communications Magazine, 63*(9), 52–59.

[9] Xu, M., Du, H., Niyato, D., Kang, J., Xiong, Z., Mao, S., Han, Z., Jamalipour, A., Kim, D. I., Shen, X., Leung, V. C. M., & Poor, H. V. (2024). Unleashing the power of edge-cloud generative AI in mobile networks: A survey of AIGC services. *IEEE Communications Surveys & Tutorials, 26*(2), 1127–1170. https://doi.org/10.1109/COMST.2024.3353265

[10] Kang, Y., Hauswald, J., Gao, C., Rovinski, A., Mudge, T., Mars, J., & Tang, L. (2017). Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. In *Proceedings of the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems* (pp. 615–629). ACM. https://doi.org/10.1145/3037697.3037698

[11] Li, E., Zeng, L., Zhou, Z., & Chen, X. (2020). Edge AI: On-demand accelerating deep neural network inference via edge computing. *IEEE Transactions on Wireless Communications, 19*(1), 447–457. https://doi.org/10.1109/TWC.2019.2946140

[12] Yi, R., Guo, L., Wei, S., Zhou, A., Wang, S., & Xu, M. (2023). *EdgeMoE: Fast on-device inference of MoE-based large language models* (arXiv:2308.14352). arXiv. https://doi.org/10.48550/arXiv.2308.14352

[13] Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., & Stoica, I. (2023). Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles* (pp. 611–626). ACM. https://doi.org/10.1145/3600006.3613165

[14] Sheng, Y., Zheng, L., Yuan, B., Li, Z., Ryabinin, M., Chen, B., Liang, P., Ré, C., Stoica, I., & Zhang, C. (2023). FlexGen: High-throughput generative inference of large language models with a single GPU. In *Proceedings of the 40th International Conference on Machine Learning* (Vol. 202, pp. 31094–31116). PMLR.

[15] Borzunov, A., Baranchuk, D., Dettmers, T., Ryabinin, M., Belkada, Y., Chumachenko, A., Samygin, P., & Raffel, C. (2023). Petals: Collaborative inference and fine-tuning of large models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)* (pp. 558–568). Association for Computational Linguistics. https://doi.org/10.18653/v1/2023.acl-demo.54

[16] Zhang, M., Shen, X., Cao, J., Cui, Z., & Jiang, S. (2025). EdgeShard: Efficient LLM inference via collaborative edge computing. *IEEE Internet of Things Journal, 12*(10), 13119–13131. https://doi.org/10.1109/JIOT.2024.3524255

[17] Chen, L., Zaharia, M., & Zou, J. (2024). FrugalGPT: How to use large language models while reducing cost and improving performance. *Transactions on Machine Learning Research*.

[18] Ding, D., Mallick, A., Wang, C., Sim, R., Mukherjee, S., Rühle, V., Lakshmanan, L. V. S., & Awadallah, A. H. (2024). Hybrid LLM: Cost-efficient and quality-aware query routing. In *Proceedings of the Twelfth International Conference on Learning Representations (ICLR 2024)*.

[19] Leviathan, Y., Kalman, M., & Matias, Y. (2023). Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning* (Vol. 202, pp. 19274–19286). PMLR.

[20] Huang, L., Bi, S., & Zhang, Y.-J. A. (2020). Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks. *IEEE Transactions on Mobile Computing, 19*(11), 2581–2593. https://doi.org/10.1109/TMC.2019.2928811

[21] Dettmers, T., Lewis, M., Belkada, Y., & Zettlemoyer, L. (2022). LLM.int8(): 8-bit matrix multiplication for transformers at scale. In *Advances in Neural Information Processing Systems* (Vol. 35). Curran Associates.

[22] Frantar, E., Ashkboos, S., Hoefler, T., & Alistarh, D. (2023). GPTQ: Accurate post-training quantization for generative pre-trained transformers. In *Proceedings of the Eleventh International Conference on Learning Representations (ICLR 2023)*.

[23] Lin, J., Tang, J., Tang, H., Yang, S., Chen, W.-M., Wang, W.-C., Xiao, G., Dang, X., Gan, C., & Han, S. (2024). AWQ: Activation-aware weight quantization for on-device LLM compression and acceleration. In *Proceedings of Machine Learning and Systems* (Vol. 6).

[24] Hinton, G., Vinyals, O., & Dean, J. (2015). *Distilling the knowledge in a neural network* (arXiv:1503.02531). arXiv. https://doi.org/10.48550/arXiv.1503.02531

[25] Alizadeh, K., Mirzadeh, S. I., Belenko, D., Khatamifard, S. K., Cho, M., Del Mundo, C. C., Rastegari, M., & Farajtabar, M. (2024). LLM in a flash: Efficient large language model inference with limited memory. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics.