

Research on Lightweight Deployment and Reasoning Acceleration of Large Language Models in Resource Constrained Environments

Yiling Yuan

Information Networking Institute, Carnegie Mellon University, Pittsburgh, 15213, Pennsylvania, United States

Keywords: Large language model; resource-constrained environment; model quantization; key-value caching; inference acceleration; edge deployment

Abstract: As large language models are increasingly applied in various places, such as centralised cloud services, terminals, edge servers and private industry networks, issues such as model size, GPU memory capacity, memory bandwidth, first-word latency and energy consumption constraints have become progressively more serious. This paper studies the light-weight deployment and inference acceleration of large language models in resource-constrained environments. It summarizes the research progress in the past three years for low-bit quantization, lightweight model design, key-value cache compression, attention operator optimisation and parallel decoding, and builds a deployment framework of "model compression - cache governance - operator fusion - scheduling coordination". Therefore, it has been concluded that a single compression strategy is unable to meet all the demands for accuracy, low latency and energy efficiency. For practical implementation at the edge, a verifiable, rollbackable and portable inference optimisation process needs to be established, and the joint objectives of task quality degradation, peak memory usage, throughput and energy consumption should be optimised.

1 Introduction

Large Language Models are now general-purpose platforms supporting many functions of artificial intelligence, such as question-answering, code generation, knowledge retrieval, equipment maintenance and multimodal interaction. The inference stage does not need to be based on high computing power; rather, it is also a complex system engineering issue that includes factors such as the scale of parameters, memory bandwidth, growth of the key-value cache, context length, and seriality of autoregressive decoding. Large cloud clusters can extend memory, tensor parallelism and batch-processing scheduling to reduce the effect of model overhead. In resource-constrained environments, such as mobile terminals, embedded gateways, industrial edge servers, private network all-in-one machines and offline security devices, the typical models have limited memory, low-power processors and unstable cooling conditions. Any extra weight reading, cache movement or scheduling waits will increase the perceived latency for users directly.

Recently, low-bit weight quantization methods for edge-side operation have been developing rapidly. AWQ identifies and protects a small number of important weight channels through activation-aware methods, enabling 4-bit weight quantization to have a small accuracy loss in multi-language and multi-modal tasks, and is combined with TinyChat for efficient inference on desktop and mobile GPUs [1]. MobileLLM is also a focus for models with less than one billion parameters; therefore, in mobile applications, the goal is not to build very large models but rather to enhance the efficiency per unit of parameter through deeply and narrowly structured networks, embedding sharing, and grouped query attention [2]. Based on the above research, to address the problem of resource-constrained deployment, it is necessary to improve the collaboration among model structure, numerical representation, runtime memory management and hardware execution paths.

The research content of this paper is to build a system that can deploy and run large language models effectively in resource-constrained environments, and then conduct experiments and analysis. The sections of this paper are as follows: Introduction, Analysis of the current state of research, Problem statement, Solutions, and Conclusion. First, what bottlenecks in the existing lightweighting and acceleration technologies have been addressed? Second, what prominent coupling contradictions remain in the edge deployment? Thirdly, how to develop a feasible deployment plan, evaluation indicators and risk control mechanism for mission-critical components without affecting their availability?

2. Current Status Analysis of the Research Topic

2.1 Lightweight Model and Evolution of Edge Usability

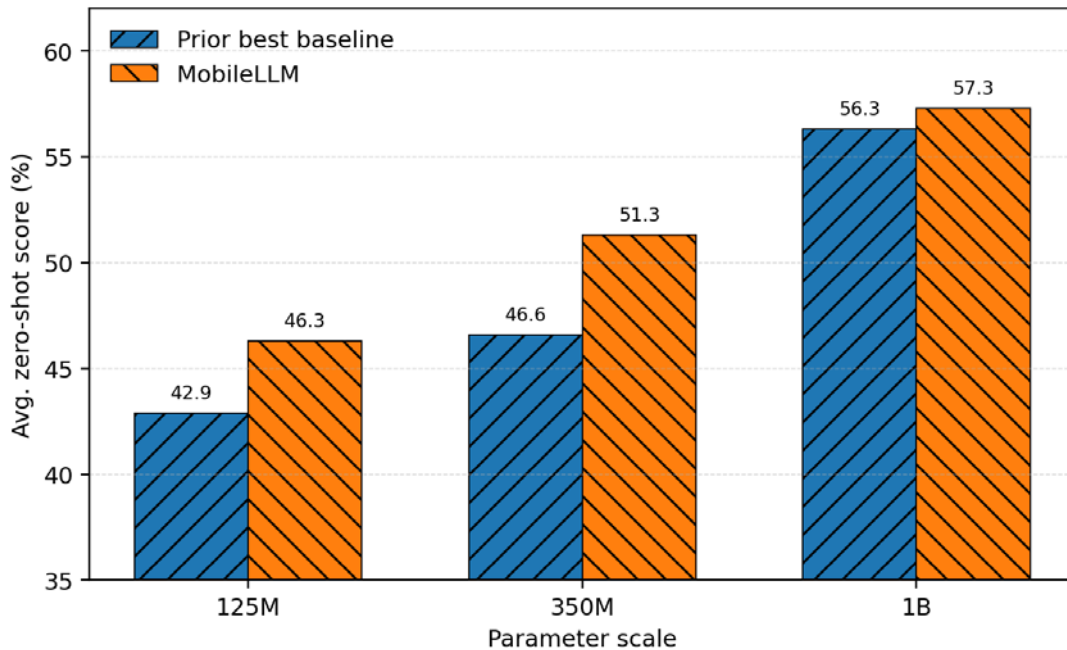


Figure 1. Comparison of Average Zero-shot Commonsense Reasoning Scores for MobileLLM at Different Parameter Scales.

Lightweight deployment of large language models is motivated by reduced model size and shorter context length. On the one hand, the general-purpose basic model will continue to improve its reasoning ability, context length and modal capacity; at the same time, industrial terminals require a low-latency, low-cost, locally private model. Large-scale dense Transformer models are

not directly suitable for edge reasoning; instead, they serve as an upper bound for task distillation, model pruning and small-model routing. MobileLLM is designed to optimise the structure of models with fewer than a billion parameters for mobile devices. By reducing the size of the parameters and enhancing the structural efficiency, small models can outperform well-known baselines of the same scale in commonsense reasoning average score [2].

Figure 1 Analysis: The data in the figure are from the publicly available experimental results of MobileLLM. The average zero-sample commonsense reasoning score is shown on the vertical axis, and the size of the parameters is on the horizontal axis. It can be seen from the above that MobileLLM outperforms the best baseline at the same size in the 125M, 350M and 1B ranges, and structural efficiency is also necessary for deployment on edge devices. Therefore, it can be concluded that lightweighting should not be regarded as the compression of the existing large model, but rather as improving the computational density, attention patterns and embedding reusability of a small model architecture.

2.2 Runtime System, Cache Compression and Decoding Acceleration

At the service system level, vLLM introduces PagedAttention to split the key-value cache into non-contiguous blocks that are similar to the page tables of the operating system; thus, fragmentation and redundant storage are significantly reduced, and throughput at the same latency can be improved [3]. For long-context tasks, the main reason why the key-value cache is a large-memory consumer now is that it is an auxiliary structure. Therefore, KIVI proposes 2-bit asymmetric quantization for Keys by channel and Values by token to reduce peak memory and improve the throughput of the real inference workload, while maintaining the model quality to within a small margin [4]. KVQuant further points out that low-bit KV cache needs to handle distribution changes before and after RoPE, sparse representation of outliers, and non-uniform quantized data types simultaneously. Its 3-bit KV compression can expand the context carrying capacity considerably, and the perplexity increase is less than 0.1 [5].

In terms of operators and decoding, FlashAttention-2 has reduced the overhead of non-matrix multiplication through reasonable parallel partitioning and workload allocation, bringing the attention calculation closer to the theoretical peak of hardware at a higher proportion [6]. Medusa adds several decoding heads to the model and uses tree attention to verify multiple candidate subsequent tokens simultaneously to reduce the serial steps of autoregressive generation [7]. SGLang uses RadixAttention and compressed finite state machines to improve the execution efficiency of multi-turn calls, structured output and prompt reuse in complex language model programs [8]. At the same time, according to the Llama 3 technical report, its largest dense Transformer model has 405 billion parameters and supports a maximum of a 128,000-token context window; both the scale of the base model and the demand for context length are still expanding [9]. OmniVLM reduces the first-word latency of the edge multimodal model through visual token compression and shows that lightweight deployment is expanding from pure text models to visual-language scenarios [10]. These studies have shown that inference acceleration is not achieved by a single algorithm at the local level, but rather through a system-level collaboration of memory layout, attention operators, cache reuse, decoding strategies and front-end program semantics.

Figure 2 Analysis: The figure shows the speedup results reported in publicly available papers for AWQ/TinyChat, vLLM, KIVI, KVQuant, FlashAttention-2, Medusa and SGLang. Due to differences in hardware, workload, baselines and metrics of the methods, the results in the figure should not be regarded as a ranking within the same platform, but rather as the speedup potential of each technique under the original experimental conditions. Generally speaking, the runtime cache reuse and program-level scheduling are both relatively weak, and low-bit quantization and decoding

parallelism are more suitable for combination with them.

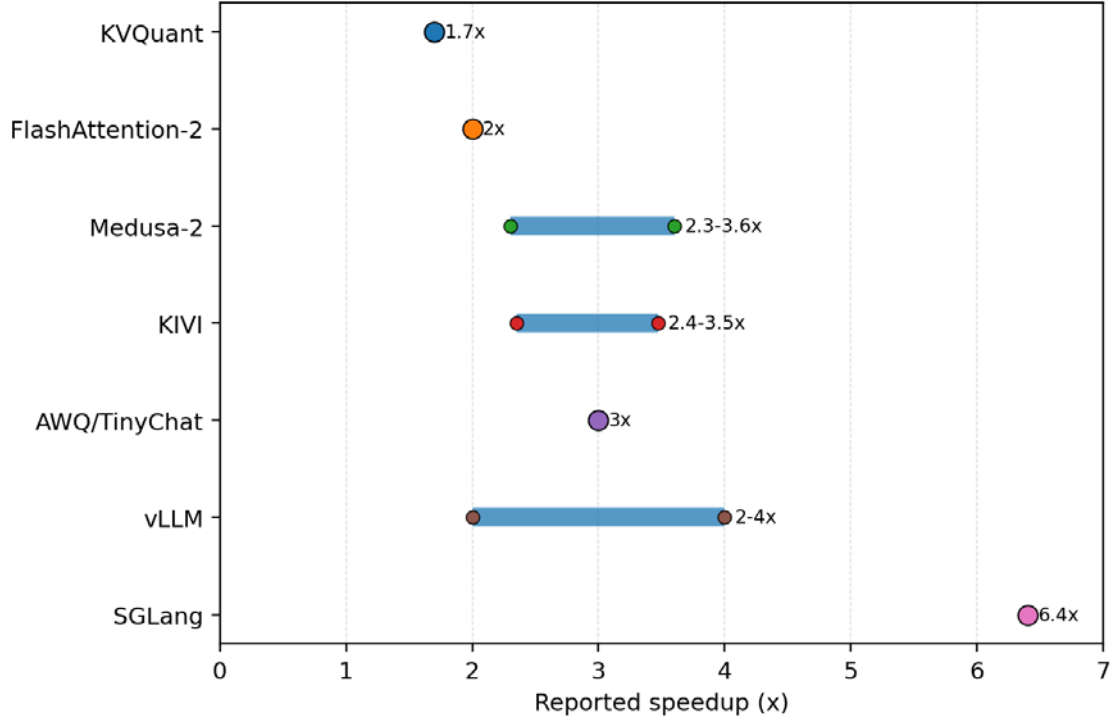


Figure 2. Speedup Range of Publicly Reported Typical Inference Acceleration Methods in the Past Three Years

3. Raise questions

3.1 Multi-objective constraints in resource-constrained environments

The deployment of large language models in resource-constrained environments faces the problem that many components, such as model weights, key-value caches, temporary activations and runtime frameworks, are all memory-intensive. Not all available memory on the edge device is used for inference; other parts of the OS, applications and security components also need to run there. Even if the model can be loaded, memory paging or process reclamation may be triggered due to the linear growth of the key-value cache in extended contexts or multi-turn sessions. Therefore, before deployment, memory for model loading, context length, concurrent batches, and key-value accuracy should also be included in the same constraint expression, and it should not be based on whether the model can run due to the number of parameters.

$$M_{\text{req}} = \frac{N_w b_w}{8} + \frac{2LHd_h B S b_{kv}}{8} + M_{\text{act}} + M_{\text{runtime}}, \quad M_{\text{req}} \leq M_{\text{max}} \quad (1)$$

M_{req} is the memory required for inference in equation (1); N_w is the number of model weight parameters; b_w is the weight bit width; L is the number of Transformer layers; H is the number of attention heads; d_h is the single-head dimension; B is the batch size; S is the context sequence length; b_{kv} is the KV cache bit width; M_{act} is the temporary activation memory; M_{runtime} is the overhead of the runtime framework, operator working area and memory pool; and M_{max} is the upper limit of available memory on the device. The above equation shows that edge deployment should compress the weights and manage or compress the KV cache at the same time.

Table 1 Resource-Constrained Deployment Scenarios and Key Constraints

Device tier	Memory budget	Compute feature	Primary bottleneck	Priority objective
Smartphone	6-12 GB RAM	Mobile CPU/GPU/NPU	Cold-start latency	Low TTFT
Industrial gateway	4-16 GB RAM	ARM CPU + small GPU	Thermal and memory cap	Stable throughput
Edge server	16-64 GB RAM	Entry GPU or iGPU	KV cache growth	Long-context support
Private appliance	8-32 GB RAM	CPU-only or low-end GPU	Bandwidth limit	Offline reliability
Embedded board	2-8 GB RAM	ARM SoC	Weight loading	Energy per token

Table 1 shows typical resource-constrained devices classified by memory budget, computing features and main bottlenecks. The problems for each device are not the same; mobile phones are more focused on first-word latency, industrial gateways need high thermal stability, edge servers are more sensitive to key-value caching, and CPU-only dedicated devices are readily constrained by weight loading and memory bandwidth. Therefore, before using any of the above lightweight deployment strategies, one needs to determine whether the bottleneck is of a particular type.

3.2 The conflict between accuracy, latency, and energy consumption

Secondly, it is not necessary for both accuracy and speed to be low. Low-bit quantization reduces memory and bandwidth, but if the step size is too large, there will be errors in the channels and thus damage mathematical reasoning, code generation or factual accuracy. Smaller models have a lower load-and-decode cost, but for complex tasks, more hint engineering, retrieval enhancement or multiple rounds of self-correction may be needed, which can increase the total number of tokens. Decoding acceleration methods can reduce the number of calls to the target model, but they are subject to the candidate token acceptance rate; in complex task distributions, high sampling temperature or an inadequate draft model will significantly reduce the benefits of acceleration. Therefore, the "optimal deployment" in a resource-constrained environment is not the lowest bit count or the smallest model, but rather to find a trade-off among memory, latency and energy consumption that still meets the quality requirements of the task.

$$q_i = \text{clip} \left(\text{round} \left(\frac{x_i}{s} \right) + z, q_{\min}, q_{\max} \right), \quad \hat{x}_i = s(q_i - z) \quad (2)$$

As shown in equation (2), x_i is the original weight or activation value; s is the quantization scale; z is the zero point; $q_{\min}$ and $q_{\max}$ are the quantization integer range bounds; q_i is the quantized integer value; $\hat{x}_i$ is the approximate value after dequantization; and clip is a cutoff function. This equation shows the initial process of uniform quantization. At the time of actual deployment, channel sensitivity, outlier distribution and calibration sample selection must also be taken into account jointly to avoid the accumulation of local quantization errors in multi-layer networks.

3.3 Bottleneck Migration in the Inference Path

The third is that the bottleneck may move during optimisation. After weight quantization, the pressure on reading model parameters is reduced, but the overhead of key-value caching and scheduling is relatively high; after FlashAttention-type operators reduce attention memory access,

serial dependencies in the decoding stage may become dominant; after increasing batch throughput, first-word latency and queue waiting may worsen; after adding retrieval enhancements, edge-side vector retrieval, document slicing and context compression will become new resource overheads. Without end-to-end observation, although some individual indicators may have increased in the deployment, overall user experience remains unaffected.

Table 2 Applicability Matrix of Lightweighting and Inference Acceleration Methods

Method	Main target	Typical gain	Risk factor	Best use case
Weight quantization	Model memory	Lower bandwidth	Accuracy drop	Small batch decoding
KV cache compression	Context memory	Longer context	Outlier sensitivity	Chat and RAG
Attention kernel fusion	Operator latency	Higher FLOPs use	Hardware dependence	Prefill-heavy tasks
Paged cache runtime	Serving throughput	Less fragmentation	Scheduler complexity	Multi-request serving
Speculative decoding	Serial decoding	Fewer target steps	Acceptance instability	Low-temperature generation
Small model design	On-device quality	Lower footprint	Capability ceiling	Private local assistant

As shown in Table 2, the reasons for the various shortcomings are listed below. Weight quantization mainly reduces the pressure on the model's resident memory and bandwidth; key-value compression is suitable for long contexts and multi-turn sessions; attention operator optimisation is more geared towards the prefill stage; paged cache runtime is more suitable for multi-request services; speculative or multi-head decoding depends on the candidate acceptance rate; and small model structure design is suitable for privacy and offline scenarios. In practice, they need to be used together rather than selected independently.

4. Problem Solving and Strategies

4.1 Building a Layered Lightweight Deployment Framework

A multi-level lightweight deployment framework for resource-constrained environments is proposed here. The first is the task-profiling layer; it determines whether the business logic is fact-finding, short-text rewriting, retrieval-enhanced question answering, device command generation or multimodal understanding, and sets corresponding quality thresholds, context upper bounds and response latency targets. The second layer is the model selection layer, and among the large model distillation, small model architecture, command-tuned models and domain-specific models, the smallest usable model is selected. The third layer is a numerical compression layer that determines the bit width of weights, activations and KV caches to prevent long-context failures due to only compressing weights. The fourth layer is the runtime execution layer, and attention operators, cache paging, batch processing strategies and hardware backends are used to optimise the inference graph. The fifth is the monitoring and rollback layer that determines whether to trigger a degradation strategy based on quality samples, latency quantiles, memory peaks, energy consumption logs, and so on.

$$T_{\text{tok}} = \max\left(\frac{F_{\text{tok}}}{C_{\text{eff}}}, \frac{R_w + R_{kv}}{B_{\text{mem}}}\right) + T_{\text{sched}} + T_{\text{sync}} \quad (3)$$

T_{tok} is the decoding latency of a single token, F_{tok} is the effective computational load required for a single token, C_{eff} is the effective computational throughput of the device, R_w is the weight read volume, R_{kv} is the KV cache read volume, B_{mem} is the effective memory bandwidth, T_{sched} is the scheduling overhead, and T_{sync} is the synchronization and communication overhead in equation (3). The formula shows that the decoding stage may be dominated by either the larger bottleneck of computation or bandwidth; therefore, quantization, cache compression and operator fusion need to be carried out in accordance with the actual bottleneck.

4.2 Co-quantization strategy of weights and KV cache

The process of weight quantization should be "calibration-protection-verification". First, collect activation statistics for samples of business prompts, domain vocabularies, long and short inputs, and abnormal commands. Second, identify channels with high sensitivity to output quality and scale up a few such channels or maintain a high accuracy. Finally, validate the quantized model with perplexity, task accuracy and human-readable samples. For 4-bit weight quantization, if only the loading of the model to the device is considered, key-value (KV) cache bloat during extended service and business logic (S) operation will be neglected; thus, the peak values of KV under various S conditions should be measured immediately after weight quantization.

Quantisation in key-value caching should distinguish between the statistical distributions of keys and values. Keys are more significantly affected by RoPE position encoding and are suitable for quantization by channel or before RoPE; values are semantically related to the token and suitable for grouping by token. For long-context question-and-answer, meeting minutes, and multi-round customer service scenarios, a hybrid caching strategy of "high-frequency nearest window high precision, long-distance context low bit depth, and retention of important fragments" can be used. It will ensure the quality of locally continuous generation and keep the linearly increasing memory pressure in long contexts under control.

4.3 Operator fusion, buffer scheduling and parallel decoding

Runtime Optimisations for inference acceleration should differentiate between the prefill and decode stages. The prefill stage has high parallelism and can therefore use FlashAttention-like operators to reduce the explicit storage of the attention matrix and HBM round trips. The decode stage produces only one or a few tokens per step and is more limited by the overhead of weight reading, KV cache reading and scheduling. Weight quantization, KV compression, PagedAttention cache management and speculative decoding can all be used. To reduce cache fragmentation and improve the composability of batch processing for multi-request services, Paged Cache can be used. Multi-head decoding or draft models can be used to improve the stability of benefits in low-temperature generation, fixed-format answers, and common instruction scenarios for single-client assistants.

$$G_{\text{spec}} = \frac{(E[a]+1)T_L}{T_D(k)+T_L+T_{\text{verify}}}, \quad 0 \leq E[a] \leq k \quad (4)$$

As shown in Equation (4), G_{spec} is the theoretical speedup ratio for speculative or multi-candidate decoding; $E[a]$ is the average number of candidate tokens accepted per round; T_L is the single-step verification latency of the target large model; $T_D(k)$ is the overhead of generating k candidate tokens in the draft model or multi-head module; and T_{verify} is the overhead of candidate tree construction, verification and sampling consistency maintenance. Based on the above, the

speed-up effect can be attributed to a certain degree of reduction in the acceptance rate of candidates and added verification costs. A relatively even distribution of tasks is more likely to attract a wider range of candidates and thus increase the actual benefit to near-theoretical levels.

4.4 Deployment Pipeline for Heterogeneous Hardware

Heterogeneous Hardware Deployment needs to manage multiple model formats, operator backends and memory pools as a single pipeline. CPU-only devices should prioritise weight partitioning, memory mapping and low-bit matrix multiplication kernels; for mobile GPU or NPU devices, support for operators by the compiler should be ensured to avoid the loss of quantization benefits caused by custom inverse quantization operators falling back to the CPU; at the edge server, a division of labour can be employed to have GPUs perform prefilling and CPUs or shared memory handle metadata retrieval and caching. For industry-specific network equipment, a stable model should be established; the offline model is to be validated, log anonymisation and anomaly rollback implemented, and service disruption caused by model updates must be avoided.

The six steps of the deployment pipeline are as follows: First, determine the business requirements for input and output length, concurrency limits, etc.; second, select candidate models and establish a full-precision baseline; third, perform weight and KV cache quantization, and record the precision loss; fourth, introduce attention operator fusion and cache pagination; fifth, test first-word latency, average token interval, P95 latency, peak memory usage, and energy consumption; finally, solidify the model, tokenizer, quantization configuration and runtime version into a rollback package. At the heart of this process is that all optimisation must have visible indicators; it should not be evaluated by the speed of just one generation.

Table 3. Evaluation Indicators for Deployment in Resource-Constrained Environments

Metric	Definition	Unit	Collection point	Decision rule
Peak memory	Maximum resident memory	GB	Runtime monitor	Must be below cap
TTFT	Time to first token	s	Client tracing	Lower is better
TPOT	Time per output token	ms/token	Decode loop	Lower is better
Throughput	Generated tokens per second	tokens/s	Server log	Higher is better
Quality delta	Score loss vs baseline	points	Task benchmark	Within tolerance
Energy/token	Energy per generated token	J/token	Power meter	Lower is better

Table 3 presents the evaluation indicators of the lightweight deployment. Peak memory determines model stability; TTFT reflects the user's initial response experience; TPOT indicates the smoothness of continuous generation; throughput is suitable for assessing multi-request situations; quality degradation can be employed to control the risk of quantization or small model replacement; and energy consumption/token are related to the mobile device's battery life and edge device heat dissipation. Only when all of the above indicators have been achieved will the deployment plan be deemed engineering-ready.

4.5 Multi-objective optimization and risk control

Lightweight deployment in practice can be regarded as a multi-objective constrained optimisation problem. Decision variables are model selection, weight quantization bits, key-value cache bits, cache eviction policy, batch processing policy, context compression policy and hardware backend. The goal is to reduce latency, but at the same time, we need to control the loss in quality, memory and power consumption, etc. For high-risk applications such as power dispatching, medical assistance, financial compliance and industrial control, the quality degradation threshold should be met before speed increases; for offline assistants, text summarisation and knowledge base question-answering, the compression ratio can be increased reasonably through manual review or evidence support from retrieval.

$$\min_{\pi, q, c} J = \alpha \Delta Q(q, c) + \beta T(\pi, q, c) + \gamma M(q, c) + \delta E(\pi, q, c), \quad \text{s.t. } M \leq M_{\max}, T \leq T_{\max} \quad (5)$$

π is the runtime scheduling and hardware mapping strategy in Equation (5); q is the bit configuration of weights, activation and KV cache; c is the context compression and cache governance strategy; ΔQ is the quality degradation relative to the full-precision baseline; T is the end-to-end latency; M is the peak memory usage; E is the energy consumption; α , β , γ and δ are the weights of different business objectives; $M_{\max}$ and $T_{\max}$ are the memory and latency constraints, respectively. The above equation can be used to make different deployment decisions under various circumstances, such as increasing γ and δ for a privacy-focused assistant, raising β for real-time customer service, or increasing α for a high-reliability question-answering system.

The three risk control devices are as follows: First, a quality regression mechanism: after every quantization, runtime upgrade, or context policy modification, regression tests need to be carried out on a fixed benchmark set, real business samples, and boundary warning sets. Second, a resource circuit breaker mechanism: If the memory exceeds a certain threshold, TPOT continuously declines; or if the device temperature is too high, reduce the context length, use a smaller model, or turn off high-cost functions automatically. Third, an interpretable logging mechanism to store model versions, quantization settings, sampling parameters, retrieval evidence and runtime logs for investigating abnormal outputs. Through the above means, lightweight deployment can be converted from a one-off compressed experiment into a continuously controllable system engineering project.

5. Conclusion

Lightweight Deployment and Inference Acceleration Methods for Large Language Models in Resource-Constrained Environments. Systematically analyze the research progress in the past three years in the areas of low-bit quantization, small model structure design, key-value cache compression, attention operator optimisation, cache paging and multi-candidate decoding. In addition, it proposes a collaborative deployment framework of "task profiling - model selection - numerical compression - runtime execution - monitoring and rollback". Research has shown that the primary problem at the edge is not a lack of GPU memory, but rather the multi-objective coupling among model weights, key-value caches, memory bandwidth, scheduling overhead, task quality and energy consumption.

(1) Lightweight deployment should start from the business quality threshold, rather than prioritizing the lowest bit. For short context and low concurrency scenarios, weight quantization can quickly reduce memory and bandwidth pressure; for long context and retrieval enhancement scenarios, KV cache compression and context governance are more critical; for multi-request

service scenarios, Paged cache and scheduling reuse can significantly improve throughput.

(2) Accelerating inference requires distinguishing between the prefill and decode stages. The prefill stage is suitable for reducing memory access through efficient attention operators and operator fusion; the decode stage relies more on weight reading optimization, KV cache compression, and parallel decoding. Confusing the two stages can easily lead to individual optimizations failing to translate into end-to-end benefits.

(3) Deployment schemes in resource-constrained environments must be observable and rollbackable. It is recommended to use peak memory, TTFT, TPOT, throughput, quality degradation, and energy consumption/token as unified evaluation indicators, and to solidify and manage model version, quantization parameters, and runtime configuration. Future research can further combine hardware-aware automatic search, dynamic bit allocation, edge retrieval compression, and security and trustworthiness assessment to promote the stable engineering application of large language models from cloud capability demonstration to edge scenarios.

References

- [1] Lin J., Tang J., Tang H., Yang S., Chen WM, Wang WC, Xiao G., Dang X., Gan C., Han S. AWQ: Activation-aware Weight Quantization for On-Device LLM Compression and Acceleration. *Proceedings of Machine Learning and Systems*, 2024, 6.
- [2] Liu Z., Zhao C., Iandola F., Lai C., Tian Y., Fedorov I., Xiong Y., Chang E., Shi Y., Krishnamoorthi R., Lai L., Chandra V. MobileLLM: Optimizing Sub-billion Parameter Language Models for On-Device Use Cases. *Proceedings of the 41st International Conference on Machine Learning*, 2024.
- [3] Kwon W., Li Z., Zhuang S., Sheng Y., Zheng L., Yu CH, Gonzalez JE, Zhang H., Stoica I. Efficient Memory Management for Large Language Model Serving with PagedAttention. *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.
- [4] Liu Z., Yuan J., Jin H., Zhong S., Xu Z., Braverman V., Chen B., Hu X. KIVI: A Tuning-Free Asymmetric 2bit Quantization for KV Cache. *Proceedings of the 41st International Conference on Machine Learning*, 2024.
- [5] Zhang, Y. (2026). Exploration of Enterprise Big Data Microservice Architecture Based on Domain-Driven Design (DDD). *Procedia Computer Science*, 282, 1994-2003.
- [6] Liu, X. (2026). Research on the Application of Artificial Intelligence in Consumer Privacy Data Protection. *Procedia Computer Science*, 279, 956-965.
- [7] Jing, X. (2025, November). In-Depth Analysis of the Optimization Decision Model of Financial Investment Portfolio Based on Machine Learning Algorithms in Python Scikit-Learn Library. In *International Conference on Computational Technologies for Research in Data Analytics* (pp. 130-140). Cham: Springer Nature Switzerland.
- [8] Yu, X. (2026). Exploration of Multi-Channel Conversion Path Optimization Methods Based on A/B Testing.
- [9] Han, W. (2025, November). An Empirical Study on Building a High-precision Forecasting System for Financial Market Trends Based on the PyTorch Deep Learning Framework. In *International Conference on Computational Technologies for Research in Data Analytics* (pp. 518-528). Cham: Springer Nature Switzerland.
- [10] Yuan, Y. (2026). Research on Memory Management and Dynamic Reasoning Methods for Intelligent Agents Based on Large Language Models.