"Don't Step on My Toes": Resolving Editing Conflicts in Real-Time Collaboration in Computational Notebooks

April Yi Wang
april.wang@inf.ethz.ch
ETH Zürich

Zihan Wu
ziwu@umich.edu
University of Michigan

Christopher Brooks
brooksch@umich.edu
University of Michigan

Steve Oney
soney@umich.edu
University of Michigan

ABSTRACT

Real-time collaborative editing in computational notebooks can improve the efficiency of teamwork for data scientists. However, working together through synchronous editing of notebooks introduces new challenges. Data scientists may inadvertently interfere with each others' work by altering the shared codebase and runtime state if they do not set up a social protocol for working together and monitoring their collaborators' progress. In this paper, we propose a real-time collaborative editing model for resolving conflict edits in computational notebooks that introduces three levels of edit protection to help collaborators avoid introducing errors to both the program source code and changes to the runtime state.

KEYWORDS

computational notebooks, data science, synchronous editing

ACM Reference Format:
April Yi Wang, Zihan Wu, Christopher Brooks, and Steve Oney. 2024. "Don't Step on My Toes": Resolving Editing Conflicts in Real-Time Collaboration in Computational Notebooks. In *2024 First IDE Workshop (IDE '24), April 20, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3643796.3648453

1 INTRODUCTION

"You work on *this* section, and I'll work on *that* one" is a familiar refrain for authors who work in teams. Working on different parts of the same document is a natural way to combine collaborators' work and avoid conflicts [15]. In data science programming, collaborators use a variety of collaborative strategies including "divide and conquer" (splitting work between team members) and "competitive authoring" (working on the same sub-problem simultaneously) [19]. However, Jupyter and other computational notebooks, which are often used by data scientists, introduce new challenges for collaboration. Although some version control tools (e.g., Git) work for computational notebooks, they mostly support the collaboration strategies for dividing work (e.g., working in independent files). Further, data scientists sometimes collaborate *synchronously*, with tools like JupyterLab [4], Google Colab [2], and Deepnote [1] that broadcast code and runtime updates to collaborators in real-time [19].

Synchronized collaborative computational notebooks allow data scientists to immediately share the notebook edits and the runtime state, which improves data science teamwork by creating a shared context, encouraging more explanation, reducing communication costs, and improving reproducibility [13, 19]. However, these synchronized notebooks also introduce many unique collaboration challenges [19]. For example, one collaborator might inadvertently change the runtime state and indirectly break another collaborator's code in a way that is difficult to debug [19].

Inspired by these challenges and opportunities, we propose a set of interactive techniques to minimize collaboration friction while maintaining the readability of the shared notebook. We instantiate these techniques in PADLOCK[1], an extension to the open source JupyterLab platform. PADLOCK provides three domain-relevant mechanisms to improve collaboration on computational narratives. *Cell-level access control*, allows collaborators to control who can view or edit cells to better support common collaboration patterns. *Variable-level access control*, extends this access control from code to runtime values to prevent implicit editing conflicts. *Parallel cell groups*, allows collabors' edits to be scoped to allow them to pursue exploratory solutions independent of collaborators. Our evaluation of PADLOCK has shown that these mechanisms can effectively prevent editing conflicts in shared notebooks and they support a wide range of collaborative workflows.

This work makes several contributions that advance the state of the art for collaborative data science tools: three new mechanisms (cell-level access controll, variable-level access control, and parallel cell groups) to improve collaborative data science work; a system (PADLOCK) that instantiates all these features in a JupyterLab plugin. To the best of our knowledge, PADLOCK is the first tool to:

- Give users the ability to specify access control constraints at the level of individual cells in computational notebooks
- Allow programmers to specify which collaborators (as opposed to which code) can access or overwrite variables
- Allow data scientists to work in "parallel cell groups", that are scoped in a way where they can access and reference each other's work without worrying about introducing conflicting code.

[1] An acronym: **P**arallelization **A**nd **D**ata **L**ocks **O**ffset **C**ollaboration **K**inks

IDE '24, April 20, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0580-9/24/04…$15.00
https://doi.org/10.1145/3643796.3648453

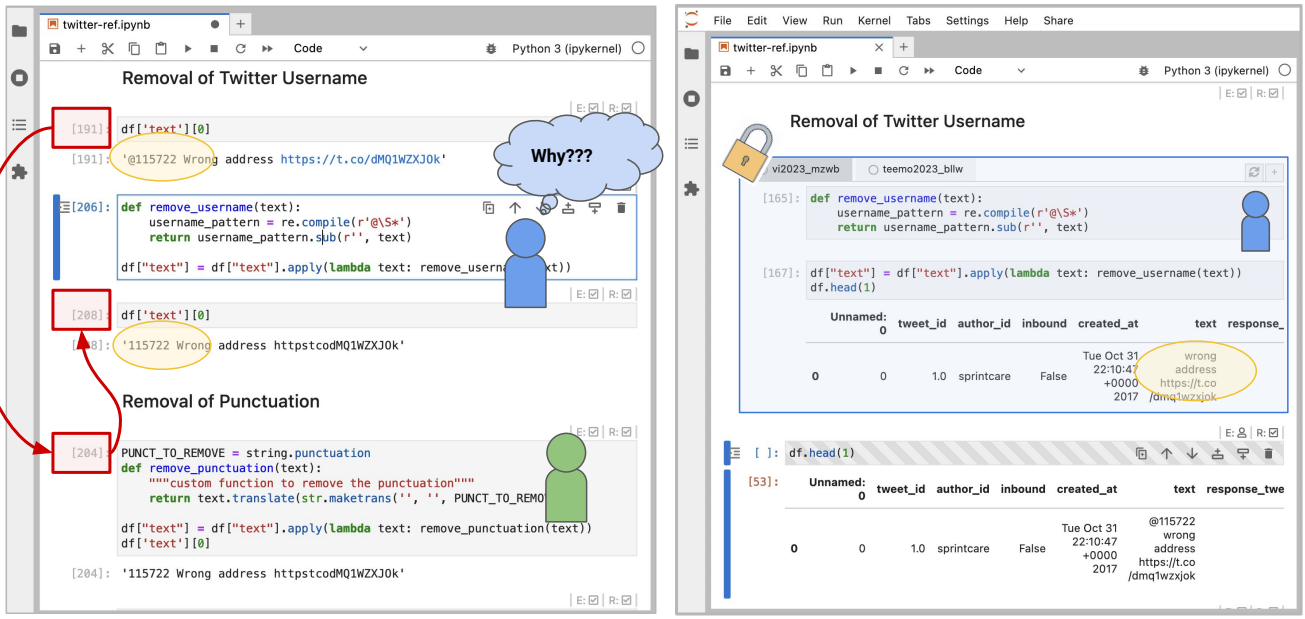

Figure 1: Editing conflicts in real-time collaborative notebooks can be implicit. As shown on the left, one can get an unexpected execution result because the collaborator accidentally changed the shared variable. As shown on the right, PADLOCK helps data scientists resolve editing conflicts in real-time collaborative editing in computational notebooks.

2 BACKGROUND

Data science programming can benefit from both synchronous and asynchronous collaboration. Zhang et al. [23] conducted a large-scale survey of data science workers and found that data science work is "extremely collaborative" and tools greatly influence their collaboration practices. Wang et al. [19] found that compared to individual programming contexts, real-time collaboration can encourage more exploration and provide a shared context for communication. They proposed four collaboration styles to characterize how data scientists work together, including *single authoring* where one collaborator does the majority of the work, *pair authoring* where one collaborator contributes to the implementation while the other collaborator participates in the discussion, *divide and conquer* where collaborators divide the task into subgoals and assign to each other, and *competitive authoring* where collaborators implement independently toward the same goal.

Researchers have proposed different systems to support programmers in working on the same code file synchronously. Targeting novice programmers, Warner and Guo created CodePilot [21], which is the first real-time collaborative programming tool that embeds coding, testing, bug reporting, and VCS features. Rädle et al. created Codestrates [16] to embed literate computing based on a shareable dynamic media system [12] and enables users to collaboratively work on authoring and debugging. There are also other tools provided in the form of IDE plugins [5, 6]. For example, Microsoft Live Share in VSCode [6] allows users to set the code to read-only for collaborators or enable server sharing for collaborating with the same variables. On a more fine-grained level, some researchers focused on resolving editing conflict of collaborative real-time editing in rich text with Conflict-Free Replicated Data Types (CRDTs) [14]; others examined collaboration in a broader context of peer assessment in programming classes for lightweight test cases [17]. Real-time collaboration in programming also brings unique challenges. Goldman [8] identified that syntax errors introduced by other collaborators might block one programmer's work. To address the issue, Goldman et al. proposed Collabode [9] which uses error-mediated integration that only integrate edits that do not cause compile errors.

For data science work, although there are many features that aims at improving awareness between collaborators and enhance communication, there has been limited features regarding preventing conflicts or interference in the collaboration process. Although the Deepnote [1] and Hex [3] computational notebook platforms provide some support for preventing conflicts—for example, both can prevent simultaneous editing of the same cell—they do not give collaborators fine-grained control over how this works, as PADLOCK does. For example, both tools still allow edits after the "cell owner" stops editing the cell, even if they were only pausing their activity and planning to resume shortly thereafter.

3 DESIGN MOTIVATIONS

We describe three examples conflict-causing real-time collaboration scenarios in data science, based on challenges identified in prior work [19].

Simultaneous Feature Implementation: Conflicts may arise during "competitive authoring" [19], where data scientists work on the same problems simultaneously. To avoid conflicts on interdependent code, collaborators must either be in close communication

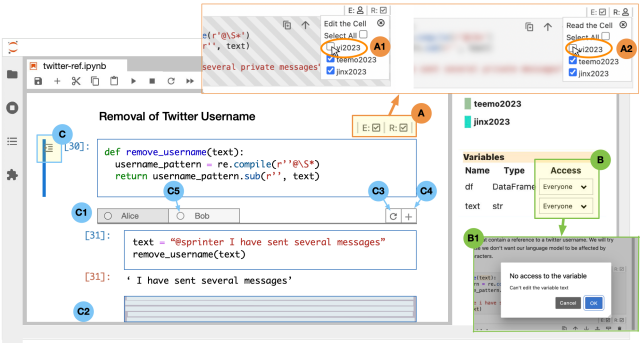

Figure 2: Overview of the three conflict-free mechanisms in PADLOCK. (A) Cell-level access control allows collaborators to claim ownership of the code cells and restrict others from editing or viewing them: (A1) Unchecking edit access for a user will change the cell background and disable editing; (A2) Unchecking read access for a user will blur the cell. (B) Variable-level access control extends the idea of access control from cells to shared variables: (B1) Unchecking a user’s variable access; (B2) After losing access, they will not be able to edit the variable and will receive warning when attempting to do so. (C) Parallel cell groups define a designated area where changes of the code and runtime state stay inside its own scope: (C1) One parallel cell group can contain multiple tabs; (C2) Each tab can contain multiple cells; (C3) Sync the variables from the global scope to the current active tab; (C4) Add a new tab; (C5) Click the radio button to mark it as the “main” tab.

or edit their own copies of the shared code and coordinate an eventual merge. This can be prohibitively difficult, particularly in large teams.

Concurrent Variable Use or Modification: Even if they are not working on the same features, collaborators often need to work on the same shared data. For example, when collaborators work on a shared dataframe, it can be easy to accidentally make edits that conflict with their collaborators’ work. However, there are still also occasions where data scientists need to synchronize changes made by their collaborators working upstream. Coordinating work on these shared variables can be difficult and error-prone.

Social Concerns in Real-Time Collaboration: Prior work has found that authors have social concerns about letting collaborators see their intermediate work [20]. For example, they might fear being judged about the quality of their intermediate code and novices may be self-conscious about their visible progress. In these scenarios, collaborators may want to take steps to gain some degree of privacy as they work. Normally, this must be done by working in a non-shared document and merging their work when they deem that it is ‘ready’ to be integrated [20]. However, this strategy can be difficult to implement in many scenarios where collaborators’ work is interdependent, as is frequently the case [19].

4 SYSTEM OVERVIEW

Our system uses three complementary techniques for conflict-free protection: cell-level access control, variable-level access control, and parallel cell groups.

4.1 Cell-Level Access Control

Computational notebooks consist of *cells*. Each cell typically represents a conceptual unit within the larger notebook. For example, a notebook might consist of one cell to fetch data from a remote API, another to clean those data, and other cells for various transformations and visualizations of the data. In PADLOCK, we leverage the structure of cells in order to allow collaborators to claim ownership of parts of larger collaborative notebooks. This helps address of the challenges of synchronous editing in traditional text-based programming tools where there are no clear “dividing lines” between different parts of the shared codebase and unclear how to localize the scope and effects of collaborators’ changes.

Specifically, PADLOCK enables *cell-level access control* where users can prevent collaborators from viewing or editing a collection of cells. As Figure 2.A shows, users can select a code cell and specify who can read or edit the code. As prior studies have found, there are many collaboration styles [19], and cell-level access control benefits multiple collaboration styles. In a “single authoring” style [19]—where one collaborator contributes the majority of ideas and code—setting cells to be only editable by the main contributor can prevent others from accidentally introducing errors. In a “divide and conquer” style [19]—where collaborators split up work—restricting view access might ease feelings of self-consciousness that authors sometimes feel when collaborators can see their writing in real-time (which might otherwise lead them to work in a private editor and then copy its contents to the main notebook, as prior work found in collaborative writing [20]).

When an author is restricted from editing a cell, the background of the cell (grey striping) indicates that edit access is not permitted. When an author is restricted from reading a cell, the content of the cell is blurred but activity (and thus awareness of contributors’ location in the narrative) is supported. Thus, the view control of the cell can allow them to focus on early explorations of ideas while still letting others know what they are working on.

4.2 Variable-Level Access Control

Restricting cell access gives collaborators control of code edits but it does not prevent collaborators from modifying the shared runtime state. For example, a user might create a cell that defines `df` as a data frame (data in a table-like structure) and restrict write access to prevent other collaborators from editing the cell that declares `df`. However, collaborators could still create a new cell that either re-declares or mutates the value of `df` and breaks downstream code that reference it.

Thus, PADLOCK also introduces *variable-level access control*. Variable-level access control extends the idea of access control from cells to shared variables—authors can determine if collaborators’ code can view or modify the values of runtime variables. PADLOCK tracks the runtime state of the notebook kernel and extracts the variable information. Users can specify the access control of every variable in a side panel (Figure 2.B). On the other collaborators’ side, the protected variable is highlighted throughout the notebook. When an individual attempts to execute a code cell a static analysis on the abstract syntax tree (AST) of the program is done to determine whether the execution would impact the value of protected variables and, if so, the execution is halted with an error.

Variable-level access control is especially beneficial for scenarios where there is a lead collaborator in charge of managing important data tables. Setting variable-level access control can encourage collaborators to either make a copy or use parallel cell groups before they do any risky explorations.

4.3 Parallel Cell Groups

Data science work is often exploratory. Authors might write code to explore an idea or approach. In the context of teams, multiple team members might simultaneously work through different approaches for the same problem [19]. In these situations, authors might want to write code that manipulates *their own version of* some subset of variables in the notebook.

PADLOCK thus also introduces *parallel cell groups* (which we will call "parallel cells"). Parallel cells define a designated area where changes of the code and runtime state stay inside its own scope. As Figure 2.C shows, users can split a regular code cell into parallel cell groups. Collaborators can create new cell groups to branch off and explore alternatives; add multiple cells to a cell group to write larger and more complex alternative code; and work individually in each cell group. The parallel cell groups are folded together into the same area in the notebook, helpings collaborators to maintain an overall coherent structure of the narrative. In addition, when collaborators are settled on a solution, they can mark a cell group as "primary", which merges the execution result into the main runtime state. Note that the parallel cell groups designed in PADLOCK are different notions than the forked cells in [22] in several ways. In terms of the usage scenario, [22] is designed for a single developer to explore alternative ideas, whereas PADLOCK is designed for synchronous computational notebooks. For the implementation, PADLOCK uses a scoping mechanism instead of spawning multiple kernels, making it easier for managing different versions of the same variable.

A key difference from prior tools for branching and managing local versions [10] is that each cell group has its own execution scope—changing the variables in one cell group would not affect others. For example, suppose there is a parallel cell group is named `plel`, and within those cells, code creates variables named `x` and `y`. Inside of the cell group, `x` and `y` can be referenced as usual. Outside of the cell group, code that references variables `x` and `y` get their 'old' values—whatever value they were assigned to outside of the cell group. However, these variables can be referenced outside of the group if the user explicitly specifies which scope they want to reference. So while `x` and `y` are not affected outside of `plel`, collaborators can refer to `_plel.x` and `_plel.y` to access the values that were set inside of `plel`.

Parallel cell groups allow collaborators to flexibly split the notebook for exploring alternatives. It is particularly designed for the "competitive authoring" collaboration style [19] where team members competitively write code for the same purpose and reach consensus when an acceptable solution is found. This allows collaborators to work independently while making concurrent edits and executions, preventing costly mismatches between programmers' mental state and the actual state of the runtime. It also provides collaborators with the shared context so they do not work too "far" away from one another, thus supporting awareness of the others' actions (e.g., working on an individual notebook for exploration). Finally, this feature preserves the structure of the narratives by grouping and folding parallel alternatives together.

In PADLOCK, parallel cell groups are represented as indented cells. Conceptually, this matches the semantic meaning of indentation in Python (specifying the bounds of a code block and potentially creating a new scope).

5 EVALUATION

We conducted a laboratory study on handling a scenario that involves editing conflicts with a paired collaborator. This conflict editing scenario is synthesized from literature [19] and reproduced by pairing participants with a member of the research team who plays the role of a "clumsy collaborator".

5.1 Study Procedure

The clumsy collaborator communicated with the participant using a simple chat interface we developed. Participants were asked to collaborate with another individual (the clumsy collaborator). They were informed that not all the study procedures would be explained until the end of the study, and we did not reveal the clumsy collaborator being a member of the research team. After a brief demonstration of the RTC feature, we asked the participants to use the chat to greet the clumsy collaborator, who introduced themselves as a data science student who knew Python and regex, but was not experienced in Pandas. Next, we explained the task. The task was adapted from a Kaggle challenge to preprocess customer support Twitter contents and has three sub-goals: lower casing (T1), removing Twitter usernames (T2), and removing URLs (T3). In the notebook, we also inserted sections on removing punctuation and removing frequent words after T3, with code already implemented.

We then asked the participant to work with the clumsy collaborator, who followed a script to create conflict editing scenarios, and provide hints when the participant was stuck for a certain period of time on each sub-goals. For participants to get familiar with the RTC feature and the clumsy collaborator, we first asked the participant to work with the clumsy collaborator to solve T1, where the clumsy collaborator will not disturb the participant's work.

Next, we asked the participant to solve T2 without the conflict editing feature. The clumsy collaborator would propose to work in separate cells below T3, and "accidentally" execute a script that changes the column the participant is working on to disturb their work. We would observe how participants reacted to the unexpected execution results.

Following a demo of PADLOCK, we asked the participant to solve T3 with the conflict editing feature. The clumsy collaborator would follow the participants' suggestion to use the notebook, and "accidentally" execute the script to change the dataset again.

5.2 Results

5.2.1 Conflict editing is hard to notice and prevent. After the second task, most participants (13/14) were not able to correctly find out what caused the code cell not to return the expected results until we explained it to them. This aligned with our observations that many participants (12/14) switched their browsers to search for API documentation and did not stay on the shared notebooks all the time. Moreover, several participants (P5, P10) did not even

notice that the output was wrong. There was an exceptional case where P9 ran the data loading cell right before executing the cell for removing the twitter username, leaving no chance for the clumsy collaborator to modify the shared variable. P9 explained that they prefer to reload data every time before executing a new cell unless the data frame is very large.

Interestingly, although several participants (4/14) were able to recover from the issue by reloading the dataframe, they still did not identify the source of the problem. Most participants (12/14) did not doubt their collaborators' actions or question what they did. Instead, they blamed themselves and looked into their own code to debug. For example, P3 said:

> I felt like I had the correct code. But I assumed something was wrong with it. I just didn't even think that it could have been the collaborator's code [that causes the issue].

5.2.2 Perceptions of PADLOCK for preventing conflict editing. In T3, participants used PADLOCK to work with the clumsy collaborator. All participants (14/14) chose to create parallel cell groups and suggested the clumsy collaborator to write their code in a parallel cell. After they finished the task, some participants (8/14) cleaned up the notebook by unindenting the parallel cell groups. Several participants (2/14) chose to keep the clumsy collaborator's parallel cell and merge their solution into the notebook by marking their solution as main.

Overall, participants reported that they felt confident about not messing up with the shared notebook. In particular, participants mentioned that the parallel cell groups made the shared notebook "neat" (P4), "organized" (P11), and "structured" (P6). Although participants did not use the cell-level access control and variable-level access control, they described scenarios where these features could be useful. P10 mentioned that both features could be helpful in the large classroom setting where an instructor has a sample notebook. P5 said that variable-level access control can be useful when the cost of restarting the kernel and running previous code cells is expensive. He described the scenario where a data science manager would not want interns to accidentally modify large-scale data tables and had to restart the kernel to recover the results. In addition, P10 mentioned that she would use the cell-level access control on finished code cells, and use parallel cell groups on work-in-progress cells. Noticeably, several participants mentioned that read access in cell access control was not necessary for themselves, but they could see it being used by other people.

5.2.3 Improvement of the Parallel Cell. Participants had several ideas on how to improve the parallel cell feature in PADLOCK. Several participants (P6, P9) mentioned adding notifications or activity histories to track if others have unindented a code cell:

> When you merge the selected tab with the main thread, that's like a commit to the main repository in GitHub. So then, you know, you need to also tell others that I have launched this, maybe a notification. I was hoping there would be some way to track that, like GitHub provides a history of commits that somebody has made to other changes.

In addition, P2 asked for a merging process where she could pull cells from various fragments:

> I wish there is an option to maybe merge different parts of the cell, like maybe one collaborator has one cell, and then you merge the second part of another collaborator.

5.2.4 Resonate with prior experience. The instance of editing conflict in task 2 resonated with participants' prior experience with real-time collaborative editing. P1 mentioned a different collaborative setting in a data science classroom. The data science classroom had around 100 students and the instructor asked everyone to join the same notebook in Deepnote. However, the instructor asked students to not directly run code cells in the notebook. Instead, students typed out solutions and commented at the same time. Several participants (P9, P13) mentioned that their prior experience with shared notebooks was mostly asynchronous collaborating.

6 DISCUSSION AND FUTURE WORK

Our work suggests exciting opportunities for supporting collaborative editing at scale. For example, instructors can share a collaborative notebook with a classroom of students; researchers can share a collaborative notebook with a broader audience for open collaboration; data science hobbyists can make their live streaming session more engaged by sharing the collaborative notebook session. Future work can use mechanisms like searching, tagging, and filtering for scaling and managing many parallel cells. Our current design of cross-referencing allows participants to computationally compare versions of variables from different parallel cells, which could be improved by integrating visualization techniques to compare data changes [18], or clustering techniques to explore variance [7]. We are also interested in incorporating domain-specific features, such as allowing students to test their code cells with peer-written test cases [17].

Another key area for future work is in improving users' awareness of collaborators' activity. For example, parallel cell groups allow multiple users to work on different versions of the same document concurrently, but makes it difficult for users to see each other's cursor movements or edits. In addition, the design of PADLOCK brings up the unique challenges in helping collaborators track and forage editing history. With non-linear notebook structures, it is worth exploring how notebook history foraging designs [11] can be extended to support the awareness of complex cell editing.

7 CONCLUSION

Real-time collaborative editing in computational notebooks requires strategic coordination between collaborators. We investigated common obstacles in real-time notebook editing and proposed a set of access control mechanisms to support conflict-free editing: *cell-level access control* (which restricts collaborators' ability to see or edit cells), *variable-level access control* (which protects runtime variables from being referenced or modified), and *parallel cell groups* (which allow collaborators to work in their own space while staying connected to the larger notebook). As we found in our user studies with PADLOCK, these features can improve collaboration within data science teams.

REFERENCES

[1] 2022. Deepnote. https://deepnote.com/
[2] 2022. Google Colab. https://colab.research.google.com/
[3] 2022. Hex - Collaborative Notebooks - A New Way to Notebook. https://www.hex.tech/
[4] 2022. JupyterLab. https://jupyter.org/
[5] 2022. Saros. https://www.saros-project.org/
[6] 2022. Use Microsoft Live Share to collaborate with Visual Studio Code. https://code.visualstudio.com/learn/collaboration/live-share
[7] Elena L Glassman, Jeremy Scott, Rishabh Singh, Philip J Guo, and Robert C Miller. 2015. OverCode: Visualizing variation in student solutions to programming problems at scale. *ACM Transactions on Computer-Human Interaction (TOCHI)* 22, 2 (2015), 1–35.
[8] Max Goldman et al. 2012. *Software development with real-time collaborative editing*. Ph. D. Dissertation. Massachusetts Institute of Technology.
[9] Max Goldman, Greg Little, and Robert C Miller. 2011. Real-time collaborative coding in a web IDE. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*. 155–164.
[10] Mary Beth Kery, Amber Horvath, and Brad A Myers. 2017. Variolite: Supporting Exploratory Programming by Data Scientists.. In *CHI*, Vol. 10. 3025453–3025626.
[11] Mary Beth Kery, Bonnie E John, Patrick O'Flaherty, Amber Horvath, and Brad A Myers. 2019. Towards effective foraging by data scientists to find past analysis choices. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–13.
[12] Clemens N Klokmose, James R Eagan, Siemen Baader, Wendy Mackay, and Michel Beaudouin-Lafon. 2015. Webstrates: shareable dynamic media. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*. 280–290.
[13] Sean Kross and Philip J Guo. 2019. Practitioners teaching data science in industry and academia: Expectations, workflows, and challenges. In *Proceedings of the 2019 CHI conference on human factors in computing systems*. 1–14.
[14] Geoffrey Litt, Sarah Lim, Martin Kleppmann, and Peter van Hardenberg. 2022. Peritext: A CRDT for Collaborative Rich Text Editing. *Proceedings of the ACM on Human-Computer Interaction* 6, CSCW2 (2022), 1–36.
[15] Ilona R Posner and Ronald M Baecker. 1992. How people write together (groupware). In *Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences*, Vol. 4. IEEE, 127–138.
[16] Roman Rädle, Midas Nouwens, Kristian Antonsen, James R. Eagan, and Clemens N. Klokmose. 2017. Codestrates: Literate Computing with Webstrates. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology* (Québec City, QC, Canada) *(UIST '17)*. Association for Computing Machinery, New York, NY, USA, 715–725. https://doi.org/10.1145/3126594.3126642
[17] April Yi Wang, Yan Chen, John Joon Young Chung, Christopher Brooks, and Steve Oney. 2021. PuzzleMe: Leveraging Peer Assessment for In-Class Programming Exercises. *Proc. ACM Hum.-Comput. Interact.* 5, CSCW2, Article 415 (oct 2021), 24 pages. https://doi.org/10.1145/3479559
[18] April Yi Wang, Will Epperson, Robert A DeLine, and Steven M Drucker. 2022. Diff in the Loop: Supporting Data Comparison in Exploratory Data Analysis. In *CHI Conference on Human Factors in Computing Systems*. 1–10.
[19] April Yi Wang, Anant Mittal, Christopher Brooks, and Steve Oney. 2019. How data scientists use computational notebooks for real-time collaboration. *Proceedings of the ACM on Human-Computer Interaction* 3, CSCW (2019), 1–30.
[20] Dakuo Wang, Haodan Tan, and Tun Lu. 2017. Why users do not want to write together when they are writing together: Users' rationales for today's collaborative writing practices. *Proceedings of the ACM on Human-Computer Interaction* 1, CSCW (2017), 1–18.
[21] Jeremy Warner and Philip J Guo. 2017. Codepilot: Scaffolding end-to-end collaborative software development for novice programmers. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. 1136–1141.
[22] Nathaniel Weinman, Steven M Drucker, Titus Barik, and Robert DeLine. 2021. Fork It: Supporting stateful alternatives in computational notebooks. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–12.
[23] Amy X. Zhang, Michael Muller, and Dakuo Wang. 2020. How Do Data Science Workers Collaborate? Roles, Workflows, and Tools. *Proc. ACM Hum.-Comput. Interact.* 4, CSCW1, Article 22 (may 2020), 23 pages. https://doi.org/10.1145/3392826