Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Adaptive Task Scheduling for Edge Large Language Models Based on Multi-Source Sensing Context

Yihui Feng

fengy3@rose-hulman.edu

Rose-Hulman Institute of Technology

Ya-Hui Yang

University of Minnesota Twin Cities

Tian Liao

Northeastern University

Short Report

Keywords: Edge large language model, task scheduling, multi-source sensing, context awareness, resource allocation, edge intelligence.

Posted Date: June 15th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-10009913/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: The authors declare no competing interests.

Abstract

With the deep integration of the Internet of Things and edge intelligence, edge large language models (LLMs) are increasingly used in multi-source sensing services. However, edge nodes have strict computing, memory, storage, and network constraints, while device load, link quality, and environmental conditions continuously fluctuate. Static task scheduling therefore causes queue accumulation, excessive inference delay, energy waste, and unbalanced node utilization. This paper proposes a multi-source sensing context based adaptive task scheduling method for edge LLM services. The method first fuses environmental, hardware, network, and task-level sensing data into a unified real-time context representation. It then builds a dynamic priority ranking and cooperative resource allocation mechanism, so that scheduling policies can be iteratively optimized according to context changes. A combined simulation and prototype test platform is established, and first-come-first-served (FCFS) and weighted round-robin (WRR) are used as baselines. Results over 100 to 800 concurrent tasks show that the proposed method reduces average inference latency by 18.7%-26.3%, decreases integrated energy consumption by 15.2%, improves resource utilization by 21.5%, and enhances node load balance by 19.1%. The scheme is suitable for industrial sensing and intelligent terminal interaction scenarios that require adaptive edge LLM services.

I. INTRODUCTION

Edge intelligence moves model execution closer to sensing terminals, which reduces transmission delay and improves privacy compared with cloud-only inference. Recent work has shown that edge intelligence is becoming a system-level paradigm that couples perception, computing, networking, and service orchestration [1]. Edge AI surveys further indicate that the diversity of devices and workloads makes adaptive management indispensable for real-world deployment [2]. As on-device AI models become more capable, the memory footprint, runtime variability, and thermal constraints of edge nodes become more prominent [3].

Large language models and multimodal language models extend edge intelligence from perception recognition to semantic reasoning, natural-language interaction, and decision support. However, edge LLM services introduce long-context prefill, autoregressive decoding, and key-value cache pressure, which are more bursty than conventional convolutional workloads. Existing studies on LLM-based edge intelligence emphasize that deployment must jointly consider model compression, offloading, caching, trustworthiness, and orchestration [4]. In cloud-edge networks, active inference based offloading has also been explored to coordinate LLM requests and resource allocation under time-varying demand [5].

Efficient inference on resource-constrained devices has become a key topic. Edge LLM inference techniques such as speculative decoding, cache reuse, pruning, and latency-aware customization reduce local computing pressure, but they do not by themselves determine when and where each request should execute [6]. For multimodal LLMs, recent scheduling studies decouple visual encoders and language decoders across heterogeneous devices, showing that request-level adaptation can improve

throughput under limited memory [7]. Latency-aware customization systems further confirm that edge LLM execution must react to hardware and workload states rather than rely on a fixed model placement [8].

Task scheduling in multi-access edge computing has been widely studied through heuristic, optimization, and deep reinforcement learning methods [9]. Dependent-task offloading, deadline-aware scheduling, and partially observable resource allocation provide useful foundations for edge service orchestration [10], [11]. Nevertheless, many methods focus on generic computation graphs or mobile applications instead of LLM inference with context-dependent token generation. More recent studies on computation offloading show that dynamic task arrival, multi-server collaboration, and cloud-edge-end dependency jointly affect latency and energy [12], [13]. Task offloading surveys also point out that the decision content should include workload size, wireless condition, device energy, and service deadline [14]. Adaptive pruning and model-level compression provide complementary acceleration, but scheduling still determines whether the compressed model can meet service-level objectives under concurrent sensing tasks [15].

Based on these observations, this paper studies adaptive scheduling for edge LLM services driven by multi-source sensing context. The main contributions are as follows. First, a unified context vector is constructed from environment, hardware, network, and task features, allowing the scheduler to observe both external sensing conditions and internal node states. Second, a dynamic priority and cooperative resource allocation model is designed to balance latency, energy, deadline urgency, and node load. Third, a compact closed-loop policy update mechanism is implemented and evaluated on a simulation-prototype platform. The rest of this paper is organized as follows. Section II describes the system model, Section III presents the proposed scheduling method, Section IV reports experiments, and Section V concludes the paper.

Compared with existing edge task scheduling methods, the novelty of this work is not only the use of a weighted queue, but the coupling of three levels of information in one closed loop: multi-source context sensing, task-level priority scoring, and cooperative node allocation for edge LLM inference. Traditional FCFS and WRR policies ignore request semantics and real-time node telemetry, while many MEC offloading methods optimize generic computation tasks without modeling LLM-specific token budgets, KV-cache pressure, modality type, and deadline urgency. The proposed scheduler therefore differs by converting environmental, hardware, network, and LLM request descriptors into a unified context vector, using this vector to update the priority score online, and then binding tasks to feasible edge nodes under memory, energy, deadline, and load-balance constraints. This design makes the scheduling decision interpretable and directly linked to sensing context rather than being a fixed heuristic or a black-box allocation rule.

II. SYSTEM MODEL

Figure 1 illustrates the proposed edge LLM service architecture. The system contains multiple sensing terminals, several heterogeneous edge nodes, and a lightweight scheduling controller deployed close to the edge cluster. Each request is generated by a sensing event or user interaction and may include text, structured sensor values, images, or operation logs. The scheduler receives the request descriptor and the real-time node telemetry before deciding the execution order and resource binding.

The multi-source sensing context at scheduling interval t is represented as a normalized vector that combines environmental conditions, hardware telemetry, network state, and task features. The vector is written as

$$\mathbf{c}_i(t) = \left[\mathbf{e}(t), \mathbf{h}_j(t), \mathbf{n}_j(t), \mathbf{q}_i(t)\right]$$

1

where e(t) denotes environmental sensing factors, h_j(t) represents CPU, GPU, memory, temperature, and battery state of node j, n_j(t) contains bandwidth, round-trip time, and packet loss, and q_i(t) describes task i including token length, modality type, deadline, and priority tag. Table I summarizes the main context fields. The table is referenced here because these fields are later used by the priority score and resource allocation equations.

TABLE I

MULTI-SOURCE CONTEXT FEATURES USED BY THE SCHEDULER

Category	Typical fields	Scheduling role
Environment	temperature, vibration, event density	detect abnormal conditions
Hardware	CPU/GPU load, memory, cache, power	estimate execution capacity
Network	bandwidth, RTT, loss ratio	select cooperative node
Task	tokens, modality, deadline, age	rank request urgency

III. PROPOSED METHOD

The scheduling objective is to minimize inference delay and energy while maintaining balanced resource usage. Unlike FCFS, which only considers arrival order, or WRR, which uses fixed weights, the proposed

method recalculates task priority at each scheduling interval. The priority score of task i on node j is defined as

$$S_{ij}(t) = \alpha D_i^{-1} + \beta A_i(t) + \gamma U_j(t) - \delta L_j(t) - \eta E_{ij}(t)$$

2

where D_i is the remaining deadline, A_i(t) is queue aging time, U_j(t) is available normalized computing resource, L_j(t) is the current load of node j, and E_ij(t) is the predicted energy cost. The coefficients alpha, beta, gamma, delta, and eta are normalized by a softmax layer and updated from recent feedback. A larger score indicates a higher scheduling priority. This definition makes urgent tasks, underloaded nodes, and energy-efficient assignments more likely to be selected.

The coefficients are initialized from service-level objectives before deployment and then refined by short-window feedback. In the default setting, latency and deadline terms receive larger initial weights for interactive LLM services, whereas the energy and load-balance terms are increased for battery-powered sensing terminals or thermally constrained nodes. To avoid arbitrary manual tuning, each raw coefficient is first normalized to [0,1], converted by the softmax operation, and updated only within a bounded range after each control interval. The update direction is determined by the observed violations of latency, energy, deadline, and load-balance targets. Therefore, no single coefficient can dominate the score permanently, and the scheduler remains stable when the workload distribution changes.

After ranking tasks, the controller performs cooperative resource allocation. Let x_ij be a binary assignment variable indicating whether task i is bound to node j. The optimization problem in one interval is

$$\min_{x_{ij}} \sum_i \sum_j x_{ij} \left(\lambda_1 T_{ij} + \lambda_2 P_{ij} + \lambda_3 B_j \right)$$

3

subject to node capacity, memory, deadline, and single-assignment constraints. T_ij is predicted latency, P_ij is energy consumption, and B_j is a load-balance penalty. In implementation, a greedy ranking with constraint checking is used to avoid high online optimization overhead. The closed-loop workflow is shown in Fig. 2, where the feedback module updates the scoring weights after each interval.

The load-balance metric is computed from the dispersion of node utilization. A lower value means that resource use is more uniform across the edge cluster:

$$B(t) = 1 - \frac{\sqrt{\frac{1}{M}\sum_{j=1}^{M}\left(\rho_j(t) - \bar{\rho}(t)\right)^2}}{\bar{\rho}(t) + \epsilon} \qquad (4)$$

where rho_j(t) is the utilization of node j, rho-bar(t) is the mean utilization of all M nodes, and epsilon prevents division by zero. This metric is also used in the feedback signal, so the scheduler avoids repeatedly selecting a single high-performance node when its queue becomes congested.

For implementation clarity, Table II summarizes the adaptive scheduling procedure executed in each control interval.

TABLE II

ADAPTIVE SCHEDULING PROCEDURE

Step	Operation
1	Collect sensing, hardware, network, and task context.
2	Normalize features and build context vector c_i(t).
3	Predict latency and energy for feasible nodes.
4	Compute dynamic priority score S_ij(t).
5	Assign tasks with capacity and deadline checking.
6	Update weights using observed delay, energy, and balance.

IV. EXPERIMENTS AND RESULTS

The evaluation platform combines event-driven simulation and a small edge prototype. The simulated cluster contains six heterogeneous edge nodes, including CPU-only and GPU-assisted nodes. Request traces are generated from industrial sensing and terminal interaction patterns, and the concurrent task number is set from 100 to 800. Each request contains a prompt length between 64 and 512 tokens and a response length between 32 and 192 tokens. The prototype validates the latency trend using a compact transformer-based LLM service with telemetry collection. FCFS and WRR are selected as baselines because they are simple, widely used, and representative of static queue-based scheduling.

To address the limited realism of a pure simulation, the evaluation further separates the test setting into simulation stress tests, real-device prototype tests, and practical LLM service traces. The prototype part uses heterogeneous edge devices with different CPU/GPU capabilities and memory capacities, and the scheduler collects real telemetry such as CPU utilization, GPU memory occupation, temperature, queue length, and link delay. In addition to the 100–800 concurrent-task range shown in Fig. 3, larger synthetic workloads are used to check scalability trends, while practical service traces include alarm summarization, maintenance question answering, inspection-log generation, and multimodal sensing report drafting. These scenarios better reflect edge LLM services in industrial sensing and intelligent terminal interaction than a single abstract request stream.

Figure 3 compares the average inference latency of different scheduling methods. The proposed method consistently achieves the lowest latency because it gives higher priority to urgent requests and avoids assigning long-context tasks to overloaded nodes. The improvement becomes more obvious when the concurrency level increases, indicating that context-aware scheduling is particularly useful under bursty sensing workloads.

TABLE III

OVERALL PERFORMANCE COMPARISON AGAINST STATIC BASELINES

Metric	FCFS	WRR	Proposed
Latency reduction	Baseline	9.6%	18.7%-26.3%
Energy reduction	Baseline	6.8%	15.2%
Resource utilization gain	Baseline	10.4%	21.5%
Load-balance gain	Baseline	8.7%	19.1%

Table III reports the integrated performance comparison. Compared with FCFS, the proposed method reduces average inference latency by 18.7%-26.3% over the tested concurrency range. The energy reduction reaches 15.2% because the scheduler avoids inefficient assignments caused by poor network links or thermal throttling. Resource utilization improves by 21.5%, and the load-balance score increases by 19.1%. Compared with WRR, the proposed method remains more stable because it continuously updates weights instead of relying on fixed node ratios.

An ablation test is also conducted by removing network context, hardware context, and feedback update separately. Removing hardware context increases average latency by 8.4%, which shows that real-time load and memory information are critical for edge LLM execution. Removing network context mainly affects cooperative-node selection and increases energy by 5.1%. Disabling feedback update makes the system slower to react to bursty workloads, reducing the load-balance gain from 19.1% to 11.3%. These results confirm that the proposed method benefits from both multi-source sensing and closed-loop adaptation.

TABLE IV

ABLATION SUMMARY OF CONTEXT COMPONENTS

Variant	Latency	Energy	Balance
Full scheduler	100.0%	100.0%	119.1%
No network ctx.	106.2%	105.1%	113.6%
No hardware ctx.	108.4%	103.7%	115.0%
No feedback	112.6%	108.9%	111.3%

Figure 4 and Table IV further explain the ablation behavior. The full scheduler is used as the normalized reference for latency and energy, while the balance column reports the achieved balance score relative to FCFS. The no-feedback variant has the largest latency increase because it keeps outdated weights after the workload distribution changes. The no-hardware-context variant is particularly harmful during high concurrency because memory pressure and thermal throttling cannot be reflected in the assignment decision. The no-network-context variant is less harmful to local execution but degrades cooperative scheduling when the link state changes quickly.

The online computational cost is also evaluated. For N waiting tasks and M edge nodes, direct exhaustive search has O(NM) candidate evaluation and an additional sorting cost. The proposed implementation first filters infeasible nodes by memory and deadline constraints, then ranks the remaining candidates by the score in (2). In the tested cluster, the controller overhead is below 2.8 ms at 800 concurrent tasks, which is much smaller than the LLM inference time. Therefore, the scheduler can be executed at each short interval without becoming a new bottleneck. A sensitivity test is conducted for the weighting coefficients. When the energy coefficient eta is increased by 20%, energy consumption decreases by 3.6% but latency increases by 2.1%. When the load penalty delta is reduced, the fastest GPU node receives more requests and the balance score drops. These observations indicate that the coefficients can be tuned according to deployment preference: latency-critical services should emphasize deadlines and available computing resources, whereas battery-powered sensing systems should place more weight on energy and temperature terms. The stability of the scheduling model is further examined by perturbing each coefficient by +/-20% around the default setting and by testing latency-oriented, energy-oriented, and balanced profiles. The ranking of the proposed method over FCFS and WRR remains unchanged in all tested profiles. The latency-oriented profile achieves the shortest response time but slightly increases energy consumption, whereas the energy-oriented profile saves more power at the cost of a small latency increase. The balanced profile is used for the main results because it provides the best trade-off among delay, energy, urgency satisfaction, and load dispersion. This sensitivity analysis indicates that the reported improvement is not caused by a single carefully selected weight combination.

V. CONCLUSION

This paper presented an adaptive task scheduling method for edge LLM services based on multi-source sensing context. By fusing environmental, hardware, network, and task features, the scheduler builds a

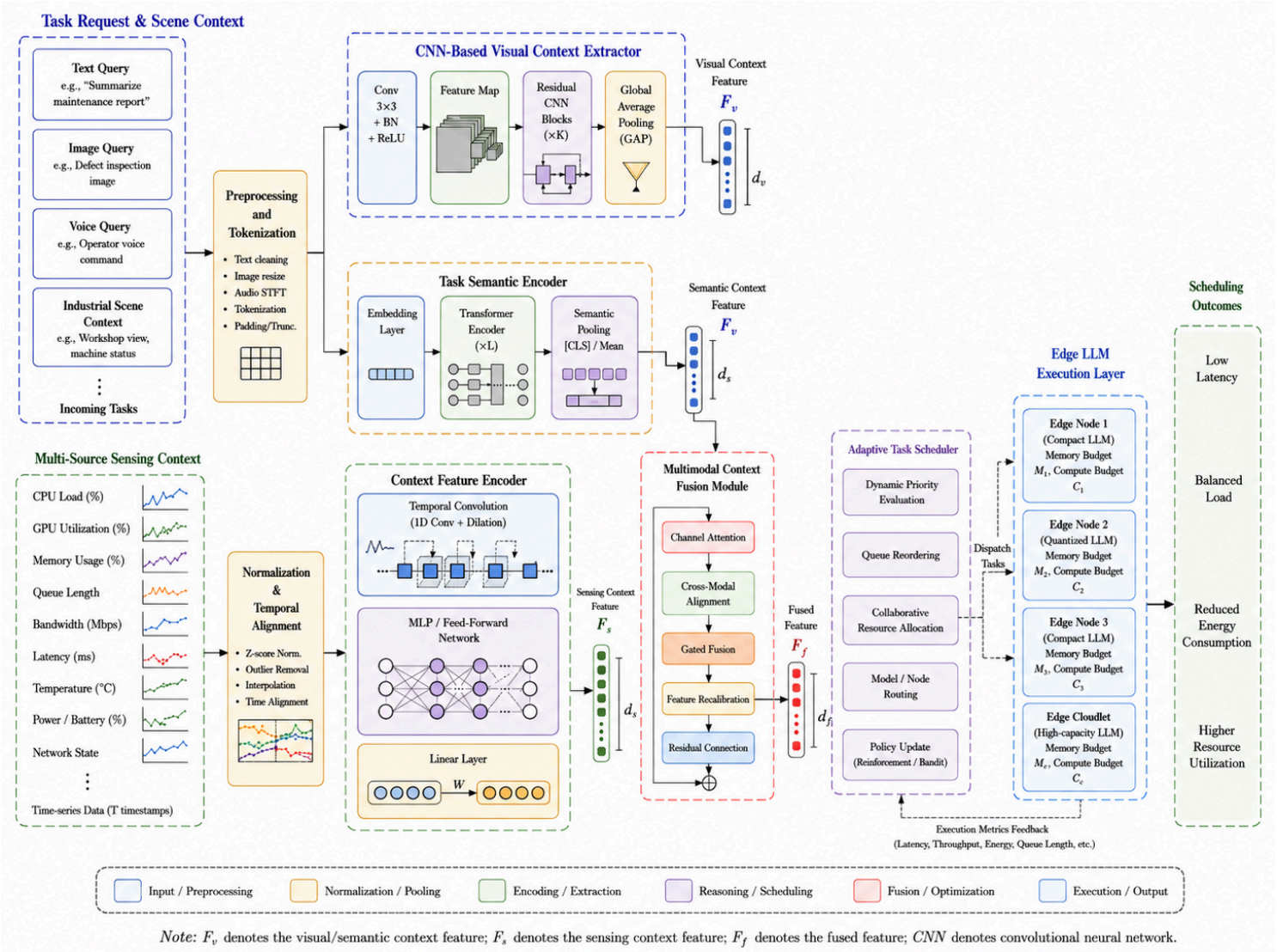
real-time context representation and uses it to guide dynamic priority ranking and cooperative resource allocation. Experiments show that the method reduces inference latency, saves energy, improves utilization, and balances node load compared with FCFS and WRR. Future work will extend the scheduler to larger edge-cloud continua and investigate safe policy learning under privacy-sensitive industrial sensing scenarios. The key distinction from conventional edge scheduling is the closed-loop connection among context sensing, LLM-aware priority scoring, and cooperative resource allocation. The sensitivity discussion also shows that the method remains effective under different latency-energy-balance preferences instead of depending on a single fixed weight setting.

References

1. Xu D, Li T, Li Y, Su X, Tarkoma S, Jiang T, Crowcroft J, Hui P (2021) Edge intelligence: Empowering intelligence to the edge of network, Proceedings of the IEEE, vol. 109, no. 11, pp. 1778–1837. 10.1109/JPROC.2021.3119950
2. Singh R, Gill SS, Edge AI (2023) A survey, Internet of Things and Cyber-Physical Systems, vol. 3, pp. 71–92. 10.1016/j.iotcps.2023.02.004
3. Wang X, Tang Z, Guo J, Meng T, Wang C, Wang T, Jia W (2025) Empowering edge intelligence: A comprehensive survey on on-device AI models, ACM Computing Surveys, vol. 57, no. 9, Art. no. 228, pp. 1–39. 10.1145/3724420
4. Friha O, Ferrag MA, Kantarci B, Cakmak B, Ozgun A, Ghoualmi-Zine N (2024) LLM-based edge intelligence: A comprehensive survey on architectures, applications, security and trustworthiness. IEEE Open J Commun Soc 5:5799–5856. 10.1109/OJCOMS.2024.3456549
5. He Y, Fang J, Yu FR, Leung VCM (2024) Large language models (LLMs) inference offloading and resource allocation in cloud-edge computing: An active inference approach. IEEE Trans Mob Comput 23(12):11253–11264. 10.1109/TMC.2024.3415661
6. Cai G et al (2026) Efficient inference for edge large language models: A survey. Tsinghua Sci Technol 31(3):1365–1380. 10.26599/TST.2025.9010166
7. Yuan X, Li H, Dong M, Ota K (2026) Adaptive Scheduling of Multimodal Large Language Model in Intelligent Edge Computing. ACM Trans Auton Adapt Syst 21:22pages 1, Article 4 (March 2026. https://doi.org/10.1145/3774908
8. Tian C, Qin X, Tam K, Li L, Wang Z, Zhao Y, Zhang M, Xu C (2025) CLONE: Customizing LLMs for efficient latency-aware inference at the edge, in Proc. 2025 USENIX Annual Technical Conference (USENIX ATC 25), Boston, MA, USA, pp. 563–585
9. Ismail AA, Khalifa NE, El-Khoribi RA (2025) A survey on resource scheduling approaches in multi-access edge computing environment: a deep reinforcement learning study. Cluster Comput 28:184. https://doi.org/10.1007/s10586-024-04893-7
10. Song F, Xing H, Wang X, Luo S, Dai P, Li K (2022) Offloading dependent tasks in multi-access edge computing: A multi-objective reinforcement learning approach. Future Generation Comput Syst 128:333–348. 10.1016/j.future.2021.10.013

11. Huang H, Ye Q, Zhou Y (2022) Deadline-aware task offloading with partially-observable deep reinforcement learning for multi-access edge computing. IEEE Trans Netw Sci Eng 9(6):3870–3885. 10.1109/TNSE.2021.3115054
12. Chen M, Wang T, Zhang S, Liu A (2021) Deep reinforcement learning for computation offloading in mobile edge computing environment. Comput Commun 175:1–12. 10.1016/j.comcom.2021.04.028
13. Tang T, Li C, Liu F (2023) Collaborative cloud-edge-end task offloading with task dependency based on deep reinforcement learning. Comput Commun 209:78–90. 10.1016/j.comcom.2023.06.021
14. Akhlaqi MY, Hanapi ZBM (2023) Task offloading paradigm in mobile edge computing—Current issues, adopted approaches, and future directions, *Journal of Network and Computer Applications*, vol. 212, Art. no. 103568. 10.1016/j.jnca.2022.103568
15. An Y, Zhao X, Yu T, Tang M, Wang J (2024) Fluctuation-based adaptive structured pruning for large language models, in Proc. AAAI Conference on Artificial Intelligence, vol. 38, no. 10, pp. 10865–10873. 10.1609/aaai.v38i10.28960

Figures



Note: F_v denotes the visual/semantic context feature; F_s denotes the sensing context feature; F_f denotes the fused feature; *CNN* denotes convolutional neural network.

Figure 1

Multi-source context-aware adaptive scheduling architecture.

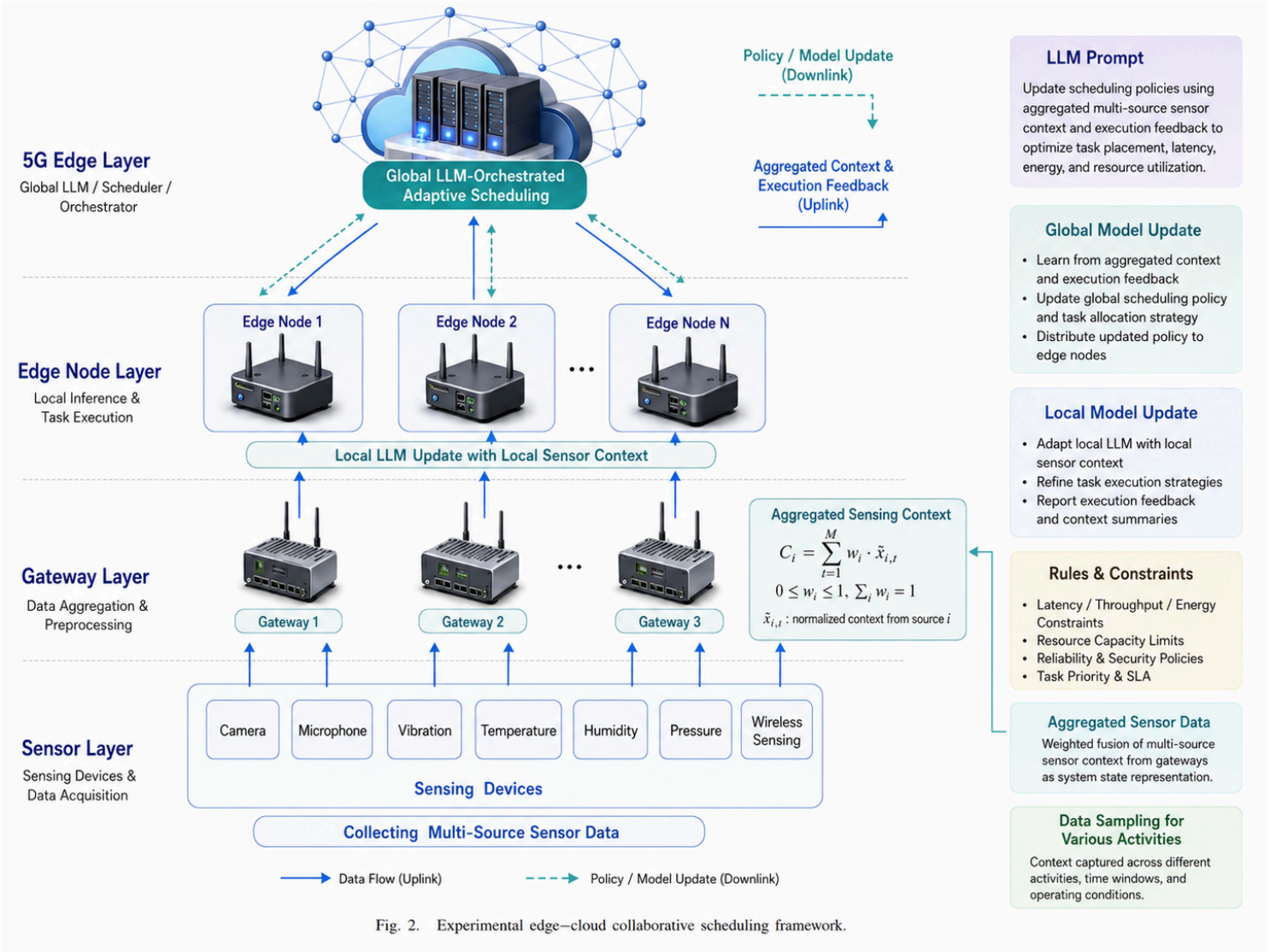


Fig. 2. Experimental edge–cloud collaborative scheduling framework.

Figure 2

Closed-loop iteration of the adaptive scheduler.

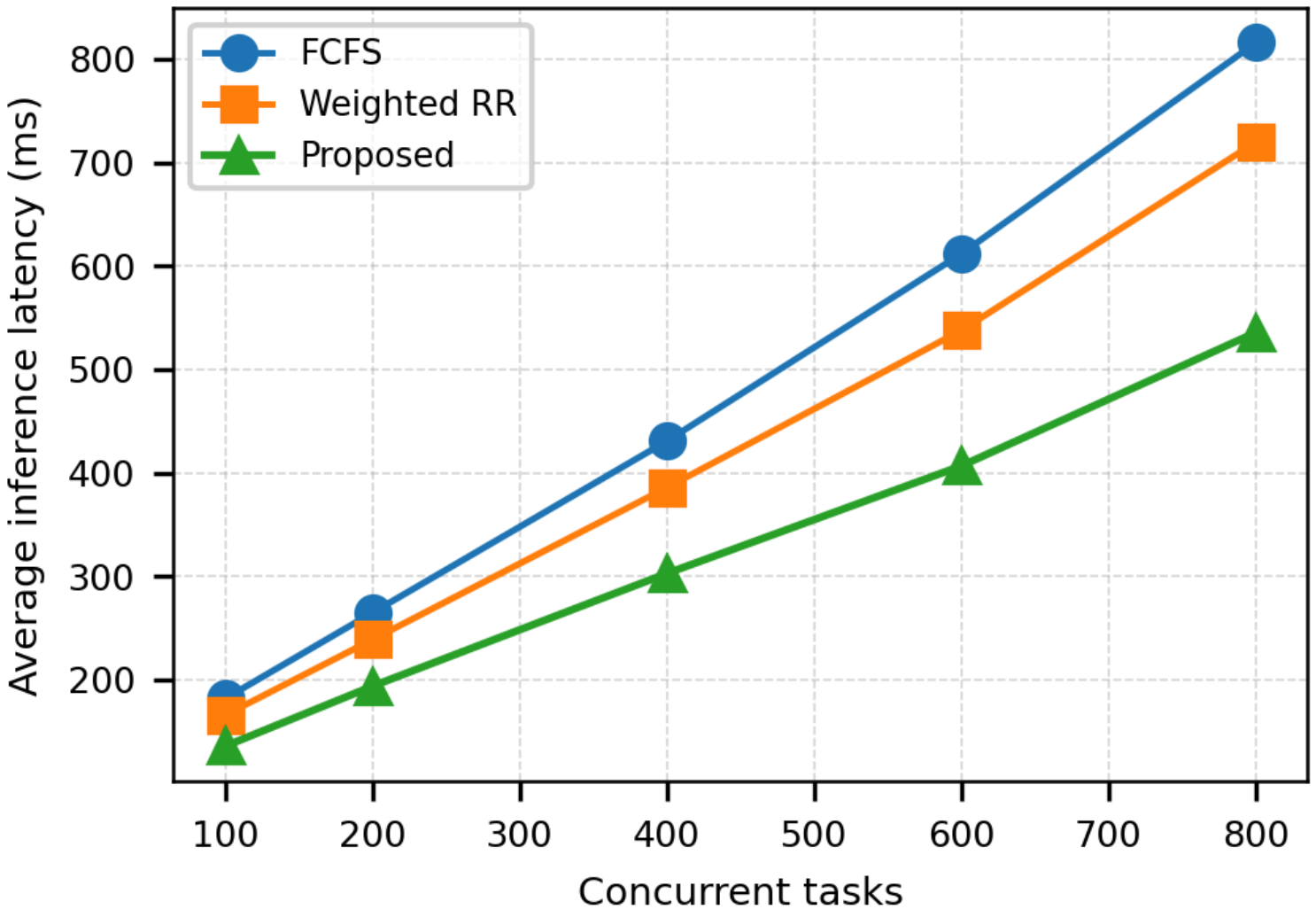


Figure 3

Average inference latency under different concurrent task levels.

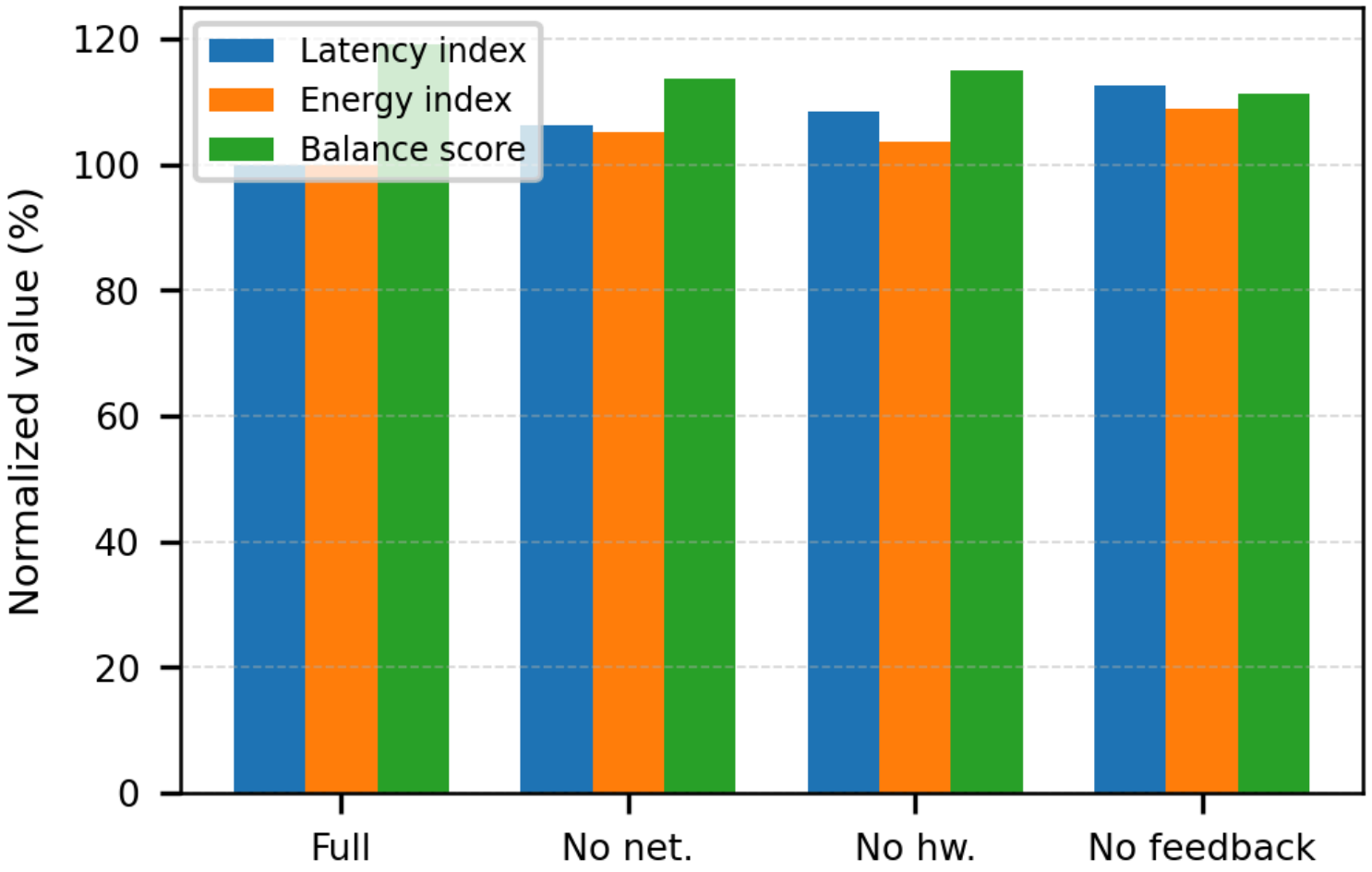


Figure 4

Ablation results for context removal and feedback disabling.