ARCHITECTURES FOR AI-AUGMENTED COLLABORATIVE DEVELOPMENT ENVIRONMENTS

Mr. Kunal Manohar Deore, Mr. Neel Ajit Patil, Mr. Gangadhar Suresh Gaikwad, Prof. K. B. Sadafale (Project Guide)[#]

Student, Department of Computer Engineering, Government College of Engineering and Research, Avasari Savitribai Phule Pune University, India
kunaldeore6531@gmail.com | patilneel288@gmail.com | gangadhargaikwad2004@gmail.com

[#]Assistant Professor, Department of Computer Engineering, Government College of Engineering & Research, Avasari, Savitribai Phule Pune University, India
kbsadafale.comp@gcoeara.ac.in

Abstract
Software development in the modern world heavily depends on platforms that support real-time multi-user editing along with artificial intelligence-assisted coding support. This systematic study aims to investigate the architectural basis of these platforms in three areas: synchronization algorithms, execution isolation, and large language model-based software agents. This study includes 35 publications from various digital libraries like the Association for Computing Machinery Digital Library, IEEE Xplore, and arXiv, from January 2022 to February 2026. The study highlights various areas of concern, even though the components of these platforms have been developed to a high degree of maturity.

Keywords: *Collaborative Development, Crdts, Software Engineering Agents, Large Language Models, Cloud Ides*

1. INTRODUCTION

In the following investigation, the architectural basis of modern AI augmented collaborative development environments will be examined, as well as a detailed analysis of the capabilities, limitations, and emerging challenges of such systems.

Software development is in the midst of rapid evolution, as teams increasingly rely on browser-based environments that allow for real-time collaborative development across geographically dispersed teams. Cloud based Integrated Development Environments allow teams to edit code in real-time, utilize isolated execution environments, and incorporate assistance from state-of-the-art large language models. These features are the result of improvements in collaborative editing technologies, containerization technologies, and AI based programming models [1]–[3]. Concurrently, modern AI systems have achieved impressive levels of proficiency in reasoning and code understanding, which can potentially boost the efficiency of human developers and automate routine software development tasks [4]–[6]. The convergence of these trends necessitates a deeper investigation into the architectural basis of AI augmented collaborative development environments.

The modern collaborative development environment must synchronize the actions of multiple users, maintain a consistent execution environment, and provide reliable assistance from AI agents. CRDT based frameworks such as Yjs, Automerge, and Peritext have achieved significant popularity in the software development industry because of the strong convergence properties of the CRDT-based systems, as well as the ease of decentralization [7]–[9]. Execution environments such as Docker Containers and Web Containers offer safe execution of workloads in cloud-based and browser-based environments [10]–[12]. Concurrently, large language models such as GPT-5.3 Codex, Claude Opus 4.6, Gemini 3, DeepSeek V3, and Qwen-2.5 Coder offer powerful features for code generation, debugging, and reasoning across multiple files [13]–[17]. Understanding the interaction of the components of the development environment is vital to the development of reliable human-AI collaborative development environments.

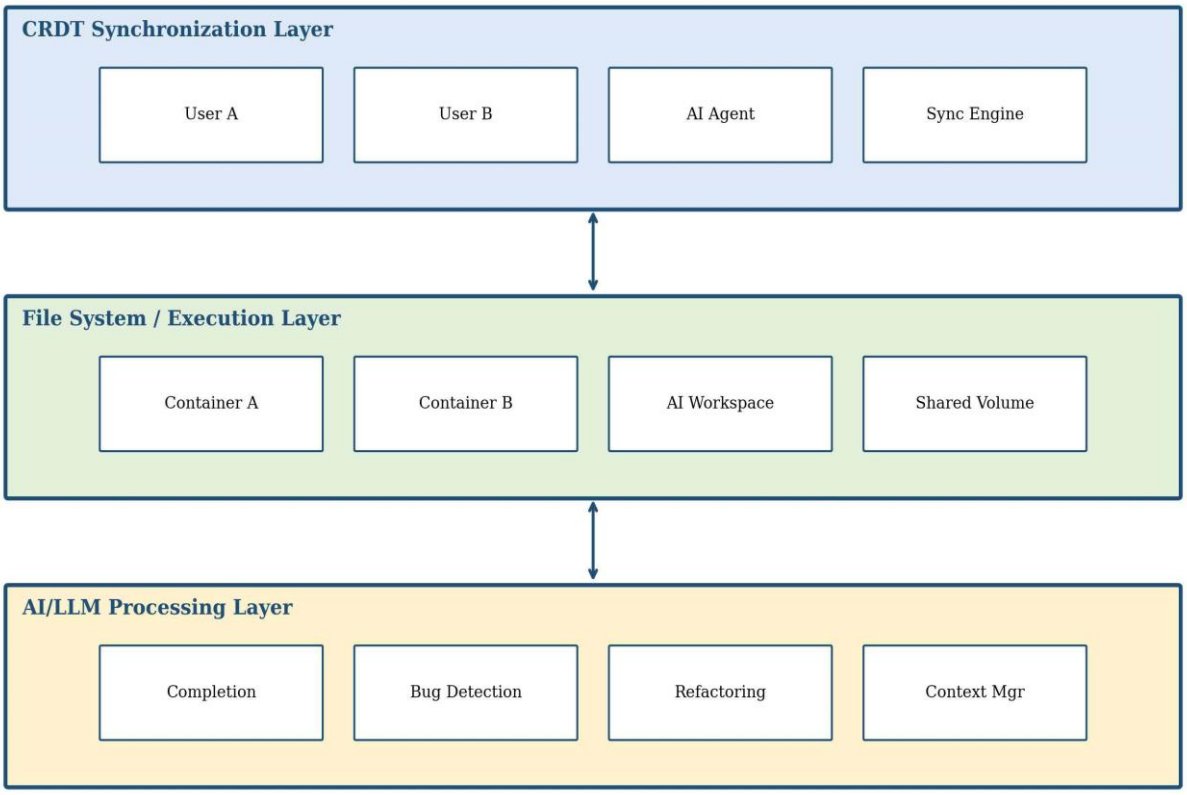

Figure 1. Three-layer architecture for AI-augmented collaborative development.

1.1 Motivation

This study is motivated by a set of observations. AI models have demonstrated remarkable progress in resolving GitHub issues faced by developers, as seen by results on SWE-bench and significant improvements achieved between 2023 and 2026 [18], [19].Cloud based development tools are also witnessing popularity because of their ease of configuration, instant sharing, and ability to maintain a constant state [2],[20]. Recent studies have shown that AI based programming can greatly impact developer productivity and reduce cognitive load, especially during collaborative development [4],[6],[21].

Despite these promising results, existing systems have maintained a separation of concern between collaborative mechanisms and AI processing. Real-time collaborative editing systems rely on OT or CRDT-based synchronization [1], [7],while AI systems utilize separate tool calls or command-based execution models. Combining these components has led to a set of challenges, including maintaining consistency, resolving conflicts, and addressing latency. While research systems such as Eg Walker and hybrid systems using CRDT and OT models have demonstrated promising avenues, unified models are still in their infancy [22], [23].

1.2 Methodology

The sources used for the current analysis include literature, open source code, technical reports, and documentation related to different platforms. Investigations related to CRDTs, OT, and hybrid synchronization approaches were carried out by using existing literature sources [1], [7]–[9], [23]. Sources related to execution were reviewed for containerization and WebAssembly-based systems [3], [10], [12], [24], [25].Evaluations related to AI model capabilities and coding performance were carried out by using technical reports and benchmarking results [13], [14], [26]–[28].The sources used include literature related to ACM, IEEE, NeurIPS, ICSE, and arXiv, along with documentation related to popular development platforms. Sources related to technical reports and architecture were used for the study.

1.3 Contributions

This study provides three major contributions to the literature:

• A systematic taxonomy of architectural components for enabling AI augmented collaborative development, including synchronization mechanisms, execution isolation strategies, and AI integration models [1], [7], [10], [13].

• A comparative overview of prominent platforms and research systems, aggregating knowledge on CRDT models, container based systems, WebAssembly execution models, and intelligent coding agents [8], [10], [11], [29].
• An identification of gaps in existing research on multi-agent collaboration, maintenance of a shared state, memory overhead minimization, and integration of large language models into collaborative development environments [28], [30]–[32]

2. SYNCHRONIZATION IN CLOUD IDES

In a setting where several individuals are working on a common codebase simultaneously, it is vital that every individual gets a clear and consistent understanding of the document being worked on. Collaborative code development tools make use of advanced synchronization techniques in managing concurrent code changes and ensuring document integrity. Operational Transformation and Conflict Free Replicated Data Types are the two primary algorithmic techniques in use.

2.1 Operational Transformation

The Operational Transformation algorithm can be regarded as one of the first techniques to have been created for collaborative editing in real time. Operational Transformation is characterized by a process in which concurrent code changes are transformed in a manner that keeps the document consistent despite receiving changes in an unpredictable order. Basic research in Operational Transformation was done by Sun and Sun in a paper that focused on the conditions for correctness and key properties of transformation that are essential for reliable distributed editing [1].

Several cloud based systems and earlier versions of browser based code editing systems make use of Operational Transformation-based systems owing to their ability to support fast and concurrent code editing. However, Operational Transformation is usually server-based and is characterized by a number of drawbacks in ensuring document integrity in a distributed environment. Operational Transformation makes use of a server for coordinating code changes and can prove to be a bottleneck in a distributed environment. Additionally, ensuring correctness is a daunting task when dealing with several transformation paths.

2.2 Conflict-Free Replicated Data Types

Conflict Free Replicated Data Types (CRDTs) offer a different solution to the OT problem by offering the guarantee of automatic convergence of all copies of the document without the need for a transformation server. CRDTs do not transform operations; instead, it assigns a unique id to the elements of the document and merges the operations in a deterministic manner. This allows the operations to be performed in any order, resulting in the same final state of the document.CRDTs have been widely adopted in modern collaborative systems. The Yjs library provides a highly optimized CRDT for text documents that can be utilized in web-based collaborative editors [7]. The Automerge library provides JSON-based CRDTs that can be utilized for the synchronization of hierarchical data [8]. The work of Peritext extends the CRDT paradigm to the realm of rich texts that can be utilized to synchronize formatting [9]. The work of Walker, as described by Gentle and Kleppmann, resulted in the reduction of memory overhead by improvements over the previous CRDT implementations [22].

CRDTs have been established as a strong contender for the development of collaborative code editing environments. The decoupled nature of CRDTs minimizes the reliance on the server, thus improving the scalability of the system.

2.3 Challenges in Rich Text and Structured Code

Though CRDTs have been successful in text synchronization, in a programming environment, there exist various structured files, their formats, and their relationships. CRDT models have been successful in preserving the sequence of characters, but they fail to support various structured elements like functions, classes, and code blocks. Peritext has been proposed to solve some of the issues in CRDT models, and various attempts have been made to amalgamate CRDT and OT models [9], [23]

As per various research studies, including the detailed study by IJRASET, it has been identified that the usage of CRDT-based collaborative editors has increased manifold after 2021, especially in web-based development platforms [2].

2.4 Assessment

Synchronization capability has been observed to have considerable variability in modern day cloud-based integrated development environments (IDEs). OT-based systems have been found to offer rapid and intuitive collaboration but require centralized coordination. On the other hand, CRDT based systems have been found to offer decentralized and fault tolerant editing but require additional metadata to be added. Large scale projects with complex formatting or inter file dependencies have been found to be problematic, especially in cases involving concurrent human and AI agent collaboration.

To conclude, it has been found that CRDTs have emerged
as the de facto paradigm for collaborative editing in real-
time due to its high degree of convergence and decentralized design. It has been observed that with recent developments, memory concerns are being alleviated, making it suitable for next generation AI augmented IDEs.

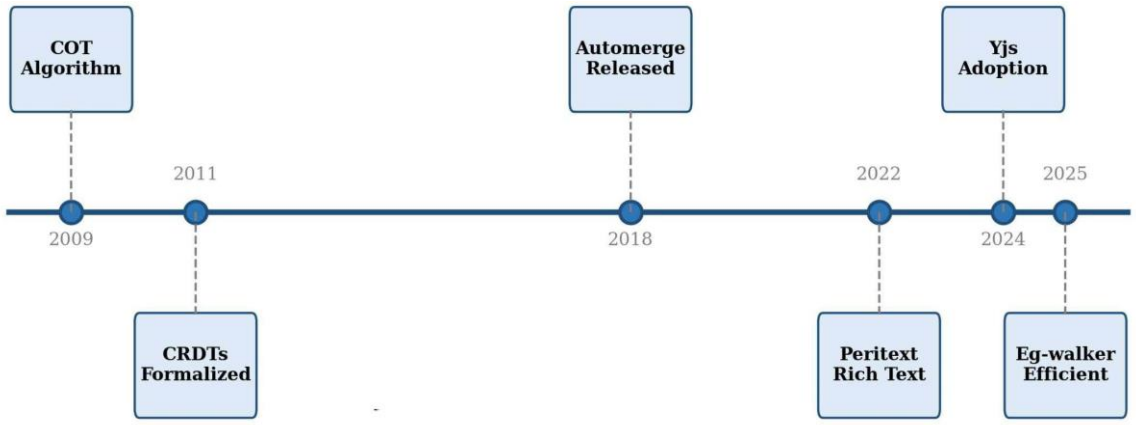

Figure 2. Timeline of key developments in collaborative editing synchronization

To contextualize these advancements, Figure 2 presents a timeline of key historical developments in collaborative editing synchronization.

3. EXECUTION ISOLATION TECHNIQUES

Cloud-based development environments should provide effective isolation of the user workloads while maintaining an authentic execution experience. The isolation of code execution, prevention of interference between workspaces, and protection of infrastructure are the key requirements of such environments. Currently, there exist multiple isolation techniques for cloud-based development environments, which provide unique trade-offs between security, performance, and portability.

3.1 Container-Based Approaches

Containers, specifically Linux containers, dominate the field of isolating cloud based development environments. Containers provide isolated process execution with negligible overhead. Docker has become the de facto standard for container based package distribution.

Malviya and Dwivedi conducted a comparative study of different cloud based development environments, including Kubernetes, Docker Swarm, Apache Mesos, and OpenShift [10]. According to their study, Kubernetes provides robust features for managing container-based development environments, but with increased operational complexity. The improvements in container orchestration and workflow for multiple cloud environments further enhance such features [24],[25].

Containers, however, provide limited isolation due to their shared kernel implementation, which makes them less secure than other alternatives, such as virtual machines, since multiple users' processes execute on the same kernel.

3.2 Lightweight Virtual Machines

Lightweight virtual machines, termed microVMs, have been developed to deliver higher isolation capabilities using hardware virtualization, yet they deliver rapid startup times, like Firecracker, which can boot in under 125 milliseconds. This allows for multi-tenant execution in a secure environment.

MicroVMs have higher security constraints than containerization and, therefore, can be utilized for execution of code written by untrusted entities. Additionally, the lower overhead of microVMs, in comparison to traditional virtualization, makes them suitable for on-demand development usage.

3.3 Browser-Based Execution

Browser-based execution has been proposed, utilizing WebAssembly for execution of isolated workloads in the browser. WebContainers have been developed to execute Node.js applications in the browser, eliminating the need for server side execution for supported projects [11]. This allows for instantaneous startup times for supported projects.An in-depth survey on WebAssembly execution environments, published by the Association for Computing Machinery, has demonstrated performance variations for

WebAssembly execution in various browsers and standalone environments [3]. Comparative studies have been conducted to assess the execution overhead and isolation capabilities of WebAssembly based execution models [12]. Browser based execution offers high portability, yet only JavaScript and associated toolchains have been supported. The memory constraints of the browser environment have been identified as a limitation for large scale execution.

3.4 Assessment

Container based isolation techniques are still predominant due to their maturity level and support for existing ecosystem and development toolchains. Browser based execution offers excellent responsiveness for frontend applications but lacks in other areas. Lightweight virtual machine techniques offer a trade-off between responsiveness and isolation and are best suited for security critical applications.

In all cases, trade-offs need to be made in terms of performance, usability, and security. Although none of these techniques meet all requirements, they form the basis of modern cloud-based application development environments and have implications for AI based collaborative application execution models.

4. LARGE LANGUAGE MODELS FOR CODE ASSISTANCE

This has transformed from basic auto-completion of codes to AI assistants that are capable of reasoning, planning, and executing tasks in a program or project. The latest cloud-based environments for software development are now employing large language models to improve collaboration, debugging, and automation in software engineering tasks.

4.1 Current Model Capabilities

As of early 2026, a number of model families show significant capabilities in software engineering tasks. Proprietary systems, such as Claude Opus 4.6, show extended reasoning and behavior in coordination with multiple agent components [13]. Independent evaluations conducted by Anthropic's security teams on Claude Opus 4.6 revealed that the model detected hundreds of previously unknown issues in widely used open source libraries [26], [33].

GPT-5.3 Codex, a model from OpenAI, has shown significant capabilities in handling contexts, agentic coding, and detailed reasoning in multiple files, suitable for integrated development environments [14]. Google's Gemini 3 model has shown significant capabilities in processing exceptionally large context windows, exceeding two million tokens, suitable for project-wide understanding [15].

Similarly, the open weight model has shown significant improvements in capabilities in software engineering tasks. DeepSeek-V3, a model from late 2024, is a mixture-of-experts model with over six hundred billion parameters and has shown significant capabilities in coding tasks at a competitive price [16]. Qwen2.5-Coder has shown significant capabilities in generating high-quality code and allows for efficient deployment in local environments, even in resource-constrained environments [17].

CodeLM, a research prototype discussed in this paper, is a model that combines a large language model with CRDT-based collaborative editing, Docker based execution, and project wide context retrieval, allowing human, AI, and file system changes to be processed in a unified and synchronized project state. A summary of these leading language models, their providers, and key capabilities is provided in Table 1.

Table 1. Selected Code Models (February 2026)

Model	Provider	Context	Key Capability
Claude Opus 4.6	Anthropic	200K	Extended reasoning; agent coordination [13]
GPT-5.3 Codex	OpenAI	128K	Agentic coding; structured reasoning [14]
Gemini 3 Pro	Google	2M+	Large-context project analysis [15]
DeepSeek-V3	DeepSeek	128K	Open-weight mixture of experts efficiency [16]
Qwen2.5-Coder	Alibaba	128K	Local deployment; strong code quality [17]

4.2 Benchmark Performance

SWE-bench, which was first introduced at ICLR 2024, has become the dominant method for measuring real world problem resolution. While initial model results were below 1 percent, it has since achieved results above 79 percent on standard configurations by 2026 [18], [19]. Figure 3 displays a detailed performance comparison of these leading LLMs across standard software engineering benchmarks. The trajectory of this rapid progress is further depicted in Figure 4, which charts the SWE-bench performance improvement from October 2023 to February 2026.

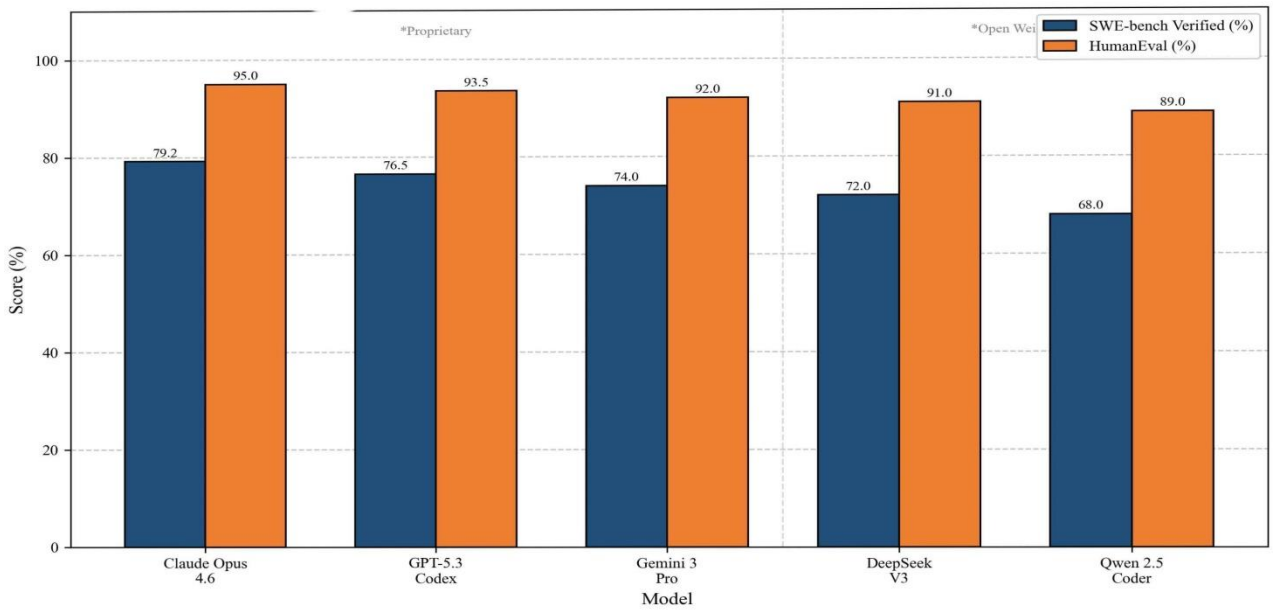

Figure 3. Performance comparison of leading LLMs on software engineering benchmarks.

Other metrics measure class-level code generation. According to Du et al., models have achieved over 80 percent accuracy on synthetic data, but results drop to 20-30 percent on real-world class-level data [27].EvoCodeBench, which was introduced at NeurIPS 2024, highlights the danger of data contamination and emphasizes the need to change the structure of data benchmarks to solve this problem [28].

4.3 Developer Productivity Studies

Empirical studies show that the application of AI development aids improves the productivity of programmers. According to studies published in the Communications of the ACM journal, the application of AI coding development aids by programmers results in improved workflow efficiency [4]. A controlled MIT study showed that significant improvements in the development speed of students using AI development aids resulted in reduced mental load [5]

Case studies by the AMICS 2024 conference show that the application of AI development aids in the industry also results in improved efficiency, especially in repetitive tasks [21]. According to studies published in Nature, collaborative programming using large language models reduces mental load and improves the computational thinking of the student [6].

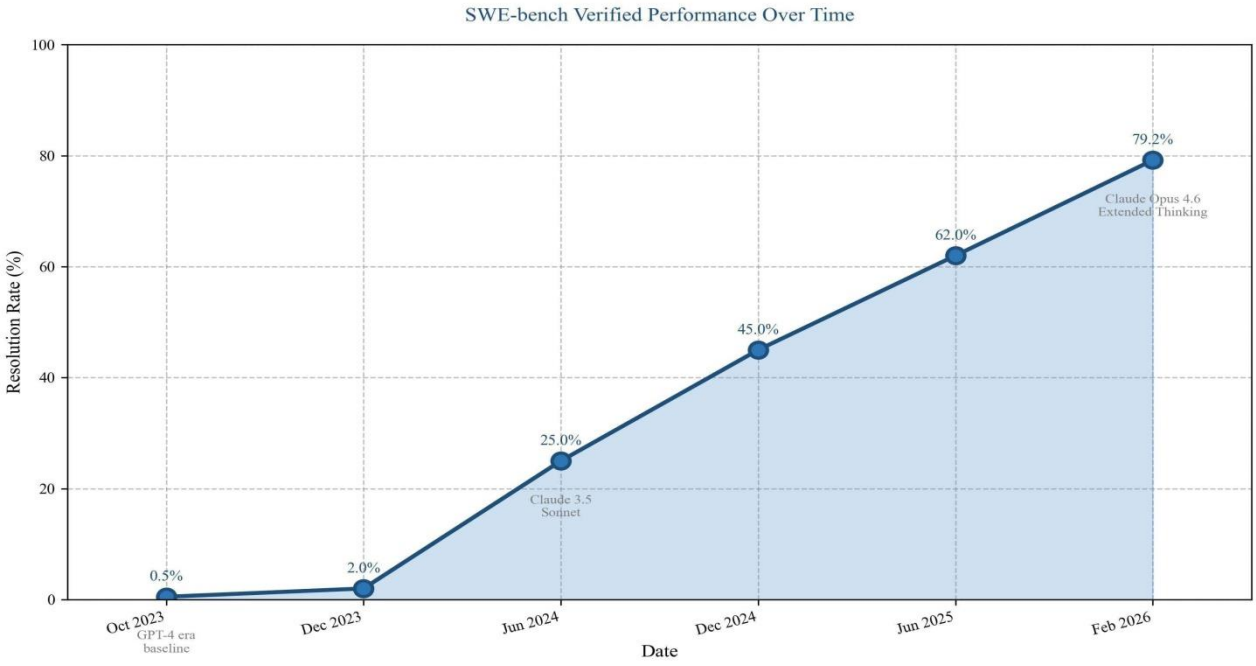

Figure 4. SWE-bench performance improvement from October 2023 to February 2026.

4.4 Agent Architectures

Agent architecture is used by modern large language model driven coding systems. The architecture used by ReAct combines reasoning and action sequences. Execute and Verify architecture breaks down complex processes into smaller steps that can be verified. Orchestration systems for multiple agents are used to solve large scale processes. The OpenHands platform, which was presented at ICLR 2025, provides open-source code for these systems, which can be used to prove these concepts for a range of model types [29]. Other literature on multi agent collaboration provides an overview of how coordination, conflicts, delegations, and context can be achieved [30]. Other experiments are exploring how these can be directly embedded into collaborative editors using a CRDT based synchronization [31]. Other literature provides an overview of how role based access control can be a challenge for collaborative coding [20].

4.5 Assessment

The practical performance of the latest generation of language models for autonomous code generation and collaborative editing support is satisfactory. The gap between proprietary weight systems and open weight systems is shrinking. The focus of the design of the architecture is on the reliable coordination of the agents and the stability of the integration with the real-time editing environment. Although the basic functionality has reached the desired level of maturity, the problems of scaling the multi agent systems, the consistency of the context for long-lived sessions, and the stability of the coordination of human users with AI agents have not yet been resolved.

5. INTEGRATION CHALLENGES IN COLLABORATIVE ENVIRONMENTS

Combining collaborative editing with AI assistance has some challenges that do not have adequate solutions in either domain individually. The synchronization mechanisms, file system operations, and AI reasoning must be consistent among all users and agents.

5.1 The State Consistency Problem

In a collaborative development environment, there can be many state representations interacting with one another. The CRDT layer manages the document content and metadata that controls the synchronization of the document. The file system inside the container manages the files that the compiler can see. The AI agent has a state representing the code base to assist the reasoning and generation of code. These states should be synchronized among the users and agents. If a user types some code, the keystroke should be propagated through the CRDT to the other users and also reach the file system so that the compiler sees the updated state of the code. If the AI agent generates code, the code should be updated based on the last synchronized document view.

Currently, all systems have ad hoc mechanisms to manage the interactions among the users and agents, but no mature system has been developed to manage the states of the collaborative development environment. As a result, many race conditions, stale state, and inconsistency in the project view among the users can happen. Let's consider a real-life scenario to understand the problem better: User A is editing a file, User B is running a terminal command that depends on the file's content, and simultaneously, an AI agent is suggesting code changes to the file. If the operations of the users and agents are not ordered correctly, the operations can happen concurrently in unpredictable ways, causing incorrect code to be built or incorrect code changes to happen. This specific state consistency challenge, involving multiple concurrent actors, is visually represented in Figure 5.

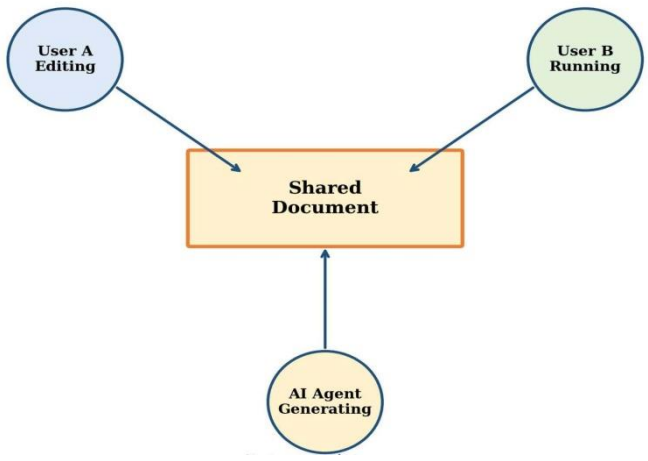

Figure 5. State consistency challenge with multiple concurrent actors.

The solution to the state consistency problem should define a source of truth that has derived views of the truth. The CRDT layer can be the source of truth because it encapsulates all the operations that happen concurrently among the users. The file system can have a derived view of the truth by projecting the state from the CRDT layer, and the AI agents can also have the same source of truth by being subscribed to the CRDT layer.

5.2 Human–AI Edit Conflicts

In the event that an AI agent is tasked with the creation of code with human intervention, conflicts can arise. Several scenarios have been identified as representative of the problems that can arise:

Scenario A: A human writes a function definition, with the AI agent inserting documentation above the function. In the process, the insertion of the documentation interrupts the typing process, leading to garbled output.

Scenario B: An AI agent identifies a bug in the code and suggests corrections. A human corrects the bug manually in an alternative way. Although the corrections are for the same bug, they are incompatible with one another.

Scenario C: In the event of concurrent human intervention, an AI agent is tasked with the refactoring of the code. However, the AI agent assumes an outdated structure of the code, leading to incorrect refactoring.

The most common solution for these problems is the use of simplistic methods. In these cases, the human intervention is prioritized, with the changes made by the AI agent being overridden or queued. Although this solution is safe, it wastes computational power and negates the advantage of the use of AI agents. Prototypes have been developed that suggest more advanced methods. One such prototype was developed by Lehmann and Shauchenka, who proposed the use of AI agents in collaborative systems with the use of CRDT based architecture. In this case, the AI agent is conceptualized as an integral part of the collaborative system. In order for this solution to work better, the use of semantic locking is required. In this case, the AI agent locks the regions of the code that it is modifying.

5.3 Latency Requirements

Different types of interactions in a collaborative environment may require different levels of latency constraint. Keystroke level operations should be synchronized within a window of fifty milliseconds to enable smooth operation in a multiplayer editing environment. AI code suggestions should be provided within a window of a few hundred milliseconds to avoid interrupting the user flow. More complex operations requiring multiple steps in reasoning may require a few seconds.

Meeting all the above requirements simultaneously creates architectural tension, where operations should be non-blocking between AI and CRDT operations, and network factors affect human and AI operations.

To address the above problems, systems are often divided into fast and slow paths, where fast paths are used for CRDT operations, and slow paths are used for AI operations with optimistic updates, rollback, and user interface indicators for user awareness.

5.4 Context Window Limitations

Large language models have limited contextual abilities. While extensive context windows, such as Gemini 3's two million token limit, can be defined, they are still limited and cannot include an entire project and its associated dependencies and history [15]. This creates a challenge for AI models to have a global and coherent view of a project.

Retrieval-based generation attempts to solve this by retrieving relevant code regions on demand. However, this has latency implications and is highly dependent on the quality of selection. Irrelevant information can hinder model reasoning and consume precious tokens, while missing context can result in incorrect suggestions.

5.5 Assessment

Among the technological pillars that have been examined in the current research, the least developed one is integration. The challenges in the field of integration are well understood, including state consistency, human AI conflicts, different tolerance levels of latency, as well as the limited context. However, the development of comprehensive solutions to address the above challenges has not been achieved yet. These areas of challenge offer significant research avenues. The development of new frameworks that can effectively manage the state of unification among CRDTs, file systems, as well as AI reasoning systems, can effectively address the challenge of inconsistency in the state of the systems. The development of better human AI coordination mechanisms can also enhance the reliability of AI based collaborative environments.

6. REVIEW OF CURRENT PLATFORMS

There are several contemporary development platforms that exemplify different collaboration methodologies, artificial intelligence integration strategies, and execution isolation techniques. The design decisions made in each platform involve trade offs between performance, synchronization guarantees, and artificial intelligence support. Table 2 outlines a comprehensive comparison of several contemporary platforms based on their collaboration methodologies, AI integration strategies, and execution environments.

Table 2. Platform Comparison (February 2026)

Platform	Collaboration	AI Integration	Execution
Replit	Real-time multiplayer	Native AI and containers	Containers
GitHub Codespaces	Limited shared sessions	GitHub Copilot	Containers
Cursor	Primarily local collaboration	Deep coding agents integration	Local runtime
Windsurf	Local-first collaboration	Agent workflows	Local runtime
StackBlitz	Real-time editing	AI assistant	WebContainers

There also exist open-source alternatives, such as Code Server, which provides a browser-based interface for Visual Studio Code, as well as the OpenHands platform, which provides a platform for agent workflow environments that can accommodate different AI architecture types [29]. Cline also provides a model agnostic assistant that can be integrated into local or remote Visual Studio Code environment

A comprehensive overview of collaborative coding platforms that incorporate role-based access control was provided by Scaliti in 2025 [20]. Another study by ACM, which examined the feasibility of real time collaboration through the Visual Studio Live Share platform, also examined the role of coordination among developers in real time collaborative environments [34]. These studies highlight the importance of synchronization, state management, and latency in such environments.

The current crop of development environments has managed to strike a good balance in terms of providing collaborative features as well as flexibility in execution environments. The browser based environments, such as StackBlitz, focus on providing a high degree of portability, whereas the container-based environments, such as Replit and Codespaces, focus on providing a high degree of isolation as well as reproducibility of environments. The local-based environments, such as Cursor and Windsurf, focus on providing a high degree of responsiveness as well as integration with AI agents.

7. RESEARCH DIRECTIONS

Our analysis has also identified areas where further investigation is required. Though significant progress has been achieved in the development of the constituent components of collaborative platforms, such as CRDT frameworks, container isolation, and large language models, the integration of the components into a common development platform has not been fully achieved. The above areas of development define the avenues for deeper investigation of the architecture of the platforms.

7.1 Unified State Management

Currently, the platforms utilize different state management systems for the code editor, file systems, and AI reasoning systems. A unified state management system that maintains the state of CRDTs, execution environments, and AI agents in a consistent manner has not been achieved in the platforms. The development of such a system could reduce synchronization time, eliminate the need for conflicts to be resolved, and enhance reliability in collaborative editing [1], [7], [9].

7.2 Human-AI Coordination Protocols

The involvement of AI agents in collaborative platforms necessitates the development of coordination protocols that define the procedures of cooperation in the development platforms. The protocols should address the areas of conflicts, intentions, and intervention limits in the platforms. The development of collaborative AI platforms has shown the importance of the development of protocols that allow human agents and AI agents to work together without overwriting each other's contributions [29], [31]; however, the development of such protocols has not been fully achieved in the platforms.

7.3 Semantic Locking Mechanisms

The involvement of AI agents in the development platforms has been identified as a challenge in the development of collaborative platforms because of the occurrence of conflicts when AI agents edit the code in the same areas as human agents. The development of the platforms has been challenged by the inefficiency of the current procedures of handling conflicts in the platforms, where AI contributions to the

code are ignored in the name of resolving conflicts [31]; however, the development of semantic locking mechanisms has been identified as a possible solution to the challenge [31].

7.4 Verification of AI Modifications

The current state of AI systems can generate code that is functionally correct but contains logical inconsistencies or security issues. Verification methods that can guarantee that AI generated code modifications do not affect the semantics of the original code can increase trust in AI systems and enable autonomous collaboration. Such verification methods could include static verification, dynamic verification, or a combination of both using inference-based verification. Benchmarks for class-level code generation emphasize the need for verification [27], [32].

7.5 Privacy-Preserving Collaboration

With the integration of AI inference using external APIs in code collaboration systems, issues related to privacy are a key concern. For instance, code containing sensitive information can be remotely executed on a server. Federated learning and secure computation can address these issues but are in their infancy for code collaboration systems and need to be evaluated for their efficiency.

8. CONCLUSION

This study aims to explore the architectural foundation of modern AI-assisted collaborative development environments. As software development increasingly adopts browser-based tools and collaboration, it is necessary to integrate three key components into collaborative development environments: synchronization mechanisms for collaborative editing in real-time, safe execution environments, and intelligent assistance through large language models. The review has found that each component has been achieved at maturity level in isolation.

For instance, CRDT based frameworks have been found to offer robust convergence guarantees for collaborative editing in real-time. Similarly, containerization and browser-based virtualization have been found to be effective in offering flexible and efficient isolation mechanisms for code execution. Moreover, it has been found that modern language models have been achieved at high maturity levels with advanced reasoning capabilities to enable effective collaboration in programming activities. However, integration has been found to be the major challenge in creating effective collaborative development environments.

For instance, maintaining consistency in editor, file system, and AI agent states has been found to be challenging in practice. Similarly, effective coordination between human and AI activities in collaborative development has been found to require mechanisms that are better than what has been achieved in modern collaborative development environments. Discrepancies in latency requirements and limited context windows have been found to introduce uncertainty in collaborative development activities. The study has found that the rapid advancements in AI have introduced promising prospects for modern collaborative development environments to be based on architectures that can effectively facilitate continuous cooperation between humans, AI agents, and execution environments in the near future.

References

[1] D. Sun and C. Sun, "Context-based operational transformation in distributed collaborative editing systems," IEEE Transactions on Parallel and Distributed Systems, vol. 20, no. 10, pp. 1454–1470, 2009.
[2] IJRASET, "A comprehensive study on real-time web IDE collaborative code editors," 2024.
[3] A. C. Surveys, "Research on WebAssembly runtimes: A survey," ACM Computing Surveys, 2025.
[4] A. R. Group, "Measuring GitHub Copilot's impact on productivity," Communications of the ACM, 2024.
[5] M. H.-C. A. Group, "Evidence from a field experiment with GitHub Copilot," MIT Research, 2024.
[6] N. C. E. Team, "LLM-based collaborative programming: Impact on students' cognitive load and computational thinking," Nature Communications, 2025.
[7] K. Jahns, "Yjs: Shared editing powered by CRDTs," https://yjs.dev, 2024.
[8] M. Kleppmann and A. R. Beresford, "Automerge: Real-time data sync between edge devices," in Proceedings of MobiUK, 2018.
[9] G. Litt, S. Lim, M. Kleppmann, and P. Van Hardenberg, "Peritext: A CRDT for collaborative rich text editing," in ACM HCI (CSCW2), 2022.
[10] S. Malviya and R. K. Dwivedi, "A comparative analysis of container orchestration tools in cloud computing," in IEEE International Conference on Cloud Computing, 2022.
[11] S. Team, "WebContainers," https://webcontainers.io, 2024.
[12] "A comparative analysis of WebAssembly workflows across browser and standalone environments," 2025, arXiv:2512.04089.

[13] Anthropic, "Introducing Claude Opus 4.6," https://www.anthropic.com/news/claude-opus-4-6, 2026.
[14] OpenAI, "GPT-5.3 Codex," https://openai.com/index/introducing-gpt-5-3-codex/, 2026.
[15] G. DeepMind, "Gemini 3," https://deepmind.google/models/gemini/, 2026.
[16] D. Team, "DeepSeek-V3 technical report," 2024, arXiv:2412.19437.
[17] Q. Team, "Qwen2.5-Coder technical report," 2024, arXiv:2409.12186.
[18] C. E. Jimenez et al., "SWE-bench: Can language models resolve real-world GitHub issues?" in Proceedings of ICLR, 2024.
[19] V. AI, "SWE-bench benchmark," https://www.vals.ai/benchmarks/swebench, 2026.
[20] Scaliti, "State-of-the-art review of real-time collaborative coding platforms," 2025.
[21] A. R. Committee, "The impact of GitHub Copilot on developer productivity from a SWEBOK perspective," in Proceedings of AMCIS, 2024.
[22] J. Gentle and M. Kleppmann, "Collaborative text editing with Eg-walker: Better, faster, smaller," in Proceedings of EuroSys, 2025.
[23] J. Li et al., "Design and implementation of a CRDT-OT hybrid system for engineering document real-time collaborative editing," 2024.
[24] IJSAT, "Container orchestration and Kubernetes enhancements," 2025.
[25] Unknown, "Containerization in multi-cloud environment: Roles, strategies, and challenges," 2024, arXiv:2403.12980.
[26] Anthropic, "Evaluating and mitigating the growing risk of LLM-discovered 0-days," https://red.anthropic.com/2026/zero-days/, 2026.
[27] Y. Du et al., "Evaluating large language models in class-level code generation," in Proceedings of ICSE, 2024.
[28] E. Team, "EvoCodeBench: An evolving code generation benchmark," in Proceedings of NeurIPS, 2024.
[29] X. Wang et al., "OpenHands: An open platform for AI software developers as generalist agents," in Proceedings of ICLR, 2025, arXiv:2407.16741.
[30] "Multi-agent collaboration mechanisms: A survey of LLMs," 2025, arXiv:2501.06322.
[31] F. Lehmann and U. Shauchenka, "Collaborative document editing with multiple users and AI," 2025, arXiv:2509.11826.
[32] "Evaluating LLM performance on real-world class-level code generation," 2025, arXiv:2510.26130.
[33] Axios, "Claude Opus 4.6 uncovers 500 zero-day flaws in open-source software," 2026.
[34] A. S. E. Group, "Understanding real-time collaborative programming: A study of Visual Studio Live Share," ACM Transactions on Software Engineering, 2024.
[35] D. Sun and J. Xu, "Real-time collaborative programming in undergraduate education," Journal of Educational Computing Research, 2024.