

Mobile Edge Computing: A Comprehensive Review of LLM Inference Offloading, Resource Optimization, and Open Research Challenges

B. RAMESH KUMAR

Research Scholar & Assistant Professor, Department of Computer Science with Data Analytics

KPR College of Arts Science and Research, Coimbatore

Email : 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 become a major foundation for intelligent applications, but their high computational, memory, and energy requirements make centralized cloud inference costly and unsuitable for many latency-sensitive mobile scenarios. Mobile Edge Computing (MEC) provides an intermediate computing layer by placing storage and processing resources close to users, creating an opportunity to execute, partition, cache, or selectively offload LLM inference tasks. This review examines the convergence of MEC and LLM inference with particular emphasis on computation offloading, model partitioning, resource allocation, model compression, caching, scheduling, and adaptive orchestration. The paper first establishes an architectural view of device–edge–cloud LLM inference and then analyzes local execution, full offloading, partial offloading, and collaborative multi-device inference. Resource optimization is discussed across latency, energy, bandwidth, memory, GPU/NPU utilization, model quality, and privacy. Recent techniques including quantization, pruning, knowledge distillation, speculative decoding, KV-cache optimization, model caching, and learning-based offloading are reviewed as complementary mechanisms. The review further identifies open challenges involving mobility, heterogeneous hardware, dynamic network conditions, multi-user contention, privacy, hallucination-aware quality of service, reproducible benchmarks, and sustainable edge intelligence. Finally, a research agenda is proposed for jointly optimizing model, network, and hardware resources rather than treating inference and communication as independent problems.

Keywords: Mobile Edge Computing, Large Language Models, Edge Intelligence, Inference Offloading, Resource Optimization, Model Partitioning, Quantization, 5G/6G, Edge AI, Speculative Decoding

1. Introduction

The rapid evolution of generative artificial intelligence has moved language models from research laboratories into mobile assistants, enterprise applications, healthcare interfaces, education platforms, industrial systems, and Internet-of-Things environments. LLMs can perform language understanding, generation, summarization, reasoning, retrieval assistance, coding, and multimodal interaction. However, the same capabilities that make these models useful also create substantial inference demands. Transformer-based models require memory for model parameters, intermediate activations, and key-value (KV) caches, while autoregressive generation requires repeated computation for every output token.

Cloud computing has traditionally addressed these requirements by providing large GPU clusters. Although cloud inference offers abundant compute capacity, it introduces network round-trip delay, bandwidth consumption, service cost, privacy concerns, and dependence on connectivity. These limitations are particularly important for mobile and IoT applications that require rapid responses or operate in bandwidth-constrained environments. MEC addresses this gap by moving computing closer to end users. Recent surveys describe task offloading as a central MEC mechanism for reducing delay and energy consumption, while emerging LLM surveys emphasize that edge inference requires coordinated model compression, runtime optimization, and offloading strategies. □cite□turn0search1□turn0search2□

The central research question is therefore not simply whether an LLM can run at the edge, but how inference should be dynamically divided among the device, nearby edge servers, and the cloud. A suitable policy must consider model size, input length, output length, device energy, memory availability, wireless bandwidth, queueing delay, edge-server load, user mobility, and application-level quality requirements. This review synthesizes these dimensions and frames LLM inference as a joint communication–computation–model optimization problem.

2. Background and Conceptual Foundations

2.1 Mobile Edge Computing

MEC extends distributed computing by placing compute and storage resources at or near the network edge. Instead of transmitting every computational task to a distant cloud, mobile devices can communicate with nearby edge servers through 5G, 6G, or Wi-Fi networks. Traditional MEC research considers offloading decisions, server selection, radio resource allocation, computation scheduling, and energy management. Surveys classify offloading objectives around latency minimization, energy reduction, high-computing workloads, and joint delay–energy optimization.

2.2 LLM Inference Characteristics

LLM inference is dominated by two phases: prompt processing (prefill) and autoregressive token generation (decode). Prefill processes many input tokens in parallel and can be compute-intensive, while decode repeatedly generates one or a small number of tokens and can become sensitive to memory bandwidth and KV-cache access. The inference footprint is influenced by parameter count, numerical precision, context length, batch size, and cache size. These characteristics make LLMs different from conventional independent MEC tasks because inference has sequential dependencies and persistent state.

2.3 Edge LLMs

Recent surveys on edge LLMs identify model compression and runtime optimization as key approaches for reducing the mismatch between LLM requirements and edge-device resources. Quantization, pruning, knowledge distillation, efficient decoding, and model/runtime co-design can reduce storage, computation, or energy requirements. Recent work also highlights speculative decoding and model offloading as two important directions for efficient edge inference.

3. MEC Architecture for LLM Inference

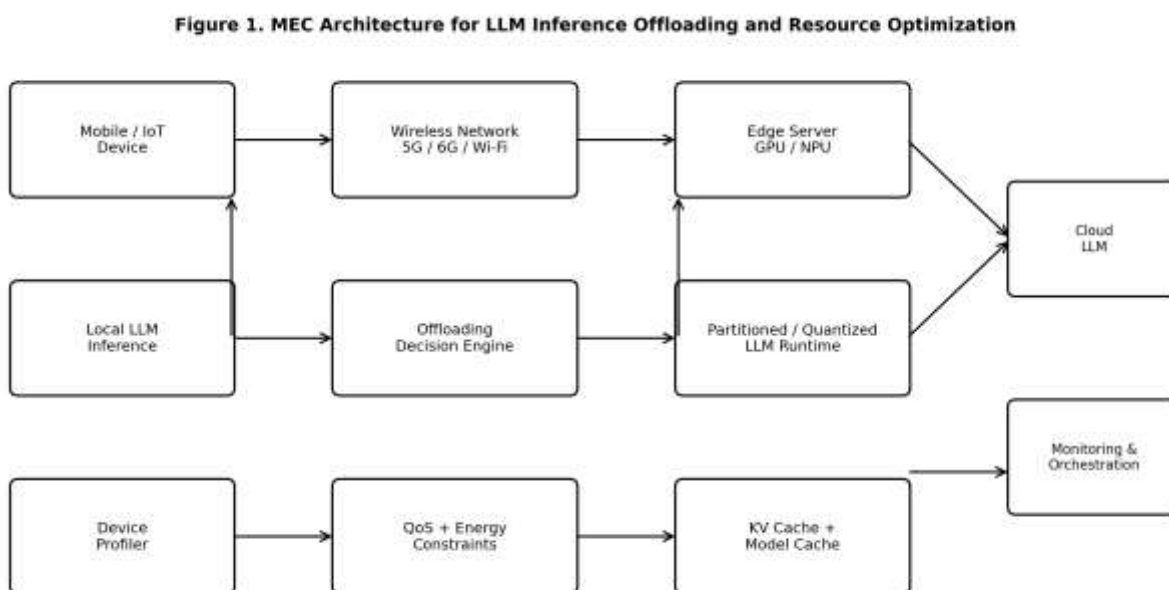


Figure 1. MEC architecture for LLM inference offloading and resource optimization.

Figure 1 illustrates a three-tier architecture comprising mobile devices, edge servers, and cloud infrastructure. The device performs profiling and may execute a compact LLM locally. The edge layer provides stronger GPU/NPU resources and can host shared model replicas, KV caches, and partitioned model components. The cloud acts as a

high-capacity fallback for large or complex requests. An orchestration layer selects execution locations and allocates resources based on current network, device, and server conditions.

3.1 Local Inference

In local inference, the entire model or a compact variant is stored and executed on the mobile device. This approach minimizes network dependency and improves data locality, but memory capacity, thermal constraints, battery consumption, and accelerator availability can limit sustained performance. On-device execution is attractive for privacy-sensitive applications because raw prompts may remain on the device.

3.2 Full Offloading

Full offloading transfers the inference request to an edge server or cloud service. It is simple to manage and can support larger models, but latency includes uplink transmission, edge queueing, inference, and response transfer. When network conditions deteriorate, offloading can become slower than local execution.

3.3 Partial Offloading and Model Partitioning

Partial offloading divides model execution between the device and edge. Layers, blocks, or selected operations can be executed locally while the remaining computation is executed remotely. Partitioning introduces additional communication overhead at the split point but may reduce device memory and compute requirements. Mobility-aware partitioning has recently been studied because user movement changes wireless conditions and the suitability of edge nodes. [\[cite\turn0search15\]](#)

3.4 Collaborative and Multi-Device Inference

Multiple edge devices can cooperate with edge servers or with each other. Collaborative inference may improve resource utilization, but it creates synchronization, privacy, scheduling, and communication challenges. A practical system needs mechanisms for selecting trustworthy peers, managing heterogeneous accelerators, and preventing one overloaded device from becoming a bottleneck.

4. LLM Inference Offloading Strategies

Offloading can be formulated as a decision problem in which an orchestrator chooses where and how much computation should be executed. Let x_d , x_e , and x_c represent local, edge, and cloud execution choices. A simplified objective is:

$$\min J = \alpha T + \beta E + \gamma B + \delta C + \epsilon R$$

where T is end-to-end latency, E is device energy consumption, B is bandwidth usage, C is monetary or infrastructure cost, and R represents a risk or quality penalty. The coefficients α – ϵ reflect application priorities. Constraints may include device battery limits, memory capacity, edge GPU availability, bandwidth limits, model quality, privacy requirements, and maximum response latency.

4.1 Latency-Aware Offloading

Latency-aware policies estimate communication time, queueing delay, and inference time before selecting an execution location. They are suitable for conversational assistants, augmented reality, industrial control, and interactive applications. A major limitation is that inference latency is not constant: it varies with prompt length, output length, concurrent requests, accelerator utilization, and KV-cache behavior.

4.2 Energy-Aware Offloading

Energy-aware strategies seek to reduce battery drain by transferring compute-intensive work to edge servers. However, communication itself consumes energy. Therefore, the correct decision depends on the relative energy costs of local computation and wireless transmission. Thermal throttling further complicates this trade-off because sustained local inference can reduce device frequency and increase response time.

4.3 Learning-Based Offloading

Machine learning and reinforcement learning can learn offloading policies under uncertain network and workload conditions. The state can include channel quality, queue length, battery level, memory pressure, model size, prompt length, and mobility. The action can specify server selection, model partition, precision, and resource allocation. The reward can combine latency, energy, cost, and quality. Earlier MEC surveys have identified reinforcement learning as an important approach for dynamic offloading, while newer LLM work extends optimization to model-specific constraints.

5. Resource Optimization Techniques

5.1 Model Quantization

Quantization reduces the numerical precision of model weights and, where supported, activations. Four-bit and eight-bit representations can substantially reduce model storage and memory bandwidth requirements. Quantization is especially useful for edge deployment because memory capacity is often the first limiting resource. Nevertheless, aggressive quantization can reduce accuracy or increase sensitivity for particular tasks, requiring calibration and model-specific evaluation.

5.2 Pruning and Knowledge Distillation

Pruning removes redundant parameters or structures, while knowledge distillation trains a smaller student model to reproduce the behavior of a larger teacher model. Structured pruning is attractive for edge devices because removing complete structures may map more effectively to hardware acceleration than unstructured sparsity. Distillation can create compact models suitable for local execution while preserving task performance.

5.3 Speculative Decoding

Speculative decoding uses a smaller draft model to propose multiple tokens and a larger target model to verify them. When proposals are accepted, several tokens can be generated with fewer expensive target-model steps. This can improve generation speed without fundamentally changing the target model. Recent edge-inference surveys identify speculative decoding as a major direction for reducing inference latency.

5.4 KV-Cache Optimization

The KV cache stores attention keys and values from previous tokens. Long-context applications can make the cache a major memory consumer. Compression, quantization, eviction, paging, sharing, and hierarchical caching can reduce memory pressure. In MEC, KV-cache placement is also a scheduling problem because moving or recreating state across edge nodes can add latency.

5.5 Model and Data Caching

Popular LLMs and frequently used prompt components can be cached at edge servers. Model caching avoids repeated downloads from the cloud, while prompt or semantic caching can reduce redundant inference for repeated requests. Cache replacement should consider model popularity, model size, loading time, tenant priority, and expected future demand.

6. Joint Network, Compute, and Model Optimization

An important limitation of conventional MEC optimization is that communication and computation are often modeled separately. LLM inference requires a more integrated formulation because changing model precision changes memory traffic and inference time; changing the partition point changes communication volume; and changing the server changes both compute capacity and queueing delay. Consequently, a good policy should jointly optimize model configuration, partitioning, wireless resources, accelerator resources, and cache state.

For example, consider three candidate actions: local 4-bit inference, edge 8-bit inference, and split inference with a 4-bit local prefix and an edge suffix. The first minimizes communication but may increase device energy. The second increases transmission but offers higher compute capacity. The third may achieve a balance but introduces split-point communication and synchronization. A dynamic controller can select among these actions using real-time profiling.

6.1 Resource Allocation

At the edge server, GPU/NPU cycles, memory, batch slots, and network bandwidth must be allocated among users. Static allocation can lead to underutilization during low demand and queue buildup during peaks. Dynamic batching can improve accelerator utilization but may increase waiting time for interactive requests. Priority-aware scheduling is therefore necessary for applications with heterogeneous service-level objectives.

6.2 Mobility-Aware Optimization

Mobile users can move between cells and edge domains while a conversation is in progress. If the user's serving edge changes, the system may need to migrate model state, KV cache, or session context. Mobility-aware policies should predict handovers and proactively replicate state. Recent research explicitly studies mobility-aware model partitioning and offloading for edge LLM inference.

7. Security, Privacy, and Reliability

Edge LLM systems distribute sensitive prompts, model parameters, intermediate representations, and cached context across devices and servers. This creates attack surfaces beyond conventional MEC. Threats include eavesdropping,

malicious edge nodes, model extraction, prompt injection, cache leakage, unauthorized model access, and compromised orchestration policies.

Privacy-preserving approaches can include encryption in transit and at rest, trusted execution environments, access control, federated learning for selected personalization tasks, and differential privacy where appropriate. However, encryption can increase computation overhead, and trusted hardware is not uniformly available across mobile devices. Reliability is equally important: the system should degrade gracefully when connectivity is lost by falling back to a compact local model or cached response strategy.

8. Evaluation Metrics and Experimental Methodology

A comprehensive evaluation should report both system-level and model-level metrics. System metrics include end-to-end latency, time-to-first-token (TTFT), tokens per second, energy per request, bandwidth consumption, GPU utilization, memory footprint, and thermal behavior. Model metrics include task accuracy, perplexity where appropriate, response quality, hallucination rate, and instruction-following performance. User-level metrics may include perceived responsiveness and service availability.

Metric	What it measures	Importance
TTFT	Time until first generated token	Critical for interactive applications
Tokens/s	Generation throughput	Measures decoding efficiency
Energy/request	Battery or device energy cost	Important for mobile sustainability
Memory	RAM/VRAM/KV-cache footprint	Determines deployability
Bandwidth	Communication volume	Important under constrained networks
Quality	Accuracy / task success / hallucination	Prevents efficiency from degrading usefulness
Availability	Successful service rate	Measures robustness under mobility and failures

Experiments should include multiple device classes, edge accelerators, network conditions, prompt lengths, output lengths, concurrency levels, and model sizes. Reproducibility requires reporting model versions, quantization settings, hardware, software runtime, network emulation configuration, workload traces, and random seeds. A useful benchmark should compare local, edge-only, cloud-only, and adaptive strategies under identical workloads.

9. Open Research Challenges

9.1 Dynamic and Predictive Offloading

Future systems should predict network quality, user mobility, workload bursts, and edge congestion rather than reacting after performance deteriorates. Prediction can reduce unnecessary migrations and improve proactive resource allocation.

9.2 Quality-of-Service-Aware Optimization

Latency alone is insufficient. An offloading controller should explicitly consider accuracy, hallucination risk, context length, safety constraints, and application-specific quality. Recent work demonstrates the value of combining latency with accuracy and hallucination constraints in MEC LLM optimization.

9.3 Heterogeneous Hardware

Mobile CPUs, GPUs, NPUs, edge GPUs, and specialized accelerators have different performance and memory characteristics. Portable runtimes and hardware-aware schedulers are required to exploit this diversity.

9.4 Long-Context and KV-Cache Management

As context windows increase, cache memory and state migration become major bottlenecks. Research is needed on cache compression, selective retention, cross-edge migration, and privacy-preserving cache sharing.

9.5 Multi-Tenant Edge LLM Services

Edge servers may host multiple models and serve users with different priorities. Efficient admission control, isolation, batching, caching, and model placement are open problems.

9.6 Security and Trust

Distributed LLM inference requires protection against malicious clients, compromised edge servers, prompt attacks, model theft, and cache leakage. Security mechanisms must be designed without eliminating the latency benefits of edge computing.

9.7 Sustainable Edge Intelligence

Energy consumption should be measured across device, network, and edge infrastructure. Future optimization should include carbon-aware scheduling and renewable-energy availability.

9.8 Standardized Benchmarks

The field needs public benchmarks combining real mobile traces, realistic network conditions, standardized model workloads, hardware diversity, and quality metrics. Without such benchmarks, comparisons between offloading methods remain difficult.

10. Future Research Directions

Future MEC–LLM systems are likely to move toward adaptive hierarchical inference. Instead of selecting a fixed execution location, a controller can continuously select model size, precision, partition point, edge server, cache policy, and decoding method. A hierarchical architecture can use a small local model for immediate responses, an edge model for moderate complexity, and a cloud model for difficult reasoning tasks.

Another promising direction is world-model or model-based reinforcement learning for offloading. Such methods can learn a compact representation of network and workload dynamics and evaluate candidate actions before deploying them. Recent work has explored world-model-assisted PPO for LLM offloading, reporting improvements in convergence and latency in simulated MEC settings.

Hardware–software co-design is also essential. Future mobile NPUs and edge accelerators should expose primitives optimized for transformer attention, quantized matrix multiplication, KV-cache management, and speculative decoding. Software runtimes should then exploit these capabilities through automatic graph partitioning and scheduling. Cross-platform runtimes that abstract hardware-specific acceleration are emerging as an important deployment direction. □cite□turn0search8□

11. Conclusion

Mobile Edge Computing offers a practical foundation for bringing LLM intelligence closer to users, but efficient deployment requires more than simply moving inference from the cloud to an edge server. LLM inference is characterized by large model footprints, autoregressive dependencies, dynamic KV-cache requirements, heterogeneous hardware, and sensitivity to network conditions. Consequently, inference location, model configuration, communication resources, and compute resources should be optimized jointly.

This review has examined local inference, full and partial offloading, model partitioning, collaborative execution, quantization, pruning, distillation, speculative decoding, KV-cache optimization, caching, learning-based orchestration, and mobility-aware resource management. The most important future opportunity is the development of adaptive systems that treat the model, device, network, and edge infrastructure as one optimization space. Such systems can provide low latency and energy consumption while preserving model quality, privacy, and reliability. As 5G-Advanced and 6G networks mature, MEC-based LLM inference is positioned to become a key enabling technology for responsive, privacy-aware, and resource-efficient edge intelligence.

References

1. Chen, Z., Zhu, B., Wang, J., Shin, H., Nallanathan, A., & Niyato, D. (2026). Network Edge Inference for Large Language Models: Principles, Techniques, and Opportunities. *ACM Computing Surveys*, 58(12). DOI: 10.1145/3809166.
2. Cai, G., Tian, R., Yang, L., Jia, Y., Li, L., & Wang, J. (2026). Efficient Inference for Edge Large Language Models: A Survey. *Tsinghua Science and Technology*, 31(3), 1365–1380. DOI: 10.26599/TST.2025.9010166.
3. Wang, R., Gao, Z., Zhang, L., Yue, S., & Gao, Z. (2025). Empowering large language models to edge intelligence: A survey of edge efficient LLMs and techniques. *Computer Science Review*, 57, 100755. DOI: 10.1016/j.cosrev.2025.100755.
4. Dong, S., Tang, J., Abbas, K., Hou, R., Kamruzzaman, J., Rutkowski, L., & Buyya, R. (2024). Task offloading strategies for mobile edge computing: A survey. *Computer Networks*, 253, 110791. DOI: 10.1016/j.comnet.2024.110791.

5. Shakarami, A., Ghobaei-Arani, M., & Shahidinejad, A. (2020). A survey on the computation offloading approaches in mobile edge computing: A machine learning-based perspective. *Computer Networks*, 182, 107496. DOI: 10.1016/j.comnet.2020.107496.
6. Qu, G., Chen, Q., Wei, W., et al. (2025). Mobile Edge Intelligence for Large Language Models: A Contemporary Survey. *IEEE Communications Surveys & Tutorials*. DOI: 10.36227/techrxiv.172115025.57884352.
7. Zhou, X., & Liu, J. (2025). DAPO: Mobility-Aware Joint Optimization of Model Partitioning and Task Offloading for Edge LLM Inference. *Electronics*, 14(19), 3929. DOI: 10.3390/electronics14193929.
8. Zhang, R., Luo, X., He, J., Niyato, D., Kang, J., Xiong, Z., & Li, Y. (2026). Compact LLM Deployment and World Model Assisted Offloading in Mobile Edge Computing. *arXiv:2602.13628*.
9. Yang, W., & Xia, S. (2026). PELM: Power Efficient On-Device LLM Inference with Speculative Decoding and Dynamic Voltage Frequency Scaling. *SenSys 2026*, 438–451. DOI: 10.1145/3774906.3802783.