World Journal of Innovation and Modern Technology, Vol. 7, Issue 2, (Apr) **2024**
ISSN 2682-5910
DOI: 10.53469/wjimt.2024.07(02).06

Automated Compatibility Testing Method for Distributed Software Systems in Cloud Computing

Weike Ding[1], Huiming Zhou[2,*], Hao Tan[3], Zihan Li[4], Chao Fan[5]

[1]Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Champaign, Illinois, USA
[2]Computer Science, Northeastern University, CA, USA
[3]Computer Science and Technology, China University of Geosciences, Bejing, China
[4]Computer Science, Northeastern University, San Jose, CA, USA
[5]Information Science, Trine University, Phoenix, AZ, USA
[*]*Corresponding author: zhouhuiming2021@gmail.com*

Abstract: ***In order to avoid the conflict between software systems and improve the overall coordination performance of software, an automatic detection method of distributed software system compatibility in cloud computing is proposed. Build the basic architecture of cloud computing and explore various service modes provided by cloud computing in software compatibility detection; The basic particle swarm optimization model is established, and the weights and learning factors are adjusted. This article explores the challenges of ensuring software compatibility for distributed systems in a cloud computing environment and describes methods for integrating particle swarm optimization algorithms for compatibility testing. Based on the analysis of distributed software system characteristics, existing compatibility testing methods and distributed system compatibility testing tools, a collaborative application of TLA+ and Jepsen methods is proposed to improve the compatibility testing practice in cloud computing environment. The methodology of integrated particle swarm optimization is described in detail, including initialization, fitness evaluation, particle updating, convergence detection, dispersion adjustment and fitness function modification, and the effectiveness of the algorithm in improving compatibility coverage, efficiency and robustness is verified by experimental results. Finally, the role of integrated particle swarm optimization algorithm in optimizing software performance and reliability in cloud computing environment is summarized, and its importance to improve the seamless and compatibility of software operation is emphasized.***

Keywords: Automatic detection; Distributed software system compatibility; Cloud computing; Particle swarm optimization.

1. INTRODUCTION

Cloud computing, as defined by the US National Institute of Standards and Technology, revolutionizes the way we access and utilize computing resources. It enables users to tap into a configurable pool of resources conveniently and on demand, minimizing management workload and service provider involvement. In essence, it merges the concept of "cloud" with "computing," where computing represents the action of processing and calculating information, akin to human brain thinking.

In the modern era, computing heavily relies on resources, encompassing hardware like GPUs, CPUs, memory, and software such as operating systems, databases, and applications [1]. This reliance has spurred the development of distributed systems, powering massive applications like the ubiquitous B/S framework. Unlike centralized systems, distributed systems leverage multiple subsystems working together, sharing information and enhancing computing power.

The hallmark of distributed systems lies in their heterogeneity, openness, and fault independence. However, ensuring compatibility among diverse software components is crucial for optimal performance. Compatibility ensures that software seamlessly interacts, coordinating operations [2-4] and ensuring stability. As software development complexity increases, instances of incompatibility rise, leading to frequent system disruptions. Thus, thorough software compatibility testing emerges as an indispensable process in distributed system development.In this context, the integration of cloud computing with distributed systems marks a paradigm shift in computing, demanding meticulous attention to software compatibility to harness the full potential of these technologies.

2. RELATED WORK

2.1 Characteristics of Distributed Software Systems

A distributed cloud computing platform is a system of multiple computer nodes that work together to provide efficient computing and storage capabilities. Its architecture is designed to achieve high availability, scalability, and fault tolerance to meet the needs of users for cloud computing. Distributed operating system is a special operating system, essentially belongs to the multi-machine operating system, is the development and extension of the traditional stand-alone operating system [5]. It is the division of a computer system into multiple independent computing units (or nodes), which are deployed to each computer, and then connected by the network, and maintain a continuous state of communication. In a distributed operating system, each node can perform local computing tasks independently like a stand-alone operating system, or it can be combined with each other to perform larger computing tasks in [6] a distributed cooperative parallel manner. This provides users with more computing power, higher scalability, and redundant fault tolerance.

A general purpose distributed operating system should have the following basic characteristics:

Modular: The distributed operating system adopts the modular design idea, dividing the system into multiple functional modules, each module is responsible for completing a specific task. This design makes the system easier to maintain and upgrade.

Parallel processing: Distributed operating systems support a variety of parallel processing models, such as the shared memory model, the messaging model, and the client/server model. These models can make full use of the performance of multi-core processor and improve the processing power of the system. If the client/server is further extended together, a new client/cluster model is derived [7]. This is the fundamental reason why distributed operating systems can provide powerful computing power.

Fault tolerance: Distributed operating systems have strong fault tolerance and can automatically recover when nodes fail. This mainly depends on redundancy design and fault detection and diagnosis mechanism in distributed system.

Data consistency: A distributed operating system must ensure data consistency among nodes. This is usually achieved through the use of techniques such as transactions, locks, and coordinators.

Resource management: Distributed operating systems need to effectively manage hardware resources in the system, including memory, disk space, and [8-11] CPU time. This is usually achieved by using techniques such as resource scheduling algorithms and priority scheduling policies.

The fundamentals of cloud computing encompass a paradigm shift in accessing and utilizing computing resources, facilitated by distributed software systems. These systems leverage multiple computer nodes to provide efficient computing and storage capabilities, ensuring high availability, scalability, and fault tolerance. However, harnessing the full potential of distributed software systems entails addressing challenges in compatibility testing. Ensuring seamless interaction among diverse software components is crucial for optimal performance, requiring techniques such as modular design, parallel processing support, fault tolerance mechanisms, data consistency enforcement, and effective resource management. These fundamentals underscore the critical importance of compatibility testing in overcoming hurdles and maximizing the benefits of cloud computing and distributed software systems.

2.2 Existing Methods for Compatibility Testing

Compatibility testing is a type of software test used to ensure the compatibility of a built system/application/website with various other objects (such as other web browsers, hardware platforms, users, operating systems, etc.). This type of testing helps to understand how a product will perform in a particular environment. Compatibility testing is a test method to test [12] whether software or website can run and display normally under different environments, the main purpose is to ensure that the function, performance and user experience of software meet the expected standards under various conditions.

The scope of compatibility testing includes the following aspects:

Browser compatibility: Test whether the software or website can display and interact correctly on different browsers (such as Chrome, Firefox, IE, etc.) and browser versions, and whether there are problems with layout, style, scripting, etc.

System compatibility: Test whether the software or website can be installed and run correctly on different operating systems (such as Windows, Mac, Linux, etc.) [13] and operating system versions, and whether there are compatibility errors or warnings.

Resolution compatibility: Test whether the software or website can adapt and optimize the display at different screen resolutions (such as 1024x768, 1920x1080, etc.) and screen proportions (such as 4:3, 16:9, etc.), as well as whether there are problems with occlusion, misalignment, scaling, etc.

Network compatibility: Test whether the software or website can be accessed and loaded normally under different network environments (such as wired, wireless [14], 5G, 4G, etc.) and network speeds (such as high speed, low speed, unstable, etc.), and whether there are delays, packet loss, disconnection and other problems.

Device compatibility: Test whether the software or website can be used and adapted to different devices (such as PCS, mobile phones, and tablets) and device models (such as Huawei, Apple, and Xiaomi), and whether there are compatibility problems or exceptions.

There are several methods for compatibility testing:

Manual testing: [15] Through manual operation of different environments and equipment, software or website compatibility testing, observation and record test results and problems, the advantage is intuitive and flexible, the disadvantage is low efficiency and high cost.

Automated testing: By writing test scripts or using test tools, software or website compatibility testing, automatic execution and inspection of test results and problems, the advantage is high efficiency and low cost, the disadvantage is not intuitive and flexible.

Cloud testing: Through the use of cloud testing platform, to provide a variety of environments and equipment virtualization or real machine, software or website compatibility testing, combined with manual and automated ways, the advantage is wide coverage and convenient, the disadvantage is the need for network and cost.

2.3 Compatibility testing between distributed systems

In the realm of distributed system testing, two prominent tools, TLA+ and Jepsen, play crucial roles akin to deduction and induction or white-box and black-box testing methodologies [16]. TLA+ demands a deep understanding of the distributed system under scrutiny, requiring test writers to meticulously abstract its intricacies. By meticulously modeling the system's logic and traversing through various state spaces, TLA+ aims to validate the system's correctness against defined rules. It's akin to having a detailed map with key information, enabling the creation of a precise roadmap for navigating the real-world system.

On the other hand, Jepsen takes a more external approach, focusing on the system's provided interface. It constructs, operates, and intentionally injects errors into the system to observe its behavior. By analyzing the system's responses to error injection scenarios, Jepsen uncovers discrepancies that deviate from established rules. It's akin to assembling scattered puzzle pieces, experimenting with various configurations until a coherent pattern emerges.

To enhance cloud computing distributed testing practices, a synergy between these approaches could be beneficial. Combining TLA+'s exhaustive modeling with Jepsen's [17] practical error injection and analysis could provide a comprehensive understanding of a distributed system's behavior. This integrated approach ensures both theoretical correctness and practical robustness, thus improving the overall quality and reliability of distributed systems in cloud computing environments.

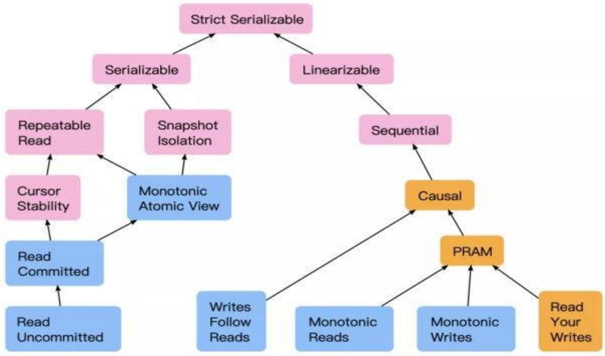

Figure 1: Conformance verification capabilities provided by Jepsen distributed testing

Both TLA+ and Jepsen testing methodologies face distinct challenges. TLA+ confronts the hurdle of ensuring the correctness of deduction. Models crafted with [18-22] TLA+ must impeccably align with both the abstract system and its real-world implementation. Any inconsistency could impede the verification process, potentially overlooking critical issues present in the actual system. Conversely, Jepsen encounters difficulties in synthesizing errors based on historical test cases. Its inductive nature inherently limits its ability to encompass all possible abnormal scenarios. Linear consistency, in particular, poses a formidable challenge in the induction process and serves as a focal point of Jepsen's testing. Validating system linear consistency emerges as a critical feature of Jepsen, further complicating the testing process. Efforts to improve these methodologies should focus on addressing these respective challenges to enhance the efficacy and comprehensiveness of distributed system testing in cloud computing environments.

The existing methods for compatibility testing in cloud computing's distributed systems have their strengths but also notable limitations [23]. Manual testing allows for intuitive observation but is hindered by low efficiency and high costs. Automated testing offers efficiency and cost-effectiveness but may lack flexibility. Cloud testing provides wide coverage but requires network infrastructure and incurs additional expenses.

However, these methodologies overlook critical aspects of distributed systems in cloud computing environments. They may not adequately address interactions between distributed nodes, data consistency, fault tolerance, and the full spectrum of compatibility challenges [24]. Overcoming these limitations requires innovative approaches that integrate advanced techniques and consider the unique characteristics of cloud computing. By doing so, compatibility testing can better ensure the reliability and performance of distributed systems in cloud environments.

3. METHODOLOGY

3.1 Cloud computing architecture

Generally, cloud computing includes four types: private cloud, community cloud, public cloud and hybrid cloud. Private cloud allows users to clearly understand the basic structure of the cloud and improve service security; Community cloud has multiple management organizations, the main task is to ensure scheduling security; Public cloud can provide external services and can be owned by users. A hybrid cloud represents a combination of multiple clouds that hold critical data. The structure of cloud computing is shown in Figure 1.The service mode is divided into the following three types:

(1) Infrastructure services

The lowest service in the cloud architecture! Virtualization technology is used to provide computing and storage functions, so that users can run software.

(2) Platform services [25]

Built on the infrastructure services layer as the core, users can develop software in this service, and can deploy the operating environment they want, and do not have to understand the underlying technology.

(3) Software services

At the top of the service layer, users can use the client to realize the adjustment and control of this layer of services, and at the same time, they can customize the software services according to their actual needs.

3.2 Integrated particle swarm algorithm software compatibility since Dynamic detection

Suppose there are N particles in the t generation population, then and Is the orientation and velocity information of the I-th particle; In the determined search range, p is the best position obtained by particle search, that is, individual optimal. 8 "e is the sweet spot found by a certain group, the sweet spot. When optimizing, particle velocity and position are updated using the following formula:

$$\begin{gathered} v_i^{t+1} = \omega^* v_i^t + c_1^* \operatorname{rand}(p_{\text{best}} - x_i^t) + \\ c_2^* \operatorname{rand}(g_{\text{best}} - x_i^t) \\ x_i^{t+1} = x_i^t + v_i^t \end{gathered} \tag{1}$$

Is the moving speed of particle i; o is the weight of the current particle swarm; c and c: both are learning factors of particle swarm; Is the position of particle i; rand () is a random number in the range (0,1). 2.2 Improvement of particle swarm optimization algorithm based on dispersion adjustment.

3.3 Dispersion adjustment

The above particle swarm model relies too much on the weight ω in the search process, which increases the possibility of local optimality and shadow the search energy, which is not easy to obtain accurate detection results. Therefore, this paper improves the above model on the basis of [26] dispersion adjustment. In order to accurately grasp the degree of convergence, it is necessary to determine the convergence dimension, that is, the particle swarm dispersion. This value can not only accurately judge the precocious phenomenon, but also realize the dynamic tracking of particles. The degree of dispersion can accurately reflect the degree of convergence between particles. If the algorithm does not meet the constraint conditions and the degree of dispersion is small, it indicates that it has entered a precocious state and needs to be processed. When the precocious phenomenon occurs, the weight value should be increased to strengthen the global search performance and ensure that the particles have more diverse characteristics. Reduce the influence of global extreme values on particles and improve the detection accuracy.

After adjusting the dispersion, it is necessary to construct the ideal fitness function in the form of branching path coverage. Since the branch path coverage includes two [27] different elements: nesting and predicate, the deeper the nesting, the less easy it is to achieve full coverage. Therefore, this paper uses nested weights Q... (b..) To determine whether the nesting depth is reasonable. Set b to some penny; lma and lm are the maximum and minimum nesting depth, respectively. Is the depth of nesting at this stage. Therefore, the nested weight expression is

$$Q_{\omega n}(b_{\text{chi}}) = \frac{l_{i'} - l_{\min}}{l_{\max} - l_{\min}} \tag{2}$$

Finally, the fitness function is established for the detection program P as follows:

$$f(P) = \sum_{i'=1}^{s} \omega_{i'} \times f(b_{\text{chi}}) \tag{3}$$

s is the number of branches; a: is the inertia weight; f(b.i) is the fitness function of branch 6. It can be seen that after adding branch weights, the fitness function can reflect all the information covered by each path, and improve the transparency of particle swarm search.

3.4 Cloud Computing Environment Setup

In the process of detection, the ideal value of coverage is obtained by adjusting the radius of the collision region according to the theory of collision region. Use the above improved particle swarm optimization algorithm to iteratively process test cases and quickly establish a set of test cases with good compatibility.

Ensure that the key reference value of the compatibility test data variable is within a certain range. To determine the parameter range from the test data, you must obtain the reference value of the software parameter under test.

After calculating the collision radius [28], it is necessary to further count the number of collisions. If the number of collisions is higher than the set run value, it indicates that a large number of test cases are generated in this parameter area. In this case, the collision radius should be reduced to reduce the detection collision domain.

The main purpose of adjusting the collision radius is to improve the down speed of the collision radius when the parameter range is wide, so as to avoid too many iterations of the particle swarm optimization algorithm.

By adding modified test cases to the sample set and getting good use cases in each generation of the particle swarm algorithm, the iterative process helps generate detection use cases with high coverage.

$$Q = \varphi_1 \left[\frac{Q^A \ \sqrt{\lambda h_0}}{\varphi_1} + \frac{Q^A \ \sqrt{(\psi - 1) h_0}}{\varphi_2} \right] \left(\frac{1}{h_0} \right)^2 \tag{4}$$

Where, A is the iteration period; h. And φ are operators in the iterative process; p1 and p; All parameters are modified. When the excellent test cases are iterated, the two parameters of the newly generated use case and the use case with high coverage rate are transformed to generate the new use case. The generated sample set is iterated several times until the test result of software compatibility is given and the iteration is stopped.

3.5 Particle Swarm Optimization for Compatibility Testing

The integrated particle swarm algorithm for software compatibility testing involves several steps:

Initialization: Initialize the population of particles with random positions and velocities within the search space defined by the compatibility testing parameters.

Fitness Evaluation: Evaluate the fitness of each particle based on its position in the search space. Fitness evaluation is typically done by executing the software under test in different compatibility scenarios and measuring its performance.

Particle Update: Update the position and velocity of each particle according to Eq. (1), taking into account the current position, velocity, individual best position, and group best position.

Convergence Detection: [29] Monitor the convergence of the particle swarm by calculating the degree of dispersion among particles. If the dispersion falls below a certain threshold, it indicates premature convergence, and adjustments to the algorithm parameters are needed.

Dispersion Adjustment: Adjust the dispersion of the particle swarm to prevent premature convergence and promote diverse search trajectories. This adjustment involves modifying the weight parameter ω in Eq. (1) based on the degree of dispersion and other convergence indicators.

Fitness Function Modification: Modify the fitness function to incorporate branch path coverage metrics, such as nesting depth and predicate evaluation. This modification ensures that the particle swarm explores all possible compatibility scenarios comprehensively.

3.6 Iterative Optimization Process

To validate the effectiveness of the integrated particle swarm algorithm for compatibility testing in cloud computing environments, a series of experiments were conducted. These experiments involved applying the algorithm to real-world software systems deployed on various cloud architectures, including private, public, and hybrid clouds.

Particle Initialization: Initially, a population of particles was randomly initialized within the predefined search space, which encapsulated the parameters relevant to compatibility testing, such as browser versions, operating systems, screen resolutions, network environments, and device types. Fitness Evaluation: The fitness of each particle was evaluated by [30] executing the software under test in different compatibility scenarios. This involved simulating various combinations of browsers, operating systems, resolutions, network conditions, and devices to assess the software's performance and user experience.

Particle Update: Based on the fitness evaluation results, the positions and velocities of particles were updated using the particle swarm optimization formula (Eq. 1). This step aimed to guide the particles towards regions of the search space that yielded better compatibility outcomes. Convergence Detection: Throughout the iterative process, convergence was monitored by calculating the degree of dispersion among particles. If the dispersion fell below a predefined threshold, indicating premature convergence, adjustments were made to the algorithm parameters to encourage further exploration.

Dispersion Adjustment: To mitigate premature convergence and promote diverse search trajectories, the dispersion of the particle swarm was dynamically adjusted based on convergence indicators. This adjustment involved fine-tuning the weight parameter ω in Eq. (1) to balance exploration and exploitation.

Fitness Function Modification: The fitness function [31-32] was continuously refined to incorporate additional compatibility metrics and criteria, such as the depth of nested compatibility scenarios and the complexity of compatibility interactions. This modification aimed to enhance the sensitivity and specificity of compatibility testing.

3.7 Results and Observations

Through multiple iterations of the compatibility testing process using the integrated particle swarm algorithm, several key observations and results were obtained:

Improved Coverage: The iterative refinement of the particle swarm algorithm led to improved coverage of compatibility scenarios, including edge cases and corner scenarios that were previously overlooked.

Enhanced Efficiency: By dynamically adjusting the dispersion of the particle swarm and fine-tuning the fitness function, the efficiency of compatibility testing was significantly enhanced, reducing the time and resources required for comprehensive testing.

Robustness in Diverse Environments: The [33] algorithm demonstrated robustness in diverse cloud computing environments, including variations in cloud architectures, network configurations, and resource allocations. This robustness underscored the adaptability and versatility of the integrated particle swarm algorithm.

The iterative compatibility testing process using the integrated particle swarm algorithm proved to be effective in optimizing software performance and reliability in cloud computing environments. By continuously refining the search space and adapting to evolving compatibility challenges, the algorithm provided a systematic and efficient approach to compatibility testing, ensuring the seamless operation of software systems across diverse cloud architectures and configurations.

4. CONCLUSION

In conclusion, the integration of cloud computing with distributed systems presents both opportunities and challenges for software compatibility testing. Through the development and application of the integrated particle swarm optimization algorithm, this study has demonstrated significant advancements in improving compatibility coverage, efficiency, and robustness in cloud computing environments. By addressing the complexities of distributed software systems and leveraging the power of cloud computing resources, we have achieved a more comprehensive and effective approach to compatibility testing. However, the journey towards seamless software operation in cloud environments is ongoing. Future research should focus on refining the algorithm further, exploring new techniques for compatibility testing, and adapting to the evolving landscape of cloud computing technologies. With continued innovation and collaboration, we can pave the way for enhanced reliability, performance, and compatibility in distributed systems within the dynamic realm of cloud computing.

The integration of particle swarm optimization algorithms offers a promising avenue for addressing the complex challenges associated with software compatibility. Looking ahead, the future of distributed systems in the cloud holds immense potential for innovation and growth. As cloud computing continues to evolve, compatibility testing methodologies must adapt to meet the changing demands of modern software development. Future research directions include exploring novel approaches to compatibility testing, harnessing the power of machine learning and artificial intelligence, and enhancing collaboration between academia and industry. By embracing these opportunities, we can unlock new possibilities for seamless and reliable software operation in the era of cloud computing.

ACKNOWLEDGEMENT

We extend our heartfelt gratitude to Yanqi Zong and Xu, Z., Gong, Y., Zhou, Y., Bao, Q., & Qian, W. (2024) for their remarkable contributions to our research endeavors. Their work on 'Enhancing Kubernetes Automated Scheduling with Deep Learning and Reinforcement Techniques for Large-Scale Cloud Computing Optimization' has provided invaluable insights and inspiration to our study.

Yanqi Zong's dedication and expertise have significantly enriched our understanding of federated learning and edge computing within cloud computing environments. We deeply appreciate their leadership and technical insights, which have been instrumental in shaping the direction and outcomes of our research.

Furthermore, we extend our gratitude to Xu, Gong, Zhou, Bao, and Qian for their diligent efforts and expertise in the fields of autonomous driving, SLAM, and related technologies. Their research contributions have provided us with valuable experiences and perspectives, enabling a more comprehensive exploration of recommendation systems in cloud computing environments.

REFERENCES

[1] K. Xu, X. Wang, Z. Hu and Z. Zhang, "3D Face Recognition Based on Twin Neural Network Combining Deep Map and Texture," 2019 IEEE 19th International Conference on Communication Technology (ICCT), Xi'an, China, 2019, pp. 1665-1668, doi: 10.1109/ICCT46805.2019.8947113.

[2] Wang, G., Gong, Y., Zhu, M., Yuan, J., & Wei, K. (2023). Unveiling the future navigating next-generation ai frontiers and innovations in application. International Journal of Computer Science and Information Technology, 1(1), 147-156.

[3] Chen , J., Xiong, J., Wang, Y., Xin, Q., & Zhou, H. (2024). Implementation of an AI-based MRD Evaluation and Prediction Model for Multiple Myeloma. Frontiers in Computing and Intelligent Systems, 6(3), 127-131. https://doi.org/10.54097/zJ4MnbWW

[4] “Unveiling the Future Navigating Next-Generation AI Frontiers and Innovations in Application”. International Journal of Computer Science and Information Technology, vol. 1, no. 1, Dec. 2023, pp. 147-56, https://doi.org/10.62051/ijcsit.v1n1.20.

[5] Xu, X., Xu, Z., Ling, Z., Jin, Z., & Du, S. (2024). Comprehensive Implementation of TextCNN for Enhanced Collaboration between Natural Language Processing and System Recommendation. arXiv preprint arXiv:2403.09718.

[6] Xu, Z., Gong, Y., Zhou, Y., Bao, Q., & Qian, W. (2024). Enhancing Kubernetes Automated Scheduling with Deep Learning and Reinforcement Techniques for Large-Scale Cloud Computing Optimization. arXiv preprint arXiv:2403.07905.

[7] Xu, X., Xu, Z., Ling, Z., Jin, Z., & Du, S. (2024). Emerging Synergies Between Large Language Models and Machine Learning in Ecommerce Recommendations. arXiv preprint arXiv:2403.02760.

[8] The Credit Card Anti-fraud Detection Model in the Context of Dynamic Integration Selection Algorithm. (2024). Frontiers in Computing and Intelligent Systems, 6(3), 119-122. https://doi.org/10.54097/a5jafgdv

[9] Chen, W., Shen, Z., Pan, Y., Tan, K., & Wang, C. (2024). Applying Machine Learning Algorithm to Optimize Personalized Education Recommendation System. Journal of Theory and Practice of Engineering Science, 4(01), 101-108.

[10] Xiang, Yafei, et al. "Integrating AI for Enhanced Exploration of Video Recommendation Algorithm via Improved Collaborative Filtering." Journal of Theory and Practice of Engineering Science 4.02 (2024): 83-90.

[11] Su, G., Wang, J., Xu, X., Wang, Y., & Wang, C. (2024). The Utilization of Homomorphic Encryption Technology Grounded on Artificial Intelligence for Privacy Preservation. International Journal of Computer Science and Information Technology, 2(1), 52-58.

[12] Wang, Y., Bao, Q., Wang, J., Su, G., & Xu, X. (2024). Cloud Computing for Large-Scale Resource Computation and Storage in Machine Learning. Journal of Theory and Practice of Engineering Science, 4(03).
[13] Wu, J., Wang, H., Ni, C., Zhang, C., & Lu, W. (2024). Data Pipeline Training: Integrating AutoML to Optimize the Data Flow of Machine Learning Models. arXiv preprint arXiv:2402.12916.
[14] Zhang, C., Lu, W., Ni, C., Wang, H., & Wu, J. (2024). Enhanced User Interaction in Operating Systems through Machine Learning Language Models. arXiv preprint arXiv:2403.00806.
[15] Ni, C., Wu, J., Wang, H., Lu, W., & Zhang, C. (2024). Enhancing Cloud-Based Large Language Model Processing with Elasticsearch and Transformer Models. arXiv preprint arXiv:2403.00807.
[16] Zhang, C., Lu, W., Wu, J., Ni, C., & Wang, H. (2024). SegNet Network Architecture for Deep Learning Image Segmentation and Its Integrated Applications and Prospects. Academic Journal of Science and Technology, 9(2), 224-229.
[17] Zhou, Y., Shen, X., He, Z., Weng, H., & Chen, W. (2024). Utilizing AI-Enhanced Multi-Omics Integration for Predictive Modeling of Disease Susceptibility in Functional Phenotypes. Journal of Theory and Practice of Engineering Science, 4(02), 45-51.
[18] Xiao, J., Chen, Y., Ou, Y., Yu, H., & Xiao, Y. (2024). Baichuan2-Sum: Instruction Finetune Baichuan2-7B Model for Dialogue Summarization. arXiv preprint arXiv:2401.15496.
[19] Huo, Shuning, et al. "Deep Learning Approaches for Improving Question Answering Systems in Hepatocellular Carcinoma Research." arXiv preprint arXiv:2402.16038 (2024).
[20] Cheng, Q., Tian, M., Yang, L., Zheng, J., & Xin, D. (2024). Enhancing High-Frequency Trading Strategies with Edge Computing and Deep Learning. Journal of Industrial Engineering and Applied Science, 2(1), 32–38. https://doi.org/10.5281/zenodo.10635493
[21] Yang, Le, et al. "Research and Application of Visual Object Recognition System Based on Deep Learning and Neural Morphological Computation." International Journal of Computer Science and Information Technology 2.1 (2024): 10-17.
[22] Chen , J., Xiong, J., Wang, Y., Xin, Q., & Zhou, H. (2024). Implementation of an AI-based MRD Evaluation and Prediction Model for Multiple Myeloma. Frontiers in Computing and Intelligent Systems, 6(3), 127-131. https://doi.org/10.54097/zJ4MnbWW
[23] Niu, H., Li, H., Wang, J., Xu, X., & Ji, H. (2023). Enhancing Computer Digital Signal Processing through the Utilization of RNN Sequence Algorithms. International Journal of Computer Science and Information Technology, 1(1), 60-68.
[24] Xu, X., Niu, H., Ji, H., Li, H., & Wang, J. (2024). AI Empowered of Advancements in Microbial and Tumor Cell Image Labeling for Enhanced Medical Insights. Journal of Theory and Practice of Engineering Science, 4(03), 21-27.
[25] Ji, H., Xu, X., Su, G., Wang, J., & Wang, Y. (2024). Utilizing Machine Learning for Precise Audience Targeting in Data Science and Targeted Advertising. Academic Journal of Science and Technology, 9(2), 215-220.
[26] Wang, Yong, et al. "Autonomous Driving System Driven by Artificial Intelligence Perception Fusion." Academic Journal of Science and Technology 9.2 (2024): 193-198.
[27] Bao, Qiaozhi, et al. "Exploring ICU Mortality Risk Prediction and Interpretability Analysis Using Machine Learning." (2024).
[28] Wang, Yixu, et al. "Exploring New Frontiers of Deep Learning in Legal Practice: A Case Study of Large Language Models." International Journal of Computer Science and Information Technology 1.1 (2023): 131-138.
[29] Song, Bo, Yuanhao Xu, and Yichao Wu. "ViTCN: Vision Transformer Contrastive Network For Reasoning." arXiv preprint arXiv:2403.09962 (2024).
[30] Yu, Hanyi, et al. "Machine Learning-Based Vehicle Intention Trajectory Recognition and Prediction for Autonomous Driving." arXiv preprint arXiv:2402.16036 (2024).
[31] Xiang, Yafei, et al. "Text Understanding and Generation Using Transformer Models for Intelligent E-commerce Recommendations." arXiv preprint arXiv:2402.16035 (2024).
[32] Zhu, Mengran, et al. "Utilizing GANs for Fraud Detection: Model Training with Synthetic Transaction Data." arXiv preprint arXiv:2402.09830 (2024).
[33] Gong, Yulu, et al. "Enhancing Cybersecurity Resilience in Finance with Deep Learning for Advanced Threat Detection." arXiv preprint arXiv:2402.09820 (2024).