Callisto: Capturing the "Why" by Connecting Conversations with Computational Narratives

April Yi Wang[†], **Zihan Wu**[‡], **Christopher Brooks**[†], **Steve Oney**[†]
[†]University of Michigan School of Information, [‡]Tsinghua University
{aprilww, ziwu, brooksch, soney}@umich.edu

ABSTRACT
When teams of data scientists collaborate on computational notebooks, their discussions often contain valuable insight into their design decisions. These discussions not only explain analysis in the current notebook but also alternative paths, which are often poorly documented. However, these discussions are disconnected from the notebooks for which they could provide valuable context. We propose Callisto, an extension to computational notebooks that captures and stores contextual links between discussion messages and notebook elements with minimal effort from users. Callisto allows notebook readers to better understand the current notebook content and the overall problem-solving process that led to it, by making it possible to browse the discussions and code history relevant to any part of the notebook. This is particularly helpful for onboarding new notebook collaborators to avoid misinterpretations and duplicated work, as we found in a two-stage evaluation with 32 data science students.

Author Keywords
Computational Notebooks; Collaborative Systems; Data Science; Literate Programming

CCS Concepts
•**Human-centered computing** → **User interface programming; Synchronous editors;**

INTRODUCTION
Data scientists benefit from collaborations to leverage expertise from each other and to improve the efficiency of their work. Computational notebooks are powerful tools for collaborative data science because they allow data scientists to document and replicate the exploration process through the creation of computational narratives—documents that combine code, explanatory text, and intermediate output. New tools like Google Colab [4] and Deepnote [3] enable data science teams to work in the same notebook in real time, creating new possibilities for collaboration.

CHI'20, April 25–30, 2020, Honolulu, HI, USA
 ISBN 978-1-4503-6708-0/20/04…$15.00
DOI: 10.1145/3313831.3376740

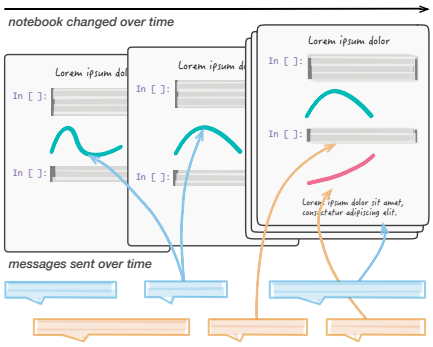

Figure 1. Callisto captures and stores contextual links between discussion messages and notebook elements with minimal effort from users.

Effective communication between data science team members is critical for productive teamwork. Collaborators need to understand what their teammates have done so far, what they plan to do, what they have given up, and how their work fits in with the team's overall goals. Team members can improve their shared understanding by *(a)* writing clearer code, *(b)* documenting their work, and *(c)* discussing as a team. Improving code clarity (a) and writing clear documentation (b) are often impractical for data scientists, who frequently write makeshift code to experiment, explore, and test hypotheses [42, 27, 22, 39, 40].

Data science teams often have rich team discussions (c) through communication channels such as e-mail, instant messenger, and face-to-face meetings [42, 27]. These discussions are often crucial for collaborators to work together effectively, as they can provide valuable context about notebook authors' goals and design rationales. However, these discussions are disconnected from the computational notebooks being discussed—they typically occur in channels outside of the notebook and references to notebook content are implicit (e.g., "there's a bug in the second cell" or using a notebook screenshot). This means that team members typically need to have a shared context to make sense of the discussion (i.e., they need to understand what "the second cell" means or which part of the notebook a screenshot refers to). This can be particularly challenging as the notebook evolves (for example, if the "second cell" is moved after it is referred to). As a result, although discussions are helpful for collaborators who are actively involved in it, they can be difficult to understand for new team members or anyone catching up on the discussion [44] who does not have this shared context.

In this paper, we propose to improve collaborative data science by connecting discussions with computational notebooks.

We first describe the results of a formative study where we found that chat messages can be invaluable to understanding collaborative computational notebooks but are difficult to comprehend afterwards. We then introduce *Callisto*, a plugin for Jupyter [19]. We designed Callisto with the insight that *discussions are an integral part of collaborative computational notebooks* and connecting discussions with notebook content can make the notebook easier to understand for its authors and for subsequent readers. Callisto augments Jupyter with several collaborative features—most notably, the ability to explicitly reference notebook elements in chat messages. These connections make it easier to understand the context of a given message and to find discussions that are relevant to a specific notebook element.

We conducted a two-stage evaluation of Callisto. In the first stage, we evaluated how Callisto supports real-time team communication and found that it can reduce communication costs. In the second stage, we evaluated whether Callisto can help data scientists better understand a discussion that has already taken place. We found that Callisto can ease user onboarding to new notebooks by helping them understand the design rationales of its authors. As one of our participants put it, *"by reading the code, I know what they were doing. But with the chat messages, I can know what they were thinking."*

This paper contributes:

- empirical evidence of the challenges that data scientists encounter when catching up with an ongoing group project,
- the design of Callisto with a set of features to make chat messages more useful for understanding the past exploration process in the notebook,
- empirical insights into how users engage with and perceive these features, and
- evidence that creating mappings between messages, notebook elements, and versions helps data scientists understand and follow up on the exploration pipeline.

RELATED WORK

Computational Notebook

Computational notebooks are widely used to create and share the exploration process for data science [36]. As a practice of *exploratory programming*, data science often involves writing code to actively experiment [22]. The design of computational notebooks makes it easy for data scientists to rapidly iterate on code chunks and inspect the intermediate results during the exploration process. Data scientists also benefit from combining exploratory code with human-readable explanations to create *computational narratives*—a practice of *literate programming* [26], and a medium for data science practitioners [23, 27, 32, 35, 24], scientific programmers [15, 37], and data science educators [29] to share and reproduce work with a low cost for setting up the environment. Recent groupware tools like Google Colab [4] and Deepnote [3] enable synchronous editing for data scientists to collaboratively author computational notebooks, which encourages more exploration and discussions over the shared context [42].

Although having a well-documented computational narrative offers many benefits, it is challenging for data scientists to maintain an updated explanation and a clean notebook during the exploration process [39]. When the problem gets complex, data scientists tend to write lower quality code, leave documentation incomplete, change the execution order, or accidentally overwrite important analyses while iterating on different ideas [21, 18]. These tensions can be amplified in a collaborative setting where it is important to keep a shared understanding of past design decisions across team members [42, 27, 22].

Prior studies have explored ways to reduce the messiness of computational notebooks. These include introducing local versioning to better document past analysis [21], using code-gathering techniques to curate and reorganize code chunks [18], and folding code chunks with annotations [38]. These tools are useful for improving the structure of the notebook, but they do not directly help with documenting design decisions; it remains a challenge to help explain the analysis process. The interpretation and reasoning process of the intermediate results plays an important role in exploration. By looking at the same output, data scientists may come up with different understandings, which lead to different actions they may take next. In a collaborative setting, data scientists may externalize such explanations through the discussion with each other. Callisto aims to curate these valuable clues to help notebook readers better understand the exploration process.

Linking Code and Discussion

Callisto extends the idea of *post-literate programming*, where discussions are gathered and incorporated into the code [34]. For instance, Github allows users to create an "issue" to start a discussion and link the issue to a commit or pull request (submission of code change) [2]; Clerkbot allows software developers to mark and summarize their chat messages through a chatbot and link it to their code repository [34]. It is difficult to directly apply the idea to computational notebooks. First, data scientists rarely use conventional version control tools like Git because they are not convenient for versioning and tracking rapid explorations in computational notebooks [20]. Second, it relies fully on the users to initiate the linking, which may hinder exploration and communication in collaborative data science. Data scientists need a lighter way to link the related discussion to the notebook content.

On the other hand, there are many communication tools that tie the code content in the discussion to build common ground [13, 14] in collaborative programming tasks such as tutoring, help-seeking, and pair programming. For instance, sharing gaze information among programmers can help communicate locations in code [12]; online Integrated Development Environments (IDEs) like JSFiddle [5] and Stack Snippets [17] allow users to create a minimal, complete, and verifiable example to directly illustrate the problem on forum posts; Codeon [8] enables remote helpers to point out locations in code using annotations; and chat.codes [33] introduces code pointers that create deictic references to regions of code.

Creating Contextual References

Creating contextual references to reduce communication costs has been explored in a variety of other contexts [45, 10, 11, 16, 9]. Contextual references can be used to point to a specific

place in a shared context. For example, NB [45] supports connecting social conversations to a specific part of a document's content and is particularly useful in the educational context when students want to start a discussion on lecture notes or textbooks. Contextual references can also be used to support the notion of time when the shared context is dynamic. For example, ConceptScape [30] connects nodes in a learner-sourced concept map with timestamps in the educational video; and chat.codes [33] connects both the code location and the version history with chat logs.

Inspired by chat.codes [33], Callisto connects chat messages with notebook content and tracks the timestamp, version history, and location of elements that are discussed. To capture the location of elements, Callisto extends the design of code pointers as one data source to collect deictic references between notebook and discussions. This approach captures accurate contextual links in a convenient way, but still requires explicit effort from users. Unlike chat.codes, Callisto also collects users' implicit interactions (e.g., cursor position) in the notebook to automatically create contextual links between code and discussion when the messages do not contain explicit pointers. Callisto enables two-way interaction between messages and notebook content, whereas chat.codes only supports navigating from messages to code.

FORMATIVE STUDY

To better understand how discussions can be useful for explaining the data-exploration process, we analyzed chat messages collected from three data science group projects. In doing so, we aimed to investigate three questions: (1) Why do collaborators send messages to one another? (2) How do messages connect with the evolving notebook? and (3) What aspects of the notebook do collaborators talk about?

Method

We recruited six data science students from data science special interest groups in both university and online learning environments. We asked participants to work remotely in pairs on a beginner-level data science task* using a collaborative Jupyter editor for four hours. This collaborative Jupyter editor synchronizes edits between users and allows collaborators to see each other's cursors. Pairs also worked in a shared runtime, meaning that code ran on a single interpreter, with outputs shared between collaborators. There were no other explicit communication mechanisms (chat, voice, etc.) enabled in the editor, and participants were given access to a third-party text chat (Slack) for communication. We collected chat messages, final notebooks, and screen recordings during the study.

Data Analysis

Our formative study uses a similar data analysis approach (in a different setting) to Yarman et al.'s work on cross-media referencing [43]. Two members of the research team used open coding to classify the collected data. We used the first 50 messages to create an initial code list, using final notebooks and video recordings as secondary evidence to help recall messages' context. After discussing and merging the code list, the two members independently coded the same sample of 50 messages and achieved an agreement of $\kappa = 0.40$. We revised codes to reduce ambiguity and achieved an agreement of $\kappa = 0.83$ between raters after two rounds of iteration.

*https://kaggle.com/c/house-prices-advanced-regression-techniques

Results

In total, we analyzed 760 chat messages to better understand their purpose, their relationship to the evolving notebook, and the specific aspects of the notebook they mention.

Purpose

We found five broad purpose categories: reflection (244), planning (87), check-in (121), cooperation (67), and out-of-scope messages (244), as Table 1 shows. Messages could fit into multiple categories, such as planning and cooperation. Messages coded as *reflection* tended to expand on the reasoning behind past decisions, while *planning* messages discussed potential features that had not yet been implemented. *Check-in* messages were updates between collaborators on what they had done, while *cooperation* messages generally discussed collaboration strategies.

These four categories show that chat messages can help explain the data-exploration process, describe the purpose of the code, and provide a high-level interpretation of the results. Not surprisingly, however, we categorized an additional 30% of messages as *out-of-scope*, because they did not convey useful information for explaining the exploration process and instead contained duplicate information or socialization messages.

Relevance

Our analysis (shown in Table 2) revealed that the final notebook is generally not reflective of the whole exploration process. Data scientists often explored ideas in discussions that they later rejected and wrote no analysis for. Even if they did implement an analysis to explore an idea, the code was often modified or removed during a "cleanup" stage [23, 18]. As such, messages between collaborators can fill in missing details of the exploration process. We suggest that revealing this information to new collaborators (e.g. someone taking over a project or another data scientist helping with an analysis) may avoid duplication of work while simultaneously revealing hidden assumptions. However, given that the Jupyter Notebook's current design does not have built-in collaboration features nor track edit histories (which can give context to discussions), it is difficult to use chat messages to understand a previous exploration process.

Granularity

We also investigated the granularity of the notebook elements collaborators referred to (Table 3), finding that 97 messages directly referred to a specific line of code. These messages were related to Application Programming Interface (API) usage, debugging, or sharing the current status. A further 119 messages directly related to the output of a cell, including the data frame, visualizations, and statistical values. Finally, 206 messages described high-level ideas implemented across one or multiple cells.

Implications

We derive three design implications from our findings:

Purpose	Example	*n*
Reflecting	"This plot confirms the correlation for sure."	244
Planning	"Let's throw away columns that have lots of missing values."	87
Check-in	"Just did a square root."	121
Cooperation	"Ok, while you fix the stuff, I'll create one hot encoding for categorical [variables]."	67
Out-of-scope	"Oh no!!"	244

Table 1. Purpose of sending a message: reflecting, planning, check-in, cooperation, and out-of-scope.

Relevance	Example	*n*
Ideas that were only discussed but never implemented	"Do you think we can just assign 1,2,3,4,5 to it and create one column instead?" ("No, I am afraid ...")	29
Ideas that had not yet been implemented when the message was sent, but appeared in the notebook later	"How about we start with numerical columns?"	150
Ideas that had been implemented in the notebook when the message was sent, but did not appear in the final notebook	"Something went off. The MSE [mean square error] is huge."	72
Ideas that had been implemented when the message was sent and appeared in the final notebook	"For the test data I did a `fillna` with 0."	108

Table 2. Relevance between messages, the notebook history, and the final notebook.

Granularity	Example	*n*
Directly referred to a specific line of code	"I think `LabelEconder` is going to treat NA as a new encoding."	97
Directly referred to the output of a cell	"I am not too convinced if our MSE values are good enough."	119
High-level ideas across multiple cells	"I just converted the categorical [data] to numerical [data]."	206

Table 3. Granularity: the level of detail of the referenced elements

Chat messages are useful for explaining the exploration process. We were able to better understand the motivations for doing specific analyses, the purpose of the code written to run them, the interpretation of their results, and alternative analysis paths (tested or rejected without implementation). These details are often missing or poorly captured in traditional Jupyter Notebook artifacts.

Chat messages are difficult to follow. Chat messages are long and tedious to read because of scattered insights, a large amount of out-of-scope information, and information that requires notebook context to understand (which is likely to change before being finalized, as Table 2 shows). This makes it difficult for newcomers to build on earlier work.

Notebook elements are frequently referred to in chat messages. Notebook elements, such as fragments of code, output of executions, and cells containing a variety of statements, are frequently mentioned in chat messages. The lack of connection between these elements and discourse limits insight into decisions and results.

DESIGN OF CALLISTO

We designed Callisto to improve collaborative data science by better connecting discussions with notebook content. Callisto extends the Jupyter Notebook platform in several ways. First, it allows users to share notebooks, collaborate in real time, and discuss with collaborators. Second, it enables users to connect discussions with elements in the shared notebook, including code, output, individual cells, or edits. Third, it leverages these connections to make it easier to navigate discussions and notebook content—for example, to find discussions about a particular part of the notebook. We describe the design of each of these facets of Callisto in more detail below.

Enabling Sharing and Real-Time Collaboration

Although the creators of Jupyter recognized the importance of real-time collaboration, they left it as future work[†] [25]. Several offshoots of the Jupyter project [4, 3] have incorporated collaborative features such as synchronized editing and shared cursors. Callisto starts by enabling notebook sharing and edit synchronization in Jupyter. We designed Callisto as a Jupyter plugin, rather than as a fork of the codebase, to allow users to easily share any standard Jupyter notebook and maintain compatibility with future versions of the Jupyter platform.

Basic Collaboration Features

The Callisto plugin augments the standard Jupyter User Interface (UI) with several widgets, as Figure 2 shows. First, Callisto adds a "share" button that generates a unique URL for collaborators to join the shared notebook session. A panel lists the collaborators that are connected to the notebook (Figure 2.D). When collaborators join the notebook, their edits are synchronized in real time with other collaborators. They can also see every other user's cursor location and selection (Figure 2.F) and navigate to any other user's location by clicking on their name in the list of collaborators (Figure 2.D).

Shared Runtime and Outputs

One important difference between computational notebooks and standard code is that computational notebooks are divided into smaller *cells* that can be run individually. Cells run in a

[†]At the time of writing, Jupyter does not support real-time collaboration.

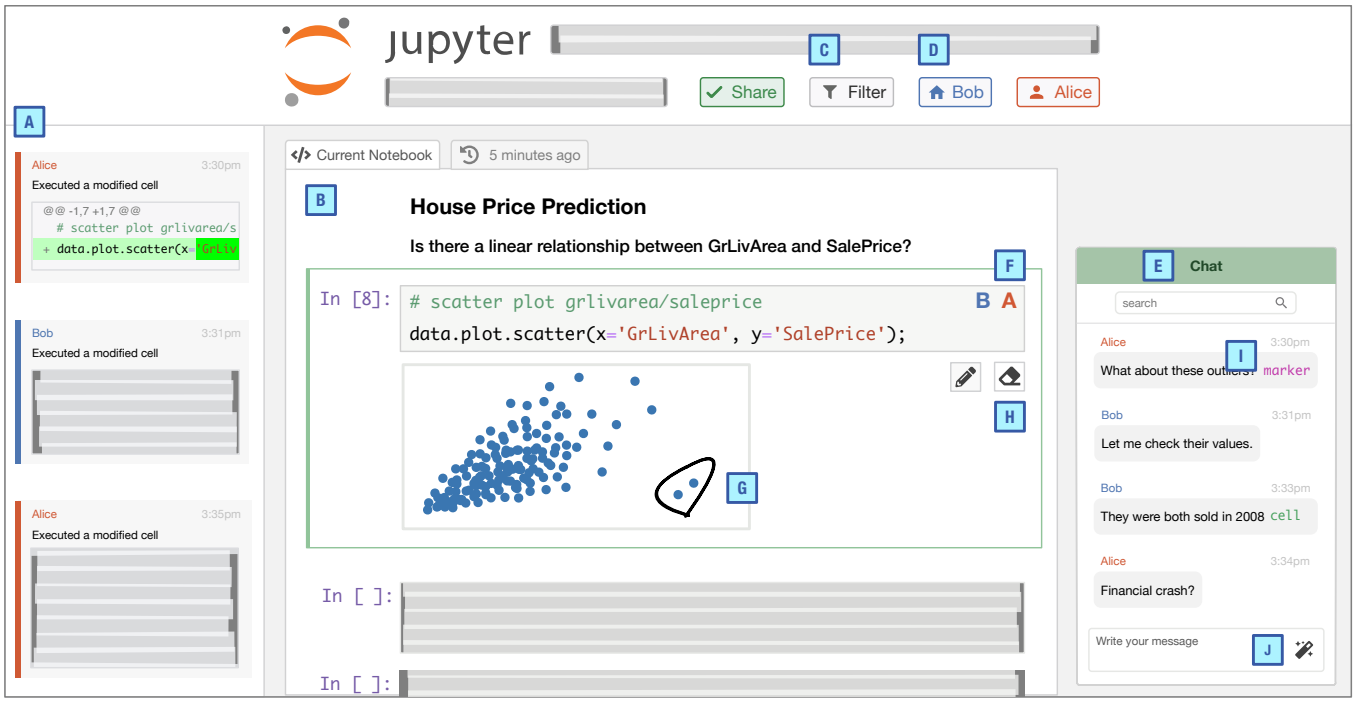

Figure 2. Overview of Callisto: (A) The changelog panel shows users' edit histories; (B) The collaborative notebook editor synchronizes edits, runtime variables, outputs, annotations (see G, H), and cursors (see F) among collaborators; (C) The filter button enables the filtering mode (see Figure 3); (D) The user panel lists collaborators that are connected to the notebook. Users can navigate to others' cursor locations by clicking on their name; (E) The embedded synchronous chat pane creates connections between messages and notebook content. Messages mapped to the selected cell are highlighted in light green. Users can create explicit references by clicking the magic wand (see J) and then selecting the relevant part of the notebook—for example, to create an annotation reference (see I).

common variable space, meaning that the ordering and timing of cell execution can (and typically does) influence execution outputs. This can be confusing for users, particularly in situations where one user's output cannot be replicated by other users who have different runtime states. Thus, rather than giving users their own runtime, Callisto connects every collaborator to a single shared runtime. This means that the state of the program is shared—if the value of a variable is modified (by executing code that modifies its value), its value is updated for every collaborator. Cell outputs (the results of running a cell, which can be textual, graphical, or shared data frames) are also shared automatically, which gives all collaborators a shared point of reference.

Synchronous Chat

Jupyter does not have built-in messaging features, which means that data science teams typically communicate through external tools such as e-mail or Slack [42]. As we found in our formative study, these communications can be valuable for understanding the design behind a notebook, but there is a cost in switching between applications for writing and communicating. Thus, Callisto embeds a synchronous chat pane directly in the shared notebook (Figure 2.E). This built-in chat pane allows us to capture contextual information and create connections between messages and notebook content, which we will introduce later.

Edit and Version History

Prior work has found that shared editors and cursors are helpful for collaborators but are not enough to build awareness of what they have worked on [42]. This is partly because they only allow users to see what collaborators are working on *at that specific moment*. Building awareness of collaborators' activity instead requires a more complete view of their actions. To provide this, Callisto includes a panel showing users' edit histories (Figure 2.A). This panel shows a history of notebook versions and a preview of user edits (which are displayed as diffs—additions and deletions from the previous snapshot). Every user action (such as cell edits, deletions, insertions, and executions) is recorded and displayed for users to see and better understand what their collaborators are working on. This panel also allows users to check notebook diffs in a complete view (by clicking on any diff summary), which will show the code and output differences (see Figure 4).

Connecting Messages and Notebook Content

As we found in our formative study, data scientists often refer to the computational notebook in their discussions. Prior work [33] has proposed enabling chat messages to refer to regions of code. However, our study participants referenced more than code; they referenced program output (which can be graphical or textual) and specific notebook cells. They also referenced things that were not explicitly part of the computational notebook, such as prior notebook versions or code edits themselves (e.g., *"I made this change…"* referring to edits they made to fix the buggy code). These references were implicit; they required readers to infer what they referred to. Callisto is the first system to explicitly encode these references.

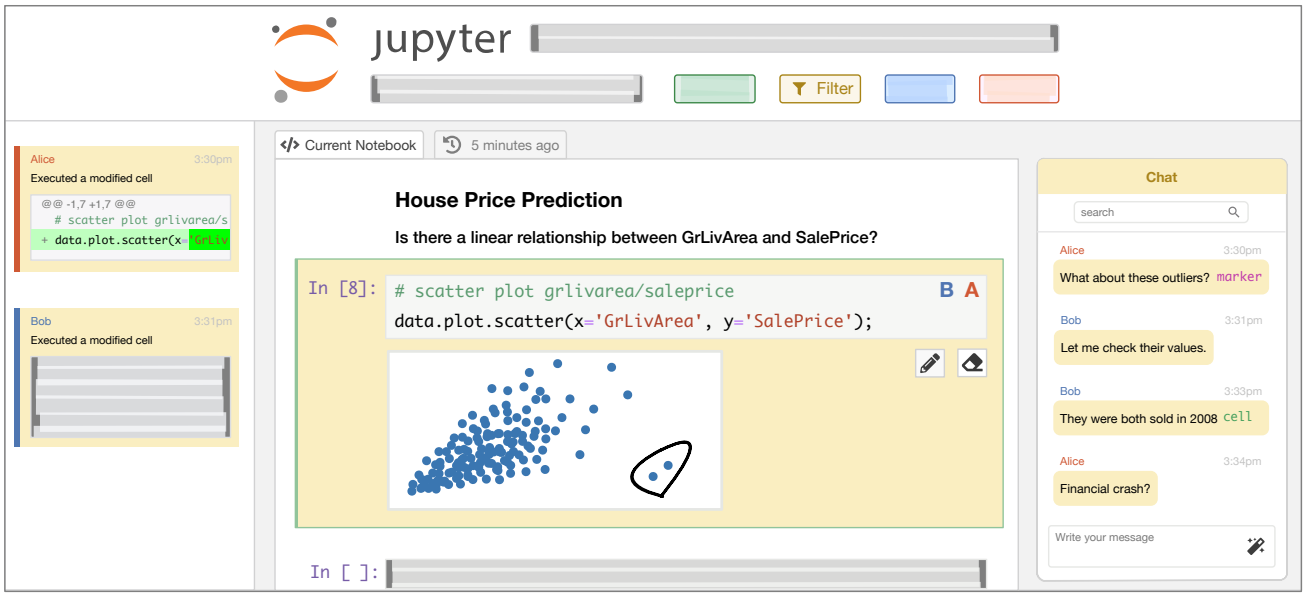

Figure 3. Filter Mode. When filter mode is enabled, it only displays messages and edits that are marked as relevant to the selected cell.

Encoding connections between messages and notebook content allows users to give their messages clear context and can make the computational notebook easier to interpret and navigate for future readers. Inspired by our formative study analyzing the granularity of the notebook elements collaborators referred to, messages make five types of references to notebook elements in Callisto:

- *code references* are associated with a specific range of code at the time when that reference was created,
- *cell references* are associated with a cell in the notebook,
- *snapshot references* point to a previous notebook version,
- *annotation references* allow users to refer to a specific portion of output (images or tables) by drawing annotations on that output and referencing those annotations, and
- *diff references* point to an edit in the notebook.

References can either be created explicitly by users or inferred by Callisto through context (as we describe in more detail below). Users explicitly create references by clicking the chat input's magic wand (Figure 2.J) and then selecting the relevant part of the notebook or version history panel.

Automatically Inferring References from Context

Although explicitly creating references requires little overhead (clicking the "edit link" button (Figure 5.B) and then the relevant part of the notebook), we built features to further reduce the effort required by automatically inferring references from users' work context—the cell that is currently selected or that they are editing, which their message likely pertains to. Although active collaborators might have no trouble decoding these messages' context (possibly by looking at where that user's cursor currently is), it can be more difficult for future collaborators as they catch up on prior discussions. Thus, Callisto automatically attaches a cell reference to the currently selected cell if users do not add an explicit reference.

This method of inference might produce erroneous references. For example, if the purpose of the message is planning, the message might relate to the cell that the user is going to edit, instead of the cell he just edited. However, we believe false negatives (when relevant context is not captured) are much more costly than false positives (when the context captured is not relevant) for users, as it is easier to ignore extraneous information than to recover missing information. Users can also manually correct errors from automatic inferences.

Navigating Messages and Notebook Content

By connecting messages and notebook content, Callisto gives a richer context to notebook elements and makes it easier to understand prior discussions. This can be helpful for both current collaborators and future readers. There are two broad uses for these connections: to understand the context of a given message (from messages to relevant notebook content) or to find the part of the discussion that is relevant to a specific part of the notebook (from notebook content to relevant messages). The former demonstrates "what changes were made" while the latter explains "why changes were made" [41].

From Messages to Notebook Content

While collaborating, data scientists often need to determine which part of a notebook a given message pertains to. Active collaborators and users reading past discussions benefit from certainty about a given message's context. In order to help build this context for messages, Callisto allows users to navigate from a reference in the chat panel to the relevant part of the notebook. References in the discussion panel appear like Web links. When a user clicks on the reference, Callisto highlights the relevant elements in the notebook (and scrolls to them if necessary). Messages might become "out of date" if they reference an element of the notebook that is later modified or deleted. To ensure references stay relevant, Callisto automatically "backtracks" references; if a user clicks on a reference to an element that was later changed, Callisto shows them the referenced content in a snapshot view.

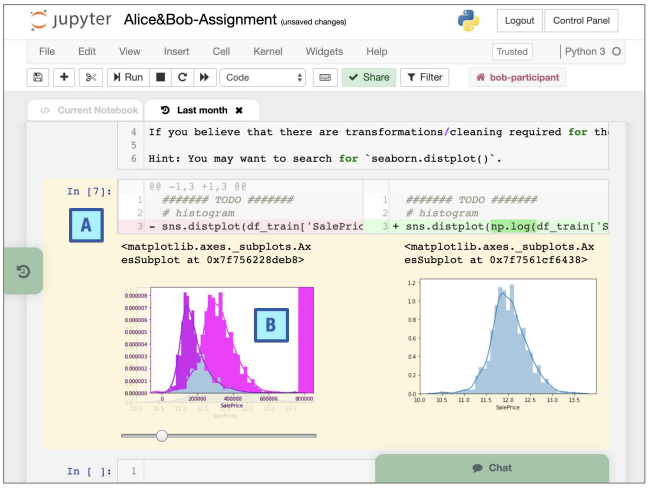

Figure 4. Diff View. Code differences (see A) and output differences (see B) are highlighted in a diff view. The new and old outputs are overlapped for comparison: hovering the mouse over the output will highlight the difference in purple and pink; the slider underneath controls the transparency between new and old output.

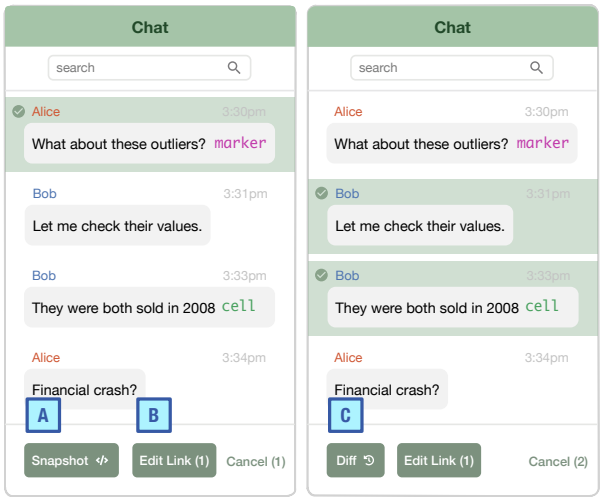

Figure 5. Chat Panel. When selecting one message, a snapshot button (see A) will navigate users to the snapshot of the notebook. When selecting two messages, a diff button (see C) will navigate users to the diff view comparing two snapshots (see Figure 4). Users can manually refine the links using the edit button (see B).

Subsequent notebook readers might also want to understand how the content of the notebook changed as the discussion moved on—what collaborators were doing between messages. To allow readers to understand how the notebook evolved through the discussion, Callisto enables them to compute the difference between any set of notebook versions. For example, if a user selects two chat messages, a diff button will appear in the chat panel, as Figure 5 shows. This will trigger Callisto to render the code and output differences between the state of the notebook when each of those messages was sent.

From Notebook Content to Messages

Computational notebooks are often shaped by many design decisions, failed experiments, and progressive iteration. For collaborative computational notebooks, explanations of *why* the notebook ended up the way it did can often be inferred through careful examination of discussions between collaborators. By linking notebook content to discussion messages, Callisto allows users to see which parts of a discussion are relevant for a particular part of the notebook. As Figure 3 shows, users can click on a cell to display relevant discussions.

EVALUATION

We designed a two-stage evaluation study with 32 data science students to assess how Callisto assists new collaborators when joining the collaborative notebook. We first observed participants working in pairs on a data science task in real time to test Callisto's usability (the *real-time collaboration study*). We then conducted a comparison study with a third individual joining the shared project using Callisto or a lite version of the system with no contextual links (the *follow-up study*).

General Study Protocol (for Both Stages)

The real-time collaboration study and the follow-up study follow a similar study protocol. We invited each participant for a 90-minute lab session. Before the study, participants reported their data science backgrounds on a pre-task questionnaire. Each participant was given a 15–20-minute training session on the tool, with example tasks to complete. After the study, we conducted a 10–15-minute semi-structured interview with each participant. We collected data from server-side usage logs, screen recordings, and post-task interviews. We also took observational notes during the study.

Participants (for Both Stages)

We reached out to data science programs and interest groups on campus, filtering qualified participants based on the courses they had taken and other data science-related experience. Overall, qualified participants were familiar with Jupyter Notebook, Python, and common exploratory data analysis packages (e.g., Pandas, NumPy). Most of them had experience of collaborating on an exploratory data analysis project.

We recruited 32 participants in total (11 female, 20 male, 1 non-binary, average age = 25). Participants were from a variety of data science-related programs (8 undergraduate students, 5 master's students, 18 Ph.D. students, and 1 full-time employee who recently graduated; students' majors included computer science, information science, health information, statistics, and economics).

Based on participants' prior knowledge, we rated their experience level as beginner[‡] (n=6), intermediate[§] (n=10), or expert[¶] (n=16). We randomly assigned participants into one of the two stages with a balanced distribution of experience level. There was no overlap in participants across study stages. We compensated participants with $25 USD gift cards.

Stage 1: Real-time Collaboration

In Stage 1, we investigated the perceived usability of Callisto for real-time collaboration. We observed eight participants

[‡]**Beginner:** has taken 1–2 data science classes, basic experience with Pandas and Python, but little experience with data science problems
[§]**Intermediate:** limited experience with data science problems
[¶]**Expert:** is familiar with libraries frequently used in data science, and very experienced in solving data science problems

S1 Real-time Collaboration (4 pairs)	$\bar{x}$	σ
Cell Edits	64.50	34.57
Messages	101.00	76.40
Creating Pointers	7.25	1.71
Making Annotations	11.25	7.45
Erroneous References	8	15.3
S2 Follow-up (10 individuals)	$\bar{x}$	σ
Clicking on Pointers	8.40	4.51
Viewing Notebook Snapshot	10.20	5.45
Viewing Notebook Diffs	15.30	7.60
Inspecting Cells for Curated Messages	44.80	48.93
Inspecting Messages for Related Cells	40.00	20.95

Table 4. Server-side Usage Logs (mean: $\bar{x}$, standard deviation: σ): (1) Most contextual links were created by inferred references; (2) The two navigating features were used equally to understand past decisions.

(P1–P8) working in pairs to solve a data science task together using the full version of Callisto. Participants were invited to the study site at the same time and sat in separate rooms. We informed participants at the outset that they would not have enough time to complete the task and a new collaborator would take over the remaining work.

The data science task was modified from a Kaggle competition (predicting house sale price). To scope the task within the study duration, we asked participants to only perform exploratory data analysis. We provided a basic framework of the notebook for participants to begin with, as well as example API usage code for common data analysis packages.

Overall Usage

As Table 4 shows, each group edited the notebook an average of 64.50 times during the 45-minute exploration. We recorded a cell editing event when a cell being executed was modified from its last execution. Participants frequently used the annotation feature (11.25 times per group) when discussing outputs. However, not all annotations were used for creating references. In fact, only 7% of the messages contained references that participants manually created. Most of these manually created references (26 out of 29) were cell pointers.

Creating References (Manual and Automatically Inferred)

In most cases where participants manually created a reference in a message, they used cell pointers with the default textual description "cell". Participants gave a variety of reasons for using cell pointers over other pointers, including that discussions are often not around a particular piece of code, and that pointing to a cell requires less effort than creating other pointers. Participants also mentioned that the ability to check their collaborator's cursor served the same function as the pointer when they were talking about code cells in real time:

> I can know my collaborator's cursor so it is easy to know what she is talking about. So we didn't use much references, only a few cell links. (P3, expert)

We identified five cases where participants could have used references to make their communication more efficient. For example, one participant could have directly pointed to a number in the table by creating an annotation reference, but instead he described it as *"the two values with bigger* `GrLivArea` *as rows with IDs 1182 & 691"*. Worse still, some participants described locations relative to their field of view, which further increased the difficulty for new collaborators to parse the message (*"If you scroll up to the cell above, it looks like the ID is always one higher than the Pandas index."*).

Because participants created relatively few manual references (7.25 out of 101 messages on average per group), Callisto's ability to automatically infer relationships between messages and notebook cells is crucial. In order to understand how well Callisto's reference inference feature works, we manually checked the messages and found that 92% of messages were connected to the correct context (only a total of eight messages were mismatched with the inferred cell references).

Annotations Aid Communication

Participants used the annotation feature frequently, and we investigated its popularity in the post-task interview. Most participants agreed that the annotation feature reduced communication costs:

> A lot of our discussions are about the graphs. I really like the ability to draw on the graphs so we knew what exactly we were talking about. (P4, intermediate)

> [When using Slack] I have to make a screenshot and save it on desktop. I do not like saving too many images on the desktop so I like this tool. (P1, beginner)

Stage 2: Following up with the Collaboration Process

In Stage 2, we evaluated how a new collaborator better followed up with an ongoing collaborative project. We compared two versions of Callisto in this stage: a *lite version* where no contextual links are captured and stored, only basic collaboration features are enabled; and the *full version*.

Content Preparation

We designed the assets (the notebook history, chat messages, and their connections) for the "ongoing collaborative project" by merging and modifying the collaboration assets produced in Stage 1. The combined project used in Stage 2 contained 42 cell edits, 132 messages, and 19 manual references. In the lite version of Callisto, we replaced the manual references with a textual description of the location in the notebook. To ensure these textual descriptions were realistic, we observed two more groups (in addition to the pairs described in the previous section) doing tasks in Stage 1 using the lite version. We identified several strategies that participants used to point to notebook elements, and replaced the references based on the three most common strategies: cell execution number, pasting the content directly, and describing the location of the content.

Study Setup

We recruited 20 participants and randomly assigned them to one of the two conditions: the experimental condition using Callisto, and the control condition using the lite version. We informed participants that the previous collaborators (Alice and Bob) were in a rush and did not finish the exploration.

		Callisto	Control
Questionnaire Score*	$\bar{x}$	30.05	23.75
	σ	6.27	3.85
Time (sec)	$\bar{x}$	1577.21	1416.05
	σ	237.35	320.28
Self-reported Confidence	$\bar{x}$	5.00	5.26
	σ	0.66	1.16

Table 5. Comparing the outcomes from the second stage of the evaluation (mean: $\bar{x}$, standard deviation: σ). Callisto helps new collaborators achieve a better understanding of an ongoing project.

We asked participants to explore the notebook and answer five questions[‖] related to Alice and Bob's prior analysis. The questions were designed using Revised Bloom's Taxonomy (RBT) to assess participants' understanding [28]. For example, *outlining* features that Alice and Bob have explored, and *summarizing* their findings about the distribution of sale price. Participants had six minutes to read details from the notebook and answer each question. We collected the answers and measured time and participants' self-reported confidence level (on a seven-point Likert scale) for each question. At the end, we gave participants 10 minutes to use the tool in depth to follow up on their work (e.g., clean the notebook, add more explanations, or continue exploring the problem).

To assess how well participants understood the ongoing collaboration process, we designed a rubric to grade their answers to the five questions (maximum score = 50). Two external data science experts independently graded their answers. We performed a Pearson correlation coefficient test and found a strong agreement on the rating ($r = 0.97$, $p < 0.001$).

Overall Performance

As Table 5 shows, participants in the experimental condition (avg = 30.05) achieved a higher score than participants in the control condition (avg = 23.75), with a two-sample t-test suggesting that the difference is significant ($p = 0.014$). There was no significant difference in the time costs or the self-reported confidence level between the two conditions.

To investigate why participants performed better in the experimental condition, we studied their usage logs and screen recordings. As Table 4 shows, participants in the experimental condition used the two navigating features in Callisto equally to understand past decisions and discussions.

Understanding Discussions Around the Cell

Participants in both conditions reported a need to check the chat messages even though the notebook already contained some code comments and explanatory texts. They complained that the code comments were not well written:

> Some comments are hard to parse. (P22, expert, experimental condition)

> They could have used the markdown cells more to conclude the results. (P16, expert, experimental condition)

Comparatively, participants in the control condition found it difficult to follow the chat messages due to the sheer quantity. We observed that three participants in the control condition misaligned the chat messages with the notebook content when answering a question about how Alice and Bob analyzed the linear relationship between `SalePrice` and `YearBuilt`. They answered the question incorrectly because they described the discussions about the linear relationship between `SalePrice` and another feature (`GrLivArea`, which appeared earlier in the analysis). Two participants in the control condition wanted chat messages to be mapped with notebook content:

> I wish there is a way to attach the messages to the cell that they discussed. It will save me time. (P30, intermediate, control condition)

While participants in the experimental condition benefited from the established connection between messages and notebook, they further reported the filter feature helpful in curating discussions around cells. On average, each participant inspected cells 44.8 times to filter related messages, checking 14.9 unique cells. In addition, we observed that most participants (9 out of 10) preferred to keep the filtering mode enabled as they dove deeper in the notebook:

> Because the chat is so long, I think it is not useful until I filter it down. (P18, expert, experimental condition)

Understanding the Context of the Message

We observed participants used the contextual links in Callisto the other way (from messages to notebook content) to understand the context of a message. Participants inspected messages 40 times to check related cells in the final notebook, or perform further actions such as checking snapshots (10.2 times) or comparing diffs (15.3 times). 27.4 unique messages were inspected by each participant, indicating that participants may go back and forth to check messages and related cells.

We further investigated why checking and comparing notebook edits from messages helped participants better understand the analysis from observation notes and screen recordings. We illustrated one interesting case of how participants approached the answers in the questionnaire differently. One question asked how Alice and Bob analyzed the outliers in the `GrLivArea`. Participants from the experimental condition were able to find all relevant analyses with two alternative hypotheses, where the code cell for testing one hypothesis was overwritten by the code cell that tested the second hypothesis. However, most participants from the control condition only reported the second hypothesis on the final notebook.

In addition, Callisto helped participants better understand how a code change resulted in an output change. As shown in Figure 4, Alice and Bob applied a log transformation to correct the distribution of `SalePrice`, only commenting vaguely on the results in the chat (*"the result looks much better"*). Participants in the experimental condition were able to compare the notebook diffs between this message and the one above, gaining an intuitive comparison of how the output changed from the diff view (see Figure 4.B). In contrast, participants in the control condition needed to first guess what might have

[‖]See supplementary materials for the full rubric and set of questions.

changed in the notebook, then revert changes (e.g., remove the `np.log`) and execute the cell to compare the output.

DISCUSSION

Reflecting on Callisto's design, we discuss how future tool builders of computational notebooks and data science researchers can build on our work.

Reducing the Burden of Communication

Our findings revealed that participants in the real-time collaboration setting are hesitant to make accurate and polished references, or to create references, even if the interaction takes only two clicks. This corresponds to studies in other domains that report user reluctance to write quality annotations or comments during active work [7, 31, 6]. Future work should consider optimizing this process by designing shortcuts or providing suggested references inferred by edits in the notebook (e.g., a newly added annotation).

As prior work [42] shows, data scientists use a variety of communication tools, including high-bandwidth communication channels such as video conferencing or face-to-face meetings. Capturing information exchanged in these channels is difficult yet important to reduce the burden of text-based communication. It is worth studying the benefits and challenges of using different communication channels in data scientists' daily work to leverage past discussions for a better understanding of shared work.

In addition, we believe similar techniques could work in other domains where remote collaborators co-design a shared artifact that changes over time, as long as the reference types are domain appropriate. For example, Callisto's features could be adapted for a shared CAD tool where designers collaborate on a 3D model, but our results and designs may not apply for highly modular work (such as multiple authors writing different chapters of a textbook with minimal interaction).

Improving the Accuracy of Contextual Links

As most of the messages (around 93%) relied on inferred references, we believe that it is important to explore ways to further improve the accuracy and recall of inferred references. Mismatched contextual links happened for several reasons. If a message describes a future action, the relevant cell may not exist when the message is sent. In this case, we may consider using Natural Language Processing (NLP) techniques to infer whether the message should be connected to the cell edited before sending the message or the cell edited after sending the message. Another possible reason is that a message might reference a cell the writer's collaborator is working on, instead of the one the writer is working on. It is worth exploring other strategies (e.g., considering common cells that nearby messages connect to) to automatically infer the context.

Towards Generating Meta-Narratives

New collaborators not only need to understand the computational narrative itself but also how that narrative evolved—the *meta-narrative* behind the narrative. Callisto is a representation of meta-narratives for computational notebooks. Creating an explicit meta-narrative object can be useful for onboarding new collaborators during the data-exploration process, as we found in our evaluation. These meta-narratives could also be useful in education; many programming lectures involve creating a form of meta-narrative. They could also be used in "traditional" writing. Future research could explore alternative representations for meta-narratives for a variety of domains.

Limitations

Callisto is designed and evaluated in the scope of within-notebook collaboration, where collaborators work in the same notebook and treat the final narrative as an end goal. The setup of the formative study is designed to encourage real-time chatting and collaboration, which may not be an accurate representation of most collaboration and communication scenarios. In addition, our in-lab evaluation contains several limits to external validity: participants are all students from the authors' home institution; participants may not be proficient enough in Callisto given the short training time; we only evaluated one type of data science problem and provided the framework of the notebook rather than asking them to start from scratch.

SYSTEM IMPLEMENTATION

Callisto** consists of two Jupyter Notebook extensions—one small extension for Jupyter's file browser (to make it easier to join shared notebooks) and the "main" extension for Jupyter Notebooks (Figure 2)—and a Node.js backend. Callisto keeps collaborators in sync (including notebook content, chat, lists of collaborators, and runtime state) through Operational Transformations (OTs), as implemented through ShareDB [1]. To maintain connections between messages and cells, Callisto tracks the edit history through the lifetime of the notebook and stores a unique id in the metadata of each cell, which stays constant as cells are inserted, deleted, and rearranged.

CONCLUSION

In conclusion, we have proposed the design of Callisto to leverage valuable chat messages in collaborative data science. Our two-stage evaluation study with 32 data science students confirmed that Callisto eases new-collaborator onboarding by helping them understand the design rationales of the notebook's authors. In particular, Callisto successfully captures contextual links during the real-time collaborative creation of the notebook without hindering exploration, while the establishment of contextual links and the set of interactions for navigating the notebook significantly improve new notebook collaborators' understanding of past discussions and decisions.

ACKNOWLEDGMENTS

We thank all of our participants and our reviewers for their valuable feedback. We also thank the Michigan Institute for Data Science (MIDAS). This material is based upon work supported by the National Science Foundation under Grant Numbers IIS 1755908 and EHR 1915515.

REFERENCES

[1] 2013. ShareDB. (2013). `https://github.com/share/sharedb`.

**Callisto's source is available at github.com/littleaprilfool/callisto

[2] 2019. Autolinked references and URLs - GitHub Help. (2019). https://help.github.com/en/articles/autolinked-references-and-urls

[3] 2019. Deepnote. (2019). https://www.deepnote.com.

[4] 2019. Google Colaboratory. (2019). https://colab.research.google.com.

[5] 2019. JSFiddle. (2019). https://jsfiddle.net.

[6] Adrian Bachmann and Abraham Bernstein. 2009. Software process data quality and characteristics: a historical view on open and closed source projects. In *Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops*. ACM, 119–128.

[7] Raymond PL Buse and Westley Weimer. 2010. Automatically documenting program changes.. In *ASE*, Vol. 10. 33–42.

[8] Yan Chen, Sang Won Lee, Yin Xie, YiWei Yang, Walter S Lasecki, and Steve Oney. 2017. Codeon: On-Demand Software Development Assistance. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. ACM.

[9] Parmit K. Chilana, Nathaniel Hudson, Srinjita Bhaduri, Prashant Shashikumar, and Shaun Kane. 2018. Supporting Remote Real-Time Expert Help: Opportunities and Challenges for Novice 3D Modelers. In *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 157–166. DOI: http://dx.doi.org/10.1109/VLHCC.2018.8506568

[10] Parmit K. Chilana, Amy J. Ko, and Jacob O. Wobbrock. 2012. LemonAid: Selection-based Crowdsourced Contextual Help for Web Applications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*. ACM, New York, NY, USA, 1549–1558. DOI: http://dx.doi.org/10.1145/2207676.2208620

[11] Soon Hau Chua, Toni-Jan Keith Palma Monserrat, Dongwook Yoon, Juho Kim, and Shengdong Zhao. 2017. Korero: Facilitating Complex Referencing of Visual Materials in Asynchronous Discussion Interface. *Proc. ACM Hum.-Comput. Interact.* 1, CSCW, Article 34 (Dec. 2017), 19 pages. DOI: http://dx.doi.org/10.1145/3134669

[12] Sarah D'Angelo and Andrew Begel. 2017. Improving communication between pair programmers using shared gaze awareness. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. ACM, 6245–6290.

[13] Susan R Fussell and Robert M Krauss. 1992. Coordination of knowledge in communication: Effects of speakers' assumptions about what others know. *Journal of personality and Social Psychology* 62, 3 (1992), 378.

[14] Susan R Fussell, Robert E Kraut, and Jane Siegel. 2000. Coordination of communication: Effects of shared visual context on collaborative work. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work*. ACM, 21–30.

[15] Philip J. Guo. 2012. *Software tools to facilitate research programming*. Ph.D. Dissertation. Stanford University Stanford, CA.

[16] Pavel Gurevich, Joel Lanir, Benjamin Cohen, and Ran Stone. 2012. TeleAdvisor: A Versatile Augmented Reality Tool for Remote Assistance. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*. ACM, New York, NY, USA, 619–622. DOI: http://dx.doi.org/10.1145/2207676.2207763

[17] David Haney. 2014. Introducing Runnable JavaScript, CSS, and HTML Code Snippets. (2014). Retrieved September 15, 2019 from https://stackoverflow.blog/2014/09/16/introducing-runnable-javascript-css-and-html-code-snippets/.

[18] Andrew Head, Fred Hohman, Titus Barik, Steven M. Drucker, and Robert DeLine. 2019. Managing Messes in Computational Notebooks. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*. ACM, New York, NY, USA, Article 270, 12 pages. DOI: http://dx.doi.org/10.1145/3290605.3300500

[19] Project Jupyter. 2015. Project Jupyter: Computational Narratives as the Engine of Collaborative Data Science. (2015). Retrieved September 15, 2019 from https://blog.jupyter.org/project-jupyter-computational-narratives-as-the-engine-of-collaborative-data-science-2b5fb94c3c58.

[20] Mary Beth Kery, Amber Horvath, and Brad Myers. 2017. Variolite: Supporting Exploratory Programming by Data Scientists. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM, New York, NY, USA, 1265–1276. DOI:http://dx.doi.org/10.1145/3025453.3025626

[21] 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 (CHI '19)*. ACM, New York, NY, USA, Article 92, 13 pages. DOI:http://dx.doi.org/10.1145/3290605.3300322

[22] Mary Beth Kery and Brad A. Myers. 2017. Exploring exploratory programming. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (2017-10). 25–29. DOI: http://dx.doi.org/10.1109/VLHCC.2017.8103446

[23] Mary Beth Kery, Marissa Radensky, Mahima Arya, Bonnie E. John, and Brad A. Myers. 2018. The Story in the Notebook: Exploratory Data Science Using a Literate Programming Tool. In *Proceedings of the 2018*

CHI Conference on Human Factors in Computing Systems (CHI '18). ACM, New York, NY, USA, Article 174, 11 pages. DOI: http://dx.doi.org/10.1145/3173574.3173748

[24] Miryung Kim, Thomas Zimmermann, Robert DeLine, and Andrew Begel. 2016. The Emerging Role of Data Scientists on Software Development Teams. In *Proceedings of the 38th International Conference on Software Engineering* (2016) *(ICSE '16)*. ACM, 96–107. DOI:http://dx.doi.org/10.1145/2884781.2884783

[25] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian E Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica B Hamrick, Jason Grout, Sylvain Corlay, and others. 2016. Jupyter Notebooks-a publishing format for reproducible computational workflows.. In *ELPUB*. 87–90.

[26] Donald E. Knuth. 1984. Literate Programming. *Comput. J.* 27, 2 (1984), 97–111. DOI: http://dx.doi.org/10.1093/comjnl/27.2.97

[27] Laura Koesten, Emilia Kacprzak, Jeni Tennison, and Elena Simperl. 2019. Collaborative Practices with Structured Data: Do Tools Support What Users Need?. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*. ACM, New York, NY, USA, Article 100, 14 pages. DOI: http://dx.doi.org/10.1145/3290605.3300330

[28] David R Krathwohl. 2002. A revision of Bloom's taxonomy: An overview. *Theory into practice* 41, 4 (2002), 212–218.

[29] 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*. DOI: http://dx.doi.org/10.1145/3290605.3300493

[30] Ching Liu, Juho Kim, and Hao-Chuan Wang. 2018. ConceptScape: Collaborative Concept Mapping for Video Learning. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. ACM, New York, NY, USA, Article 387, 12 pages. DOI:http://dx.doi.org/10.1145/3173574.3173961

[31] Walid Maalej and Hans-Jorg Happel. 2009. From work to word: How do software developers describe their work?. In *2009 6th IEEE International Working Conference on Mining Software Repositories*. IEEE, 121–130.

[32] Michael Muller, Ingrid Lange, Dakuo Wang, David Piorkowski, Jason Tsay, Q. Vera Liao, Casey Dugan, and Thomas Erickson. 2019. How Data Science Workers Work with Data: Discovery, Capture, Curation, Design, Creation. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*. ACM, New York, NY, USA, Article 126, 15 pages. DOI: http://dx.doi.org/10.1145/3290605.3300356

[33] Steve Oney, Christopher Brooks, and Paul Resnick. 2018. Creating Guided Code Explanations with chat. codes. *Proceedings of the ACM on Human-Computer Interaction* 2, CSCW (2018), 131.

[34] Soya Park, Amy X. Zhang, and David R. Karger. 2018. Post-literate Programming: Linking Discussion and Code in Software Development Teams. In *The 31st Annual ACM Symposium on User Interface Software and Technology Adjunct Proceedings (UIST '18 Adjunct)*. ACM, New York, NY, USA, 51–53. DOI: http://dx.doi.org/10.1145/3266037.3266098

[35] Samir Passi and Steven J. Jackson. 2018. Trust in Data Science: Collaboration, Translation, and Accountability in Corporate Data Science Projects. 2 (2018), 136:1–136:28. Issue CSCW. DOI: http://dx.doi.org/10.1145/3274405

[36] Jeffrey M. Perkel. 2018. Why Jupyter is data scientists' computational notebook of choice. *Nature* 563 (2018), 145. DOI: http://dx.doi.org/10.1038/d41586-018-07196-1

[37] Bernadette M. Randles, Irene V. Pasquetto, Milena S. Golshan, and Christine L. Borgman. 2017. Using the Jupyter Notebook as a Tool for Open Science: An Empirical Study. In *2017 ACM/IEEE Joint Conference on Digital Libraries (JCDL)*. 1–2. DOI: http://dx.doi.org/10.1109/JCDL.2017.7991618

[38] Adam Rule, Ian Drosos, Aurélien Tabard, and James D. Hollan. 2018a. Aiding Collaborative Reuse of Computational Notebooks with Annotated Cell Folding. *Proc. ACM Hum.-Comput. Interact.* 2 (2018), 150:1–150:12. Issue CSCW. DOI: http://dx.doi.org/10.1145/3274419

[39] Adam Rule, Aurélien Tabard, and James D. Hollan. 2018b. Exploration and Explanation in Computational Notebooks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. ACM, New York, NY, USA, Article 32, 12 pages. DOI: http://dx.doi.org/10.1145/3173574.3173606

[40] Adam Carl Rule. 2018. *Design and Use of Computational Notebooks*. Ph.D. Dissertation. University of California San Diego.

[41] James Tam and Saul Greenberg. 2006. A Framework for Asynchronous Change Awareness in Collaborative Documents and Workspaces. *Int. J. Hum.-Comput. Stud.* 64, 7 (July 2006), 583–598. DOI: http://dx.doi.org/10.1016/j.ijhcs.2006.02.004

[42] April Yi Wang, Anant Mittal, Christopher Brooks, and Steve Oney. 2019. How Data Scientists Use Computational Notebooks for Real-Time Collaboration. 3 (2019), 39:1–39:30. Issue CSCW. DOI: http://dx.doi.org/10.1145/3359141

[43] Matin Yarmand, Dongwook Yoon, Samuel Dodson, Ido Roll, and Sidney S. Fels. 2019. "Can You Believe [1:21]?!": Content and Time-Based Reference Patterns

in Video Comments. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*. ACM, New York, NY, USA, Article 489, 12 pages. DOI:http://dx.doi.org/10.1145/3290605.3300719

[44] Amy X. Zhang and Justin Cranshaw. 2018. Making Sense of Group Chat Through Collaborative Tagging and Summarization. *Proc. ACM Hum.-Comput. Interact.* 2, CSCW, Article 196 (Nov. 2018), 27 pages. DOI: http://dx.doi.org/10.1145/3274465

[45] Sacha Zyto, David Karger, Mark Ackerman, and Sanjoy Mahajan. 2012. Successful Classroom Deployment of a Social Document Annotation System. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*. ACM, New York, NY, USA, 1883–1892. DOI: http://dx.doi.org/10.1145/2207676.2208326