

Design of Collaborative Architecture for Android Mobile Smart Systems Based on Edge Computing

Zheng Li

Jacobs School of Engineering, University of California, San Diego, La Jolla, 92093, California, United States

Keywords: Android; edge computing; mobile intelligent system; task offloading; resource collaboration; low latency

Abstract: Given the problems of limited computing resources, unstable network conditions, high cloud round-trip latency, and privacy risks in complex perception, real-time inference, and multimedia processing of Android mobile intelligent applications, this paper proposes an edge-cloud collaborative architecture. Add a task profiling, device state awareness and security caching module on the Android side, and set up service registration, dynamic scheduling, model caching and elastic resource pools at the edge. A multi-objective cost function is employed to optimise the ratio of task segmentation, edge node selection and computing resource allocation simultaneously. Monte Carlo simulations are carried out with publicly available reproducible parameters, and each data point is independently sampled 1200 times. Based on the above results, the proposed architecture can achieve an average response latency of 436.2 ms for a 6 MB task; this is a reduction of 60.3% and 59.0% compared with pure local execution and cloud execution, respectively, and reduces device-side power consumption by 84.1% over local execution. Therefore, the cooperative offloading of Android intelligent systems should not be based on a simple binary decision of "local or cloud", but should incorporate factors such as task granularity, link quality, device power consumption and edge load into a unified control loop.

1. The Driving Forces Behind the Increased Computing Load of Android Smart Applications and the Collaboration Between End-Edge-Cloud

Android mobile phones have developed from basic display devices into intelligent nodes that can perceive, interact, reason and control. Computer vision applications, speech recognition, augmented reality, mobile health and intelligent inspection all need to process high-dimensional sensor data continuously and simultaneously meet constraints on response latency, energy consumption and privacy protection. The old local execution has limitations in CPU, memory and battery capacity, and a full switch to the central cloud will be affected by wide-area network jitter, link congestion and remote service unreliability. Edge computing places computing, caching and service capabilities close to the access network to provide a low-latency execution area for Android applications in the local-cloud environment.

Although the previous studies have put forward task offloading to address the problem of weak

terminal computing power, engineering deployment still faces three types of contradictions: First, applications have dependencies and different security requirements, and coarse-grained whole-packet offloading may significantly increase transmission costs; Second, due to changes in terminal mobility, wireless bandwidth and edge load, stable static thresholds are difficult to set; Third, there is a trade-off among latency, energy consumption, reliability and privacy, and optimal performance in a single indicator does not guarantee a good system experience. To solve the above problems, this paper puts forward an edge-cloud collaborative architecture for the Android mobile intelligent system and builds a decision-making closed-loop based on a computable task model and multi-objective strategies. The first three contributions of this paper are as follows: Construction of a task profiling mechanism that focuses on the Android lifecycle and component-based development; Design of an edge collaboration layer supporting parallel partitioning, node selection and fault fallback; Establishment of a joint optimisation model for latency, energy consumption and risk; Verification of the performance limits of the architecture through statistical simulation.

2. Research Progress on Android Edge Task Unloading and Resource Co-operation Optimization

Akhlaqi and Mohd Hanapi have systematically studied the task model, resource allocation and future problems of mobile edge computing offloading, and indicated that multi-objective decision-making in dynamic network environments is still a major issue [1]. Lu and Yan proposed an offloadable task partitioning method based on the reconstruction of mobile applications, and showed that the internal structure of an application directly affects offloading benefits [2]. Chen and others proposed FUNOff to refine the application offloading granularity to the function level, and verified the feasibility of dynamically adjusting the offloading scheme at mobile terminals [3].

Wang and others have used online learning to address the distributed offloading problem in the presence of unknown system states, and a long-term decision-making under uncertain feedback is required [4]. Kim and others have jointly modelled user association, resource allocation and full task offloading, as well as battery status, and have shown that cooperative optimisation can reduce both latency and energy consumption [5]. Peng and others have pointed out that deep reinforcement learning faces problems of state-modeling, action-space and training-stability for edge offloading, and both model complexity and online inference overhead need to be addressed at the same time [6].

Dong et al. have classified offloading strategies as latency minimisation, energy consumption minimisation and service quality balance, and further indicated that the unified evaluation framework is still missing [7]. Dayong and others have summarized the task offloading mechanism in multi-access edge computing for IoT terminals and indicated that device heterogeneity and multiple communication standards will increase the difficulty of strategy migration [8]. Loutfi and others have studied the problem of edge node switching in terms of mobility perception and service migration, and believe that mobility management is required for uninterrupted service [9]. Lu and others have added data security to the deep reinforcement learning offloading framework, and security processing will add extra latency and energy consumption overhead [10].

In short, the main directions of recent studies have been task granularity, online learning, resource allocation, mobility and security. Most of the above works only focus on a particular algorithm or scenario and lack an engineering-based collaborative architecture that integrates Android application components, terminal state management, edge service governance and cloud policy updates. Therefore, this paper uses "task separability—resource availability—link reachability—data sensitivity" as the unified decision inputs to improve the feasibility of the solution in practice for mobile intelligent systems.

3 System Constraints and Joint Cost Model in Dynamic Task Unloading Scenarios

3.1 Android Task Profiling and Edge-Cloud Operation Status Modeling

Suppose that an Android terminal generates a set of tasks within a time slice $N=\{1,2,\dots,n\}$. Each task i is described by the amount of input data, the required CPU cycles, the maximum allowable latency, and the security level.

$$J_i=(d_i, c_i, \tau_i, s_i) \quad (1)$$

In the formula, d_i represents the amount of data to be processed, c_i represents the number of CPU cycles required to complete the task, τ_i represents the business deadline, and s_i represents the data sensitivity level. The Android task profiler extracts context from WorkManager, foreground/background status, network type, battery temperature, and memory usage, and determines whether the task can be split, cached, or executed remotely based on this context.

3.2 Terminal Local Execution Latency and Processing Load Model

If only the mobile phone is used to complete the task, the calculation delay will be relatively long because it has a small number of cores.

$$T_i^{\text{loc}} = \frac{c_i}{f_i^m} \quad (2)$$

Here, f_i^m represents the effective CPU frequency allocated to the task by the terminal i . This formula reflects that, with a fixed workload, increasing the frequency can shorten latency, but it will exacerbate dynamic power consumption and heat accumulation. Therefore, the system cannot meet real-time requirements solely by increasing the terminal frequency.

3.3 Edge Delay and Terminal Energy Consumption Model under Partial Offloading Conditions

Let $x_i \in [0,1]$ represents the task offloading ratio, and j represents the selected edge node. Edge-side execution latency consists of uplink transmission, edge computing, result return, and scheduling overhead.

$$T_{i,j}^{\text{edge}} = \frac{x_i d_i}{r_{i,j}^{\uparrow}} + \frac{x_i c_i}{f_{i,j}^e} + \frac{x_i d_i^{\text{out}}}{r_{j,i}^{\downarrow}} + t_{i,j}^q \quad (3)$$

In the formula, $r_{i,j}^{\uparrow}$ and $r_{j,i}^{\downarrow}$ represent the uplink and downlink rates, respectively; $f_{i,j}^e$ is the computation frequency allocated to the edge node; d_i^{out} is the output data volume; $t_{i,j}^q$ is the queuing and scheduling overhead. The unloaded portion and the edge portion are executed in parallel, and the task completion time is the maximum of the two, thereby avoiding the accumulation of waiting caused by serial unloading.

Device-side energy consumption includes local computing energy consumption, data transmission energy consumption and waiting energy consumption.

$$E_i = E_i^{\text{loc}} + E_i^{\text{tx}} + E_i^{\text{wait}}$$

$$E_i^{\text{loc}} = \kappa_i (1-x_i) c_i (f_i^m)^2$$

$$E_i^{\text{tx}} = p_i^{\text{tx}} \frac{x_i d_i}{r_{i,j}^{\uparrow}}$$

$$E_i^{\text{wait}} = p_i^{\text{idle}} \left(\frac{x_i c_i}{f_{i,j}^e} + t_{i,j}^q \right) \quad (4)$$

Where , E_i^{loc} and E_i^{tx} represent E_i^{wait} the energy consumption for local computation, data transmission, and waiting, respectively; κ_i is the effective switching capacitor coefficient of the chip; p_i^{tx} and p_i^{idle} represent the transmission power and waiting power, respectively. This model illustrates that edge computing is not always energy-efficient; when link rates are low or queuing times are long, transmission and waiting energy consumption may offset offloading benefits.

3.4 Definition of Multi-Constraint Unloading Ratio Node Selection and Resource Allocation Problem

The core problem this paper addresses is: how to select the offloading ratio x_i , target nodes j , and edge computing share to minimize global service costs while ensuring resource capacity and service deadlines, given changes in task dependencies, terminal resources, edge load, link status, and security constraints. Using only fixed thresholds leads to frequent policy switching during network fluctuations; relying entirely on complex reinforcement learning increases edge-side inference overhead. Therefore, it is necessary to combine an interpretable cost model with lightweight policy updates.

4. Edge-Cloud Collaborative Architecture and Multi-Objective Scheduling Strategy for Android Intelligent Tasks

4.1 A three-tier collaborative architecture integrating task profiling and elastic execution pools: edge-cloud and endpoint-based architecture

Table 1. Module Configuration and Functional Responsibilities of the Three-Tier Collaborative Architecture for End-Edge-Cloud

Layer	Main Components	Main Input	Main Output	Design Role
Android Device	UI, Sensor Adapter, Task Profiler, Local Executor, Secure Cache	Sensor streams, user requests, device state	Local result, task descriptor, encrypted payload	Context sensing and lightweight execution
Edge Collaboration	API Gateway, Service Registry, Scheduler, Model Cache, Resource Monitor	Task descriptor, channel state, security level	Execution result, resource decision, migration event	Low-latency processing and cross-node coordination
Cloud Control	Global Trainer, Policy Repository, Historical Analytics, Orchestrator	Aggregated metadata, anonymized logs	Model version, policy parameters, global rules	Long-term optimization and lifecycle governance

The three layers of the system are an Android device, an edge collaboration area and a cloud control centre. The modules of the Android device layer are as follows: A task profiler to determine data volume, CPU cycles and deadlines; a status sensor that collects CPU, memory, temperature, battery level, network type and signal strength; a local executor for handling non-unloadable tasks;

and a secure cache to store intermediate results, model versions and failure retry information. Components of the edge collaboration layer include an API gateway, service registry, task splitter, resource monitor, elastic execution pool and model cache to support service migration or fault rollback across multiple edge nodes. Not all the real-time calculations are performed by the cloud control layer; it is instead used for the global model training, historical log analysis, publishing policy parameters and cross-regional resource orchestration.

Table 1 shows that the three-tier structure separates real-time closed-loop processing from long-term optimisation. The Android client keeps interaction and state acquisition to perform some local processing in the event of a network failure; the edge layer handles high-frequency decision-making and low-latency execution; and the cloud is only used for cross-regional training, historical analysis and policy deployment to reduce the strong dependency of real-time services on the central cloud

4.2 Dynamic Offloading Decision Mechanism Driven by Joint Delay, Energy Consumption, and Security Risks

Given the latency, energy consumption and security risks, the general optimisation objectives are as follows:

$$\begin{aligned} \min_{\{x_{i,j}, f_{i,j}^e\}} & \sum_{i=1}^n [\alpha T_i + \beta E_i + \gamma R_i(s_{i,j})] \\ \text{s.t.} \quad & T_i \leq \tau_i, \quad \sum_i f_{i,j}^e \leq F_j^e, \quad 0 \leq x_i \leq 1 \end{aligned} \quad (5)$$

In the formula, T_i represents the completion latency after parallel execution at both local and edge, $R_i(s_{i,j})$ represents the risk cost resulting from the mismatch between security level and node trustworthiness, F_j^e represents j th the available computing capacity of the node, α and β and γ represent the configurable weights for the business. Improvements are made for interactive applications α , low-power scenarios β , and when identity, medical, or location data are involved γ . The policy execution adopts a four-step process: "candidate generation—constraint filtering—cost ranking—running feedback." First, candidates for local, partial, complete, and cloud rollback are generated based on task separability; then, overdue, resource-insufficient, and security-level mismatched solutions are eliminated; subsequently, they are ranked according to the objective function; finally, the actual latency, energy consumption, and failure status are fed back to the edge policy cache.

4.3 Reliable security mechanism based on checkpoint migration and least privilege control

The end point has an idempotent flag and checkpoints for all tasks to ensure fault tolerance. If an edge node fails, it can be moved to the nearest checkpoint without re-uploading all the input data. For high-real-time applications, a small fraction of local execution paths have been preserved in the system to reduce the problem of edge unreachability. Mobile scenarios with frequent network switching use a hysteresis window strategy to reduce jitter, and migration is only triggered when a new solution is better than the current one for several consecutive time slices.

For security reasons, the task profiler decides whether to allow the data to leave the terminal based on the sensitivity of the data; unloadable data is encrypted at the session level and uses short-term tokens, and edge services only have the minimum permissions required to complete the current task. Logs only upload anonymous statistics and do not save raw perception data. A model and policy package are added to the Android security cache after signature verification, and if a version anomaly occurs, the latest trusted version is automatically reverted.

5 Monte Carlo simulation verification of collaborative offloading delay and energy consumption under multi-task scale

5.1 Setting Random Parameters and Comparison Schemes for Edge-Cloud Execution Scenarios

Build an edge-cloud task execution simulation in this paper to test how the architecture's performance changes. Task sizes are set to 1, 2, 4, 6, 8 and 10 MB, and computational density, terminal frequency, link rate and node computing power are distributed by truncated normal or log-normal random distributions. Sample each data point 1200 times independently, and the error bars in the figure are 95% confidence intervals. The three comparison modes are local execution only, full cloud offloading, and the collaborative edge strategy proposed in this paper. The simulation parameters are not the same as those of some commercial products, so a comparison of the architecture mechanism under uniform random conditions was carried out. The task-load and communication/computation modeling choices follow the reproducible MEC simulation setting reported by Kim et al. [5], including 3GPP small-cell channel modeling and AI-task load profiles; the numerical means in Table 2 are controlled scenario parameters for comparative simulation rather than measurements of a specific commercial platform [11-12].

Table 2: Parameter Range and Probability Distribution of Edge-Cloud Task Offloading Simulation

Parameter	Distribution or Value	Unit	Purpose
Task Size	1, 2, 4, 6, 8, 10	MB	Input workload scale
CPU Cycles per MB	Lognormal, mean 4.2×10^8	cycles/MB	Computational intensity
Mobile CPU	Normal, mean 2.35	GHz	Local execution capacity
Edge CPU Allocation	Normal, mean 18	GHz	Near-edge execution capacity
Cloud CPU Allocation	Normal, mean 45	GHz	Remote execution capacity
Edge Uplink Rate	Normal, mean 155	Mbps	Access-network transmission
Cloud Uplink Rate	Normal, mean 58	Mbps	Wide-area transmission
Monte Carlo Repetitions	1200	runs/point	Statistical estimation

Table 2 shows that the parameters jointly represent the end-point difference, wireless fluctuations and server resource changes. Edge links have higher average speeds and lower round-trip latencies, but cloud computing power is relatively strong and wide-area transmission costs are relatively high. Repeatedly sample at all levels to prevent the excessive influence of a single random state on the results and provide a confidence interval for the mean estimate.

As shown in Figure 1, all three schemes can complete the processing at a task size of 1 MB; however, the collaborative edge strategy has a lower average latency. As the input size increases to 6-10 MB, the latency of local computing and cloud transmission rises almost linearly; at this time, the collaborative strategy can use task partitioning and edge parallel execution to significantly reduce the growth slope and is thus more suitable for continuous sensing and real-time inference workloads [13-14].

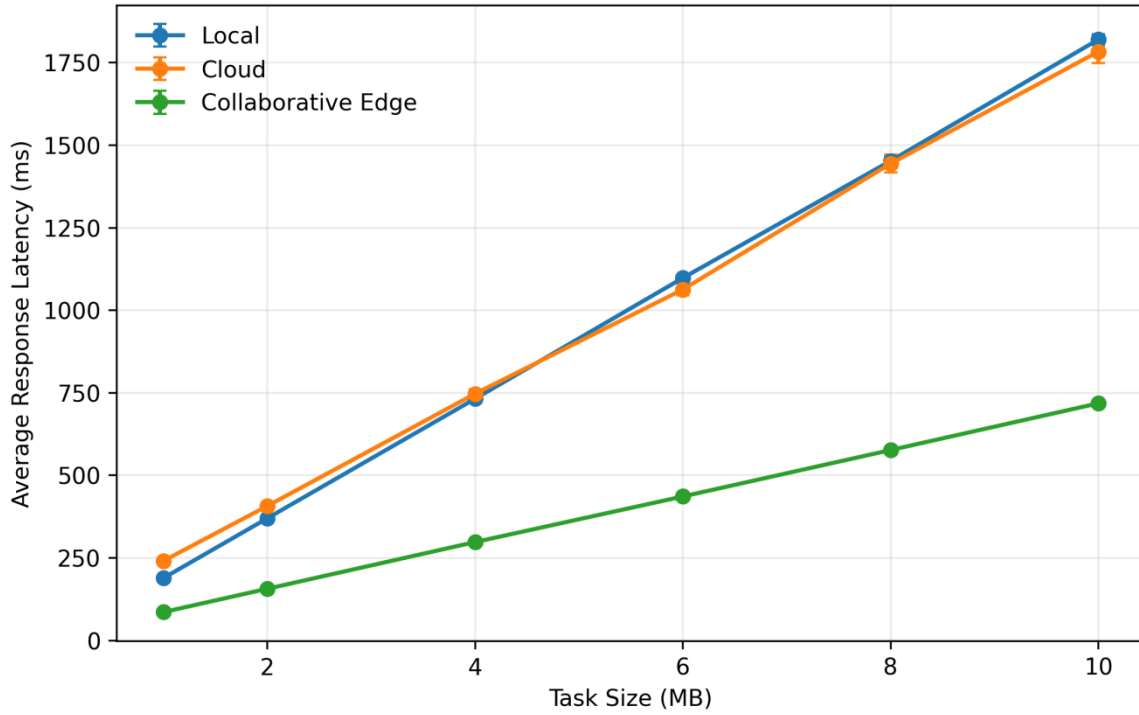


Figure 1. Average Response Latency of Local Cloud and Collaborative Edge Strategies at Different Task Scales.

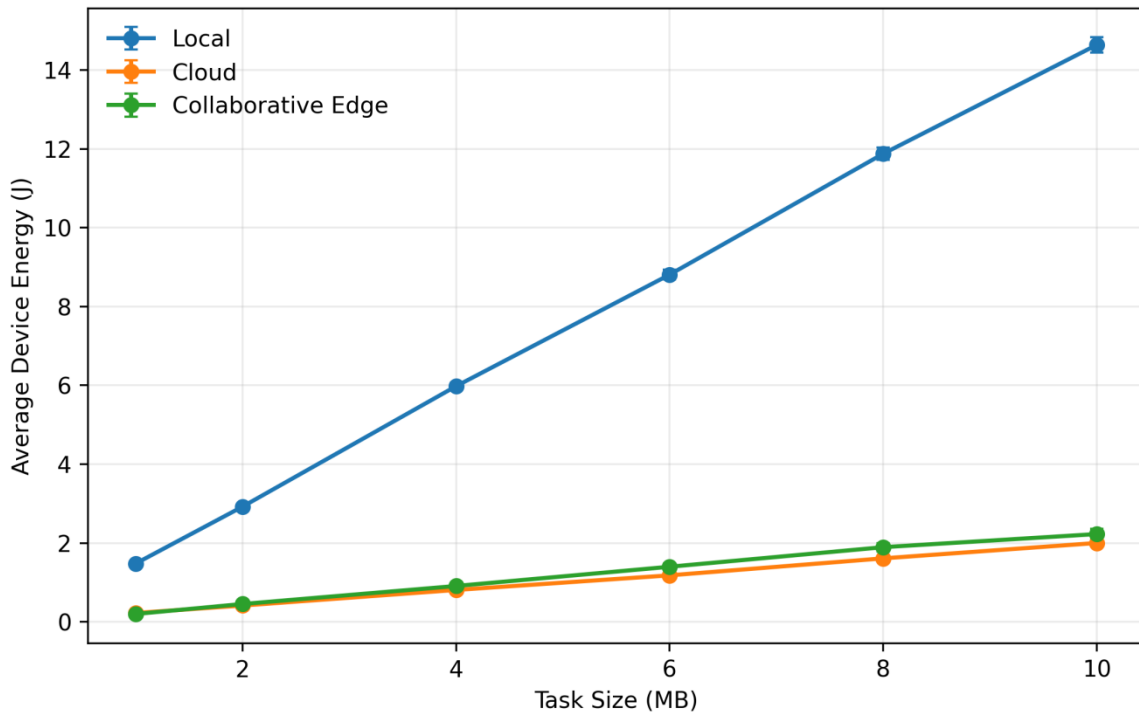


Figure 2. Device-side Power Consumption of Local Cloud and Collaborative Edge Strategies at Different Task Scales.

As shown in Figure 2, the energy consumption of purely local execution increases rapidly with the size of the task, and the average energy consumption for a 10MB task exceeds 14J. Full cloud

offloading reduces device energy consumption but adds high-latency networks. The collaborative edge strategy consumes slightly more energy than the cloud solution but is significantly lower than local execution, and as shown in Figure 1, achieves a considerable latency advantage; therefore, it can be regarded as "a small transmission energy consumption for a large response improvement".

5.2 Performance Improvements and Applicability Boundaries of Three Execution Strategies under Representative Loads

Table 3. Comparison of Response Latency and Device Power Consumption for the Three Execution Strategies of a 6MB Task.

Method	Average Latency (ms)	Device Energy (J)	Latency Reduction vs Local	Energy Reduction vs Local
Local	1097.9	8.81	0.0%	0.0%
Cloud	1063.2	1.18	3.2%	86.6%
Collaborative Edge	436.2	1.40	60.3%	84.1%

As shown in Table 3, for a 6MB task, although the pure cloud solution significantly reduces device power consumption, its latency is only 3.2% lower than that of the local solution; thus, the advantage of remote computing power is offset by wide-area transmission. The cooperative edge solution has reduced the average latency to 436.2ms and maintained the 84.1% reduction in local power consumption; thus, by dividing tasks, near-end execution and dynamic resource allocation, a better quality of service can be achieved.

Based on the above results, the strengths of the collaborative architecture can be summarised as follows. First, tasks are no longer all-or-nothing in terms of offloading; instead, they dynamically select a ratio based on link and computing power to achieve parallel local and edge computing. Second, the low round-trip latency of the edge node reduces the waiting time for data upload and result return. It should be noted that when the volume of task data is small, the terminal has a relatively high battery, the network is unstable, and local execution may still be more practical; when the task does not require low latency and the device battery is very low, the cloud solution will have lower terminal power consumption. Therefore, the architecture shown in this paper uses switching based on business importance rather than employing fixed edge nodes.

6 Performance conclusions and prototype verification directions of the Android edge-cloud collaborative architecture

This paper proposes an edge-cloud cooperative architecture for Android mobile intelligent systems that integrates task profiling, terminal state awareness, partial offloading, edge resource scheduling, security control and cloud policy updates into a single closed-loop system. Five interpretable models are used to describe the characteristics of the task, local latency, edge latency, device power consumption and multi-objective optimisation goals, and statistical simulations are performed under the same random parameters. Based on the above results, a cooperative edge strategy can reduce the response delay of medium- to high-load scenarios substantially and lower power consumption at the device level compared with standalone operation. The above architecture is suitable for mobile vision, voice interaction, intelligent inspection and multi-sensor fusion.

There are still some deficiencies in the study: temperature-controlled frequency reduction, cellular network switching and cold start for edge containers on actual Android devices have not

been covered; security risks have been expressed in terms of cost, and the edge-side overhead of specific encryption algorithms has not been measured. Future work may include building prototypes on real Android devices and Kubernetes edge clusters, collecting real-world data with Perfetto, Battery Historian and network tracing tools, and further exploring multi-device cooperation, federated policy learning and cross-edge node service migration.

References

- [1] Akhlaqi MY, Mohd Hanapi ZB Task offloading paradigm in mobile edge computing—current issues, adopted approaches, and future directions. *Journal of Network and Computer Applications* , 2023, 212: 103568. DOI: 10.1016/j.jnca.2022.103568.
- [2] Lu H., Yan Z. Mobile application tasks refactoring and offloading in Edge Environments. In: *Proceedings of the 2023 7th International Conference on Control Engineering and Artificial Intelligence (CCEAI 2023)* , 2023: 17–22. DOI: 10.1145/3580219.3580224.
- [3] Chen X., Li M., Zhong H., Chen X., Ma Y., Hsu C.-H. FUNOff: Offloading Applications at Function Granularity for Mobile Edge Computing. *IEEE Transactions on Mobile Computing* , 2024, 23(2): 1717–1734. DOI: 10.1109/TMC.2023.3240741.
- [4] Wang, B. (2025). *Methods of Load Optimization for Computer Systems Based on Physical Principles*.
- [5] Yu, X. (2026). *Exploration of Multi-Channel Conversion Path Optimization Methods Based on A/B Testing*.
- [6] Guo, X. (2025, March). *Research on Blockchain-Based Financial AI Algorithm Integration Methods and Systems*. In *2025 IEEE International Conference on Electronics, Energy Systems and Power Engineering (EESPE)* (pp. 816-821). IEEE.
- [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] Liu, X. (2026). *Research on the Application of Artificial Intelligence in Consumer Privacy Data Protection*. *Procedia Computer Science*, 279, 956-965.
- [9] Guo, X. (2025, April). *Research on Financial Trading Algorithms Based on Deep Reinforcement Learning*. In *2025 IEEE 3rd International Conference on Control, Electronics and Computer Technology (ICCECT)* (pp. 1711-1715). IEEE.
- [10] Liu, X. (2026). *Construction of Consumer Data Privacy Protection System Based On Blockchain Technology*. *Procedia Computer Science*, 282, 2082-2091.
- [11] Yang, Y. (2026). *Causal Inference-based Identification of Incremental Effects in Digital Advertising and Optimization Pathways for Resource Allocation*. *Advances in Computer and Communication*, 7(2).
- [12] 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.
- [13] Yuan, Y. (2026). *Research on Memory Management and Dynamic Reasoning Methods for Intelligent Agents Based on Large Language Models*.
- [14] Sun, L. (2026). *Consistency Optimization Strategies for Distributed Advertising Systems Under Cross-regional Deployment*. *Advances in Computer and Communication*, 7(2).