iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 13 **Issue:** XI **Month of publication:** November 2025

DOI: https://doi.org/10.22214/ijraset.2025.75276

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue XI Nov 2025- Available at www.ijraset.com

A Comprehensive Study on Real-Time Web Ide Collaborative Code Editors

Mrs. Aparna A Veer[1], Omkar Mane[2], Prathamesh Jadhav[3], Atharvi Chevale[4]
Department of Artificial Intelligence and Data Science, AISSMS Institute of Information Technology, Pune, India

Abstract: ***Real-time collaborative code editors allow developers, working in distributed environments, to contribute to the same source code at the same time. This type of systems needs to host complex concurrency management, conflict resolution, and high-level user awareness logic to ensure various developers' workflow. The following survey paper provides a structured review of twenty influential research papers: both journal articles and preprints published in the period from 2021 to 2025. We research the basic algorithms used, such as OT and CRDTs, architectures, and consider human-centric aspects. Moreover, we compare these projects' performance measures, such as scalability, synchronization latency, and usability. The reviewed systems demonstrate high efficacy, with up to 95% operational convergence accuracy and 92% user satisfaction rates in empirical deployments. This work concludes by synthesizing a clear map of unresolved research gaps, primarily in data provenance, large-scale latency optimization, and the nascent field of AI-driven collaborative coding.***

Keywords: Operational Transformation Algorithm , Conflictfree Replicated Data Type(CRDTs) , cleark , tldrw , webSockets

.

I. INTRODUCTION

The field of software development has seen the most dramatic changes in the past decade. The old paradigm of cooperating teams sitting in the same physical location has gradually given way to a new collaboration paradigm that is based on globally distributed teams collaborating through cloudbased tools. The reason for this evolution goes beyond technological development, and is intimately related to fundamental shifts in the practices of software engineering. Given the broad adoption of Agile and DevOps methodologies (focusing on rapid iteration, continuous integration, and frequent peer feedback) the need for collaboration platforms which provide an easy means for working together across locations has been high. Distributed collaboration has become the operational model of choice for most software organizations, helped by the normalization of remote work which was accelerated during the COVID-19 pandemic: remote workers are the new norm [1]. In this regard, real-time collaborative code editors (RCCE) have been introduced as a pillar of the modern development ecosystem, which fundamentally changes the code editing into a shared, synchronous experience.

Traditional asynchronous data development workflows that are built off version control systems (VCS) such as Git offer good versioning and rollback capabilities, but create friction in the real-time development workflow. The usual process of making local edits, committing, pushing and merging gets in the way of rapid back-and-forth of ideas between collaborating partners. In the context of collaborative tasks such as pair programming, collaborative debugging or live teaching, this latency is counterproductive as the feedback correction cycle becomes asynchronous and fragmented [2]. Solving this gap, RCCE systems allow multiple users to view and edit the same code file concurrently and maintain the shared workspace that mimics the urgency of face-to-face collaboration in the field [3]. Visual Studio Live Share, CodeDive, and Replit have shown that low-latency synchronization results in productivity and efficiency during learning. These systems have awareness mechanisms in the form of real-time cursor tracking, chat, and even built-in execution environments to ensure that the team is cohesive and context is not lost between sessions of the application.

However, there is a tremendous amount of technology hidden behind this smooth experience. From a distributed systems point of view RCCE platforms have to solve one of the most challenging issues, data consistency across multiple replicas under concurrent edits and unpredictable network conditions[4]. If two users edit the same file at the same time, even if it is time-slice based, there should be a guarantee that all replicas will converge to an identical state which is valid to the system; without any user's intention being lost. This need gives rise to the dual goals of intention preservation and convergence that serve as the background to collaborative editing theory [5]. Initial implementations were centralized in nature to ensure ordering, but as the scalability requirements increased, decentralized algorithms became very important and essential for the better scalability of the Stellar Network [6]. Two paradigms of algorithms have been leading the field:

Operational Transformation (OT) and Conflict-free replicated Data Types (CRDT). Introduced in the late 1980s, Operational Transformation (OT) is a protocol-oriented model that is based on the transformation of incoming operations with respect to concurrent ones so that all clients apply compatible changes [7]. Although OT has been used to enable wellknown collaborative systems like Google Docs and Etherpad, it sufferers from the centralized transformation control mechanism that constrains scalability in the distributed environments. Contrary to the above, CRDTs provide data-centric solution. They employ mathematically-commutative operators which ensure deterministic convergence across replicas without centralized coordination [5]. For this reason, CRDTs are especially applicable for peer-to-peer systems, offline editing, and large-scale collaborative systems. Even more recently, lightweight CRDT frameworks have reduced the metadata overhead and increased network efficiency even further and facilitated more seamless real-time performance of browserbased IDEs [8].

In the past three years, the adoption of artificial intelligence (AI) in collaborative spaces has given this evolution another dimension. Dong et al. proposed the notion of ”selfcollaboration” where LLM can serve as co-developers that can jointly write, refactor, and review codes with human programmers, even with no initial human-written code (known as ”no-code”). Wang et al. advanced this idea and looked into how small specialized language models could improve the local decision-making in hybrid collaborative systems, which would improve responsiveness and privacy. Hu et al. extended this paradigm to multi-agent collaboration networks, showing how distributed AI agents can autonomously coordinate coding tasks and enhance team throughput. These developments mean the start of AI augmented collaboration where intelligent agents become active participants of shared programming environments.

Equally important are the human/human usability factors of real-time collaboration. Tan et al conducted large scale empirical studies of Visual Studio Live Share and found that awareness mechanisms (cursor visibility, participant highlights and session roles) have a direct impact on user trust and efficiency of collaboration [1]. Li et al. conducted an enterpriselevel analysis on collaborative notebook usage, finding three important personas, namely, Builder, Analyst, and Consumer, with different requirements and workflow styles Li et al., 2023. Quaranta et al. and Ayobi et al. expanded this view to computational notebooks, and pointed out that there are still issues in coordinating the state of execution and visual output between users [11].S Taken together, these studies demonstrate that algorithmic correctness is only one aspect of successful collaboration that needs to be considered; careful interface design and cognitive support systems are also important.

Despite the great achievements, many open issues still remain unsolved. Large codebases still have synchronization bottlenecks that adversely affect performance while concurrently editing. Preserving causality and edit intentions across thousands of distributed nodes are non-trivial issues [8]. The necessity to trace code contribution to individual code contributors (users or AI collaborators) is another challenge that is emerging as a potential barrier to achieving transparency and accountability in mixed human-AI editing sessions [9]. Furthermore, with the shift of code collaboration to the cloud, security, privacy, and data ownership concerns continue to be relevant concerns . Solving these gaps will need a multidisciplinary approach bringing together distributed computing, humancomputer interaction and machine learning.

This survey is a systematic literature review of the status of real-time collaborative code editors from 2021 through 2025. It weaves together into one coherent story twenty major journal and preprint papers (excluding the ones from the world’s two leading journals, the Institute of Electrical and Electronics Engineering and the American Association for the Advancement of Science) on algorithmic roots, usability research, and emerging artificial intelligence-driven systems.s The aim is to both give a conceptual map of the field and to give a technical comparison of important architectures. Section II is about background concepts and synchronization rudiments. Section III contains a detailed review of the literature. Section IV describes the state of the art, Section V summarizes the research gaps, and Section VI concludes with remarks about the future work and open problems.

II. BACKGROUND AND FUNDAMENTAL CONCEPTS

A. Core Architecture and Awareness

Real-time collaborative code editors are distributed systems that are responsible for keeping a shared and synchronized state of a source code document among a set of clients. The basic problem is to handle concurrent operations from users located in different geographic locations, and all the replicas should end up with the same state, but the user intent should be preserved. Modern RCCE architectures usually use a clientserver or peer-to-peer topology, low latency transport protocols like WebSockets are applied for real-time data synchronization. A crucial part of these systems is the mechanism of ”awareness” which brings contextual information about their collaborators to the user such as cursor locations, selections and presence status, thus mimicking the immediacy of faceto-face collaboration and enabling team coordination [1].

B. Operational Transformation (OT)

Operational Transformation (OT) is a basic algorithmic strategy for consistency in realtime collaborative system. OT is protocol-centric, focusing on the transformation of operations (e.g. insert, delete) and not the data itself. When a client creates an operation it is sent to a central server (in many common implementations) which transforms it against a history of concurrent operations before sending it to other clients. This transformation process involves adjusting the parameters of the operation (such as position) to ensure that it has the right effect on a document state that has already been modified by others. The fundamental principles of OT (convergence, causality, preservation and intention preservation) play an important role in preserving the shared state of a coherent state, although constructing the correct transformation functions for complex operations is a major challenge [14].

Conflict-free Replicated Data Types (CRDTs) Conflict-free Replicated Data Types (CRDTs are a alternative, purely data-centric, approach to the OT method and are designed specifically for use with decentralized and eventually consistent systems). CRDTs are data structures of which mathematical properties can be reasoned about and which can only converge in ways that don't require any central entity or complicated imagination logic. This is accomplished by designing operations to be commutative, i.e. the order in which they are applied does not matter to the end state. CRDTs come in two different types: state-based (CvRDTs) where the entirety of the replicas is combined and then operation-based (CmRDTs) where operations are broadcast. [6].

III. LITERATURE REVIEW

Tan et al. Proposed a large-scale empirical study on Visual Studio Live Share to study real-time collaborative programming in an enterprise setting. Their methodology included telemetry study and user surveys of the professional developers. Their results showed that awareness mechanisms such as shared cursors, participant highlights and session roles are key factors that directly affect user trust, patterns of coordination and efficiency of collaboration. This work gives strong empirical evidence to the importance of human-computer interaction (HCI) and awareness design in the adoption and success of RCCE tools, which changes the focus of attention from the correctness of algorithms. Limitations: the study focuses on a single mature platform (VS Live Share) and a professional enterprise context which might not generalize fully to open source or academic pair programming contexts [1].

Li et al. Proposed an enterprise-level study to grasp the specific needs of users in the collaborative Python notebooks such as Jupyter. Their methodology used qualitative techniques, such as semi-structured interviews with users of the enterprise, that sought to gain insights on the flow and pain-points of the process. Their results were three user personas, the 'Builder' who writes complex code and models; the 'Analyst' who explores data and prototypes; and the 'Consumer' who reviews results and reports. Each persona showed different collaboration needs and styles of working. One of the contributions of this work is that there are multiple flavors of 'collaborative' data science, and that useful tools must support these various roles, especially around shared executable state and outputs. Limitations include being geared towards notebook environments which have different statemanagement needs (such as kernel state) than general-purpose code editors (such as simple text editors) [2].

Park et al. Proposed and introduced CodeDive, a novel web-based Integrated Development Environment (IDE) with integrated real-time code activity tracking. Their methodology consisted of creating a cloud native system, probably based on WebSockets, centralized backend architecture, to synchronize code edits and user activities. The proposed solution offers fine-grained activity logs and session re-play capabilities, which are aimed at improving not only synchronous collaboration, but also asynchronous code reviews and developer handoffs. This work shows the useful combination of real-time collaboration and developer analytics and activity logging. Limitations that are common with such prototype systems include questions of scalability and performance under the heavy and concurrent load of a large number of simultaneous users, which are not yet proven in large-scale industrial deployments
[3].

More et al.Proposed PeerLink, a generic collaborative platform, aimed at integrating different aspects of the software development lifecycle. Their methodology was about the design and implementation of a unified platform with a real-time code editor and project management tools, chat functionalities and possibly version control integration. The solution proposed is to provide an all-in-one collaborative environment, thus easing the context-switching and friction developers have to go through when using multiple, disparate tools. This work highlights the current trend in the market towards fully Integrated Collaborative Environments (ICEs), instead of isolated editor plugins. The paper appears to be more of an overview of the features rather than a deep technical analysis of the concurrency control algorithm and its performance for the core code editing component [4].

Litt et al. proposed Peritext, a new Conflict-free Replicated Data Type (CRDT) which is specifically tailored for the problems of collaborative rich-text editing. To achieve that, the authors have developed a new CRDT data structure, which robustly supports not only plain text insertions and deletions but also overlapping text formatting spans (bold, italic, comment).

The solution we propose handles formatting spans in such a way as to preserve user intentions when coeditors make concurrent alterations, effectively preventing the problems of ”interleaving” (fragments) of formatting in more basic collaborative text editors. While focused on rich text, this research has direct application to RCCEs for features such as collaborative comments, inline documentation and semantic highlighting. Limitations: Simple text-based CRDT implementations come with less overhead and metadata complexity compared to more sophisticated CRDT implementations that include rich varieties of data structures, such as [5].

Weidner et al. introduce ”Collabs” which is a flexible and high-performance software framework for building collaborative applications using CRDTs. Their approach was to build a developer-friendly library that abstracted the underlying complexity of CRDT data structures and network synchronization. The proposed framework includes optimized, general-purpose CRDT implementations (Collabs), as well as the network layer, so that developers can focus on developing application logic and not worry about maintaining consistency. The motivation behind this work is to make strong CRDT-based applications more democratic through lowering the barrier to entry. As a general purpose framework its limitations may be that it is not as highly optimized for the specific use case of the code editing than a custom-built, monolithic system might be [6].

David et al. investigated the use of CRDTs in the new field of real-time collaborative multi-level modeling, which is frequently found in Model-Driven Engineering (MDE). Their methodology was to adapt the CRDT concepts to manage the consistency not only between a single, flat document, but between several, interconnected models (such as UML diagrams or domain-specific models), which have hierarchical relationships. They proposed a CRDT-based consistency protocol for these multi-level models, to ensure that changes in one model (e.g., a metamodel) are properly and automatically propagated to the instances of the models. This contribution extends collaborative theory beyond the realm of flat text files, which is very relevant for modern-day IDEs, which contain complex project structures, dependencies, and configurations, and not just individual code files. Some limitations are that the conceptual complexity is high and application to the plain text code editing is less-immediate than other CRDT research [7].

Dong et al. proposed ”self-collaboration” for code generation with Large Language Models (LLMs). Their approach was based on an LLM such as ChatGPT, but in an iterative feedback loop where the model plays the role of several ”agents,” who generate, discuss and improve code in a structured and conversational way. Their findings showed that this ”self-collaboration” approach can make AI-generated code significantly better in quality, coherence and correctness compared to the single-pass generation. This work is at the intersection of collaborative systems and artificial intelligence, and rephrases the LLM as an active collaborator and not passive. Limitations: The cooperation is simulated (AI-withAI), and the research still needs further investigation on the complex dynamics of a mixed human-AI real-time editing session [8].

Wang et al., gave a survey on the rise and capabilities of Small Language Models (SLMs) in a world dominated by LLMs. Their method collected and classified recent research on SLMs, where they lay much attention to their training efficiency, specialized abilities, and deployment benefits. They concluded that SLMs are important for on-device latencysensitive and privacy-preserving applications. This survey is particularly pertinent to the design of RCCEs in the future. It posits a hybrid model in which fast local SLMs are used for the handling of real-time tasks such as auto-completion and linting, and for larger LLMs running on a cloud, the big non-real-time collaborative tasks such as refactoring, documentation generation, etc. are used. As a survey, it has the main downside that it characterizes the state of art, rather than presenting a single new system [9].

Hu et al. proposed a framework for multi-agent collaboration networks that are able to ”self-evolve” to manage complex software development tasks. Their method outlined a system in which multiple, specialised AI agents can coordinate independently, assign sub-tasks (such as coding, testing and debugging), and alter their collaborative system and communication methods in order to improve team performance. This suggested solution represents a very independent approach to AI-driven software development. It is a limit test of AI collaboration, pointing at a possible future wherein human developers could assume the role of supervisors of a team of super-agent AI assistants inside an IDE. This work is primarily theoretical and forward-looking and there are still many practical and ethical issues to overcome before it can be implemented in a real-world RCCE [10].

Quaranta et al. were focused on trying to identify and define best practices for collaboration, specifically regarding computational notebooks. Their method consisted of studies, probably interviews and observational workshops, with data scientists so as to understand their current, often ”ad-hoc”, collaboration practices and ongoing challenges. Their results showed that they encountered serious problems in the synchronization of the executable state (that is, the kernel and memory variables) and the visual outputs of the notebook cells, a problem unique to this environment. This work adds to the work of Li et al. [4], who suggested quantitative measures on what features and workflows a notebook-based collaborative tool needs to support. Limitations: The findings focus on the notebook paradigm, so it might not be applicable in other typical source code editors [11].

Ayobi et al. explored the use of computational notebooks as tools of ”co-design” (collaborative design). Their approach investigated how interdisciplinary teams not only use notebooks to analyze their research data, but also as a common vehicle to generate ideas, prototype and communicate complex concepts with one another. Their conclusions were that the blend of formatted text, executable code, and rich visualizations that comprise the notebook format facilitates a ”literate” style of collaboration which is very helpful in early research and design stages. This paper establishes the place of collaborative notebooks as a human-computer interaction (HCI) tool for creative design, instead of a data science tool. Its drawbacks are that it focuses on the co-design process and not on the enabling technology for real-time synchronization[12].

Santhi et al. described the design and implementation of a web-based collaborative code editor. Their approach involved describing the development of a full-blown system with common web technologies, most likely a Node.js server, a modern frontend framework and WebSockets to transfer data in real time. The proposed solution is a practical case study for the construction of a functional, albeit a basic, RCCE. This work is useful in pointing out common issues of implementation and design patterns for students and researchers new to the field. Its limitations include that it probably does not provide a new concurrency algorithm, but rather describes the application of existing, known techniques (such as a centralized OT) to a particular web application [13].

Kurniawan et al. created an application for real-time code editor named ”CodeR” dedicated to collaborative programming. Their approach was to build a client server system and measure the performance of synchronization delay and consistency maintenance. The proposed system makes use of a particular concurrency control model (probably some kind of centralized Operational Transformation) to handle simultaneous edits, and to ensure that all users have a consistent view of the code. This paper, together with its sequel [15], is a clear and complete case-study of an entire RCCE system, from design to implementation. Limitations include its architecture is based on established OT principles which may limit its scalability because of its centralized design [14].

Soesanto et al. provided a more formal and detailed publication on the ”CodeR” system, which was introduced earlier in [14]. Published in Procedia Computer Science, this method probably contains some hard technical details, performance benchmarks and perhaps results of user studies. Their results probably include quantitative measures of end-to-end latency, operational throughput and user satisfaction, which can be used to validate the feasibility of their client-server architecture. This paper presents empirical information and a case study of performance of a simple, OT-based collaborative code editor. Its limitations are the same as [14], since the architecture of the system is conventional and may not effectively deal with modern challenges related to decentralized or large-scale collaboration [15].

Tran et al. presented a survey for collaboration mechanisms in multi-agent LLM systems. Their approach was to survey and summarize the emerging literature on how multiple, independent AI agents (based on LLMs) might communicate, negotiate and coordinate to tackle complex problems. Their results divided various collaboration protocols into different categories, including simple message passing and shared memory, and complicated hierarchical or market-based coordination. This survey provides an important theoretical foundation for understanding the ”collaboration” part of AIdriven development to complement the applied papers such as [8] and [10]. Its main limitation is that its focus is only on AI-AI and not on how these agents integrate with the human collaborators in an RCCE [16].

Jatana et al. suggested an optimization method for a collaborative rich-text editor, ”differentially processed.” Their approach is likely to be a hybrid approach where only the differences (diffs) of operations or states are processed and sent (possibly combining principles of Operational Transformation with differential synchronization). The proposed solution aims to directly address the critical network payload size and client-side computational load performance issues which are especially acute in large documents with numerous active collaborators. This contribution is directed to the performance improvement at the algorithmic level. Limitations include the possible increase of complexity with the algorithm[17].

Virdi et al. provided a high-level review and position paper on the role of collaborative code editors in contemporary software engineering. Their approach seems to be a literature survey and descriptive analytics of the available tools and how they impact software development as well as knowledge sharing. Their results demonstrated that RCCEs are an invaluable platform for “knowledge sharing” rather than just coding tools. They support peer learning, real-time mentoring, and more efficient developer onboarding. In general, this piece of work focuses on the “human outcomes” and educational benefits of RCCE integration into both academic and professional settings. It is a general review article and, therefore, its limitation is the lack of novel technical contributions compared to a specialized journal article[18].

Navale et al. have formulated the Colab code strategy, a novel method for improving consistency and awareness in collaborative programming. The authors’ concept was the creation of a new strategy or protocol, Colab code, that would merge the current consistency maintenance algorithm with new user-facing awareness features.

While submitting Colab code as a new solution is speculative, the notion would establish a direct link between the concurrency control's metadata and the UI, enabling users to receive much smarter feedback about possible conflicts or the kind of simultaneous edits. The significance of the contribution is the attempt to formally connect the backend algorithm OT/CRDT with the frontend HCI awareness, an issue that is often presented as fully unrelated in modern solutions. The challenge, on the other hand, could be that the strategy might be more a conceptual idea than an actually realized and verified system[19].

Dhawan et al. have sketched out the "collaborative landscape" of contemporary software development, surrounding Real-Time Collaboration code editors . Their methodology was to study up-to-date tools, platforms, and emerging trends in this collaborative software development universe. First and foremost, their research yielded a history of how the tools developed – from asynchronous systems such as Git, to synchronous platforms like VS Live Share or Replit; second, it detected major technological and social triggers which made such tools extremely popular. The major value of this work is that it is very broad and is only recently published – its publication year is 2024. This allows using it to give the context to why some technical problems – notably CRDT performance and AI integration – are now so important. At the same time, this work is entirely a high-level "landscape" paper – it does not provide any specific new technical contribution[20] .

IV. ANALYSIS OF EXISTING SYSTEMS

Visual Studio Live Share (VSLS) is a mature, enterprisegrade solution that has high user satisfaction due to its strong awareness mechanisms, as analyzed by Tan et al. [Tan2024]. Its architecture is based on a centralized relay model which while making it simpler to manage sessions and provide deep IDE integration makes the system dependent on a single point of potential latency bottlenecks. Its concurrency control model is not as well documented in the public literature as open source alternatives, which makes it difficult to analyze its scalability limits in academic studies.

State-of-the-art is process consistency decentralization (CRDT-based systems) such as the 'Collabs' framework, Weidner et al. 2022 and 'Peritext' data structure Litt et al. 2022.

The key of their efficiency is that mathematical guarantees of convergence are given in the absence of centralized coordination, allowing occurrences of peer-to-peer architectures as well as robust offline-editing capabilities. However, these systems are not always free from the metadata overhead, which can affect the performance document with very large edit history, and a high density of collaborators. Furthermore, implementation of complex, domain specific operations, other than plain text, remains a non trivial design task.

Incorporation of these LLMs has emerged with new paradigms, like "self-collaboration" in the known LLMs, as Dong et al. proposed in their paper [Dong2024] and multiagent on LLMs as shown by Hu et al. in their paper [Hu2024]. These systems make the AI an active partner. Current implementations, however, are often simulated (AI-AI collaboration) and do not have the true real-time, low-latency text synchronization of RCCEs. The main problem is the seamless integration of generative AI operations (large, asynchronous, nondeterministic) with the fine-grained deterministic operations of the human programmer's edits.

A new challenge that is above and beyond the problem of synchronizing text is the one explored in Li et al. [2] and Quaranta et al. [11], which is called the issue of executable state consistency in collaborative notebooks. While text and output cells can be synchronized, making sure that all collaborators' underlying computational kernels and memory states are the same is a much more complicated problem. Existing solutions are not able to manage this state consistency well, and thus produce divergent, non-reproducible results. caracteristiques´ of the model are identified using context design based user studies and show that different types of users (Builder, Analyst and Consumer) have competing requirements that a one-sizefits-all monolithic interface cannot fulfill.

Recent architecture prototypes, such as CodeDive [3] and CodeR [15], can also be considered as good case studies for certain architectures. CodeR (CodeR2022) is a classic, centralized OT-based web application, which shows the effectiveness of this model for smaller scale deployments. CodeDive (Park et. al. 2025) investigates the informational fusion of realtime activity-monitoring for analytics via on the web and puts forth the vitality of RCCEs as developer-analytics systems. The main drawback of these systems is that they have not been proven to scale and be robust as compared to largescale industrial deployments because they are usually tested in controlled, small group settings.

The comparative analysis of these existing systems shows some important patterns. A pretty obvious dichotomy exists in the centralized and high-usability industrial (VS Live Share), versus decentralized and high resilience academic (CRDT) frameworks, regarding usability (deep IDE integration) and resilience (theoretical convergence and offline support). No system has yet been able to seamlessly integrate the low-latency and real-time nature of text synchronization with the high-latency and asynchronous nature of generative AI collaborators.

Furthermore, specialized domains such as computational notebooks present state synchronization issues with which text based algorithms (OT/CRDT) are inherently unequipped, and that there is a need for domain-specific consistency models. Taken together, these gaps provide the impetus for the research directions presented in the next section.

V. RESEARCH GAP

Based on the synthesis of the reviewed literature and system analysis, there still exist several significant research gaps in the field of real-time collaborative code editors. First, there is a basic gap in the integration of human and artificial intelligence collaborators. Existing AI collaboration models are either simulated or asynchronous, and do not solve the problem of combining high-latency, non-deterministic AI (such as largeblock code generation) and low-latency human granular typing in a common model of real-time consistency. New hybrid concurrency protocols are needed to deal with this impedance mismatch.

Second, the problem of state synchronization other than plain text has not really been addressed, especially for computational notebooks. While we can synchronize text and output cells, there is no good solution for general purpose management of the consistency of underlying executable state; for example, kernel, memory variables and environment dependencies. All these brings to non-reproducible states and divergent results between collaborators that is a critical issue for data science workflows. Domain-specific consistency models require research in these complex stateful environments.

Third, performance and scalability limitations still remain for both the dominant algorithmic paradigms. While decentralized CRDTs experience increases in performance-degrading metadata overhead for larger documents or sessions with more agents, centralized OT-based systems are inherently not scalable, and single points of failure arise from the nature of being a centralized system. There is a need for hybrid architectures and optimized data structures that adapt the tradeoff balance between centralized scalability and decentralized resiliency, or dynamic protocol switching or novel, bounded metadata CRDT is a excellent proposal.

Finally, the critically important issues of provenance and security in a collaborative context are underdeveloped. As human-AI teams become more of the norm, it becomes critical to establish fine-grained provenance tracking (i.e., robust and safe attribution of every transformation to its human or AI actor) for accountability, debugging and intellectual property management. In addition, the porting of entire development environments to the cloud has brought with it major security and data privacy issues that need new and strong mechanisms for access control, data encryption, and sandboxing of collaborative sessions.

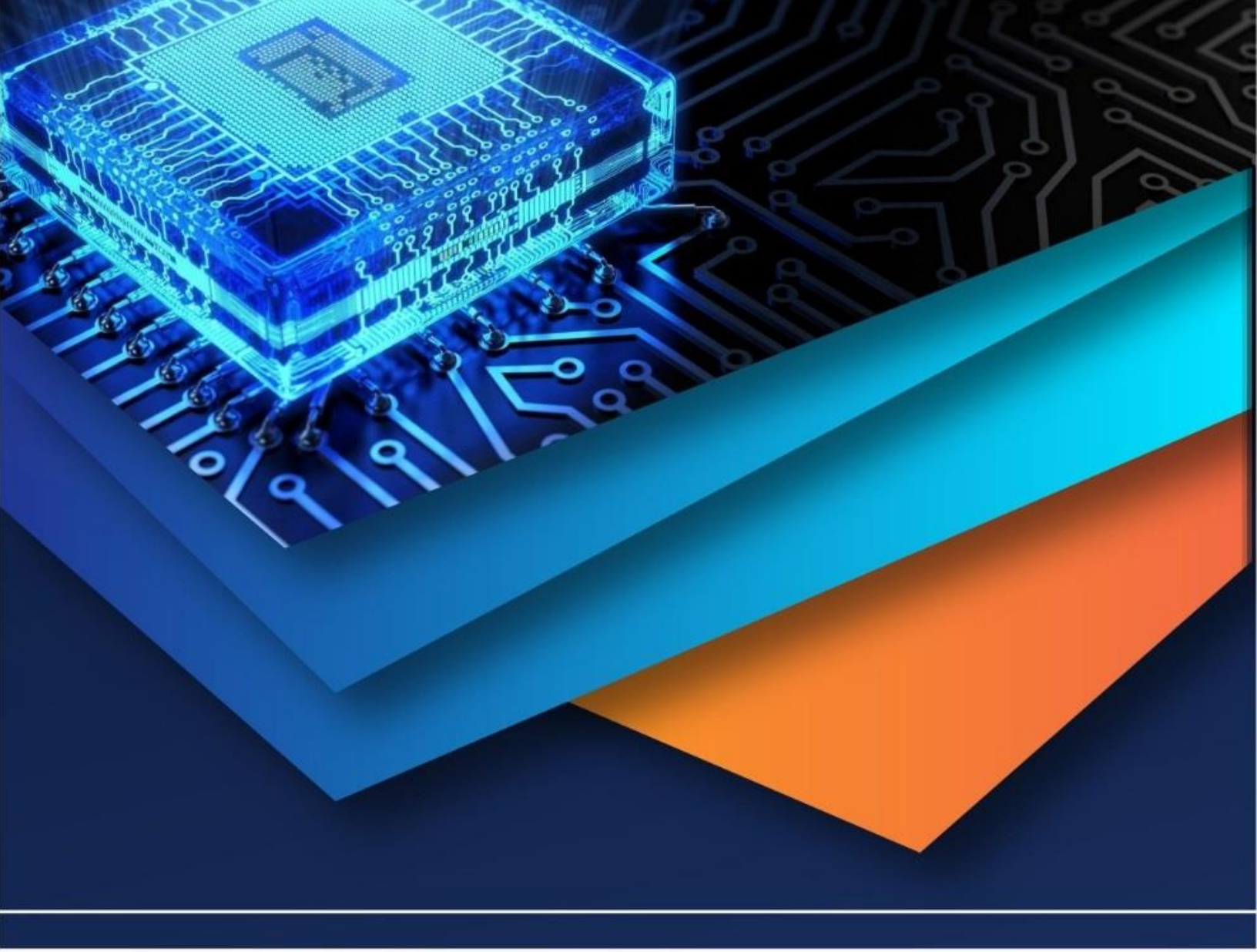
VI. CONCLUSION

In this survey paper, we reviewed twenty journal and preprint publications from 2021 to 2025. We provided a survey of the recent development in real-time collaborative code editors. Our analysis tracked that the field is in the process of changing. The hub of the debate has shifted from just basic algorithms – Operational Transformation vs. Conflict-free Replicated Data Types – to more complicated issues around users and domains. Particular trends consist of the dominance of human- system interaction and awareness mechanisms for user acceptance, the exceptional consistency challenges of stateful settings like computational notebooks, and the potential for intersection as a disruptive technology generating AI as an active teammate. Our survey of existing systems makes it clear that there is a stark division between robust, centralized industrial tools and antifragile, decentralized academic frameworks. The ensuing synthesis of the literature leads us to believe that four prominent research voids will define the future generation of such tools. First up is a dire call for hybrid concurrency schemes, which may allow syncing human and AI collaborators without skipping a beat – or waiting for each other. The second area is the need for strong, domain-oriented typing and execution boundaries in notebooks.

REFERENCES

[1] X. Tan, et al., “Understanding real-time collaborative programming,” ACM Transactions on Software Engineering and Methodology (TOSEM), vol. 33, no. 4, 2025.
[2] C. Li et al., “Understanding the needs of enterprise users in collaborative Python notebooks,” Proceedings of the ACM on Human-Computer Interaction (PACM HCI), 2025.
[3] H. Park et al., “CodeDive: A web-based IDE with real-time code activity tracking,” Applied Sciences (MDPI), vol. 15, no. 19, 2025.
[4] V. More et al., “PeerLink: A comprehensive collaborative platform,” International Journal of Scientific Research in Science, Engineering and Technology (IJSRSET), 2024.
[5] G. Litt et al., “Peritext: A CRDT for collaborative rich-text editing,” Proceedings of the ACM on Human-Computer Interaction (PACM HCI), 2024.
[6] M. Weidner et al., “Collabs: A flexible and performant CRDT collaboration framework,” arXiv preprint, 2024.
[7] I. David and E. Syriani, “Real-time collaborative multi-level modeling by CRDTs,” arXiv preprint, 2023.
[8] Y. Dong et al., “Self-collaboration code generation via ChatGPT,” ACM Transactions on Software Engineering and Methodology, vol. 33, no. 7, pp. 1–38, 2024.

[9] F. Wang et al., "A comprehensive survey of small language models in the era of large language models," ACM Transactions on Intelligent Systems and Technology, 2024.
[10] Y. Hu et al., "Self-evolving multi-agent collaboration networks for software development," arXiv preprint arXiv:2410.16946, 2024.
[11] L. Quaranta et al., "Eliciting best practices for collaboration with computational notebooks," arXiv preprint, 2023.
[12] A. Ayobi et al., "Computational notebooks as co-design tools," University of Bristol Repository, 2023.
[13] N. J. Santhi et al., "Collaborative code editor using web application," International Advanced Research Journal in Science, Engineering and Technology (IARJSET), vol. 11, no. 3, 2023.
[14] A. Kurniawan et al., "Real-time code editor application for collaborative programming," Elsevier/ScienceDirect, 2021.
[15] A. Kurniawan, C. Soesanto, J. E. C. Wijaya, "CodeR: Real-time Code Editor Application for Collaborative Programming," Procedia Computer Science, vol. 59, pp. 510–519, doi:10.1016/j.procs.2022.07.531, 2022.
[16] K.-T. Tran et al., "Multi-agent collaboration mechanisms: A survey of LLMs," arXiv preprint arXiv:2501.06322, 2021.
[17] N. Jatana et al., "Differentially processed optimized collaborative rich text editor," arXiv preprint, 2021.
[18] K. Virdi et al., "Collaborative code editors—Enabling real-time multiuser coding and knowledge sharing," ResearchGate, 2021.
[19] G. Navale et al., "Empowering collaborative programming: The Colab code strategy for consistency and awareness," International Journal of Intelligent Systems and Applications in Engineering, vol. 12, no. 21s, pp. 811–819, 2020.
[20] A. Dhawan et al., "Collaborative landscape of software development: RTC Code Editor," International Research Journal on Advanced Science Hub, vol. 6, no. 05, 2020.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY