Serverless Cloud IDE with Real-Time Collaboration Using Docker

Senthamarai N
Dept of Networking And Communications,
SRM Institute of Science and Technology,
Kattankulathur, India
senthamn@srmist.edu.in

Pragada Sai Ramesh
Dept of Networking And Communications,
SRM Institute of Science and Technology,
Kattankulathur, India
sp2610@srmist.edu.in

Mannam Arjun
Dept of Networking And Communications,
SRM Institute of Science and Technology,
Kattankulathur, India
am3547@srmist.edu.in

Abstract*— The fast development of cloud computing has had vast changes in the current software development practice, especially in facilitating remote co-operation and scalability of the infrastructure. Nevertheless, the existing traditional development environments continue to be based on manual installations, tedious configurations, and fragmented tool chains, which makes these systems less productive and not as accessible as they could be. This study aims at suggesting a scalable Integrated Development Environment (IDE) on the cloud, serverless, and supporting real-time collaborative programming through Docker-based containerization. The system allows users to code, edit, execute and share code directly via a browser without the need to set it up locally which minimizes overhead and optimises efficiency. Serverless backend architecture provides Serverless backend architecture provides automatic scalability based on demand and ensures optimal resource utilization and optimum utilization of resources and Docker containers offer secure and isolated execution space. Moreover, with the assistance of AI-driven code editor, the productivity of the development team can be improved with smart recommendations and auto-complete technology. Experimental results show that there is increased collaboration efficiency, less time has been taken to set up and also has more scalability in the system than the traditional methods. The offered solution helps in the enhancement of the cloud-based development platforms by incorporating the collaboration, execution, and intelligence as a single system.

Keywords - Cloud IDE, Serverless Computing, Docker, Real-Time Collaboration, AI-Assisted Development, Cloud-Native Architecture.

I. INTRODUCTION

The intensive advancement of cloud computing and distributed systems has essentially changed the scene of any software development process, as the developers can reach resources, tools, and environments anywhere on Earth. The flexibility, scalability, and collaboration requirements have become prominent in the recent development workflows, especially due to the emergence of remote work and remote collaborative teams. Conventional developmental environments, nevertheless, still depend on local system set-ups, manual dependence installations, and a variety of distanced tools to code, execute, versioning and collaborate. This piece-meal system brings complication, adds complexity to setup, and in most cases, creates compatibility problems even between various systems and thus, lowers overall efficiency and productivity of development [1] [2]. Some of these challenges have been tried to be resolved at the existing collaborative coding platforms, which provide an opportunity to share and be synchronized in real-time with all the users. However, most of these systems have shortcomings in the form of ineffective synchronization systems, non existence of scalability in the face of the high user load, and missing the built-in execution environment [3] [4]. Moreover, the majority of traditional solutions do not offer secure and isolated execution environments needed to perform untrusted or user-generated code in a secure environment. The lack of built-in intelligent aid also limits the ability of such platforms to guide the developer to write an error-free and optimized code [5] [6]. Recent developments of containerization technologies, especially that of Docker technology have proposed lightweight and efficient methods of providing isolated execution environment that provides consistency and security across various systems [7]. At the same time, serverless computing has become a strong paradigm, which enables the dynamic scaling of applications, without explicit managing infrastructure, enhancing performance and minimizing operational overhead [8]. The current studies and systems hardly combine serverless architecture, containerization, interactive work with AI support into a single platform, even despite these developmental improvements [9] [10]. Due to these constraints, the current study proposes a unified serverless cloud-based Integrated Development Environment, which will allow real-time collaborative The development of code, Docker-based execution, and AI-assisted development combined into a single development framework. The proposed system will make the development processes easier, build development upon more efficient collaboration, and offer scalable, secure, and smart coding environments, which can be accessed directly via web browsers. The system helps towards creating a next-generation development platform by solving the weaknesses of the existing and traditional cloud-based IDEs to meet the changes in requirements of the current software engineering practices.Also, the suggested system insists on the significance of integrating the process of development and implementation and team work into one consistent setting to erase the disintegration and enhance the continuity of the processes. Adding real-time synchronization to smart assistance, developers may not just work in harmony but also have their contextually-aware guidance, which increases the quality of writing and creates the shortest time of developing a program. The accessibility is in the form of a browser and cannot be limited to any specific platform thus making the user capable of operating on other devices without worrying about compatibility. Also, the scalable cloud infrastructure is implemented allowing to meet increasing user requirements without reducing its performance.

II. LITERATURE REVIEW

The popularity of cloud-based Integrated Development

Environments (IDEs) and group coding platforms has been increasing over the past few years as there is an increasing demand to be able to access and provide software development services remotely and make the work of developing and maintaining software a team-based effort. The first studies in the area of collaborative code editing were mainly aimed at facilitating real-time multi-user communication by means of synchronization tools like WebSockets and Operational Transformation (OT). These were more productive and efficient in terms of sharing knowledge as more than one user can write on the same codebases. It has however been noted that there is latency in several systems and conflict resolving problems especially in high concurrency environments, which may reduce user experience and system reliability [1] [11]. Later innovations saw the incorporation of browser based IDEs which combine the features of coding, editing and collaboration into one web based interface. The current systems like React and Node.js have been embraced to construct responsive and interactive development systems. These platforms enhanced access by removing the necessity of local configurations and the development could be done right through web browsers. Regardless of all these advancements, most of the systems did not have sophisticated intelligent assistance and frequently used the help of third-party tools to debug and optimize. There was also the concern of security simply because of the lack of isolated execution environments and hence they were not suitable to execute untrusted or user-generated code in a secure manner [3] [14]. The trends in the recent past have discussed the incorporation of artificial intelligence into development settings in order to increase productivity in terms of code recommendations, automatic completion and automated documentation. The AI-supported coding tools have revealed the beneficial growth in the development pace and the decrease in the time devoted to the process of manual coding. The majority of them, however, are independent services or modules, and not fully embedded in collaborative cloud services. Moreover, there exist the issue of the contextual correctness and trustworthiness of AI-generated recommendations, particularly in complicated program development cases [5] [16]. Similar studies have highlighted the use of containerization technologies and especially Docker to deliver lightweight and isolated execution environments. Container-based systems guarantee uniformity among varying platforms because they secure the dependence and runtime settings. Containers have a shorter startup time and reduced resource overhead compared to traditional virtual machines and are therefore suitable in scalable cloud applications. Nevertheless, most containerized development systems do not have in-built collaborative capabilities and smart support, where their use is constrained in contemporary dispersed development processes [7] [18]. Moreover, serverless computing has become a revolution in the development of cloud applications providing the ability to automatically expand resources and decreasing the burden on managing infrastructure. There is better cost efficiency and performance because the serverless architecture allows applications to dynamically scale resources according to demand. The effectiveness of serverless systems in the distribution of workloads and real-time applications has been proven by a number of studies [9] [19]. More studies are also indicating that secure access control mechanisms and effective data synchronization strategies should be considered as one of the solutions in collaborative settings. Any system that does not handle any simultaneous edits well is likely to have some data discrepancies and memory overloads. Intense synchronization approaches and distributed system architecture have been put forth to alleviate the problems though not generally practiced in the current IDE systems [2] [12]. Furthermore, scalable architecture of microservices to cloud-based application, with emphasis to modularity, flexibility and ease of installation, has been researched recently. Such architectures enhance the maintainability of the system and enable scaling independent components. They have not been however extensively integrated with real-time collaborative systems and execution environments to pose challenges in the fully unified development platform [13] [20]. The review of the existing literature shows that there is a gap in research. The current solutions concentrate on the individual elements, including real-time collaboration, artificial intelligence support, containerization, or serverless computing, and lack the connectivity between all these elements into a unified and scalable system [4] [15]. Moreover, the issues of secure access control, efficient synchronization in a high workload environment and the optimal use of resources remain a common concern in a wide variety of implementations. The above restrictions underscore the need to come up with an integrated framework that integrates serverless architecture, real-time collaboration, Docker-based execution and artificial intelligence assisted development into one cloud-based IDE [6] [17].

III. METHODOLOGY

The system follows a structured operational workflow to ensure seamless functionality. Initially, users authenticate through a secure login mechanism. Once authenticated, users create or access project workspaces. Code is then written and executed within Docker-based isolated environments. Real-time collaboration enables multiple users to simultaneously interact with the same codebase with instant synchronization. Finally, all project data and files are securely stored and managed using cloud storage services, ensuring persistence and accessibility. Using the benefits of modern web technologies, serverless backend services and Docker-based isolation, the system will not require local environment setup, but it will guarantee consistency, security, and performance. Its design focuses on the smooth integration between the elements like the authentication, collaboration modules, execution engines and storage service thus forming a complete development ecosystem. Additionally, this methodology will use AI-assisted code functionality to improve the productivity of developers and the quality of their code. It has been designed in a way such that real-time synchronization between several users is enabled through the effective communication protocols and data integrity and low latency is ensured. The serverless microservices are adopted in order to promote the scaling of the system resources in reaction to the user demand, and maintain high availability and optimal performance at different loads. The implementation of containerization can ensure safe implementation of user code in isolated environments with avoiding system-level problems and increasing reliability. In general, the methodological approach is focused on establishing a balance among scalability, security, usability, and intelligent assistance and helps overcome the shortcomings present in the current systems.

A. System Data Handling

According to the proposed cloud-based Integrated Development Environment, user data generated on the fly will be used, instead of using the stagnant datasets. The system mostly handles user credentials, project metadata, source code files, collaboration states, and AI interaction history. Authentication with a token-based verification and encrypted storage are other authentication mechanisms of dealing with user-related information including identity and session data securely. Attributes of project data include the

project name, project type, ownership and access permissions that are stored in a structured cloud database to have a easy way of retrieving it and scaling it. The system does not only process the content of the collaboration data, such as cursor positions, code edits, and user activity states, which are real-time project metadata. This information is kept in synch with various clients to ensure that there is consistency in a common workspace. Cloud object storage systems store file contents and make them durable and readily accessible to distributed environments. Monitoring and optimization The AI interaction records are also kept in the platform and track code suggestion usage. This is a dynamic data management technology that allows the system to efficiently serve many users and projects simultaneously without compromising on data integrity, security and performance.

B. Data Handling & Validation.

The selected system relies on the constant interaction of users with the browser-based interface, which will lead to data handling. Upon accessing the platform, token-based methods of authentication are securely tested against some authenticated data and the user is permitted to access system resources. Users create, edit, or delete projects, and thus the data are recorded in the cloud database by corresponding metadata. The data written in the editor is constantly sent with the help of real-time communication protocols that allow synchronizing collaborators. The validation process makes sure that all the data coming in is within the boundaries of the system and within the security standards. Input validation checks are done to ensure no invalid or malicious entries are made to the project names, file structures and user commands. Before the execution, the code sanitization methods are used to prevent the injection attacks and unauthorized operation. Access control policies also make sure that authorized users only modify or execute some of the resources. Moreover, the real-time synchronization data is optimized with the help of the effective state management techniques in order to reduce the latency and eliminate the conflicts. This confirmed and structured data handling and validation layer guarantees reliability and security as well as smooth user experience.

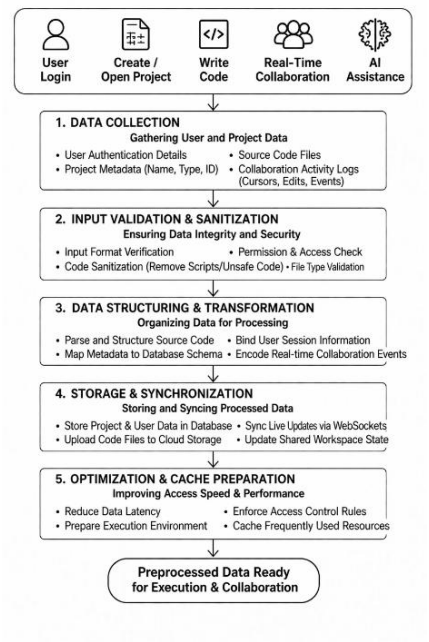

Fig. 1. Data Flow Diagram of Serverless Cloud IDE with Real-Time Collaboration

Fig. 1 illustrates the overall data flow of the proposed Serverless Cloud IDE with Real-Time Collaboration. It represents how user requests are processed through authentication, project creation, real-time collaboration, Docker-based execution, and cloud storage integration.

C. System Architecture and Design

The system is based on the serverless microservices architecture, according to which every functional component is independent and interacts via APIs. The frontend is developed on modern web framework and offers an interactive code-based interface on the browser with features like code editing, file management, accessing a terminal and dash board of the project. The serverless cloud functions are used to create the backend and they process requests regarding the authentication and project management, collaboration, and AI services. The scalability is guaranteed by this architecture, where resources are dynamically assigned as per the demand. The fundamental unit of execution is Docker containers, which attaches isolated user code execution environments. Any project correlates with a containerized working environment that has some degree of homogeneous execution irrespective of the system underneath. The collaboration module allows work of several people on the same project in real time by synchronizing the changes in code. The AI assistance integrated into the system is based on pre-trained API services and does not involve any local model training or dataset processing within the platform. This approach ensures efficient performance without additional computational overhead while maintaining the quality of intelligent code suggestions. The database houses structured data like user data and details of projects whereas file storage is done in the cloud storage. It has a special security layer allowing the encryption of data and control of access to it in addition to protection against unwarranted access.

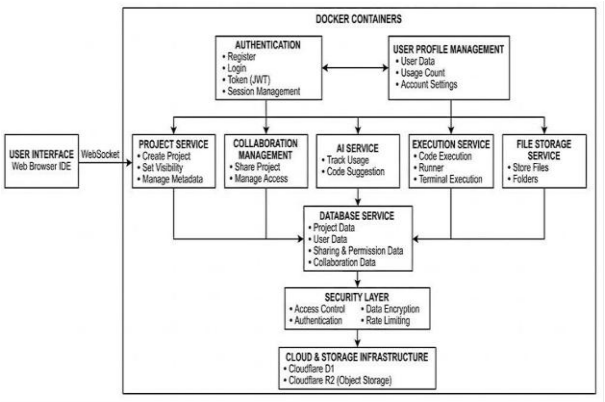

Fig. 2. System Architecture of the Cloud IDE Platform

D. System Execution & Resource Management

The suggested system does not presuppose the model training but rather emphasizes the effective performance and resource usage in the cloud infrastructure that is serverless. The code is executed in a Docker-based containerization where each request by the user is served in an isolated environment to provide security and uniformity. By executing a program, a user sends a request to a container instance, which is run on its own and the result is sent to the user interface in real-time. Serverless computing is implemented to manage resources in a dynamically scaled fashion, with backend services being dynamically scaled depending on incoming requests. This guarantees the optimal use of the computational resources without the management of the infrastructure manually. The system also has rate limiting and inactivity checks which ensure that resources are not depleted so that the system can be stable. Pre-trained API services are also part of AI-assisted coding where the services suggest code suggestions without the need to have the local training. The execution workflow is

designed so as to provide low latency, high reliability and able to handle concurrent users efficiently; hence the platform can be used in real world implementation conditions.

IV. RESULTS AND DISCUSSION

The assessment of the suggested serverless cloud IDE was carried out by evaluating the performance of the proposed system in the framework of various functional areas and domains to measure and compare the effectiveness of the proposed processing system in relation to such aspects as real-time collaboration, scalability of the system, and execution reliability and responsiveness to users. A controlled cloud environment was installed where the system was used by a number of concurrent users who could access the system using browsers. The experiment was related to testing the effectiveness of the system in maintaining discrepancies in code editing, execution requests, and intelligible interactions with minimal latency and regular performance. The findings indicate that serverless architecture and Docker-based containerization bring great flexibility to the systems and minimize the costs of infrastructures. Besides, the browser-based interface also helps in making the interface accessible on various devices without local configuration, hence offering greater usability and adoption.

Table 1. Input and Output of the Proposed System

S. No	Input	Process Performed	Output
1	User Login Credentials	Authentication using secure token mechanism	Authenticated User Session
2	Project Creation Request	Workspace initialization in cloud environment	New Project Work space
3	Code Input in Editor	Execution using Docker container	Program Output in Terminal/Preview
4	Collaboration Request	Real-time synchronization using APIs/WebSocket	Shared Workspace with Live Updates
5	AI Code Suggestion Request	API-based AI processing	Intelligent Code Suggestions
6	File Save Request	Storage in cloud object storage (R2)	Persisted Project Files

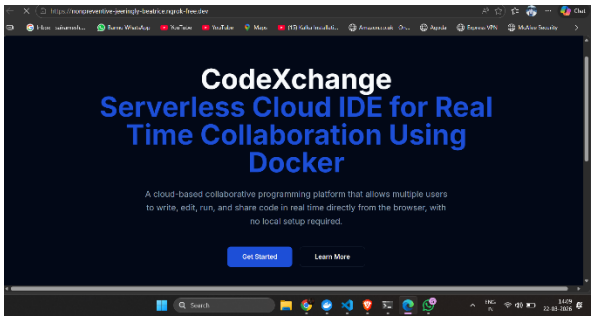

Fig. 3. Creating Project

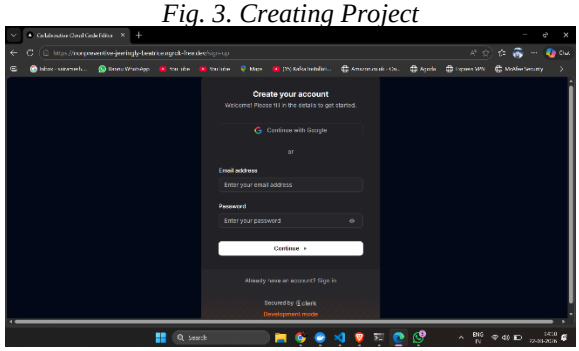

Fig. 4. User Authentication Interface

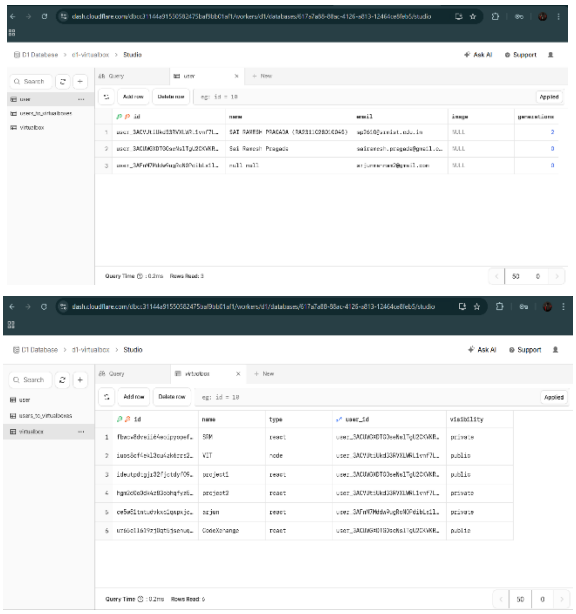

Fig. 5. User Login information

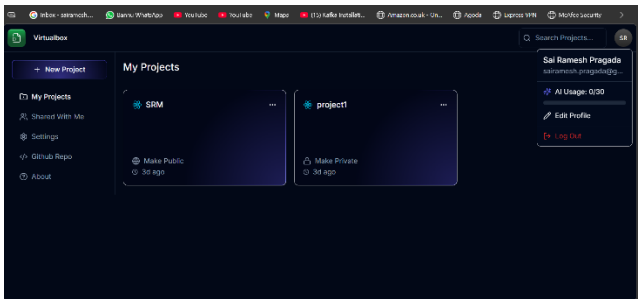

Fig. 6. User Dashboard Interface

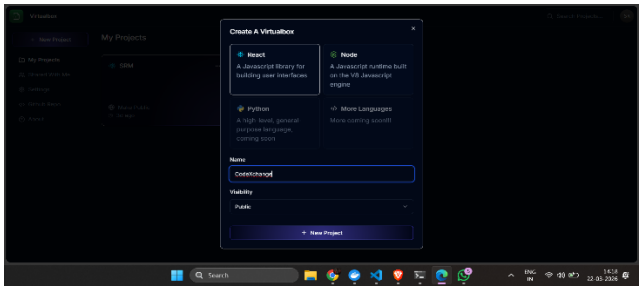

Fig. 7. Create a Project

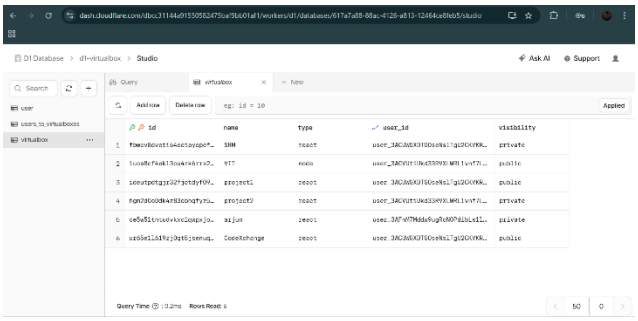

Fig. 8. Cloud File Storage System

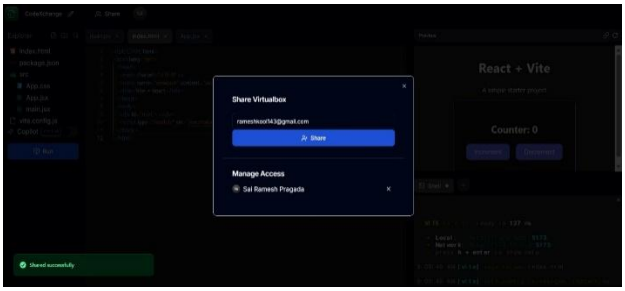

Fig. 9. Project Sharing Interface

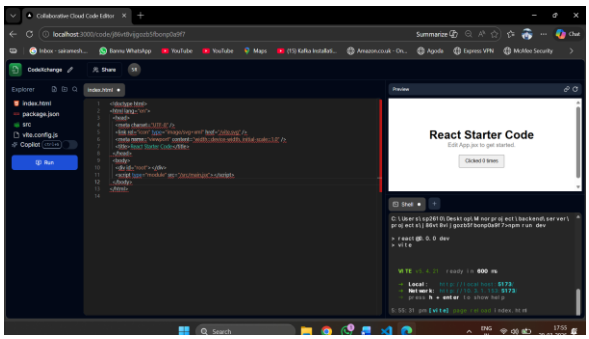

Fig. 10. Code editor with live preview and terminal.

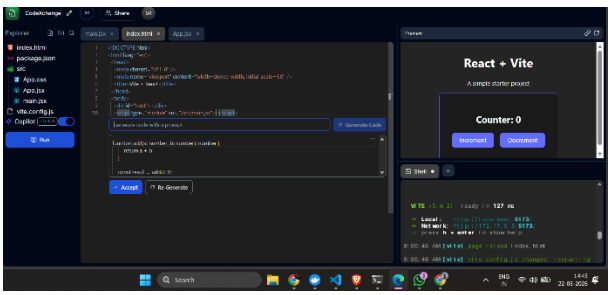

Fig. 12 AI Code Assistance Module

A. System Performance.

The system demonstrated good functionality with regard to concurrent user handling and imitative operations. Code changes were synchronized in real-time with minimal latency and all the members of a shared workspace had the same level of updates. Effective communication protocols were also utilized, which facilitated the communication between the frontend and the backend elements. In addition, the Docker containers made the code execution stable and did not interfere with any user, which allowed maintaining a reliable setup during the various sessions. The serverless backend was very efficient and did not involve scaling manually, which helped to make the system responsive. Besides a baseline performance, the stress testing was performed with the increase of simultaneously on-line users and operations. This system showed consistency in its responsiveness even during moderate to high workloads, proving the usefulness of serverless allocation of resources. The container orchestration mechanism promoted the efficient instantiation and termination of execution environments without occurrence of delays. In addition, memory and CPU consumption were also optimized since the Docker containers were lightweight in nature. These measurements suggest that the platform can serve real-life deployment applications in which multiple users can communicate with each other at the same time without major performance reduction.

B. System Performance Metrics

System-oriented measures are used to assess the functionality of the proposed system on the basis of its efficiency, scalability, and responsiveness. The most important parameters of evaluation will be the response time, success rate of execution, collaboration latency, and system throughput. Response time is the time of user request processing, which is an execution of codes and the API interactions. The lightweight nature of serverless functions and efficient request processing systems are shown to indicate consistently low response times in the system. The success rate of execution of code represents the extent to which the code can execute in isolated Docker environments. Containerization is used to ensure that the failures of execution due to differences in environment are kept at a minimum. Latency of collaboration measures the delay time needed to coordinate code changes between multiple users, and the system is close to real-time synchronized by communicating protocols which are optimized. System throughput is a gauge of the successful requests handled simultaneously implying the scalability of the architecture. All these measures confirm the efficiency of the offered system to provide high-quality and stable development environment.

C. Error Analysis.

Even though the system proved to be highly efficient, there are some limitations that were noted during testing. Synchronization delays that were minor were observed when the conditions of concurrency were extreme, and this may affect user experience when deployed at scale. Moreover, AI-based code suggestions also sometimes generated less precise or less relevant results, which demonstrates the weaknesses of the existing AI models. In certain situations, real-time collaboration was also unstable because of network instability, and temporarily the collaboration was not synchronized. The problems point to the direction of the future developments, where there is much to be done to optimize synchronization algorithms and improve AI accuracy. A further study of such errors found that the latency of the network and the overhead of keeping the same state across multiple clients were the major causes of synchronization delays. There was also observed some temporary data inconsistencies in the case where internet connectivity was unstable though it was corrected as soon as the connection was made stable. The levels of AI-related inaccuracies were higher in the more complicated coding cases that demanded domain-specific knowledge, which supports the fact that more specialized training models are necessary. Also, uncommon execution failures were witnessed during containerization where there were delays during the initiation of the containers under heavy load environment conditions. To solve these issues, network optimization, improving AI API response quality and contextual accuracy, and resource allocation approaches will need to improve the state of affairs.

D. System Effectiveness and Impact on Practice.

The given system has pronounced practical benefits compared to the classical development settings. It creates less time to boot up a development and makes the process of onboarding new users easier, as no local setup is required. The merging of collaboration, execution, and storage in a common platform enhances the efficiency of workflow and lessens relying on tools. Infrastructure wise, serverless computing is the best utilization of resources, incurring less load to operation but no compromise when it comes to performance. Educational use cases are also supported by the platform as it offers accessible coding environments to students without the need of using a high-end hardware. On balance, the system offers the scalable and efficient solution which is consistent with the demands of the modern software development and leads to the further development of cloud-native develop ecosystems. With these benefits, the system has good implications to the industry and academia. Organizations can use the platform to automate development processes, remote working, and lower the expenses of maintaining infrastructure. Scalability is dynamically, and this makes sure that a company is capable of meeting its ebbs and flows without investing in hardware. In education, the platform democratizes access to coding environments to enable students practice and work together without technical constraints. Also, AI-based assistance will add value to the learning process, particularly by offering feedback and advice in real time. These expediency gains underscore the overall effect of the system in changing the organization of software development and learning settings.

V. CONCLUSION

This study introduced a unified and scalable serverless cloud-based Integrated Development Environment that tries to mitigate the shortcomings of traditional software development processes. Combining real-time collaboration, containerized execution based on Docker and AI-assisted code generation into one platform, the suggested system is effective at making the development process simpler, improving accessibility, and efficiency. Serverless architecture provides the ability to allocate resources dynamically, which superbly scales and is highly performant, and does not require manual administration of infrastructure. Experimental findings show that the system provides a useful support to the simultaneous users, a low latency of the collaborative interactions, and a secure and consistent executing environment that is made possible by containerization. The most significant results prove that the offered platform saves a lot of time on the setup process, increases the efficiency of collaboration, and helps developers become even more productive as a result of intelligent assistance. The ability to combine various elements together like authentication, cloud storage and execution environments and collaboration modules helps to have a coherent and user friendly development ecosystem. Nonetheless, some shortcomings were observed such as small mismatched delays during extreme workloads and some irregularities in the suggestions generated by AI. Such obstacles indicate that more optimization should be added to synchronization mechanisms and that more advanced and domain-specific AI models should be introduced. The next area of work can be to optimise the system to allow large-scale deployment in an enterprise, better the accuracy of AI as fine-tuned models, and integrate more languages and communities. Also, the implementation of more sophisticated functionalities like the version control systems, automated testing pipelines and more sophisticated security measures can also push the platform to the next level. On the whole, the suggested system will be a great step to the next-generation cloud-native development environment that is flexible, scalable, and smart in accordance with the current needs of software engineering.

REFERENCE

[1] I. A. Gornev and V. V. Bondarchuk, “Towards Collaborative Coding in RIDE Web IDE,” in IEEE 16th International Conference on Actual Problems of Electronic Instrument Engineering (APEIE), 2023.

[2] Y.-Y. Chen, Y.-H. Lin, Y.-C. Hu, C.-H. Hsia, Y.-A. Lian, and S.-Y. Jhong, “Distributed Real-Time Object Detection Based on Edge-Cloud Collaboration for Smart Video Surveillance Applications,” IEEE Access, vol. 10, pp. 93745–93756, 2022.

[3] “Remote Practical Work Environment Based on Containers to Replace Virtual Machines,” in IEEE Global Engineering Education Conference (EDUCON), 2022.

[4] “Building Temporary Isolated Workspace in Real-Time Collaborative Programming Environment,” in IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2023.

[5] “Collaborative Code Editors: Enabling Real-Time Multi-User Coding and Knowledge Sharing,” in IEEE International Conference on Innovative Mechanisms for Industry Applications (ICIMIA), 2023.

[6] “CodeFlow: Real-Time Collaborative Code Editor,” in International Conference on Knowledge Engineering and Communication Systems (ICKECS), 2024.

[7] “CodeXchange: AI-Powered Code Editing,” in International Conference on Computational Intelligence and Computing Applications (ICCICA), 2024.

[8] “CodeQuest: A Comprehensive Code Learning and Collaboration Platform,” in International Conference on Progressive Innovations in Intelligent Systems and Data Science (ICPIDS), 2024.

[9] “AI-Powered Code Editors: Bridging Academia and Industry,” in International Conference for Emerging Technology (INCET), 2025.

[10] “CodeXpress: Realtime Collaborative Environment,” in International Conference on Circuit, Power and Computing Technologies (ICCPCT), 2025.

[11] J. Dean and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Clusters,” Communications of the ACM, vol. 51, no. 1, pp. 107–113, 2008.

[12] M. Fowler, “Microservices: A Definition of This New Architectural Term,” ThoughtWorks, 2014.

[13] C. Pahl, “Containerization and the PaaS Cloud,” IEEE Cloud Computing, vol. 2, no. 3, pp. 24–31, 2015.

[14] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, “Borg, Omega, and Kubernetes,” Communications of the ACM, vol. 59, no. 5, pp. 50–57, 2016.

[15] E. Jonas et al., “Cloud Programming Simplified: A Berkeley View on Serverless Computing,” University of California, Berkeley, 2019.

[16] T. White, “Hadoop: The Definitive Guide,” O’Reilly Media, 2015.

[17] A. Fox et al., “Above the Clouds: A Berkeley View of Cloud Computing,” University of California, Berkeley, 2009.

[18] D. Merkel, “Docker: Lightweight Linux Containers for Consistent Development and Deployment,” Linux Journal, 2014.

[19] N. Dragoni et al., “Microservices: Yesterday, Today, and Tomorrow,” in Present and Ulterior Software Engineering, Springer, 2017.

[20] S. Newman, “Building Microservices: Designing Fine-Grained Systems,” O’Reilly Media, 2015