Jupyter Analytics: A Toolkit for Collecting, Analyzing, and Visualizing Distributed Student Activity in Jupyter Notebooks

Zhenyu Cai
EPFL
Lausanne, Switzerland
zhenyu.cai@epfl.ch

Richard Lee Davis
KTH Royal Institute of Technology
Stockholm, Sweden
rldavis@kth.se

Raphaël Mariétan
Swisscom
Lausanne, Switzerland
raphael.marietan@swisscom.com

Roland Tormey
EPFL
Lausanne, Switzerland
roland.tormey@epfl.ch

Pierre Dillenbourg
EPFL
Lausanne, Switzerland
pierre.dillenbourg@epfl.ch

Abstract

Jupyter is a web-based, interactive computing environment that supports many commonly-used programming languages. It has been widely adopted in the CS education community and is now rapidly expanding to other STEM disciplines due to the growing integration of programming in STEM education. However, unlike other educational platforms, there is currently no integrated way to capture, analyze, and visualize student interaction data in Jupyter notebooks. This means that teachers have limited to no visibility into student activity, preventing them from drawing insights from these data and providing timely interventions on the fly. In this paper, we present Jupyter Analytics, an end-to-end solution for teachers to collect, analyze, and visualize both synchronous and asynchronous learning activities in Jupyter. The Jupyter Analytics system consists of two JupyterLab extensions connected via a cloud-based backend. On the student side, we introduce the Jupyter Analytics Telemetry extension to anonymously capture students' interaction activity with more structure and higher granularity than log data. On the teacher side, we introduce the Jupyter Analytics Dashboard extension, which visualizes real-time student data directly in the notebook interface. The Jupyter Analytics system was developed through an iterative co-design process with university instructors and teaching assistants, and has been implemented and tested in several university STEM courses. We report two use cases where Jupyter Analytics impacted teaching and learning in the context of exercise sessions, and discuss the potential value of our tools for CS education.

CCS Concepts

• **Applied computing → Education**; • **Human-centered computing → User interface toolkits**; *Participatory design*; • **Social and professional topics → Student assessment**.

Keywords

STEM education; Jupyter; Educational Dashboards; Learning Analytics; Programming

ACM Reference Format:
Zhenyu Cai, Richard Lee Davis, Raphaël Mariétan, Roland Tormey, and Pierre Dillenbourg. 2025. Jupyter Analytics: A Toolkit for Collecting, Analyzing, and Visualizing Distributed Student Activity in Jupyter Notebooks. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE TS 2025), February 26-March 1, 2025, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3641554.3701971

1 Introduction

Computational literacy has become a central part of science, technology, engineering, and mathematics (STEM). Across STEM disciplines, professionals use computers when working with data, creating models and simulations, and running experiments [4]. Because of the need for computational literacy in STEM professions, the importance of integrating computer science (CS) into STEM education has been explicitly called for in frameworks such as the Next-Generation Science Standards [29]. In response, programming and computation are becoming an integral part of STEM courses, especially in higher education [13, 22, 28].

Integrating CS into STEM courses has been shown to have meaningful learning benefits [3, 32]. However it also introduces a unique set of challenges, since non-CS majors often have distinct needs and expectations that differ from those of their CS-focused peers. Rather than seeking to develop deep programming expertise, non-majors place greater emphasis on learning computational tools and techniques to support their primary fields of study [7, 18]. Accordingly, non-majors stand to benefit from tools and resources that can be seamlessly integrated into their research and coursework without imposing a steep learning curve or exposing the low-level computational details [18, 23].

Jupyter notebooks are a tool that fulfills many of the needs of non-majors in STEM courses. Jupyter notebooks provide a literate programming environment [21] where code, data, interactive controls, and text are combined in a single shareable document [15]. Jupyter notebooks consist of a series of "cells" that can be executed in any sequence, supporting experimentation and rapid feedback. For hiding low-level details, supporting iterative and non-linear development, and packaging these features into the familiar lab-notebook format, Jupyter has been named "one of the ten computer codes that transformed science" [27] and has been widely adopted across CS and STEM disciplines [1, 5, 8, 14, 33].

In STEM courses incorporating Jupyter notebooks, however, a major problem is that teachers lack visibility into student activity in Jupyter notebooks. Jupyter was not explicitly designed as an educational tool, and as such lacks the data-collection and visualization features that are common in learning management systems. Prior work has shown that tools which provide the ability to collect, process, and visualize student interaction data provide numerous benefits for teachers and students. Examples include early detection of struggling learners [2, 11, 24], disengagement identification in large classes [20], and the ability to provide timely, explicit instruction (i.e., classroom orchestration) [12, 16].

The fact that Jupyter lacks data collection and visualization tools is particularly problematic because of its widespread and common use in STEM courses with non-CS majors. As previously mentioned, these students have unique expectations and needs that differ from those of CS majors. In particular, non-majors benefit from well-timed explicit instruction [23] and responsive support during problem solving [19]. When students use Jupyter notebooks, these forms of support can be difficult for teachers to provide because they lack tools that provide visibility into student activity. While instructors can address this problem by physically visiting students in the classroom and viewing their screens, this is less viable in courses with large enrollments, in online settings, or when students work with Jupyter notebooks outside of class.

We built Jupyter Analytics to help address this problem. Jupyter Analytics is an end-to-end data collection and visualization toolkit that was designed to meet the needs of STEM instructors in higher education. The Jupyter Analytics system consists of two JupyterLab extensions connected via a cloud-based backend. On the student side, we introduce the Jupyter Analytics Telemetry extension to anonymously capture students' interaction activity with more structure and higher granularity than existing methods. On the teacher side, we introduce the Jupyter Analytics Dashboard extension, which visualizes real-time student data directly in the notebook interface. The system was developed through an iterative co-design process with university instructors and teaching assistants (TAs), and has been implemented and tested in several university STEM courses. We report two use cases where Jupyter Analytics positively impacted teaching and learning in the context of exercise sessions, and discuss the potential value of our tools for supporting responsive teaching in computational STEM education.

2 Related Work

While there are a number of Jupyter extensions for collecting interaction data [26, 30, 31], there are only a small number of end-to-end analytics toolkits that collect, process, and visualize Jupyter data.

The first of these is nbgrader [17], which is part of the Jupyter project and has been adopted widely because of its strengths in creating and grading Jupyter-notebook-based assignments. Nbgrader collects data related to assignment grading, such as submission timestamps and counts. It also maps the contents of notebook cells with metadata, and stores grades and feedback in the database. However, because nbgrader is only used after students have submitted their final notebooks, this means the tool can't be used to provide real-time information about students' activity. Furthermore, because no interaction data is collected, nbgrader is unable to provide insights related to the students' process of working in notebooks.

A second tool that addresses some of the issues with nbgrader is Notebook Progress Tracker (NPT) [6]. After configuring a session on its website, a teacher can track students' real-time progress by checking the dashboard provided. The NPT dashboard visualizes the number of students that have completed each questions for any registered sessions, while it can also provide a list of student code and plots, or data entries. However, these data have to be submitted from students by manually calling a "send" function in the code, instead of being collected automatically. Additionally, because the NPT does not track click data, it is not able to provide real-time information about students' location in the notebook or their time spent in specific cells.

While nbgrader and NPT do provide some visibility into student activity in Jupyter notebooks, they fall short of providing a holistic solution that supports the entire workflow from automatic collection of real-time interaction data to data visualizations integrated directly into Jupyter notebooks. This is the gap that Jupyter Analytics was designed to fill.

3 Jupyter Analytics

Our primary goal was to bring teachers more visibility into students' learning activities in Jupyter notebooks. This required a toolkit built into the Jupyter platform that was easy to use, that would not affect the user experience and system performance, and that would be capable of retrieving large amounts of post-processed data and visualizing them in a responsive way.

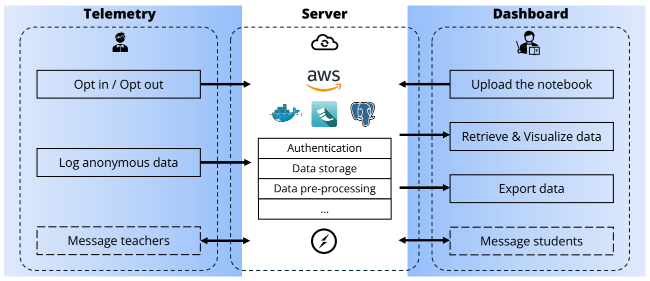


Figure 1: Overview of the system architecture

We designed and built Jupyter Analytics[1] with these constraints in mind. There are three main components of Jupyter Analytics, as Figure 1 shows:

(1) Jupyter Analytics Dashboard – a JupyterLab extension to visualize and interact with collected data;
(2) Jupyter Analytics Telemetry – a JupyterLab extension to record interactions with Jupyter notebooks;
(3) Jupyter Analytics Server – a cloud server to store and process data, as well as handling user requests and verifying user authenticity.

It should be noted that the two extensions can be easily installed in users' JupyterLab environments using common *pip install* commands.

[1]Jupyter Analytics: https://github.com/chili-epfl/jupyter-analytics

The design of Jupyter Analytics enables the collection of distributed student data from any computing environment that supports JupyterLab extensions, whether hosted individually or centrally. The extensions are compatible with JupyterLab 3.x and 4.x environments, making them suitable with most Jupyter installations.

3.1 The Jupyter Analytics Dashboard Extension

The main purpose of the dashboard extension is to provide teachers with useful data analytics features that could facilitate responsive teaching practices within specific educational contexts. To achieve this, we took both synchronous and asynchronous use cases into consideration when designing the dashboard, and built the dashboard directly into the Jupyter interface. By layering analytics information directly into the notebook, we hoped to increase usability and adoption, and to minimize task switching away from the Jupyter notebook.

In the following sections we provide additional details about the features in the dashboard extension (see Figure 2).

3.1.1 The Student Location Dashboard. The Student Location Dashboard is integrated into the left panel, and can be accessed by clicking a button on the left sidebar. The primary function of the this dashboard is to show where students are in the notebook in real time. The location of each student is visualized directly in a table-of-contents showing the structure of the code and markdown cells in the notebook. The number of students in a cell is presented next to the headings and thumbnails in the table-of-contents in the format of *active_number / total_number.* The intensity of the background colors correspond to the actual values – the darker the color is, the more the students are active on the corresponding cells or sections.

The Student Location Dashboard was designed so teachers can monitor the overall progress of their students at a glance. For example, teachers who incorporate programming activities into their lecture could use the dashboard to check if any students are falling behind, and then adjust their lesson plan accordingly.

3.1.2 Aggregate Visualization. On the right panel, another dashboard is presented to provide an aggregate view of different metrics about real-time whole-class performance on the notebook. While this dashboard is able to display any aggregate visualizations of student data, in its current configuration it shows a bar chart that visualizes the number of clicks, executions, and error-free executions of all code cells; and a bubble chart displaying the number of students who have visited a cell (size) and the average time students have spent on each cell (vertical displacement).

In real-time settings, these charts could be used to estimate class engagement, problem-solving strategies (e.g., trial-and-error), and issues with specific cells. Outside of class, aggregate visualizations could be useful for reviewing the course materials. For example, based on students' coding strategies and time-on-task, teachers could evaluate the difficulty level of the notebook and cells, and revise them if needed.

3.1.3 Cell-Level Visibility. In addition to notebook-level visualization, the right panel dashboard also provides visibility into students' programming activities at the cell level. By clicking the dashboard button next to a focused cell on the notebook, or by clicking on the visualized numbers on the Student Location Dashboard, teachers can navigate to the cell-level dashboard. Every time a student executes a code cell, the input and output (including plots) will be displayed with a timestamp. Moreover, these codes and outputs can be filtered by clicking on toggle buttons or by typing into the search box, or sorted by time, length of code, or length of output.

This dashboard could be useful if teachers want to know if there are common errors with a specific cell, which could be done by entering keywords in the search box and toggling on the "Error" filter. Additionally, when reviewing the course materials, teachers could also dig into a problematic cell, because the dashboard still keeps the latest records of students who have executed this cell.

3.1.4 Reflection & Posterior Analysis. The dashboard by default is configured as a real-time tool, however, this can be disabled in several ways, in order to benefit from the features that are useful for asynchronous settings.

Playback. To further support asynchronous use, we introduce a playback feature that allows teacher to review students' activities for each time step within a self-defined time span. This can be accessed by clicking on the playback button on the notebook toolbar, and the real-time attribute for all features will be automatically disabled. When entering the playback mode, a time slider will appear as the control panel. Teachers can move the pointer to any moment of their interest, and all the dashboard features will configured to display everything that was available at that moment. The playback feature allows student activity during a lesson or exercise session to be replayed, providing teachers with a way to revisit and reflect on the lecture or exercise session.

Data Exporting. Comprehensive interaction data can be exported by clicking on the "EXPORT" button and specifying a time range. By exporting this data, instructors can perform their own analyses outside of Jupyter Analytics, for example, performing a temporal analysis of students' programming behaviors.

3.2 The Telemetry Extension

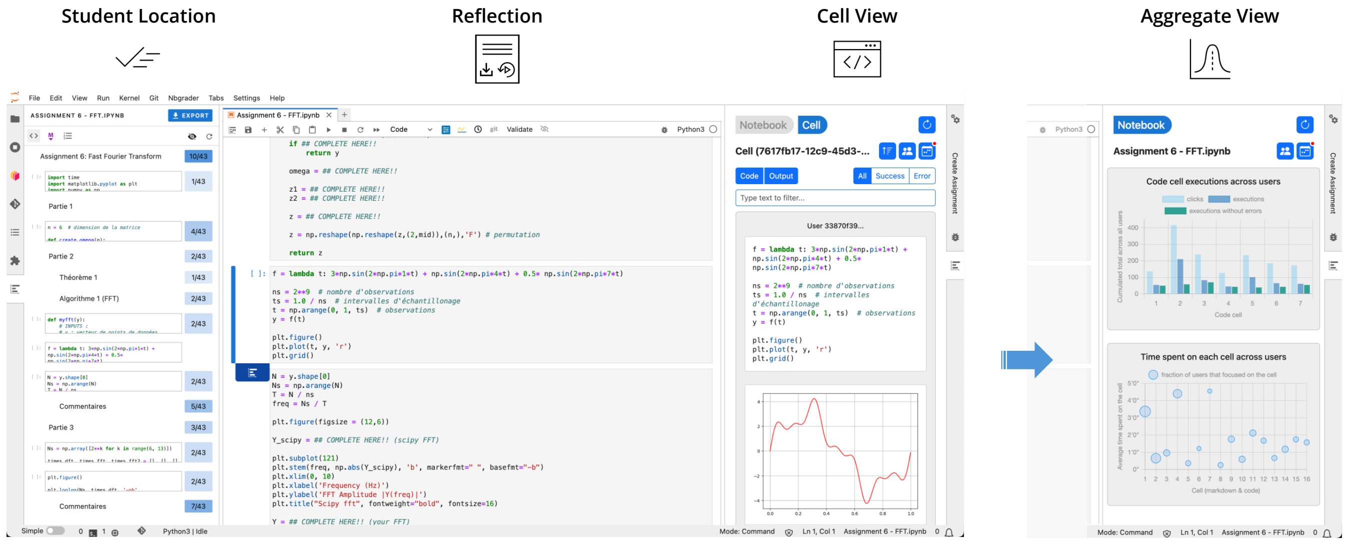
The Telemetry Extension records user actions in Jupyter notebooks and sends the fine-grained data to a cloud-based backend.

3.2.1 Respecting Privacy.

Anonymity. User IDs (if any) collected by Telemetry are automatically encrypted before being processed. No identifiable information is included in the database.

Opt In/Out. The first time students open a tagged notebook, a pop-up window requests permission to collect their data in an anonymous manner. If students choose to opt out, the Telemetry Extension deactivates and no data is sent to the server. Even after students have opted in to data collection, they are still able to opt out by toggling a switch in the Settings.

3.2.2 Collected Data. Data are sent to the backend whenever a student interacts with a notebook. Interactions include clicking on the notebook and cells, creating or removing cells, as well as executing cells – no matter whether it is a code cell or a markdown cell. For code cell execution, its output (including the error message),



```
    if ## COMPLETE HERE!!
        return y

    omega = ## COMPLETE HERE!!

    z1 = ## COMPLETE HERE!!
    z2 = ## COMPLETE HERE!!

    z = ## COMPLETE HERE!!

    z = np.reshape(np.reshape(z,(2,mid)),(n,),'F') # permutation

    return z
```

```
f = lambda t: 3*np.sin(2*np.pi*1*t) + np.sin(2*np.pi*4*t) + 0.5* np.sin(2*np.pi*7*t)

ns = 2**9  # nombre d'observations
ts = 1.0 / ns  # intervalles d'échantillonage
t = np.arange(0, 1, ts)  # observations
y = f(t)

plt.figure()
plt.plot(t, y, 'r')
plt.grid()
```

```
N = y.shape[0]
Ns = np.arange(N)
T = N / ns
freq = Ns / T

plt.figure(figsize = (12,6))

Y_scipy = ## COMPLETE HERE!! (scipy FFT)

plt.subplot(121)
plt.stem(freq, np.abs(Y_scipy), 'b', markerfmt=" ", basefmt="-b")
plt.xlim(0, 10)
plt.xlabel('Frequency (Hz)')
plt.ylabel('FFT Amplitude |Y(freq)|')
plt.title("Scipy fft", fontweight="bold", fontsize=16)

Y = ## COMPLETE HERE!! (your FFT)
```

```
f = lambda t: 3*np.sin(2*np.pi*1*t) +
np.sin(2*np.pi*4*t) + 0.5*
np.sin(2*np.pi*7*t)

ns = 2**9  # nombre d'observations
ts = 1.0 / ns  # intervalles
d'échantillonage
t = np.arange(0, 1, ts)  # observations
y = f(t)

plt.figure()
plt.plot(t, y, 'r')
plt.grid()
```

Figure 2: Overview of the dashboard interface

status, timestamp, and the programming language of the kernel are logged.

3.3 The Server

Following the client-server architecture, the server manages requests from two types of clients (i.e., Dashboard and Telemetry). It handles tasks like request processing, data storage, access control, and pre-processing raw data from Telemetry to prepare structured data for Dashboard, in conjunction with other backend components as introduced in the following sections. The backend can be deployed in both cloud and self-managed servers, and users can accordingly configure the backend endpoint URLs for the extensions.

3.3.1 Application and Database. The backend application uses the Flask framework with a PostgreSQL database. The Flask App is in charge of communicating with the database to retrieve or store the data, which are originally collected from students' interactions and pre-processed by the application.

3.3.2 Between-Extension Communication. In order to enable bidirectional real-time communication between students and teachers, Flask-SocketIO is integrated into our Flask application. On the server side, it handles connections and messages from both Dashboard and Telemetry extensions. Whenever new student data is available, Flask-SocketIO is used to send update messages to any connected Dashboard extension. This is the part of the system that enables real-time display of student interaction data.

3.4 Designing Jupyter Analytics

As outlined in Figure 3, the design process of Jupyter Analytics involved three phases.

- Phase 1: we designed a web-based mock-up, which was used in task-based interviews with 8 TAs; the outcome of this qualitative study was obtaining a better understanding of the context and surfacing TAs' needs, while also receiving insightful feedback on the mock-up itself.
- Phase 2: we implemented Jupyter Analytics in a university physics course, in which we went through a iterative co-design process with 7 TAs; more details and use cases are reported in section 4.2.
- Phase 3: we conducted another classroom study in a university mathematics course through a longer iterative co-design process with one instructor and 5 TAs; more details and use cases are reported in section 4.3.

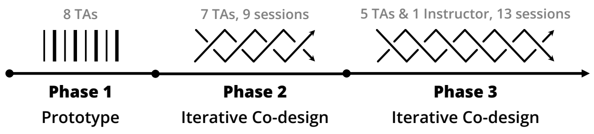


Figure 3: Design process of Jupyter Analytics

4 Implementation and Evaluation

4.1 Context: University Exercise Sessions

The university at the center of our research is a research institute specializing in STEM disciplines. Exercise sessions (lab sessions) are part of most courses, complementing lectures with practical activities. In general, these sessions involve students working individually or collectively on problems with access to instructors or TAs for assistance. This was the context in which we introduced and evaluated Jupyter Analytics.

4.2 Classroom Study 1: Supporting Classroom Orchestration

We conducted a pilot study over nine exercise sessions in an undergraduate physics course. All exercises were Jupyter notebooks, and the number of students attending varied between 40 and 60 per session. In the fifth session, Jupyter Analytics was installed for the class.

We observed these sessions and took field notes. The TA team in this class comprised 6 PhD students and 1 Msc student (2 females, 5 males); each week one of the PhD students was responsible for correcting the exercise. We triangulated the observation notes through talking or exchanging emails with the TAs, as well as checking the log data we collected from their interactions with the dashboard.

Normally in the middle of these sessions, TAs would stop the class to correct the first part, ensuring most students were making steady progress and able to work on the second part on they own. These routines were usually delivered via announcements, leading most students to pause their work and follow the instructions.

After Jupyter Analytics was installed, TAs used the Student Location Dashboard to help them decide when to stop the class. Because it visually displayed the real-time distribution of class activity on the notebook, this provided objective indicators to complement TAs' subjective judgment on when to start correcting the exercise. Based on our observations, TAs would check the dashboard over and over again to identify the moment when most students had reached an important milestone, and would then stop the class.

Similar behaviors of bringing the class together were captured by the events logged when TAs interacted with the dashboard, as shown in Figure 4: (a) TAs' interactions over the whole exercise session; (b) TAs' interactions when making an announcement. Typically, when TAs speculated that students were making common mistakes, they would go to the dashboard, check students' codes and error messages to test this hypothesis. As a TA described after one exercise session: *"it was that one other TA had realized it and was asking around with the other TAs, I had noticed the mistake once too and then I went to check the code cells, but we already had a pretty big suspicion at the point we were checking it. Meanwhile [the head TA] said that he had seen it too, so then we decided together to make an announcement."*

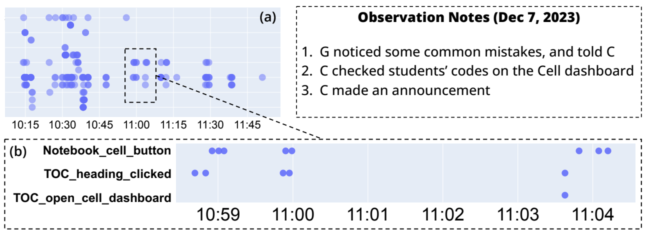


Figure 4: TAs' interactions with the dashboard

The act of bringing students together to discuss important topics or reflect is an important part of "classroom orchestration" [10]. Effective classroom orchestration involves knowing when to bring students together, and has been shown to have positive impacts on learning [9]. Additionally, effective classroom orchestration (i.e., timed explicit instruction) has been identified as a way to support non-majors in STEM courses [23]. This use case helps demonstrate that by providing improved visibility into student behavior, Jupyter Analytics can support more effective classroom orchestration in computational STEM courses.

4.3 Classroom Study 2: Identifying Collaborative Learning Patterns

The second study was implemented in a bachelor-level mathematics course with a total of 71 students who consented to have their data collected. There were thirteen exercise sessions over the semester, six of which were graded. All exercises were Jupyter notebooks, and needed to be completed in groups of three or four. Jupyter analytics was installed in the second week of the course.

In this course, we adopted an iterative co-design process with the teaching team. In addition to observing exercise sessions and taking field notes, we also attended the weekly teaching-team meetings before each exercise session. In this way, we were able to follow their lesson plans, and proactively discuss with the instructor and 5 TAs (3 females, 2 males; 3 PhD students, 2 MSc students) regarding the usage of our tools, as well as their perceived needs in the same context. This made it possible to use an agile development approach to rapidly develop new features based on their needs and integrate them into Jupyter Analytics during the semester.

Each TA in the course was responsible for supporting 4-5 groups. In one of the weekly TA meetings, TAs requested the ability to filter data by the groups they were responsible for to reduce unnecessary noise in the dashboard. In response, we implemented a group filtering feature midway through the semester. Based on TAs' feedback, this feature helped them concentrate their efforts on monitoring and assisting their groups, while minimizing the distractions and cognitive load brought by extra information about other groups.

After implementing the ability to filter by groups, TAs used this feature to identify a new issue in the course. By visualizing individual student activity within each group, they noticed different approaches to collaborating on the exercises. TAs had concerns about one of these approaches—the parallel approach—which involved each student in a group working on a different problem in the notebook at the same time. This was believed to be problematic because of the design of the notebooks, which consisted of problems that were designed to be worked on sequentially. Not only did this approach reduce opportunities for collaborative learning, it also led to situations where those responsible for later parts were unable to make progress because they relied on results from prior problems, leaving them with nothing to do while waiting for their teammates finished the earlier sections.

Different collaboration strategies were identified from students' interaction data collected by Telemetry. Three groups – A-4, B-3, C-1 – were selected based on a preliminary analysis on their activities. Figure 5 shows timelines of their programming processes throughout the session, illustrating their different collaboration patterns. Observation notes indicated that B-3 worked in pairs, A-4 collaborated using a single notebook, and C-1 used multiple notebooks but worked closely together, while one student moved around to assist her peers and seek help as needed.

These observations were validated through follow-up interviews with the groups. For A-4, typically they would first review the assignment together to understand the tasks at hand. Then, one was designated to write the code, while *"the other two usually look at the lesson and the PDFs, etc."* They adopted this strategy because they realized the exercise followed a serial format. As one student noted, "we can't split the code. It doesn't make sense".

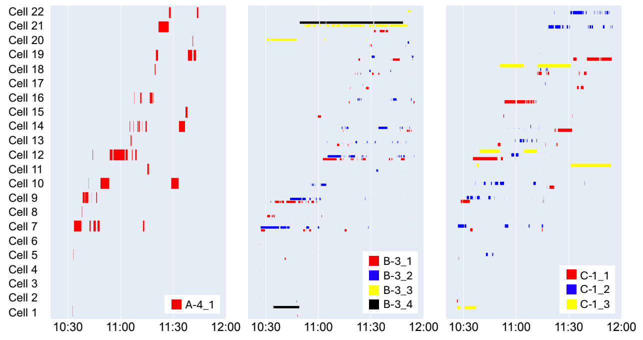


Figure 5: Students' different collaboration strategies

In contrast, B-3 and C-1 preferred splitting the work. C-1 was divided into two pairs with different sets of skills. One pair included a student skilled in programming to handle heavy coding tasks, working alongside another student. And the other two students *"usually do commentaries and graphs and stuff like that"*. B-3 also worked in two pairs, but without a precise plan. One student referred to this strategy as *"an improvisation"*, explaining, *"we just went with whatever was happening at the moment"*. Since they worked separately, there was no collaboration between pairs, unlike C-1 in which one person coordinated the group work. This situation highlighted the concern that TAs had, as described by the student: "we took more time on exercise five because we had some problems. They continued going on, we stayed on."

In addressing concerns about group dynamics, TAs appreciated the visualizations for providing insight into various collaboration patterns. In response to identifying unproductive strategies, one TA suggested a solution: instead of releasing the notebook at the beginning, they would first provide a non-interactive version of the notebook and give students 10 minutes to plan their collaboration. This intervention was carried out for the next two graded exercises, and led to improved collaboration strategies.

This use case helps demonstrate that the data collected by Jupyter Analytics made it possible to visualize and analyze complex collaborative behaviors. Furthermore, TAs were able to design an effective intervention in response, and to evaluate the intervention's effectiveness by consulting the data. While this did not require making any changes to the data collected by the system, it did require the development of new analysis methods and visualizations. This points to the importance of the agile, iterative co-design process in integrating analytics tools into computational STEM courses.

5 Discussion

Taken together, the two use cases described above indicate that Jupyter Analytics helps to fill an important gap in computational STEM courses where Jupyter notebooks are used. By capturing, processing, and visualizing student interaction data, Jupyter Analytics provides much-needed visibility into student activity. Our goal in building Jupyter Analytics was to provide teachers with increased visibility so that they could be more responsive in their teaching; and in each of the use cases, we described how teachers used the novel sources of feedback provided by Jupyter Analytics to respond to students' needs.

Each of the use cases took place in a different domain (Math vs. Physics), with different activities and learning goals. Nevertheless, Jupyter Analytics provided useful information that supported instructors in productively modifying their teaching practices. We interpret this as providing evidence that increased visibility into student activity in Jupyter notebooks is valuable and needed, and that our approach of building a holistic analytics toolkit that is directly integrated into Jupyter provides benefits over other systems.

5.1 Lessons Learned

Overall, we received positive feedback on the Jupyter Analytics toolkit. TAs found that it helped address their needs with minimal effort, which led to TAs' continuous usage and proactive ideation throughout the iterative co-design process. We also found that our co-design approach, which involved attending TA meetings during the semester to debrief about tool use and identify emerging needs, helped with tool implementation, adoption, and use.

We also identified obstacles to overcome. For example, while our iterative design process allowed us to rapidly implement features and improvements that could be tested during the semester, it also decreased system stability and introduced bugs. Additionally, our rapid development approach was sometimes at odds with the pace of the institution, leading to delays in deployment due to the institutional decision-making process. These lessons highlighted the importance of careful planning, thorough testing, and fostering and promoting cross-organizational collaboration [25], given that there is usually no chance to restart or redo an educational session.

In the context of exercise sessions, we found that as TAs' workload (e.g., answering questions) increased, their use of the tool decreased. This suggests that there is still work to be done in integrating the toolkit into their practices, such that the toolkit is able to seamlessly support teaching practices regardless of the workload.

5.2 Conclusion

In this paper, we introduced Jupyter Analytics, an end-to-end toolkit for teachers to collect, analyze, and visualize distributed student activity in Jupyter notebooks. Two JupyterLab extensions were developed and integrated through a cloud-based backend: a Telemetry Extension for data collection on the student side and a Dashboard Extension for data presentation on the teacher side. The system was implemented and tested in two university STEM courses through an iterative co-design process with a total of twelve teaching assistants. The results from these use cases showed that Jupyter Analytics provided improved visibility into student activity, and that instructors were able to use this information to make their teaching more responsive to students' behaviors and needs. This supported our hypothesis that building a holistic analytics toolkit that was directly integrated into Jupyter would provide benefits over other systems in computational STEM courses.

Acknowledgments

The project was funded by Swiss National Science Foundation (SNSF) under grant number 407740_187534.

References

[1] Abdulmalek Al-Gahmi, Yong Zhang, and Hugo Valle. 2022. Jupyter in the Classroom: An Experience Report. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education - Volume 1* (Providence, RI, USA) *(SIGCSE 2022)*. Association for Computing Machinery, New York, NY, USA, 425–431. https://doi.org/10.1145/3478431.3499379

[2] Kai Arakawa, Qiang Hao, Wesley Deneke, Indie Cowan, Steven Wolfman, and Abigayle Peterson. 2022. Early Identification of Student Struggles at the Topic Level Using Context-Agnostic Features. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education - Volume 1* (Providence, RI, USA) *(SIGCSE 2022)*. Association for Computing Machinery, New York, NY, USA, 147–153. https://doi.org/10.1145/3478431.3499298

[3] Golnaz Arastoopour Irgens, Sugat Dabholkar, Connor Bain, Philip Woods, Kevin Hall, Hillary Swanson, Michael Horn, and Uri Wilensky. 2020. Modeling and measuring high school students' computational thinking practices in science. *Journal of Science Education and Technology* 29 (2020), 137–161.

[4] Elham Beheshti. 2017. Computational thinking in practice: How STEM professionals use CT in their work. In *American Education Research Association Annual Meeting 2017*. San Antonio, TX.

[5] Sarah D. Castle. 2023. Leveraging Computational Science Students' Coding Strengths for Mathematics Learning. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (Toronto ON, Canada) *(SIGCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 263–269. https://doi.org/10.1145/3545945.3569861

[6] Evann Courdier and Aymeric Dieuleveut. 2024. *Notebook Progress Tracker*. Retrieved July 4, 2024 from https://courdier.pythonanywhere.com

[7] Jessica Q. Dawson, Meghan Allen, Alice Campbell, and Anasazi Valair. 2018. Designing an Introductory Programming Course to Improve Non-Majors' Experiences. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (Baltimore, Maryland, USA) *(SIGCSE '18)*. Association for Computing Machinery, New York, NY, USA, 26–31. https://doi.org/10.1145/3159450.3159548

[8] Alessio De Santo, Juan Carlos Farah, Marc Lafuente Martínez, Arielle Moro, Kristoffer Bergram, Aditya Kumar Purohit, Pascal Felber, Denis Gillet, and Adrian Holzer. 2022. Promoting Computational Thinking Skills in Non-Computer-Science Students: Gamifying Computational Notebooks to Increase Student Engagement. *IEEE Transactions on Learning Technologies* 15, 3 (2022), 392–405. https://doi.org/10.1109/TLT.2022.3180588

[9] Pierre Dillenbourg and Patrick Jermann. 2010. Technology for classroom orchestration. In *New science of learning: Cognition, computers and collaboration in education*. Springer, New York, NY, 525–552.

[10] Pierre Dillenbourg, Luis P Prieto, and Jennifer K Olsen. 2018. Classroom orchestration. In *International handbook of the learning sciences*. Routledge, New York, NY, 180–190.

[11] Barbara J. Ericson, Hisamitsu Maeda, and Paramveer S. Dhillon. 2022. Detecting Struggling Students from Interactive Ebook Data: A Case Study Using CSAwesome. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education - Volume 1* (Providence, RI, USA) *(SIGCSE 2022)*. Association for Computing Machinery, New York, NY, USA, 418–424. https://doi.org/10.1145/3478431.3499354

[12] Louis Faucon, Jennifer K Olsen, Stian Haklev, and Pierre Dillenbourg. 2020. Real-Time Prediction of Students' Activity Progress and Completion Rates. *Journal of Learning Analytics* 7, 2 (2020), 18–44.

[13] Robin Flatland, Darren Lim, James Matthews, and Scott Vandenberg. 2015. Supporting CS10K: A New Computer Science Methods Course for Mathematics Education Students. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (Kansas City, Missouri, USA) *(SIGCSE '15)*. Association for Computing Machinery, New York, NY, USA, 302–307. https://doi.org/10.1145/2676723.2677274

[14] A. Gajdoš, J. Hanč, and M. Hančová. 2022. Interactive Jupyter Notebooks with SageMath in Number Theory, Algebra, Calculus, and Numerical Methods. In *2022 20th International Conference on Emerging eLearning Technologies and Applications (ICETA)*. IEEE, Stary Smokovec, Slovakia, 178–183. https://doi.org/10.1109/ICETA57911.2022.9974868

[15] Brian E Granger and Fernando Pérez. 2021. Jupyter: Thinking and storytelling with code and data. *Computing in Science & Engineering* 23, 2 (2021), 7–14.

[16] Stian Haklev, Louis Pierre Faucon, Thanasis Hadzilacos, and Pierre Dillenbourg. 2017. FROG: rapid prototyping of collaborative learning scenarios. In *Data Driven Approaches in Digital Education: 12th European Conference on Technology Enhanced Learning, EC-TEL 2017, Tallinn, Estonia, September 12–15, 2017, Proceedings*. Springer, Tallinn, Estonia.

[17] Jessica B. Hamrick. 2016. Creating and Grading IPython/Jupyter Notebook Assignments with NbGrader. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (Memphis, Tennessee, USA) *(SIGCSE '16)*. Association for Computing Machinery, New York, NY, USA, 242. https://doi.org/10.1145/2839509.2850507

[18] Meng Han, Zhigang Li, Jing He, and Xin Tian. 2019. What are the Non-majors Looking for in CS Classes?. In *2019 IEEE Frontiers in Education Conference (FIE)*. IEEE, Covington, KY, USA, 1–5. https://doi.org/10.1109/FIE43999.2019.9028448

[19] Emma Hogan, Ruoxuan Li, and Adalbert Gerald Soosai Raj. 2023. CS0 vs. CS1: Understanding Fears and Confidence amongst Non-majors in Introductory CS Courses. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (Toronto ON, Canada) *(SIGCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 25–31. https://doi.org/10.1145/3545945.3569865

[20] Hassan Khosravi and Kendra M.L. Cooper. 2017. Using Learning Analytics to Investigate Patterns of Performance and Engagement in Large Classes. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (Seattle, Washington, USA) *(SIGCSE '17)*. Association for Computing Machinery, New York, NY, USA, 309–314. https://doi.org/10.1145/3017680.3017711

[21] Donald Ervin Knuth. 1984. Literate programming. *The computer journal* 27, 2 (1984), 97–111.

[22] Rubin Landau. 2006. Computational physics: A better model for physics education? *Computing in science & engineering* 8, 5 (2006), 22–30.

[23] Kathy A. Mills, Jen Cope, Laura Scholes, and Luke Rowe. 0. Coding and Computational Thinking Across the Curriculum: A Review of Educational Outcomes. *Review of Educational Research* 0, 0 (0), 00346543241241327. https://doi.org/10.3102/00346543241241327 arXiv:https://doi.org/10.3102/00346543241241327

[24] Jonathan P. Munson and Joshua P. Zitovsky. 2018. Models for Early Identification of Struggling Novice Programmers. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (Baltimore, Maryland, USA) *(SIGCSE '18)*. Association for Computing Machinery, New York, NY, USA, 699–704. https://doi.org/10.1145/3159450.3159476

[25] Lucía Márquez, Valeria Henríquez, Henrique Chevreux, Eliana Scheihing, and Julio Guerra. 2024. Adoption of learning analytics in higher education institutions: A systematic literature review. *British Journal of Educational Technology* 55, 2 (2024), 439–459. https://doi.org/10.1111/bjet.13385 arXiv:https://bera-journals.onlinelibrary.wiley.com/doi/pdf/10.1111/bjet.13385

[26] University of Michigan. 2024. *JupyterLab Pioneer*. University of Michigan. Retrieved July 4, 2024 from https://jupyterlab-pioneer.readthedocs.io/en/latest

[27] Jeffrey M Perkel. 2021. Ten computer codes that transformed science. *Nature* 589, 7842 (2021), 344–349.

[28] Amir Rubinstein and Benny Chor. 2014. Computational thinking in life science education. *PLoS computational biology* 10, 11 (2014), e1003897.

[29] NGSS Lead States. 2013. *Next generation science standards: For states, by states*. National Academies Press, Washington, DC. https://doi.org/10.17226/18290

[30] Jupyter Development Team. 2022. *Jupyter Events*. Project Jupyter. Retrieved July 4, 2024 from https://jupyter-events.readthedocs.io/en/latest

[31] Jupyter Development Team. 2024. *Jupyter Telemetry*. Project Jupyter. Retrieved July 4, 2024 from https://jupyter-telemetry.readthedocs.io/en/latest

[32] Camilo Vieira, Alejandra J Magana, R Edwin García, Aniruddha Jana, and Matthew Krafcik. 2018. Integrating computational science tools into a thermodynamics course. *Journal of Science Education and Technology* 27 (2018), 322–333.

[33] Austin L. Zuckerman and Ashley L. Juavinett. 2024. When Coding Meets Biology: The Tension Between Access and Authenticity in a Contextualized Coding Class. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1* (Portland, OR, USA) *(SIGCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 1491–1497. https://doi.org/10.1145/3626252.3630966