Scholar Publishing Group
International Journal of Big Data Intelligent Technology
https://doi.org/10.38007/IJBDIT.2026.070203
ISSN 2790-0932 Vol. 7, Issue 2: 20-30

Research on Cloud Native Lightweight Container Layout Scheme for Edge Computing Scenarios

Chenghao Shi

Georgia Institue of Technology, Atlanta, GA, 30332, USA

Keywords: Edge Computing; Cloud-Native; Lightweight Containers; Container Layout; Layered Orchestration; Resource Scheduling

Abstract: Edge computing places computing, storage and network functions at the site where data is generated to provide fundamental support for the Industrial Internet, smart transportation, urban sensing and other low-latency intelligent applications. As cloud-native technologies move from the central cloud to the edge, lightweight containers have gradually been used to run edge microservices, support device protocol adaptation and real-time inference. Edge nodes are highly heterogeneous in terms of processor architecture, memory capacity, network stability and security boundaries, and thus traditional container orchestration methods that assume a central cluster cannot be applied directly. Based on a review of research and industry data over the past three years on edge cloud-native container orchestration, this paper establishes a layered layout scheme of "cloud-side control, edge autonomy, lightweight nodes, and service proximity" to address the problem and puts forward optimisation strategies in terms of resource constraints, latency requirements, image startup speed, state persistence, and security isolation. The study proposes that container layout at the edge should aim to maximise the use of a single node and build a complete-stack decision-making system that covers all levels, networks and security to enhance business continuity, deployment efficiency and operational control.

1. Introduction

The development of edge computing has progressed rapidly, and at this time, the traditional paradigm of cloud computing – "centralised processing and unified deployment" – can no longer apply. If all of the above data were sent back to the central cloud for processing, there would be problems of link congestion, response delay, privacy infringement and increased energy consumption. Therefore, computing tasks need to be preprocessed, inferred and filtered close to the data source, and control decisions should be made locally; only the central cloud will be used for overall coordination, policy dissemination, model management and long-term data storage. Cloud-native technologies offer containerised encapsulation, declarative configuration, elastic scaling and automated deployment, and serve as the main technology path for connecting central cloud and edge resources.

However, the edge is not a reduced scale of the central cloud. The central cloud data centre is

generally well-connected to the network, uses relatively uniform server hardware, and has a mature operation and maintenance system; on the other hand, edge nodes are often placed in factories, parks, roadsides, stores or equipment rooms. The nodes have different attributes, the network links are unstable, on-site maintenance costs are high, and the security zones are scattered. Recent reviews have pointed out that although containerisation and orchestration capabilities have provided a relatively stable foundation for migrating cloud-native workloads to the edge, edge-native deployment and continuous delivery/orchestration mechanisms are still in their infancy; furthermore, standardized solutions for adapting to resource-constrained and disconnected environments are lacking [1].

Therefore, at present, research on cloud-native lightweight container deployment schemes is highly practical. Container deployment is not only about scheduling the placement of container instances on a node; it also includes all-encompassing design such as container image layering, node role allocation, service chain topology, data caching location, fault migration strategies and security isolation boundaries. Lightweight containers can reduce operating costs through optimisation; however, they may lead to issues such as frequent cross-layer communication, excessively high cold-start latency, poor edge autonomy, state data drift, and broken observation links. This paper puts forward a cloud-native lightweight container-layered deployment scheme for edge computing and aims to achieve a "lightweight, local, autonomous and controllable" deployment in edge environments, providing a reference for platform construction and engineering implementation.

2. Current Status Analysis of the Research Topic

2.1 Technical Foundation of Edge Cloud Native Container Orchestration

Containerized Deployment has changed from a single-machine-based run environment to a cross-cluster orchestration system in terms of technology. Containers group an application's runtime environment, dependencies and configurations into a single image so that the application runs consistently in all environments; the orchestration system is responsible for replica scheduling, service discovery, configuration injection, health checks, rolling updates and fault recovery. Edge scenarios place new demands on this system: the control plane should be highly centralized and stable; the data plane needs to be near the field and self-regulating in the absence of connections; node agents should reduce memory consumption and the number of background processes; and service communication should minimise the number of hops over wide-area links.

Research has mainly focused on the performance of container orchestration tools, lightweight distributions and service deployment modes so far. Performance tests of container orchestration in edge computing show that these tools have distinct differences in cluster initialization and service start-up, scaling capabilities, network throughput and resource utilization. Edge deployment should not use the default parameters in the central cloud [2]. Other research has shown that, for example, in terms of the lowest resource usage, ease of installation, and load capability, the orchestration platform for constrained devices is not ideal. The deployment modes that are suitable for industrial control gateways, edge stations and micro data centres are not all the same [3].

Table 1 shows the different requirements for edge container deployment according to the type of business. Protocol gateways and industrial control tasks generally need to be close to the source and deterministic; therefore, less emphasis is placed on accelerators and model caching for video analytics. Data aggregation tasks do not have a high requirement for low latency and focus more on continuous buffering and regional cooperation. Therefore, at the edge container deployment stage, the type of workload needs to be determined first, followed by the location of the node, runtime characteristics and state handling method.

Table 1 Edge Workloads and Container Layout Requirements

Workload class	Latency focus	State pattern	Runtime choice	Layout priority
Protocol gateway	Sub-10 ms	Local queue	Minimal container	Device-adjacent node
Sensor filtering	20-50 ms	Mostly stateless	Container or Wasm	Nearest edge node
Video analytics	50-100 ms	Model cache	Container with GPU	Accelerated edge zone
Industrial control	Deterministic	Local checkpoint	Hardened container	Isolated node pool
Data aggregation	Minute-level	Persistent buffer	Standard container	Regional edge cluster

2.2 Trends in Container Production and Automated Delivery

According to the data in the industry, containerisation has begun to appear in production environments. Cloud-native surveys show that the proportion of organisations running "most or all of their production applications" in containers is continuously rising; thus, containers have moved from being only development and testing tools to enterprise-level production platforms. With the rise of edge computing, the new requirements for edge applications include a unified image governance and version release/rollback mechanism; however, due to their weak networks and limited resources, these edge nodes must be equipped with a lighter, more stable, and highly recoverable deployment process.

Although it can be shown that moving microservices from a central cloud to the edge reduces end-to-end latency and backhaul traffic according to research, the complexity of service dependency, data consistency and location selection will also increase [4]. More research on service placement and scheduling has found that, in the edge-cloud continuum, schedules based only on the quantity of unused resources cannot achieve low cost and low delay globally simultaneously. Consider Node prices, link conditions, user mobility and service chain constraints simultaneously [5]. Therefore, this serves as the problem basis for building the layered container layout model in this paper.

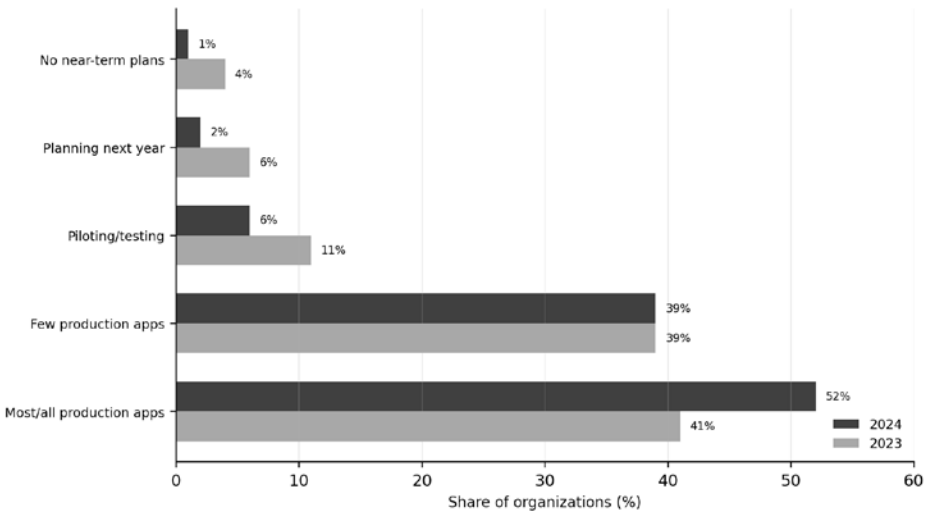

Figure 1. Statistics of Changes in Container Usage Stages

Figure 1 is redrawn based on the annual cloud-native survey data. The English labels are kept in the figure but not in the figure title. According to the above data, the proportion of container use in most or all production applications has risen from 41 per cent in 2023 to 52 per cent in 2024, while the proportions of no near-term usage plans, only planned use, and testing and evaluation stages have all declined. It can be seen from this that the main direction of container production has been adopted, and therefore, the deployment of containers in edge scenarios also needs to shift from a pilot project to an engineering system that is operationally viable, observable, and rollback-capable. The Data Source is Reference [10].

2.3 Research progress on edge lightweighting solutions

In recent years, scholars at home and abroad have begun to study how to optimise the deployment of cloud-native applications in the edge-cloud continuum. Several methods have been used in the research to automatically distribute microservices, reduce communication delay, and improve resource utilisation via mixed-integer programming, greedy heuristics and reinforcement learning [6]. Some research has also put forward context-aware container orchestration frameworks in the area of serverless edge computing that consider both computing resources and wireless bandwidth together to avoid resource fragmentation and reduce convergence time [7]. Based on the above data, edge container arrangement will be implemented dynamically rather than fixedly.

In addition to traditional container runtimes, there is also lightweighting in the form of image and execution environment innovation. Research on lightweight workloads for the edge aims to build a unified scheduling interface for containers, micro-virtual machines and dedicated runtimes to reduce idle agent overhead and expand the resources available to constrained nodes [8]. Recently, comparative studies of lightweight orchestration distributions have also pointed out that different platforms differ in security, maintainability, network disconnection recovery and resource consumption, and engineering selection should be combined with business requirements and on-site operation and maintenance capabilities [9].

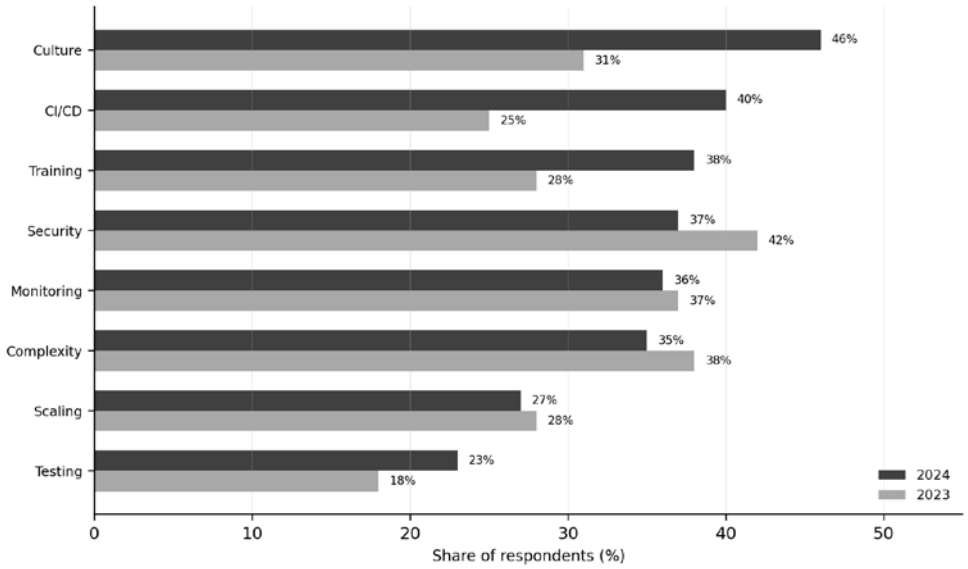

Figure 2. Statistics on the Changing Challenges of Container Deployment

Figure 2 is a reconstruction of the container deployment problem based on the data in the same year's survey. Only the English categories, legends and coordinate descriptions have been kept in the figure. Based on the results, in 2024, the main problems were a lack of cultural collaboration,

continuous integration and delivery, and insufficient training; although security and complexity remained relatively high compared to 2023, they had decreased somewhat. Therefore, the problems in the deployment of edge containers are not limited to technical issues; there are also issues in the organisation's delivery process, skills of personnel on the ground, and platform governance standards. The Data Source is Reference [10].

3. Problem Statement: The Key Challenges of Lightweight Container Layout in Edge Scenarios

3.1 Resource heterogeneity and capacity constraint mismatch

Edge nodes generally include industrial control computers, gateways, mini-servers, micro data centers and dedicated acceleration devices; these have various processor architectures, core counts, memory sizes, disk speeds and accelerator card types. Traditional schedulers are prone to matching static resource requests with the remaining resources on nodes, and edge tasks are often bursty and site-specific; therefore, resource demand changes according to the number of devices, sampling frequency and business hours. If only the average load for container deployment is taken into account, there will be memory jitter, processing queue backlogs and service chain congestion at the peak time.

$$\sum_{i=1}^{M} x_{in}\, c_i \le C_n, \qquad \sum_{i=1}^{M} x_{in}\, m_i \le M_n \tag{1}$$

Equation (1) is the fundamental capacity constraint of the edge node. Here, x is the layout decision between the container instance and the node; c is the container processor requirement; m is the container memory requirement; C is the available processor capacity of the node; M is the available memory capacity of the node; i denotes the container instance; and n denotes the edge node. Therefore, the first requirement for a layout plan is that its resources can be carried by a single node, but even if this condition is met, an ideal layout cannot be achieved solely through capacity constraints.

3.2 Network instability and cumulative service chain latency

The service chain of the edge application generally includes multiple links, such as device access, data cleaning, rule judgment, model inference, result feedback and alarm linkage. If these containers are distributed at different levels of the nodes, link latency will increase with an increase in the number of network hops; if they are all concentrated in one node, it may result in local resource congestion. In the event that a mixed deployment of campus private networks, cellular networks and field LANs is chosen, fluctuations in link quality may make a static layout inefficient. Therefore, when deploying containers, all three will be considered in the overall decision-making process: network distance, queue waiting time and computing service time.

$$L_{in} = d_{un} + q_n + \frac{s_i}{\mu_n} + \delta_i \tag{2}$$

Equation (2) is the total delay when container instance i is deployed on node n. L is the total delay in the formula; d is the time for network transmission and propagation from a user or device to a node; q is the queueing delay at the node; s is the scale of the requested computation; μ is the service rate of the node; and δ is the runtime overhead. The above formula is to show that the purpose of the edge container is not necessarily to be close to the user; it should be close enough so that the network link, queueing delay and compute resources are all acceptable.

3.3 Conflicts between mirrored cold start, state preservation and security boundary

Lightweight containers are fast to start, relatively flexible in migration and have clear image dependencies. The distance to the image repository, link bandwidth, image layer reuse rate and local caching strategy at the edge will all affect startup speed. Only a small image cache is maintained in the nodes to reduce storage, but it will be slow to start up initially; otherwise, large-scale image warming up will take up local disk space and increase version management complexity. Edge containers often deal with sensitive on-site data, and therefore, state data and key materials should not be moved freely across security boundaries.

Table 2 Container Layout Variables and Constraints

Variable	Meaning	Measurement	Constraint role
CPU request	Compute demand per container	Millicores	Capacity check
Memory request	Resident memory footprint	MiB	Admission control
Image size	Compressed image layers	MiB	Cold-start control
Link delay	Device-to-node latency	ms	QoS constraint
State class	Stateless or stateful workload	Class label	Migration policy
Security zone	Trust boundary of node pool	Zone ID	Isolation rule

Table 2 shows the main indicators to be monitored and restricted for container deployment. Processor and memory are required for admission; image size will affect cold start; link latency is related to service quality; state category controls migration permission; and security domain dictates whether containers can be deployed across different domains. This table shows that edge deployment decisions are complex and have many constraints; therefore, a single resource index cannot be used.

4. Problem Solving and Strategies: Cloud-Native Lightweight Container Layered Layout Solution

4.1 Overall Approach: Collaboration between Cloud-based Control and Edge Autonomy

To address the above problems, this paper puts forward a four-tier architecture: "Central Cloud - Regional Edge - Field Nodes - Device-Side Agents". The Central Cloud is responsible for global policies, image repositories, version baselines, policy auditing and cross-regional resource views; the Regional Edge is responsible for batch image caching, model distribution, service gateways and fault tolerance; the Field Nodes are responsible for low-latency service containers, protocol adaptation, real-time analysis and local state preservation; and the device-side agents only perform lightweight data collection, filtering and heartbeat functions. A divided Structure of control and data processing can be set up to run basic business functions at the edge node without a stable network.

Deployment aims to lower the total cost of ownership of a resource, not to maximise its use rate. The three components of the total cost are latency cost, load imbalance cost and energy consumption cost. At the same time, hard constraints are set to exclude illegal deployments, such as resource over-provisioning, security domain conflicts and non-transferable states. Thus, all containers are far from the end node and there will be no single cloud bottleneck.

$$X^* = \arg\min_X [\alpha L(X) + \beta U(X) + \gamma E(X)] \tag{3}$$

Equation (3) is the all-encompassing optimisation objective of the container layout. X* is the optimal layout decision matrix in the equation, X is the set of candidate layouts, L(X), U(X) and E(X) are the latency cost, load imbalance cost and energy consumption cost of the layout scheme, respectively, and α, β and γ are the weights of the three types of costs. The above equation shows that lightweight container layout should aim for an engineering-feasible solution through a multi-objective weighted approach rather than pursuing the theoretical optimum of a single indicator.

4.2 Node layering and container grouping strategy

The first layer of a multi-layered layout is node designation. Central cloud nodes do not directly handle millisecond-level response services but are responsible for maintaining policy consistency and version authority; regional edge nodes have high caching and computing capabilities to handle cross-site shared model inference, message aggregation and service gateways; field nodes are close to the devices and should be used to deploy protocol parsing, anomaly detection and real-time control containers; and device-side proxies are responsible for lightweight sampling, edge filtering and breakpoint resumption. Design roles to reduce unnecessary cross-layer migration in the system and set a stable boundary for container placement.

The three-category separation rule for container grouping is as follows: separate real-time paths from non-real-time paths, separate stateful containers from stateless containers, and separate security-sensitive containers from ordinary business containers. Real-time path containers should be deployed first at the on-site nodes or the edge of the region, and non-real-time statistics and reporting containers can be aggregated in the cloud. Stateless containers can be used for elastic scaling, and stateful containers need to be bound to local volumes or replicated in a region. Security-sensitive containers must be restricted to trusted node pools, and access should be controlled by image signing, least privilege and network policies.

Table 3 Comparison of Various Container Layout Strategies

Strategy	Latency	Resource use	Autonomy	Operation risk	Suitable scenario
Cloud-only layout	High	Centralized	Low	Backhaul pressure	Offline analytics
Single-edge layout	Low	Local bottleneck	Medium	Node overload	Small site
Regional-edge layout	Medium	Balanced	Medium	Gateway dependency	Campus network
Layered lightweight layout	Low-medium	Adaptive	High	Policy complexity	Industrial edge
Hybrid runtime layout	Low-medium	Fine-grained	High	Runtime governance	Heterogeneous nodes

Table 3 shows the different Container Deployment Strategies. Centralised cloud deployment is simple to operate and maintain but has high latency; a single edge node deployment offers a fast response but is susceptible to local bottlenecks; regional edge solutions offer a good trade-off but rely on the stability of the gateway. Layered lightweight deployment can achieve a better trade-off among low latency, reduced resources and weaker autonomy; therefore, stronger policy governance and monitoring are required.

4.3 Lightweight Image, Cache Preheating and Startup Path Optimization

Optimisation of edge container startup will be carried out in the three stages of image building, distribution and runtime. Reduce the number of base layers in image building, remove compile-time dependencies, use a multi-stage build approach, and separate common dependency layers from business code layers to improve edge cache hit rate. A relay cache for images should be set up at the regional edge during image distribution, and only high-frequency business images and the most recently rolled-back image need to be stored locally in the node. At runtime, the critical containers need to be pre-pulled, pre-decompressed and checked for health before the initial response time of the alerting, control and inference services.

$$T_i^s = \frac{S_i}{B_n} + T_i^u + T_i^a \tag{4}$$

Equation (4) is the container startup time. In the equation, the superscript s of T is the total startup time, S is the image size, B is the available pull bandwidth of the node, the superscript u of T is the image decompression time, and the superscript a of T is the application initialization time. Therefore, it can be seen that edge-side cold starts are not solely caused by the container runtime; image size, cache location, network bandwidth, etc., are also factors. At present, image-layer compression will be used to improve the local hit rate, and preheating strategies will be set up for high-priority containers in the cache.

4.4 Comprehensive scoring, scheduling, and closed-loop operation and maintenance mechanism

The closed-loop mechanism for the specific scheduling process is "hard constraint filtering - comprehensive scoring and ranking - local autonomy confirmation - operational status feedback". First, nodes that lack sufficient resources, are not in the right security domain, have non-transferable states, or are missing dependent services are excluded; then, a comprehensive score is calculated for the candidate nodes; next, the edge-side autonomous controller makes the final confirmation based on local links, queues and health status; finally, the deployment results, performance indicators and abnormal events are sent back to the cloud side for subsequent policy updates. Thus, a central cloud system is established to serve as a global hub for real-time data processing at the edge.

$$P_{in} = \omega_1\left(1 - \hat{L}_{in}\right) + \omega_2\left(1 - \hat{U}_n\right) + \omega_3\hat{R}_n + \omega_4\hat{H}_n \tag{5}$$

Equation (5) is the score of the candidate node. P is the overall score of container i deployed on node n, the capped variable is a normalised index, L is latency, U is utilization, R is node reliability, H is cache hit or health, and ω is the weight of each index. A relatively large value is expected for a node that can host a particular container. By normalisation and weight adjustment, various deployment strategies for different situations in industrial control, video analytics and park management can be built on the platform.

5. Implementation Evaluation and Application Discussion

The four steps in the application of this solution are: establishment of a base, building of layers, policy improvement and continuous supervision. The construction of the base phase needs to finish node resource profiling, network quality measurement, service latency classification and security domain division. The Layered Deployment Phase needs to build a central cloud control plane, regional edge caching planes and on-site execution planes. Revise latency weights, caching thresholds and migration rules in the policy optimisation stage according to the operational data. Set

up an all-weather normal mode for managing image vulnerabilities, node health, service alerts and version rollbacks.

In resource-constrained environments, the full central cloud orchestration function of a platform does not need to be deployed at all edge nodes. Therefore, lightweight node proxies, edge autonomous controllers and regional caching services should be selected in combination. For gateways with weak computing power, only protocol conversion, data filtering and a small rule engine need to be used. For small edge stations that have a graphics processor or a neural network accelerator, a video analytics and model inference container can be deployed. Regional edge clusters can offer cross-site load balancing, image distribution and a unified observer.

Reliability: Edge container deployments must consider the possibility of a network outage at any time. When the central cloud cannot be contacted, the local nodes should still operate based on local policies, cache essential data, and maintain container health checks and local alerts; upon the restoration of the network, event logs, operational indicators, and incremental changes in state need to be synchronized with the cloud. Data consistency, volume reachability and security domain authorisation must be confirmed for stateful containers before migration; for stateless containers, multiple replicas and proximity scheduling can be used to improve elasticity.

Lightweight Design does not mean that it is not secure. Sign and Scan Edge Container Images for Vulnerabilities. Limit the scope of privileged mode activation, mount paths and host network access at runtime. Limit the privileges of inter-service communication via a least-privilege model. Given that the edge nodes are often in unattended environments, trusted boot, key rotation, log tamper-proofing, and node anomaly isolation mechanisms need to be implemented to prevent a single point of failure from propagating to the regional edge or the central cloud.

In terms of cost-effectiveness, a multi-tier, lightweight structure can reduce backhaul pressure on the central cloud and lower the latency of on-site response; however, this will increase the costs of edge-side caching, observation and policy governance. Engineering Decisions should consider business value, node scale, operating conditions and security requirements all at the same time. High-real-time, high-value and high-risk enterprises can be assigned an on-site-first and regional backup layout; low-real-time and offline-processable enterprises can be opted for a regional aggregation or cloud-based centralized layout.

6. Conclusion

This paper introduces the deployment of cloud-native lightweight containers in edge computing environments. Based on English literature from the past three years and annual cloud-native survey statistics, this paper analyzes the development trends of container production, lightweight orchestration and edge autonomy, and proposes a layered deployment scheme for engineering implementation. The research results are as follows.

First, the main problem of edge container deployment is that the orchestration capabilities of the central cloud do not match the resource-constrained environment at the edge. Reducing the default orchestration mode of the central cloud simply by deploying it at the edge will not solve problems such as heterogeneous resources, weak networks, disconnected autonomy and on-site security.

Second, the Design of a lightweight container deployment also needs to coordinate several modules, such as node layering, container classification, image caching, state management, and security isolation. Real-time services should be deployed closer to the device, and non-real-time services can be converged at the regional edge or in the central cloud. State Containers and security-sensitive Containers should have stricter migration and access boundaries.

Third, the all-encompassing optimisation model in this paper balances several goals simultaneously, such as low latency and load, energy consumption and reliability, and high cache

hit rate. A hard-constraint filter and a full-scoring mechanism can be used to construct an explainable, adjustable and feasible container placement strategy for all kinds of edge business cases.

Fourth, in the future, some real-world edge cluster operation data can also be used to conduct empirical investigations across various sites, networks and runtime environments. Intelligent prediction, anomaly detection and adaptive weight adjustment can be added to the closed-loop system of container layout to improve the long-term autonomy of edge cloud-native platforms.

References

[1] *R. Vaño, I. Lacalle, P. Sowiński, R. S-Julián, and CE Palau, "Cloud-Native Workload Orchestration at the Edge: A Deployment Review and Future Directions," Sensors, vol. 23, no. 4, 2215, 2023.*

[2] *I. Čilić, P. Krivić, I. Podnar Žarko, and M. Kušek, "Performance Evaluation of Container Orchestration Tools in Edge Computing Environments," Sensors, vol. 23, no. 8, 4008, 2023,*

[3] *H. Koziolek and N. Eskandani, "Lightweight Kubernetes Distributions: A Performance Comparison of MicroK8s, k3s, k0s, and Microshift," in Proceedings of the 2023 ACM/SPEC International Conference on Performance Engineering, 2023..*

[4] *A. Detti, "Microservices From Cloud to Edge: An Analytical Discussion on Risks, Opportunities and Enablers," IEEE Access, vol. 11, pp. 49924–49942, 2023.*

[5] *Wu, Y. (2026). Research on Interpretable Confidence Rule Base Modeling Method Integrating Data-Driven and Constrained K-Means Optimization. Procedia Computer Science, 279, 612-619.*

[6] *Sun, J. (2026). Automated Feature Engineering and Screening System for Large-Scale Factor Libraries. Procedia Computer Science, 281, 1282-1290.*

[7] *Hou, Y. (2026). Collaborative Regulation for Stable Data Center Operation under Energy Efficiency Constraints. Procedia Computer Science, 281, 612-620.*

[8] *Zhang, Z. (2026). Research on Performance Optimization Methods for Resource-Aware Model Services in AI Systems. Procedia Computer Science, 281, 1310-1317.*

[9] *Chang, C. W. (2026). Privacy Infrastructure Vulnerability Mining and Automated Framework Based on Multimodal AI. Procedia Computer Science, 279, 702-711.*

[10] *Liu, Z. (2026). Research on Measuring User Behavior Response Differences Supported by Propensity Scoring Method. European Journal of AI, Computing & Informatics, 2(2), 47-53.*

[11] *Gao, Y. (2026). Governance Structures and Checks and Balances in Private Equity Funds: Practice, Problems, and Improvement Paths. Financial Economics Insights, 3(2), 82-91.*

[12] *Liang, Q. (2026). Research on Low-Energy Space Organization Methods in the Context of Building System Integration. Global Journal of Science & Innovation, 3(1), 9-15.*

[13] *Zhang, Z. (2026). Research on Performance Monitoring and Data-driven Optimization Mechanisms for AI Systems Oriented Toward Decision Support. Advances in Computer and Communication, 7(2).*

[14] *Wu, Y. (2026). Research on Interpretable Confidence Rule Base Modeling Method Integrating Data-Driven and Constrained K-Means Optimization. Procedia Computer Science, 279, 612-619.*

[15] *Wu, L. (2026). Construction and Evolutionary Analysis of a Game Model for Supply Chain Finance Funding Based on Blockchain Technology. Procedia Computer Science, 282, 2004-2012.*

[16] *Ding, J. (2025, December). Research on Logistics Cost Control and Optimization in Automotive Manufacturing Supply Chain Based on DMAIC Model. In 2025 IEEE 1st*

International Conference on Recent Trends in Computing and Smart Mobility (RCSM) (pp. 1-7). IEEE.