A Systematic Literature Review of Software Engineering Research on Jupyter Notebook

Md Saeed Siddik[a], Hao Li[b], Cor-Paul Bezemer[a]

[a] *University of Alberta, Edmonton, Canada*
[b] *Queen's University, Kingston, Canada*

Abstract

Context: Jupyter Notebook has emerged as a versatile tool that transforms how researchers, developers, and data scientists conduct and communicate their work. As the adoption of Jupyter notebooks continues to rise, so does the interest from the software engineering research community in improving the software engineering practices for Jupyter notebooks.

Objective: The purpose of this study is to analyze trends, gaps, and methodologies used in software engineering research on Jupyter notebooks.

Method: We selected 146 relevant publications from the DBLP Computer Science Bibliography up to the end of 2024, following established systematic literature review guidelines. We explored publication trends, categorized them based on software engineering topics, and reported findings based on those topics.

Results: The most popular venues for publishing software engineering research on Jupyter notebooks are related to human-computer interaction instead of traditional software engineering venues. Researchers have addressed a wide range of software engineering topics on notebooks, such as code reuse, readability, and execution environment. Although reusability is one of the research topics for Jupyter notebooks, only 64 of the 146 studies can be reused based on their provided URLs. Additionally, most replication packages are not hosted on permanent repositories for long-term availability and adherence to open science principles.

Conclusion: Solutions specific to notebooks for software engineering issues, including testing, refactoring, and documentation, are underexplored. Future research opportunities exist in automatic testing frameworks, refactoring clones between notebooks, and generating group documentation for coherent code cells.

Keywords:
Jupyter Notebook, Software Engineering, Data Analysis

1. Introduction

In recent years, Jupyter Notebook has emerged as a powerful and versatile computing platform for data science, scientific research, and software development [112, 135]. Its user-friendly interface has revolutionized the way researchers, developers, and data scientists conduct and communicate their work. Jupyter Notebook provides a unique platform for integrating code, visualizations, and narrative text, fostering reproducibility, and facilitating collaborative exploration of data and algorithms [171]. Seamlessly incorporating code, data, and output into a single file makes the Jupyter Notebook ideal for data analysis, scientific computing, and machine learning (ML) tasks. Moreover, its support for multiple programming languages, including Python, R, and Julia, makes it versatile and widely accessible to a diverse community of researchers, data scientists, and educators. However, despite their popularity, notebooks have been associated with several challenges, such as low reproducibility rates, problems with their execution environments, and excessive code duplication [46].

As Jupyter notebooks are often created and maintained by users without a software engineering background [150], there has been a growing interest from software engineering researchers to help notebook users integrate best software engineering practices into their notebooks. As a result, there has been a growing body of software engineering research that targets Jupyter Notebook. In this paper, we conduct a systematic literature review (SLR) of such research and the problems addressed by it.

We systematically searched academic publications to identify relevant studies related to software engineering research on Jupyter Notebook. We followed Kitchenham's guidelines [71] for our SLR to ensure rigor and impartiality. We comprehensively searched all the papers on Jupyter Notebook indexed in the DBLP database [80] and published until 2024. Then, we filtered the papers focused on software engineering research on Jupyter notebooks. The review process involved several steps, including screening the titles and abstracts of the identified studies, assessing their relevance based on the inclusion criteria, and extracting relevant data from the selected studies. We finally selected 146 papers as our primary studies. During our SLR, we focused on the following two research questions (RQs):

- **RQ1: How much software engineering research on Jupyter Notebook has been published?** We analyzed current trends in the publication of software engineering research on notebooks to help future researchers identify potential venues for their work. The number of publications has gradually increased over the years. Most (78 of 146 studies) software engineering research on notebooks has been published at conferences, indicating a fast-moving field. Our research found that other venues beyond core software engineering conferences published the highest number of studies; for example, the ACM Conference on Human Factors in Computing Systems (CHI) published 15 studies. Conversely, the International Conference on Software Engineering (ICSE) published six studies on software engineering research on notebooks. According to our study, a significant number of the articles we examined, specifically 31 out of 146 (21%), had affiliations with industry institutions such as Microsoft Research (12 studies) or IBM Research Lab (6 studies). This highlights the practical importance and real-world applications of the research findings. We identified 67 studies that provided URLs to replication packages. Of these, 64 URLs were deemed reusable, while three were not, due to missing source code or unclear execution instructions. Most replication packages (54 studies) are hosted in GitHub repositories, which is against the best practices for open science in software engineering [98] as such repositories can disappear over time. Only 13 replication packages were hosted in a permanent repository, such as Zenodo or Figshare.

- **RQ2: Which software engineering topics are being studied in software engineering research on Jupyter Notebook?** We categorized the primary studies into 11 software engineering topics: code reuse and provenance, managing the computational environment and workflow, readability of notebooks, datasets of notebooks, documentation of notebooks, testing and debugging, visualization in notebooks, best practices, cell execution order, notebook code generation, and supporting other programming paradigms. Our findings indicate that code reuse and provenance is the most extensively researched topic related to Jupyter Notebook, with 32 studies focusing on it. This is followed by managing computational environment and workflow with 29 studies, and readability of notebooks with 11 studies. These results suggest that researchers are primarily concerned with code cells in notebooks,

which often require significant human effort to understand. We also identified 19 publicly available Jupyter Notebook datasets used in the studies, most of which were sourced from Kaggle and GitHub repositories.

Our systematic literature review reveals that software engineering research on Jupyter notebooks is an active research direction with diverse publication venues beyond core software engineering. Our research indicates that studies on notebooks are highly focused on dealing with code cells in notebooks, which require considerable human effort to understand and manage. Although there are techniques and tools for software engineering in Jupyter notebooks, such as automated refactoring tools, testing frameworks, and strong documentation practices, they are often insufficient. These present a chance for improvement, since better tools can enhance the usability and quality of projects in Jupyter notebooks. Future research can focus on improving these tools to support collaboration and reproducibility in scientific computing and data analysis.

The remainder of this paper is organized as follows. Section 2 gives an overview of the Jupyter Notebook platform. Section 3 describes our methodology. Sections 4 and 5 present the results of the research questions. Section 6 presents the future research directions derived from this SLR. Section 7 identifies the threats to validity and Section 8 concludes the paper.

2. The Jupyter Notebook Platform

Jupyter Notebook allows users to edit and execute code inside a web-based interface. Unlike traditional IDEs like PyCharm, Jupyter Notebook provides a cell-based interface that seamlessly integrates code with output and allows individual code cell execution in any order. In this section, we explore the platform's features and its utilization in various fields.

2.1. Unique Features of Notebooks

Jupyter Notebook enables users to manage code, documentation, and output into a single document. Users can execute any code cell from anywhere in the notebook; they are not bound to follow the code cell's order sequence. Two main features distinguish notebooks from a traditional source code file: the cell-based structure and execution order.

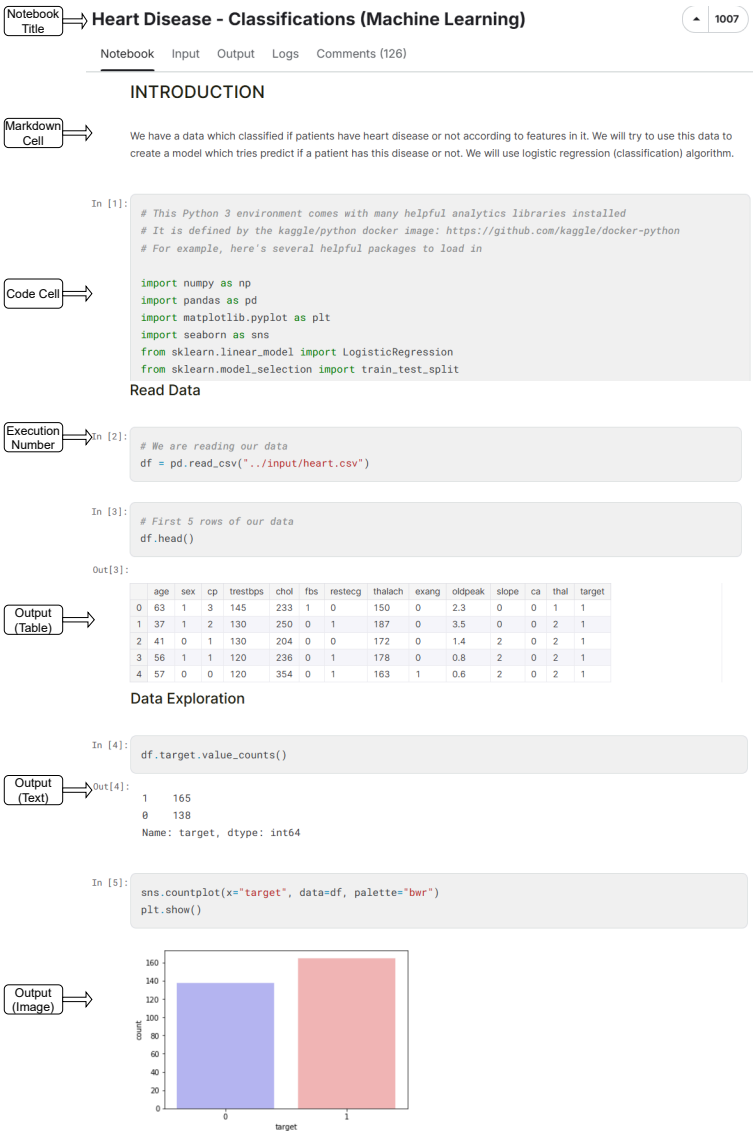

Figure 1: An example of Jupyter Notebook's structure

2.1.1. Cell-Based Structure

A Jupyter Notebook is composed of several types of cells: code cells for executing source code, Markdown cells for documentation, and raw cells that contain non-rendered contents. Furthermore, every code cell has an execution

number, which records the order of cell execution, and an output cell that displays the result of the corresponding code cell. An illustrative example of this structure is shown in Figure 1, sourced from a publicly accessible notebook on Kaggle.[1] Below is a detailed exploration of each cell type:

Code Cell: Code cells are the fundamental components of a Jupyter Notebook, where the source code (Python or other supported languages depending on the kernel) is written and executed. After execution, each code cell shows the results immediately below the cell. A notebook enables users to edit and execute each code cell independently, allowing them the flexibility to run cells in any order they prefer. This means that programmers are not obligated to execute their code sequentially from top to bottom or limited to making changes only at the end of the notebook. Instead, they can selectively modify and execute specific code cells at any point, fostering a more dynamic and interactive coding experience [21, 46].

Markdown Cell: Jupyter Notebook's Markdown cells provide the narrative to a notebook. They use plain text with Markdown syntax to create formatted text and media, including headings, lists, links, images, and formatted text to explain and guide the reader through the notebook's purpose, underlying logic, and methodology [115, 165]. This aspect of notebooks is crucial for making them comprehensive and user-friendly, especially when sharing with peers who may not be familiar with the code. Comments in traditional programming are quite similar to this Markdown cell. However, a Markdown cell generally contains more information than traditional textual comments [91, 100].

Raw NBConvert Cell: A raw cell[2] in a Jupyter notebook is a cell whose contents are included without modification when converted using nbconvert.[3] Unlike other cell types, raw cells are neither processed nor rendered within the notebook itself and are primarily intended for direct inclusion in exported outputs such as LaTeX documents or HTML files. This feature offers users flexibility in formatting and exporting notebook content for various publication or documentation purposes.

Output Cell: The output of a code cell displays the results generated by the execution of the corresponding code cell. These can include textual

[1] https://www.kaggle.com/code/cdabakoglu/heart-disease-classifications-machine-learning

[2] https://ipython.org/ipython-doc/3/notebook/nbformat.html#raw-nbconvert-cells

[3] https://ipython.org/ipython-doc/3/notebook/nbconvert.html#nbconvert

data, tables, error messages, or visualizations created using libraries such as Matplotlib,[4] Seaborn,[5] or Plotly.[6] The outputs are automatically updated whenever the associated code cell is executed. This capability makes Jupyter notebooks especially valuable for data science and research, as outputs can immediately visualize and communicate results alongside code, enhancing understanding and analysis [124, 118].

2.1.2. Execution Order

The execution order in a Jupyter notebook refers to the sequence in which the code cells are executed. Unlike traditional scripts, code cells in a notebook can be executed non-linearly, depending on the user's needs and exploration path. The execution sequence is indicated by a number in brackets next to each code cell (e.g., `In [1]:`), which shows the most recent execution order (see Figure 1 for the execution number). Jupyter Notebook allows users to execute any code cell multiple times or out-of-order. This flexibility allows notebook users to test specific parts of their code, troubleshoot issues, or make adjustments without rerunning the entire notebook [140, 116, 55]. This type of execution in notebooks also helps the iterative nature of exploratory data analysis [33]. However, out-of-order execution can also cause problems, such as failure to load dependencies if they are not properly managed in the code [178]. Researchers found that about 36% of notebooks on GitHub did not execute in linear order [99].

2.2. Applications of Jupyter Notebooks

Jupyter Notebook is a popular tool for data analysis and data science, often used for interactive exploration during experiments [112, 67, 190]. It is considered an effective tool for both experienced and novice data scientists [36, 18]. Its flexibility in code execution facilitates short feedback cycles, which are essential for the iterative data analysis process [33]. While notebooks have been used with static visualizations, interactive visualizations can also be embedded and supported, as well as advanced visual analysis [106, 184]. These allow users to quickly try out different data analysis options and observe the results with minimal effort. Notebooks are also used

[4] `https://pypi.org/project/matplotlib/`
[5] `https://pypi.org/project/seaborn/`
[6] `https://pypi.org/project/plotly/`

for generating personalized data narrations over a given dataset for interactive data exploration [19] and sharing data science work through presentation slides [202, 176, 81, 107].

Jupyter notebooks are widely used in the education sector and in academic teaching [6, 157, 113, 17]. They provide an intuitive and user-friendly platform for teaching programming concepts and designing course structures [129, 65]. Educators use Jupyter notebooks to create coding tutorials, assignments, and lecture materials that foster active learning and engagement among students [6, 4]. Research has shown that the use of Jupyter notebooks in educational settings significantly enhances the student's understanding of course materials [9]. Additionally, studies demonstrate that Jupyter notebooks are particularly convenient for teaching data science, as they eliminate the need for students to install software locally or use specific machines [163]. Despite their popularity for managing coding-based assignments, automatic assignment grading in Jupyter notebooks presents challenges. The standard automated assessment grading tool for Jupyter notebooks, such as *nbgrader* [52]) has limitations; for example, it cannot automatically submit scores to a Learning Management System (LMS) like Canvas and does not provide automated feedback to students [95]. To address the automatic score submission, Malone et al. developed an automated assignment grading system for Jupyter notebooks that focuses on gamified cybersecurity exercises [94].

Jupyter Notebook is popular for modeling and analyzing scientific tasks and performing experimental simulations [11, 142, 110, 164]. For example, Tran et al. developed a library for Jupyter Notebook that can set up and control additive manufacturing machines such as 3D printers [161]. Wilsdorf et al. presented a Jupyter Notebook extension that lends support to modelers by automatically specifying and running suitable simulation experiments [192]. Jupyter Notebook is also popular for analyzing GIS data to deal with big geospatial data [200, 199, 51]. For example, Valentine et al. implemented a data discovery studio for geoscience data discovery and exploration using Jupyter Notebook [162]. Yin et al. presented a cyberGIS framework to achieve data-intensive and scalable geospatial analytics using the Jupyter Notebook [200]. Jupyter notebooks are also used as a toolbox for interactive surface water mapping [108], geospatial environmental data processing [158], and astrophysical data-proximate analysis [62].

Notebooks are often used in the healthcare sector to analyze complex medical data [48, 5, 8]. For example, Hao et al. from IBM Research devel-

oped customized healthcare data analysis pipelines in Jupyter Notebook to support healthcare users on user-friendly and reusable data analytics [53]. Almugbel et al. developed a tool that allows users to easily distribute their biomedical data analysis through notebooks uploaded to a GitHub repository or a private server [5]. The study provided four different Jupyter notebooks to infer differential gene expression, analyze cross-platform datasets, and process RNA sequence data. Biomedical researchers prefer notebooks to document and share their research with their community [5, 8]. For example, researchers used Jupyter Notebook as a co-design tool that combines static illustrations and interactive ML model explanations to predict health risk in diabetes care [8]. Furthermore, Llaunet et al. developed and shared a solution with centralized Jupyter Notebook code to support various medical applications such as medical image processing [79]. Beyond the above sectors, people use Jupyter Notebook in different areas. For example, journalists and media organizations use notebooks to analyze journalism content and visualize trends and patterns of newspaper data [148, 153].

3. Research Methodology

For our systematic literature review of software engineering research on Jupyter notebooks, we followed the guidelines proposed by Kitchenham [71]. The planning stage of our work includes two steps: (1) identifying the need for a systematic review and (2) developing the review protocol. In the conducting stage, based on the review protocol from the planning stage, we searched and selected the primary studies. We considered DBLP[7] as the search space, as it covers all the major software engineering journals and conference proceedings published by renowned publishers, e.g., IEEE, ACM, Springer, and Elsevier. Then, we extracted data from DBLP and synthesized the data. We selected 146 papers as our primary studies after completing all the steps in this process. Finally, in the reporting stage, we concluded the systematic review by reporting the collected data and findings. Figure 2 summarizes the steps of our methodology. In this section, we explain each step in more detail.

[7] `https://dblp.org/`

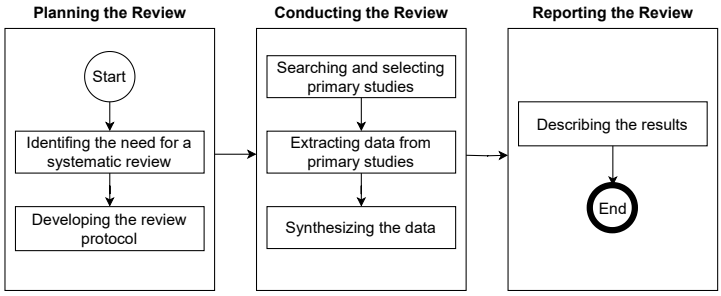

Figure 2: The steps of our SLR on software engineering research on Jupyter Notebook based on Kitchenham's guidelines [71]

3.1. Planning the Review

3.1.1. Identifying the Need for a Systematic Review

The need for a systematic review of software engineering research on Jupyter Notebook arises from the growing popularity and widespread adoption of Jupyter notebooks in various research fields. The widespread use of Jupyter notebooks has highlighted software engineering challenges such as poor code quality, lack of coding standards, and code duplication [1, 179, 73]. These difficulties have sparked notable interest among software engineering researchers in helping notebook users adopt software engineering practices in their work. As a result, more research has been conducted on these issues in Jupyter notebooks. This highlights the need to conduct a systematic literature review of this research and the challenges it addresses, as well as identify what should be studied next.

Table 1: Questions in the data extraction form

Question (Q) - Description (D) - Rationale (R)	**Target RQ**
Q: In which year was the study published? D: The publication year of the corresponding study. R: The publication year helps to indicate the interest in this research topic across the years.	RQ1
Q: What is the publication type? D: The publication type of the study (*Journal*, *Conference*, *Book*, *Technical report*, or *Other*). R: Understanding the distribution of research across these publication types provides insights into the preferred channels for sharing knowledge in the field of software engineering research on Jupyter notebooks.	RQ1
Q: Was the research done with an industry collaborator?	RQ1

Continued on next page

Question (Q) - Description (D) - Rationale (R)	**Target RQ**
D: *Yes* if any of the co-authors have an industry affiliation in the study, *No* otherwise. R: Collaborations between academia and industry indicate that the research is conducted towards solving practical problems and has potential direct applications in industry settings. Understanding the extent of industry collaboration can help in assessing the balance between theoretical and applied research in software engineering research on Jupyter notebooks.	
Q: What type of solution has been proposed? D: The type of proposed solution (*theoretical framework*, *developed solution*, or *empirical analysis*). R: Identifying the type of solution proposed in each study helps to understand the nature of the study: knowing the type of solution provides insights into the focus of the research and its methodology - whether it is more conceptual, application-based, or data-driven.	RQ1
Q: Where are the solutions publicly available? D: The public URL of the provided solution or study to reuse the solution. R: This question seeks the specific URLs or platforms where researchers have made their solutions or study results available. These resources allow for a practical evaluation of how easily other researchers and practitioners can access, test, and potentially replicate the study's findings. It also contributes to understanding the commitment of software engineering for the Jupyter Notebook research community to open science and sharing knowledge and tools.	RQ1
Q: Which specific software engineering problems are addressed by the study? D: The software engineering problem(s) that the study targets (e.g., *code quality*, *reproducibility* or *usability*). R: This question helps to understand which challenges are focused on by software engineering researchers for Jupyter notebooks.	RQ2
Q: How do software engineering researchers evaluate their solutions? D: The steps to evaluate or measure the provided solution or study. R: Determining how software engineering researchers evaluate their solutions in studies on Jupyter notebooks is crucial for assessing the validity and effectiveness of their findings. This question aims to understand the methodologies and metrics used for evaluation, such as experimental designs, case studies, user surveys, performance metrics, or qualitative analysis. Understanding these evaluation methods allows for critically assessing the research's reliability and applicability.	RQ2
Q: What do the authors mention as the main contributions? D: List of the contributions of the study. R: This question helps to map out the progress that has been made in the field so far.	RQ2
Q: What do the authors mention as the implications? D: Implications of the results reported in the study. R: The implications are critical for understanding the real-world impact and broader significance of a study. This question seeks to uncover how the results of each study might influence future research, industry practices, educational methodologies, or software development processes.	RQ2
Q: What are the limitations of the study? D: List of the limitations presented in the study. R: This question aims to uncover the acknowledged weaknesses, constraints, or aspects that were not covered in the studies. Knowing these limitations helps evaluate the robustness of the software engineering research on notebooks. It also provides insights into areas that need further investigation or improvement in future studies.	RQ2

3.1.2. Developing the Review Protocol

A review protocol is necessary to outline the procedure for conducting the systematic review to limit the likelihood of researcher bias [71]. To meet these objectives, our protocol involves: (1) defining research questions, (2) searching and selecting primary studies, (3) extracting data from primary studies, and (4) synthesizing the data.

As a starting point, we formulated research questions that guided this systematic literature review. Our goal is to give an overview of the academic software engineering research on Jupyter Notebook. To achieve this, we defined two research questions (RQs):

- RQ1: *How much software engineering research on Jupyter Notebook has been published?* (Section 4)

  Motivation: The first research question gives a quantitative overview of the software engineering research on Jupyter Notebook. RQ1 aims to provide a high-level overview of how the field of software engineering research on Jupyter Notebook is evolving and where it is getting attention. RQ1 also explores what kinds of solutions are being proposed by software engineering researchers to deal with problems in Jupyter notebooks. By answering RQ1, we can identify trends and patterns of software engineering research targeted at Jupyter notebooks over time. Furthermore, this question explores whether research in this area follows best practices for open science [98].

- RQ2: *Which software engineering topics are being studied in software engineering research on Jupyter Notebook?* (Section 5)

  Motivation: The second research question is motivated by the idea that gaining a detailed understanding of the topics explored in software engineering research on Jupyter Notebook will help to pinpoint areas that have not yet received sufficient attention and form promising future research directions.

The specific procedures for searching and selecting primary studies, extracting data from primary studies, and synthesizing the data identified from these studies are presented in the next section.

3.2. Conducting the Review

3.2.1. Searching and Selecting Primary Studies

We searched all the indexed articles in the DBLP [8] to find relevant papers for this systematic review. DBLP is a computer science bibliography website that provides free access to bibliographic information on major computer science publications. It includes conference papers, journal articles, and other academic reports related to computer science. It is a reputable and comprehensive source of academic papers, and many researchers [193, 32, 82] used it to find relevant studies for their systematic literature reviews.

The focus of our review is software engineering research on Jupyter Notebook. This type of research includes work that analyzes or improves current software engineering practices in notebooks. To find relevant studies for our review, we searched the title of the papers with the following query: *"Notebook"* or *"Jupyter"*. The literature list was compiled up to the end of 2024, capturing the latest software engineering research on Jupyter notebooks.

To make sure that we include only studies on software engineering research on Jupyter Notebook, we define the following inclusion criteria:

- The subject of the study should be Jupyter Notebook (and not, e.g., physical notebooks)
- The study should focus on at least one software engineering topic
- The study must be available online to ensure its accessibility
- If a study has both an official and pre-print available, we picked the official one as the most recent
- The study must be written in English

3.2.2. Extracting Data from Primary Studies

We created a set of questions for the data extraction form. These questions were designed to gather the necessary information from the primary studies. To streamline the data extraction process, we linked each question to one of our research questions and provided the rationale for the question. We refined this form through several iterations with randomly selected studies. Table 1 presents the list of questions, their descriptions, rationales, and

[8] `https://dblp.org/`

corresponding research questions. We combined the data collected from the data extraction forms based on the questions. Then, we synthesized them to answer our research questions in this SLR. This compilation of data will give an overview of the existing software engineering research on Jupyter notebooks.

3.2.3. Synthesizing the Data

We manually explored the title of the online version of all those papers. In this stage, we excluded several studies that did not focus on computational notebooks but on physical notebooks (e.g., laptops). Then, we explored the studies' abstracts and selected the papers that target software engineering-related topics on Jupyter Notebook. We also excluded studies that focused on topics that are adjacent to software engineering research on Jupyter Notebook, such as those on using Jupyter Notebook for software engineering education. For example, we excluded the experience report by Al-Gahmi et al. [4] on using Jupyter Notebook in classroom programming.

We manually reviewed the list of studies to exclude duplicates, e.g., sometimes a study's final official version and unofficial pre-print version are available online. For example, Wang et al. published a pre-print of their paper titled "Themisto: Towards Automated Documentation Generation in Computational Notebooks" [173]. After that, the same paper was published officially in the next year with a different title "Documentation Matters: Human-Centered AI System to Assist Data Science Code Documentation in Computational Notebooks" [174]. We manually checked both versions of that paper and excluded the earlier version.

To ensure the reliability of our study selection procedure, the first and the third authors independently went through the list of studies following the procedure above. We used Cohen's Kappa statistic to measure the Inter-Rater Reliability (IRR) [96], which indicates the level of agreement between two raters in a classification task. We found that Cohen's Kappa value is 0.79, which indicates substantial agreement between the raters. We measured the IRR into two stages. In the first stage, the first and third authors discussed and resolved each disagreement to come up with a list of selected studies up to 2023. In the second stage, studies from 2024 were discussed and resolved by the first and second authors. The disagreements mostly arose from the first author's misunderstanding of defining the scope of provenance in the notebook. After discussing and resolving these disagreements, the Kappa value became 1.0 after resolving the disagreements. As a result, we ended

up with a list of 146 studies for our SLR.

3.3. Reporting the Review

After conducting the review, we began reporting the results. Using the data extraction strategy outlined in Section 3.1.2, we examined the primary studies to address the research questions of this literature review. We followed the data extraction questions presented in Table 1 to gather relevant data for reporting that would help us answer our research questions. We reported a quantitative overview of the software engineering research on Jupyter Notebook in Section 4 to address RQ1. We reported the publication year, publication type (e.g., journal, conference, symposium, workshop, and others), and industry collaborations. Our analysis also focused on studies that provided URLs for notebook replication, which allowed us to assess the hosting platforms used for these packages. Then in Section 5, we reported a comprehensive discussion of the various software engineering topics addressed in the literature we reviewed, aligning our analysis with RQ2. We searched, reviewed, and finalized a well-structured categorization of software engineering topics and subtopics. The first and third authors collaborated closely in a card-sorting-like process to categorize the studies based on different software engineering topics. We employed a Trello board[9] as a dynamic organizational tool to effectively manage and categorize the software engineering topics and their corresponding subtopics.

4. RQ1: How much software engineering research on Jupyter Notebook has been published?

4.1. Year-wise Distributions

The data from the SLR showcases a marked increase in software engineering studies on Jupyter notebooks. The graph in Figure 3 shows the number of publications with software engineering research on Jupyter notebooks per year. The figure outlines a growing trend. Software engineering research on notebooks began with just one study in 2015 and followed an increasing trend to 42 in 2024. This trend reflects a growth in academic interest in considering software engineering practices in Jupyter notebooks.

[9] `https://trello.com/`

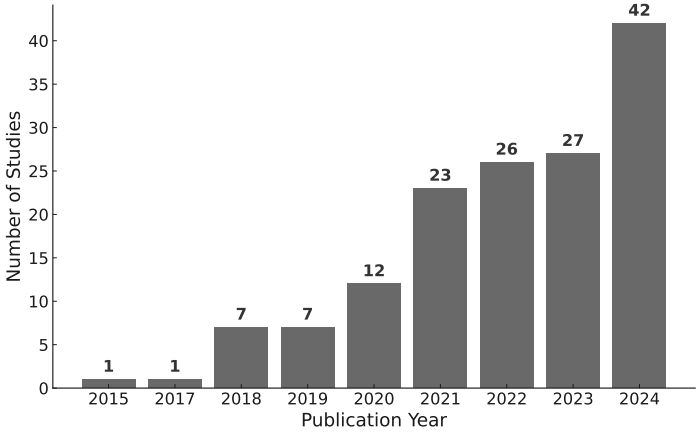

Figure 3: Number of studies published over the years

4.2. Publication Types

The majority of the work on software engineering research on notebooks was published at conferences. Figure 4 shows the number of studies per type. Our research findings showed that the majority, specifically 53.4% (78 out of 146 studies), were presented at conferences. In addition, 26 studies were published in journals, 11 in workshops, and 11 in symposia. Furthermore, we identified 20 studies that appeared in informal and other types of publications, such as book chapters, thesis dissertations, or those published in the arXiv repository. It is important to note that conferences publish findings more quickly than journals, allowing researchers to share updates with the community faster [44]. The substantial number of conference papers included in the systematic literature review indicates that software engineering research on notebooks is an active research area of study.

4.3. Publication Venues

The most popular venues for software engineering research on notebooks are focused on human-computer interaction (HCI). Table 2 presents the list of conferences, journals, symposiums, workshops, and other publication venues where the primary studies have been published. We found that the most popular venue (15 studies) was the ACM Conference on Human Factors in Computing Systems (CHI). Likewise, the most popular symposium (six studies) is the IEEE Symposium on Visual Languages and

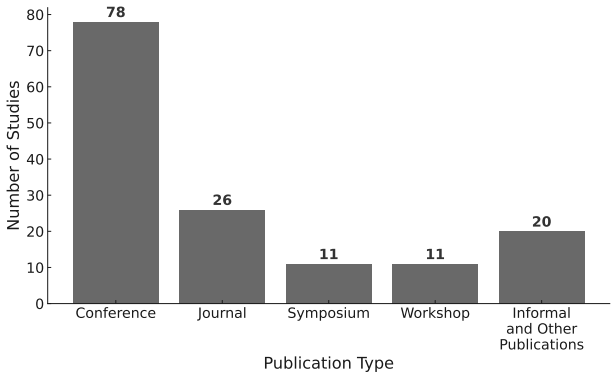

Figure 4: The distribution across publication types

Human-Centric Computing (VL/HCC), and one of the most popular journals (two studies) is the ACM Transactions on Computer-Human Interaction (TOCHI). This pattern emphasizes the interdisciplinary nature of working with Jupyter notebooks and underlines the importance of their user experience and usability (which are important topics at the HCI venues above).

Table 2: Publication outlets for software engineering research on notebooks in conferences, journals, symposiums, workshops and other platforms

Type	**Acronym**	**Venue**	**Reference**	**Count**
Conference	CHI	ACM Conference on Human Factors in Computing Systems	[55, 64, 84, 81, 97, 125, 135, 172, 183, 189, 21, 54, 60, 175, 184]	15
	ASE	Automated Software Engineering Conference	[10, 30, 100, 177, 197, 203, 59]	7
	ICSE	International Conference on Software Engineering	[109, 119, 155, 180, 178, 179]	6
	MSR	Mining Software Repositories Conference	[46, 120, 115, 40]	4
	TaPP	USENIX Conference on Theory and Practice of Provenance.	[28, 114, 75, 117]	4
	APSEC	Asia-Pacific Software Engineering Conference	[131, 145, 154]	3
	SANER	Int. Conference on Software Analysis, Evolution and Reengineering	[159, 167]	2
	CAIN	Int. Conference on AI Engineering – Software Engineering for AI	[122, 41]	2
	EDBT	International Conference on Extending Database Technology	[56, 186]	2

Continued on next page

Type	Acronym	Venue	Reference	Count
	e-Science	IEEE International Conference on eScience	[3, 25]	2
	CIDR	Int. Conference on Innovative Data Systems Research	[93, 12]	2
	SciPy	Scientific Computing with Python Conference	[99, 92]	2
	ICSME	Int. Conference on Software Maintenance and Evolution	[61, 205]	2
	FSE	ACM International Conference on the Foundations of Software Engineering	[29, 181]	2
	IUI	Annual Conference on Intelligent User Interfaces	[101, 38]	2
	SCAM	IEEE Working Conference on Source Code Analysis and Manipulation	[149]	1
	EASE	Int. Conference on Evaluation and Assessment in Software Engineering	[2]	1
	PEARC	Practice and Experience in Advanced Research Computing Conference	[185]	1
	UCC	International Conference on Utility and Cloud Computing	[196]	1
	EMNLP	Int. Conference Empirical Methods in Natural Language Processing	[91]	1
	SPLC	Int. Systems and Software Product Line Conference	[13]	1
	IJCAI	International Joint Conference on Artificial Intelligence	[169]	1
Conference	FLAIRS	Int. Florida Artificial Intelligence Research Conference	[105]	1
	WWW	ACM Web Conference	[77]	1
	ACL	Annual Meeting of the Association for Computational Linguistics	[201]	1
	ICPC	IEEE/ACM International Conference on Program Comprehension	[132]	1
	AIMLS	International Conference on AI ML Systems	[190]	1
	Programming	ACM Conference on the Art, Science, and Engineering of Programming	[104]	1
	KDIR	Int. Conference on Knowledge Discovery, Engineering, and Management	[69]	1
	ISWC	International Semantic Web Conference	[137]	1
	ICMD	International Conference on Management of Data	[130]	1
	QCE	International Conference on Quantum Computing and Engineering	[70]	1
	SEAA	Euromicro Conference on Software Engineering and Advanced Applications	[43]	1
	CSR	International Conference on Cyber Security and Resilience	[128]	1
	MOD	International Conference on Management of Data	[86]	1
	JCSSE	Int. Joint Conference on Computer Science and Software Engineering	[126]	1

Continued on next page

Type	Acronym	Venue	Reference	Count
Journal	EMSE	Empirical Software Engineering Journal	[116, 127, 166]	3
	TOCHI	ACM Transactions on Computer-Human Interaction	[121, 174]	2
	ACM-HCI	Proceedings of the ACM on Human-Computer Interaction	[134, 187]	2
	VLDB	International Journal on Very Large Data Bases	[88, 146]	2
	SoftwareX	SoftwareX	[35, 123]	2
	IEEE TVCG	IEEE Transactions on Visualization and Computer Graphics	[89, 144]	2
	TOSEM	ACM Transactions on Software Engineering and Methodology	[90, 26]	2
	ACM PL	Proceedings of the ACM on Programming Languages	[141]	1
	Sigmod Rec.	ACM SIGMOD Record Journal	[118]	1
	CGF	Computer Graphics Forum	[37]	1
	Interactions	ACM Interactions	[124]	1
	GigaScience	GigaScience	[139]	1
	BTW	Journal on Business, Technologie und Web	[68]	1
	JASEP	The Journal on the Art, Science, and Engineering of Programming	[63]	1
	PLoS CB.	PLOS Computational Biology Journal	[133]	1
	TiiS	ACM Transactions on Interactive Intelligent Systems	[85]	1
	SoftwareTT	Softwaretechnik-Trends Journal	[152]	1
	JASIST	Journal of the Association for Information Science and Technology	[15]	1
Symposium	VL/HCC	Symposium on Visual Languages and Human-Centric Computing	[73, 22, 156, 78, 66, 23]	6
	UIST	ACM Symposium on User Interface Software and Technology	[195, 107]	2
	AISTA	ACM International Symposium on Software Testing and Analysis	[165]	1
	Onward	ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software	[151]	1
	SPLASH-E	ACM SIGPLAN International Symposium on SPLASH-E	[76]	1
Workshop	IDE	ACM/IEEE Workshop on Integrated Development Environments	[170, 160]	2
	IPAW	International Provenance and Annotation Workshop	[138, 74]	2
	SC Workshop	Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis	[16, 191]	2
	SOAP	ACM Int. Workshop on the State Of the Art in Program Analysis	[103]	1
	QuASoQ	International Workshop on Quantitative Approaches to Software Quality	[7]	1

Continued on next page

Type	Acronym	Venue	Reference	Count
	HILDA	Workshop on Human-In-the-Loop Data Analytics	[14]	1
	PAINT	Workshop on Programming Abstractions and Interactive Notations, Tools, and Environments	[168]	1
	IWSC	International Workshop on Software Clones	[140]	1
Informal and Other Publications	arXiv	An archive for electronic preprints of scientific studies	[20, 24, 31, 34, 39, 45, 47, 50, 57, 87, 83, 111, 143, 147, 182, 188, 194, 198, 204]	19
	Thesis	Thesis dissertation was published at the University of California, San Diego	[151]	1

4.4. Industry Collaborations

Software engineering research on Jupyter notebooks is regularly done in collaboration with industry. Of the 146 primary studies on software engineering research on Jupyter notebooks, 31 (21.2%) were done with industry collaborations. Table 3 presents the industry affiliations of the studies included in our SLR. Microsoft Research has the highest representation among the identified industry collaborations, with a study count of 12. IBM Research has six studies on software engineering research on notebooks, followed by Microsoft. We analyzed those studies and found that IBM research mainly focused on code documentation in Jupyter Notebook [91, 101, 174, 169, 172]. On the other hand, the Microsoft Research team focuses on code management [55, 155, 118, 100, 81, 97]. We also found that JetBrains Research was involved in 5 studies, whereas Google and Fujitsu Research each have two studies. Overall, the industry affiliations reflect the widespread interest and impact of software engineering research on Jupyter notebooks in both academia and industry.

4.5. Replication Packages

Our systematic literature review discovered a gap in the availability of the resources necessary to replicate the research. Specifically, almost half of the studies (79 out of 146) did not provide publicly accessible source code or installer files of their work. This made it difficult to validate and build on their findings. In contrast, the remaining 67 studies provide source code, installer files, datasets, or a combination thereof to replicate their research results. Upon a closer manual examination of these

Table 3: Industry-affiliated software engineering research on Jupyter Notebook

Industry Name	**List of Literature**
Microsoft Research	[155, 118, 55, 100, 81, 97, 189, 21, 20, 59, 83, 107]
IBM Research	[91, 101, 169, 172, 174, 25]
JetBrains Research	[46, 159, 45, 47, 160]
Fujitsu Research of America	[10, 203]
Google Inc	[114, 201]
Apple	[23]
Adobe Research	[22]
Megagon Labs	[77]
Ploomber	[99]

studies, we found that replication packages for 64 studies are available because they provide the source code with clear instructions to execute. The remaining three studies lacked available replication packages due to the absence of source code or lack of clear execution instructions [146, 131, 145]. For example, Settewong et al. [145] shared a GitHub URL[10] of their work; however, upon examination, we found neither source code nor installation files.

Sharing replication packages of software engineering research on notebooks regularly does not follow open science principles. We found that the dominant platform of choice was GitHub, with 54 studies opting to use it to host their replication packages. Although GitHub is a widely accessible platform, any changes or deletions by repository owners can be a risk to the future accessibility of the replication packages. Best practices for open science in software engineering [98] recommend to adopt platforms and repositories designed for the long-term archiving of research outputs. We found that only 13 studies hosted their replication packages on platforms that align with these best practices (i.e., they ensure the permanent archiving of digital artifacts). Of these, 11 were hosted on Zenodo [105, 127, 115, 73, 120, 121, 7, 43, 40, 26, 182] and two on Figshare [90, 147].

4.6. Jupyter Notebook Extensions

We found that a modest 10.9% of the studies (16 of 146) explored the development and use of Jupyter Notebook extensions

[10] `https://github.com/NAIST-SE/VizJupyterNotebooks`

to enhance the user experience within notebook environments. Despite showing the potential benefits of Jupyter Notebook extensions, five studies did not provide available replication packages due to missing links or code [25, 101, 186, 168, 81]. We found that eight extensions were reusable and integrable into Jupyter notebooks, showcasing a variety of functionalities designed to improve the user experience of Jupyter notebooks [183, 10, 64, 116, 55, 68, 127, 88]. After replicating and integrating these extensions into Jupyter notebooks, we found that the majority of Jupyter Notebook extensions are designed to enhance the visualization and presentation of notebook cells and their outputs [183, 10, 127, 64, 55, 111]. However, only a few studies tackle more complex challenges related to the core functionalities of Jupyter notebooks. For example, improving the quality and reproducibility of Jupyter notebooks [116], managing data provenance through their extension [68, 37], allowing live migration of notebook environments [88], and notifying potential fairness in data science code [54].

5. RQ2: Which software engineering topics are being studied in software engineering research on Jupyter Notebook?

We have classified studies into 11 groups based on the specific software engineering topics they addressed. By grouping those studies, we aim to uncover the key challenges and solutions in software engineering that researchers have explored in the context of Jupyter Notebook. We have listed the identified topics, subtopics, and their corresponding studies in Table 4. In this section, we will provide detailed information about these software engineering topics and their associated studies.

5.1. Code Reuse and Provenance

Code reuse in Jupyter notebooks refers to the intentional act of leveraging previously written code cells. Code reuse within Jupyter notebooks differs from traditional IDEs, particularly regarding the notebook's structure and workflow. Jupyter notebooks employ a cell-based execution model that facilitates iterative coding, making them well suited to quickly test hypotheses and iterate on data by reusing existing code cells [73]. A survey among Microsoft data scientists revealed that 94% of the participants considered reusing existing code in Jupyter notebooks to be at least "important" [21]. Users often reuse code through code cloning, where cells are copied within or across notebooks, which can lead to inconsistencies and technical debt [73].

Table 4: List of software engineering topics addressed by SE research on Jupyter Notebook

Topics	**Subtopics**	**List of studies**
Code reuse and provenance (Section 5.1)	Code cloning	[73, 198, 63, 131]
	Reusing code snippets by code search	[56, 84, 186, 85, 10, 57, 7, 126, 83]
	Reproducibility	[116, 115, 143, 133, 177, 178, 139, 3, 138]
	Provenance	[74, 117, 28, 68, 69, 137, 66, 37, 54, 154]
Managing computational environment and workflow (Section 5.2)	Empirical studies on workflows	[78, 124, 24, 118, 130, 75, 156, 43, 205]
	Computational environment in notebooks	[31, 25, 21, 92, 88, 189, 128, 70, 16, 160, 86, 204, 141, 87]
	Managing dependencies	[203, 180, 35, 185]
	Performance analysis	[190, 191]
Readability of notebooks (Section 5.3)	Refactoring	[90, 140, 29, 30, 159, 146, 55]
	Non-linear visualization	[183, 64, 134, 20]
Documentation of notebooks (Section 5.4)	Empirical studies on documentation	[136, 135, 125]
	Documentation generation	[174, 169, 91, 172, 100, 101, 81, 89, 41, 175, 107]
	Cell header generation	[167, 165, 166, 111]
Testing and debugging (Section 5.5)	Empirical studies on testing and debugging	[26, 147, 182]
	Detecting bugs	[132, 196, 12, 109, 34, 181, 47, 45]
	Detecting data leakage	[197, 155, 103]
Visualization in notebooks (Section 5.6)	Empirical studies on visualizations	[2, 145, 127, 184, 194]
	Interactive visualization	[77, 195, 144, 60, 50]
Best practices (Section 5.7)	Following code style standards	[46, 179, 149, 15, 122, 123, 152]
	Best practices for collaborative use	[119, 121, 170]
Cell execution order (Section 5.8)	-	[105, 151, 61, 99, 93, 14]
AI-based coding assistance for notebooks (Section 5.9)	-	[201, 13, 97, 22, 38, 58, 23, 188, 76]
Supporting other programming paradigms (Section 5.10)	-	[104, 114, 168, 187]
Datasets of notebooks (Section 5.11)	-	[46, 121, 84, 91, 180, 120, 73, 115, 125, 26, 90, 127, 40, 147, 43, 58, 47, 182, 7, 39]

To find reusable code more efficiently, code search tools provide support by helping users locate relevant code cells or workflows [84]. However, proper provenance management is crucial for understanding the origin and evolu-

tion of reused code, ensuring that dependencies and execution history are clear [196]. Moreover, reproducibility becomes a challenge in notebooks, as reused code might not produce consistent results if dependencies or execution orders are not properly tracked [177]. In this subsection, we describe these studies that address the challenges and opportunities of reusing code within Jupyter notebooks and provenance.

5.1.1. Code Cloning

Code cloning in Jupyter notebooks refers to copying and pasting code cells within the same notebook or between different notebooks. Research has shown that code cloning is common in Jupyter notebooks. Over 70% of all notebook code cells on GitHub are exact copies of other cells, and approximately 50% of all notebooks contain no unique cells [63]. These findings are also supported by Yang et al. [198], who noted that about 32% of Jupyter notebooks hosted in GitHub repositories were directly copied from Stack Overflow. The code snippets that are most frequently cloned in Jupyter notebooks mainly relate to visualization (21%), followed by ML (15%) [73]. Among reused code, interproject clones in Jupyter notebooks are far more common than intraproject clones, which is the opposite of the prevalence of code clones in Java code [42]. Furthermore, another study indicated that top-level notebook users (i.e., Grandmasters in Kaggle) are most likely to clone common abstractions such as importing packages, configurations, file IO operations, and showing data [131].

5.1.2. Reusing Code Snippets by Code Search

Searching for code snippets in Jupyter notebooks can be different from traditional code. While users often seek specific functions or APIs in traditional code, notebook users frequently search for code cells (snippets) that analyze similar data or follow comparable workflows [84]. As highlighted by Li et al. [84], semantic code search is vital in this context, which enables natural language queries that reflect the intended functionality of code cells rather than just API names or keywords. Moreover, the flexible nature of notebooks, where code, data, and results are integrated within a single file, contrasts with the modular organization of traditional code. This integration in notebooks makes it more difficult to isolate functionality and understand the context of individual cells, which imposes challenges for code search such as tracking the logical flow of code and capturing enough context within a code cell [55].

Despite sharing common objectives, code search tools for Jupyter notebooks vary in their methodological approaches. For instance, some tools utilize keyword-based search mechanisms, where users input specific terms related to their queries. This approach, demonstrated in research by Watson et al. [186], allows users to find relevant code by matching their search terms with corresponding code repositories and snippets. On the other hand, certain advanced search tools leverage deep learning techniques to interpret natural language queries. This enables users to search for code snippets more intuitively, using conversational language instead of strict keywords [84]. Similarly, Ragkhitwetsagul et al. [126] proposed *Typhon*, which integrates traditional information retrieval techniques (BM25) with UniXcoder [49] code embeddings to perform text-to-code matching. In addition, Li and Lv [83] developed a semantic search framework explicitly tailored for Jupyter notebooks, leveraging embeddings generated by large language models (LLMs) from both markdown and code cells.

Additionally, some tools enhance search efficacy by employing graph representations to illustrate code flow relationships within a Jupyter notebook. By mapping the interactions and dependencies between different code cells, these graph-based tools can identify interrelated code cells and suggest connections that the user needs [56]. *EDAssistant* [85] introduces a methodology for context-aware code search in Jupyter Notebook. This approach prioritizes example notebooks that align with the user's intent and existing code, assisting users in finding relevant code snippets to reuse and adapt during exploratory data analysis. Similarly focused on cell-level recommendation, Aydin et al. [7] proposed a specialized cell recommendation approach targeting ML notebooks. Furthermore, some search solutions specifically target the data visualization domain by mining, extracting, and cataloging Python functions designed for visualization purposes [10]. This approach leverages reusable visualization snippets from data science notebooks to facilitate access to visualization-related code, thereby promoting the reuse and adaptation of existing visualization workflows.

5.1.3. Reproducibility

Reproducibility ensures that other notebook users can get the same results from a notebook by following the same steps, using the same data, and working in the same environment [3]. Reproducibility differs from code reuse and cloning, which typically involves adapting code or directly copying code into different environments and applying the code to different data to meet

Table 5: Good and bad practices that affect the reproducibility of a notebook

Good practice #1: Share and explain the data [133]
Reproducibility requires sharing data alongside notebooks. When full datasets are too large or sensitive to share, provide a sample or detailed descriptions of the data and processing steps. Breaking complex datasets into tiers ensures interpretability while maintaining accessibility and reproducibility.

Good practice #2: Document the process, not just the results [133]
Notebooks require documenting the process throughout the analysis, including taking notes during the analysis, capturing key decisions, reasoning, and observations, and preserving the context of the work. This approach enhances the clarity and utility of notebooks, facilitating their use for future reference and collaboration.

Good practice #3: Record and manage library dependencies [133]
The notebook format does not encode library dependencies with pinned versions, making it difficult (and sometimes impossible) to reproduce a notebook. To ensure reproducibility, users of notebooks should carefully manage their dependencies using an environment management package. This process involves creating and sharing a file, such as 'environment.yml' or 'requirements.txt', which lists all the libraries and their specific versions used in the notebook. Such a file offers a clear and accurate description of these dependencies.

Bad practice #1: Presence of non-executed code cells [116]
Non-executed cells can lead to discrepancies between the notebook's apparent logic and its actual state, making it unclear whether the code has been tested or contributes to the presented results. This practice hinders the ability of others to reproduce the workflow, as the notebook may fail to run as expected or produce inconsistent outcomes.

Bad practice #2: Out-of-order cell execution [116]
Out-of-order cell execution can create challenges for others trying to trace the steps necessary to reproduce results. This occurs because dependencies between cells may need to be clarified or temporarily broken. Such practices can lead to errors, incomplete outputs, or incorrect conclusions when the notebook is reproduced.

Bad practice #3: Presence of hidden states [115]
A hidden state in a notebook occurs when variables or data are changed in ways that are not clearly documented. Hidden states caused by cell re-execution or removal make the notebooks skip numbers in the execution counter sequence. This often results from executing cells out of order or from relying on previous sessions. Figure 5 presents an example of a hidden state. Such hidden states can make it difficult for others to understand or replicate the workflow since the notebook's output depends on undocumented or inaccessible conditions.

different objectives of users, resulting in outputs that are different from the original notebook.

Reproducibility is known as one of the key promises of Jupyter notebooks, as they are often used to share results with others, allowing users to trace the steps from raw data to final results [72]. However, achieving reproducibility remains challenging due to issues such as out-of-order execution and repetitive execution of the same cell [177]. These issues differ from traditional programming, where code execution is typically linear (from top to bottom), ensuring a clear and consistent execution flow. For example, notebook users might execute the same cell multiple times and only save the latest execu-

tion state, causing cells located at the notebook's beginning to be executed after later cells. This nonsequential execution can confuse others trying to reproduce results, affecting notebook reproducibility [177, 178].

A study by Pimentel et al. [115] analyzed more than 1.4 million Jupyter notebooks from GitHub and found significant reproducibility issues. The study noticed that only 25% of the valid notebooks executed without errors under straightforward conditions, and only 5% reproduced results identical to their original outputs. Furthermore, Schröder et al. [143] revealed that only 14% of the available notebooks in published academic research on PubMed,[11] where Jupyter notebooks are used as implementation, were reproduced successfully. The study noticed the need for comprehensive documentation and containerization of notebooks, which supports the good practices presented by Rule et al. [133]. We have analyzed the relevant studies and presented a list of good and bad practices that affect the reproducibility of a notebook in Table 5.

Researchers have demonstrated that the reproducibility of Jupyter notebooks can be significantly improved by resolving cell dependencies within the notebook's code [177, 180]. Cell dependencies represent the relationships between variables, functions, and data in different code cells. Wang et al. [178] proposed a method to build a cell dependency graph using static analysis to accurately model these relationships and found that 83% of executable notebooks can be reproduced using their approach. Samuel and König-Ries [138] developed a visualization tool *ReproduceMeGit* to check the reproducibility of Jupyter notebooks from GitHub by extending the work from Pimentel et al. [115]. This tool can automatically install dependencies from *requirements.txt* or *setup.py* files and execute the notebook to provide highlights of the reproducibility study through a user interface.

5.1.4. Provenance

Provenance in notebooks refers to the detailed history of actions, executions, and dependencies that led to the current state of a notebook. It includes the sequence of cell executions, the modifications made to cells, variable dependencies, data flow, and intermediate outputs [68]. Unlike reproducibility which emphasizes achieving identical results under a consistent initial environment, provenance addresses the historical and iterative details

[11] `https://pubmed.ncbi.nlm.nih.gov/`

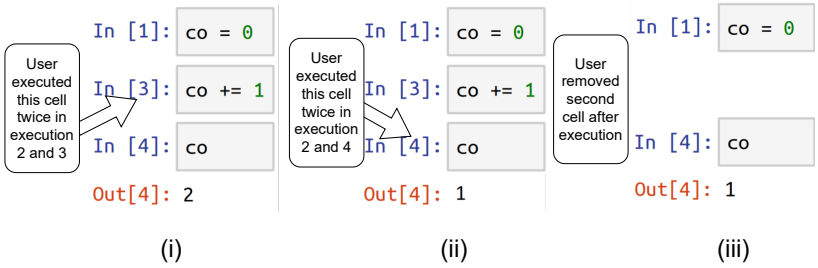

Figure 5: An example of a hidden state in Jupyter Notebook with three different execution variations

present in the notebook's creation and evolution. Managing provenance in Jupyter Notebook differs from traditional systems due to its iterative and non-linear nature. In a notebook, users can execute cells in any order, creating challenges in tracking the execution history and dependencies [69]. Additionally, Jupyter notebooks merge code, data, and outputs into a single file, whereas traditional systems follow a sequential execution process that generates output externally. As traditional systems offer clear execution pipelines and distinct output demonstrations, provenance management becomes more straightforward. In contrast, Jupyter notebooks require specialized tools for managing provenance to capture dynamic state changes and overwritten variables [68, 74].

Researchers developed provenance detection techniques specialized for Jupyter Notebook to reuse notebooks by obtaining the original results. In the early stage of Jupyter Notebook usage, Pimentel et al. [117] proposed a notebook tool to overcome the limitations in provenance support by automatically capturing and analyzing the execution history and environment settings of code within notebooks. The study extends the *noWorkFlow* tool [102] developed for Python scripts to capture the provenance of scripts, including control flow information and library dependencies. The proof can be inferred by statically analyzing code cells on execution counts and cell positions within the notebooks [74]. A scheduler that adjusts the order of cell execution based on data dependencies detected during runtime can be used to improve the accuracy of provenance of Python code in notebooks [28].

In addition to focusing on tools and execution kernels, researchers have developed extensions to capture provenance. *MLProvLab* [68] is a JupyterLab extension to track, manage, compare and visualize the provenance of computational experiments. It can capture the provenance at run-time by detecting

dependency graphs. Another notebook extension, *ProvBook*, was designed to capture and visualize the provenance of notebook executions [137]. This extension automatically stores provenance information within the notebook's metadata, detailing execution times, inputs, and outputs for each cell. It helps users to see how data and analysis have evolved over time and compare results, which helps to enable notebook reproducibility. Kery and Myers [66] developed an extension named *Verdant* that records the history of all previous changes in a notebook and quickly retrieves versions of a specific artifact from the existing versions of the entire document. It can help to compare multiple versions of different notebook artifacts, including code cells, tables, and images, which are absent in traditional version control systems like Git.

Expanding on these efforts, Studtmann et al. [154] introduced *Histree* for automatic versioning and visual branching of notebook modifications. It organizes experiment histories into tree-based structures for easier navigation. Further extending provenance into interactive visual analytics, Gadhave et al. [37] proposed *Persist*, which captures interaction provenance across code and visualizations. Empirical evaluations showed improvements in analysis efficiency, accuracy, and reproducibility. Finally, *Retrograde* [54] integrates provenance tracking with fairness auditing. It provides real-time, context-aware notifications triggered by notebook events and maintains a dynamic data ancestry graph, helping users detect and address fairness and bias concerns during data preprocessing and modeling.

Summary of code reuse and provenance

Code reuse in Jupyter notebooks is essential for improving productivity and collaboration while addressing the challenges associated with their unique structure and iterative workflows. Users can more effectively leverage existing resources and maintain reusable notebooks through techniques such as code cloning, context-aware code search, provenance management, reproducibility practices, and advanced versioning tools. The development of innovative solutions, such as context-sensitive code search tools [84, 85], code dependency resolution systems [178], and provenance tracking extensions [68, 117, 54], highlights ongoing efforts to support effective code reuse and provenance. By integrating these solutions and best practices (see Table 5), notebooks can become more reusable and maintainable, thus establishing a stronger foundation for data science workflows and collaborative research.

5.2. Managing Computational Environment and Workflow

Managing the computational environment and workflow in Jupyter notebooks is crucial to ensure the successful execution and reproducibility of data science tasks [118, 92]. Unlike traditional programming environments, which are typically well defined and managed by integrated development environments or build systems, a Jupyter Notebook does not have a separate file to manage the configuration [72]. Traditional programming environments commonly include a configuration file (e.g., "requirements.txt" in Python) to ensure a consistent and reproducible setup for code execution. In contrast, Jupyter notebooks execute code cells independently, and each notebook has its own dependencies [25, 124]. This section explores various aspects of computational environment management, including empirical studies on notebook workflows, the impact of different environments on code behaviour, and the challenges associated with managing library dependencies.

5.2.1. Empirical Studies on Workflows

Lau et al. [78] analyzed 60 notebooks and summarized four main stages of a data science workflow: importing data into notebooks, writing and editing code, running the code to generate output, and publishing the results. However, workflows can vary. For example, notebooks often start with exploration, where users write lots of code to find interesting patterns in the data, and end with a presentation or explanation [124]. In addition, recent empirical analyses have shed further light on workflow diversity. Golendukhina and Felderer [43] studied 138,376 Kaggle notebooks and identified significant variations in data preprocessing practices correlated with user expertise, highlighting a notable gap between model-centric activities and actual data cleaning efforts. Similarly, Zou et al. [205] uncovered that data scientists frequently and manually experiment with alternative ML pipeline configurations (e.g., data preparation, model selection), emphasizing the limitations of managing and systematically exploring pipeline variations in traditional notebook environments. On the other hand, Psallidas et al. [118] identified two types of pipelines in Jupyter notebooks that are key components of data science workflows: explicit and implicit. Explicit pipelines utilize tools like *sklearn.pipeline* to define structured steps for tasks such as data pre-processing and model training. In contrast, implicit pipelines rely on ad hoc function calls with libraries such as Pandas to clean, merge, and visualize data [118]. Researchers have also suggested ways to map workflows to better understand how notebooks work. One approach uses directed acyclic graphs

to show the flow of data and tasks within a notebook [130]. These graphs can highlight where variables are reused (if a cycle exists) or how tasks are connected. Another solution is a Jupyter Notebook extension that assigns unique IDs to each cell, making it easier for users to track how cells depend on each other [75]. These tools help manage and simplify the often complex workflows in notebooks.

5.2.2. Computational Environments in Notebooks

The computational environment refers to the environment in which Jupyter notebooks run [72]. This environment can affect the code execution process because different execution environments may have different dependencies, causing inconsistencies in code behaviour [31]. Researchers have found that data scientists face computational environment-related challenges throughout the analytics workflow, from setting up the notebook to deploying it to production [21].

Although notebooks typically run in a single computational environment, there are advantages to running a notebook in multiple computational environments [25, 31]. For example, parts of a notebook, such as model training algorithms, may require specialized computational resources that are unavailable in a standard environment designed primarily for data exploration or visualization. Cunha et al. [25] presented a solution developed as a Jupyter Notebook extension to automatically determine and migrate selected notebook cells to appropriate computational environments for execution. Their solution leverages abstract syntax trees and dependency tracking to extract the selected notebook cells and the dependencies. Another study discussed the possibility of automating the migration of a computational environment for Jupyter notebooks to a distributed Kubernetes[12] environment [31]. This approach aims to overcome the limitations of Jupyter notebooks, such as scalability and fault tolerance, by dividing notebooks into executable steps and deploying them in a *Kubernetes* cluster. In addition, *ElasticNotebook* [88, 86] provides live migration through optimized checkpointing and restoration to reduce migration overhead. Expanding this concept further, Kinanen et al. [70] introduced a custom Jupyter kernel to simplify offloading quantum computation tasks directly from the notebook to remote Kubernetes clusters, thus lowering access barriers to high-performance quantum computing with-

[12] `https://kubernetes.io/`

out requiring deep infrastructure expertise.

Recognizing security as a crucial factor in computational environment design, researchers have explored specific strategies for improving Jupyter notebook security. For instance, Lu et al. [92] proposed an architectural security framework leveraging containerization, load balancing, authentication, and encryption mechanisms, thus safeguarding sensitive data within cloud-based notebook execution environments. In addition, Ramsingh and Verma [128] conducted empirical analyses revealing gaps in current notebook security practices, such as poor awareness of threats and insufficient technical expertise. Based on their findings, they proposed the Jupyter multi-layer security (JMLS) defence model explicitly designed to reinforce notebook security. Complementing these perspectives, Cao [16] developed a systematic taxonomy to classify and better understand the variety of network-based security threats impacting Jupyter notebooks, especially in high-performance computing (HPC) scenarios. Their exhaustive analysis highlighted vulnerabilities involving ransomware, data exfiltration, misconfiguration, and resource misuse, providing insights for secure management of computational environments.

Researchers have also investigated various computational environments to tackle Jupyter Notebook reproducibility and usability issues. Sato and Nakamaru [141] addressed reproducibility issues arising from potentially unsafe dynamic notebook modifications by developing *Multiverse Notebook*, a computational environment that enables safe and efficient "time-travel" (cell-wise checkpointing) based on the POSIX "fork()" mechanism, where a process is assigned to each executed cell and a process tree is maintained as the entire state. To address hidden state and out-of-order execution issues, Weinman et al. [189] created a notebook extension that allows for easy branching (or "forking") of execution states into separate kernel instances to simplify concurrent exploration of different analytical approaches. Furthermore, Li et al. [87] introduced *Kishu*, which employs namespace-patching techniques to efficiently revert or undo notebook states without resorting to costly kernel restarts or complete re-executions. To improve Jupyter notebook usability and integration into software development workflows, Titov et al. [160] explored strategies for integrating computational notebooks within IDEs to better align interactive and exploratory notebook usage with standard software engineering practices. Complementarily, acknowledging the cognitive and operational challenges inherent in mixed-methods research workflows, Zhu et al. [204] proposed design concepts for notebook environments that

more effectively integrate qualitative and quantitative analysis tasks, thereby streamlining hybrid analytical processes.

5.2.3. Managing Library Dependencies

Managing dependencies in Jupyter notebooks is crucial not only to ensure their reproducibility, but also to ensure their functionality. To address this challenge, researchers have developed various tools. For example, *SnifferDog* is a dependency management tool that restores notebook execution environments by automatically identifying the required libraries and their compatible versions [180]. This tool leverages a comprehensive API bank to analyze Python code and make notebooks executable and reproducible. Similarly, *Davos* simplifies dependency management by dynamically installing and updating the correct library versions directly within notebooks, resolving common issues such as version mismatches [35]. Another tool, *RELANCER*, employs an automatic technique that restores the executability of broken Jupyter notebooks by upgrading deprecated API calls to non-deprecated ones [203].

Additionally, a framework that monitors Linux kernel system calls captures specific versions of dependencies to ensure that the computational environment can be precisely recreated across systems, addressing failures caused by missing or incompatible libraries [185]. These tools collectively aim to reduce the manual effort involved in dependency management, improving the reliability of notebooks in diverse computational settings.

5.2.4. Performance Analysis

Performance analysis in Jupyter notebooks addresses the efficient execution of notebook code. Werner et al. [190] developed a tool for Jupyter notebooks that measures the execution time and memory usage of individual cells. This tool can identify inefficiencies in runtime or the resource consumption of a notebook to detect performance issues and provide detailed feedback, enabling users to pinpoint resource-intensive areas for refining their code. Extending this idea further, Werner et al. [191] introduced *JUmPER*, which combines coarse-grained performance monitoring with fine-grained instrumentation for interactive HPC workflows. In addition, *JUmPER* leverages parallel marshalling and in-memory communication methods to minimize instrumentation overhead.

Summary of managing computational environment and workflow

Managing the computational environment and workflow in Jupyter notebooks is critical to ensuring smooth execution. Empirical studies show that notebook workflows typically involve iterative processes, from importing data and editing code to generating output and sharing results [78, 124]. Tools, such as directed acyclic graphs and unique cell identifiers, have been introduced to help map workflows and manage dependencies between cells [130]. Researchers developed an efficient computational notebook engine for enabling cell-wise checkpointing [141]. Researchers have also explored solutions for adapting notebooks to different computational environments, including tools for migrating cells to distributed platforms such as *Kubernetes* [31] and systems such as *Elastic-Notebook* [88] for checkpointing and restoration. In addition, managing library dependencies remains a key focus, with tools such as *RELANCER* [203], *SnifferDog* [180], and *Davos* [35] automating the resolution of API issues and ensuring consistency across environments. These advancements address the unique challenges of Jupyter notebooks, making them more reliable and scalable.

5.3. Readability of Notebooks

The readability of notebooks has two central dimensions: (1) the readability of the code in the notebook, which focuses on the clarity, organization and quality of the written code [90], and (2) the readability of the notebook narrative, which emphasizes how effectively the text and the output explain the workflow and analysis [133]. Unlike traditional source code files, notebooks integrate iterative code execution, visual output, and narrative explanations, contributing to a distinctively different in nature narrative-driven coding environment [135]. Researchers have addressed notebook readability challenges through various practices and tools, with the aim of improving both code and narrative readability.

5.3.1. Refactoring

Refactoring helps enhance readability by focusing primarily on improving the clarity and organization of the notebook's code itself. Due to Jupyter Notebook's distinctive characteristics, such as the cell-based structure and the iterative nature of cell-wise execution [90], traditional software refactoring methods may not always be directly applied. Often, notebooks are

developed with a focus on completing the final data analysis, and little attention is paid to code refactoring, as users prioritize drawing conclusions and sharing results [135]. This approach leads to "messy" notebooks, which are disorganized and poorly structured, and characterized by scattered cell arrangements, excessive inline outputs, and a lack of modularity [55]. Messy code reduces readability, makes notebooks harder to maintain, and complicates tracing the steps that produced the results, which can ultimately discourage sharing and collaboration [133, 55].

To address these issues, refactoring practices are employed to improve notebook readability and maintainability. Researchers found that the common refactoring practices in notebooks are extracting functions, reordering cells, renaming notebooks, splitting cells, and merging cells [90]. In another study, Dong et al. [30] expanded this list of refactorings by identifying additional activities such as removing commented blocks, adding comments, and renaming variables. These refactoring practices are not universally applied but vary depending on the authors' backgrounds (e.g., data scientists versus computer scientists) [90]. These also vary from the intended use of the notebook [29]. Dong [29] noticed that notebooks for sharing results with others saw mostly refactorings related to Markdown, whereas, for production notebooks, the most refactorings occurred in the logical code in the transition between notebook and Python file.

In addition, tools such as *nbslicer*, a hybrid static-dynamic slicing tool, were developed to help cleaning and re-executing notebooks more effectively [146]. Using backward and forward program slicing, *nbslicer* helps reduce messy code and improves the efficiency of refactoring while addressing challenges such as achieving accurate program slices without excessive overhead or performance issues. These tools and practices collectively aim to improve the readability and maintainability of Jupyter notebooks. Another tool named *ReSplit* helps to split the notebook code cell so that coherent code lines are placed together [159]. In the first step, the algorithm of that tool suggests merging some of the cells, and in the second step, it suggests extracting specific code fragments from the original cells into new ones [159].

Refactoring should not affect the functionality of the code. However, verifying the correctness of a notebook after refactoring can be challenging. For example, when code clones are refactored into functions, users must ensure that the output remains consistent. To address this issue, [140] proposed a method to verify correctness by comparing API calls and textual output to ensure that the same API calls with identical parameters produce consistent

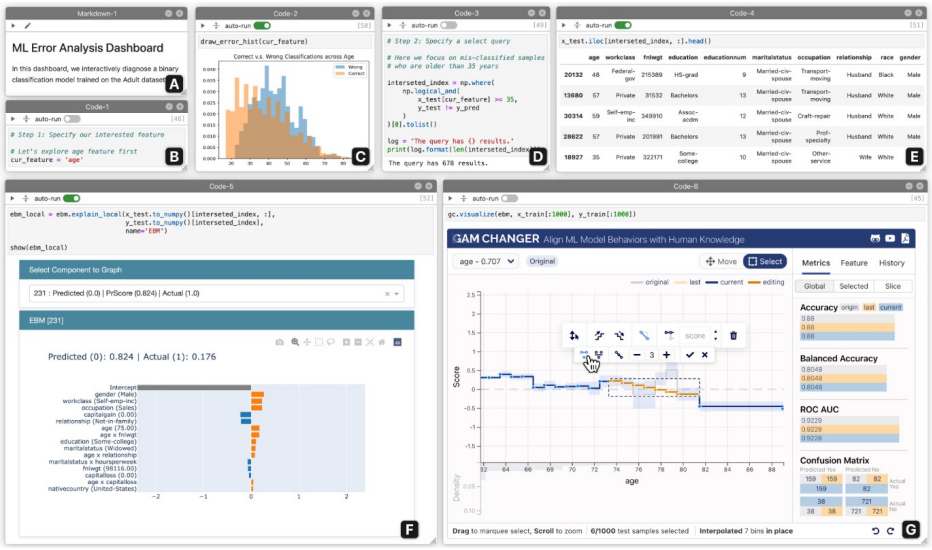

Figure 6: A screenshot of the *StickyLand* tool, which helps to place notebook cells in any orientation (not in a fixed top-to-bottom position) (taken from `https://github.com/xiaohk/stickyland`)

results, simplifying the validation process.

5.3.2. Nonlinear Visualization of Notebooks

While refactoring primarily focuses on code readability, nonlinear visualization focuses on improving readability through narrative clarity and interactive structuring. Due to the inherent cell-based organization of Jupyter notebooks which follow a linear, top-to-bottom format, the narrative of notebooks can sometimes limit readability. Researchers addressed narrative readability by proposing nonlinear organization tools that enable visualization and organization beyond the linear convention. For example, tools such as *StickyLand* [183] and *ToonNote* [64] allow users to rearrange notebook cells based on their priorities, such as by grouping related sections or highlighting key parts of the workflow. These tools offer new options for organizing, navigating, and displaying notebooks.

StickyLand [183] provides a visual interface where users can freely drag and drop cells to create a customized layout, breaking away from the conventional top-to-bottom order. This allows users to better align the notebook's structure with their thought process or the natural flow of their analysis. Figure 6 demonstrates how *StickyLand* transforms the notebook interface

into a more intuitive and visually organized workspace.

Furthermore, Chattopadhyay et al. [20] explored the cognitive processes involved in understanding computational notebooks, identifying key tasks such as comprehension, mental modelling, and contextual inference. They mapped these tasks to practical design elements, including navigation panels, annotations, and structured titles, and incorporated these elements into *Porpoise*, an interactive overlay tool specifically designed to enhance notebook sense making and navigation.

Summary of readability of notebooks

Notebook readability refers to readability and organization of code and other content within Jupyter notebooks. Researchers have examined refactoring practices to improve code readability, such as reorganizing cells, extracting functions, and splitting or merging cells. Tools such as *nbslicer* [146] and *ReSplit* [159] support these efforts by providing semi-automated solutions to clean up and optimize notebook structures. Moreover, tools such as *StickyLand* [183] and *ToonNote* [64] facilitate non-linear visualization of notebooks, enabling users to organize cells and better reflect their analytical processes dynamically. Collectively, these advancements improve the readability and maintainability of Jupyter notebooks.

5.4. Documentation of Notebooks

Research on notebook documentation is crucial because Jupyter Notebook allows combining code with narrative, data visualization, and exploratory analysis in a way that traditional source code files do not [72]. However, notebook users often pay attention only to the code without creating or updating their documentation [174]. For our purposes, we define documentation to include any content within Markdown cells and comments in code cells. In this section, we will discuss the key areas that researchers have explored regarding notebook documentation, including document generation, cell header creation, and storytelling.

5.4.1. Empirical Studies on Documentation

Integrating explanatory text in notebook documentation is essential to improve the understandability and shareability of notebooks. These texts

make a well-documented notebook that describes workflows, provides context, and explains the purpose of code. Although studies have shown that almost all notebooks (99%) include at least one Markdown cell [135], the quality of documentation inside markdowns is not up to mark [172]. To improve notebook documentation, Wang et al. [172] analyzed highly voted notebooks and identified nine guidelines for Markdown content (such as adding sections or subsections) in notebooks. Researchers also showed that although using code comments can enhance readability by organizing and annotating the flow of code [125], only a small fraction of notebooks use comments to explain the reasoning (10%) or expected results (4%) of the corresponding code cells [135, 136].

5.4.2. Documentation Generation

Researchers developed several tools to generate notebook documentation. To tackle the challenge of insufficient documentation in data science notebooks, *Themisto*, an AI system to generate code documentation, has been proposed [174, 169]. It works in three phases, i.e., retrieving relevant API documentation, generating documentation automatically, and prompting users to add their documentation. Themisto was reported to reduce documentation time, increase user satisfaction, and encourage documentation of previously overlooked code. Another tool *HAConvGNN*, uses a hierarchical attention mechanism to focus on relevant code snippets and tokens for accurate documentation generation [91]. In addition, *Cell2Doc* uses ML pipelines to generate documentation for code cells [100] and *InkSight* [89] documents insights from charts using sketch-based interactions. To enhance the quality and relevance of automatically generated documentation, Ghahfarokhi et al. [41] proposed a CNN-RNN-based approach leveraging diverse code metrics (such as cell complexity and API popularity). Empirical results from Muller et al. [101] show that in 41% of the cases, users still need to modify automatically generated notebook documentation.

Storytelling for Jupyter notebooks takes documentation even further by transforming raw analyses into coherent narratives tailored for communication. For instance, *Notable* [81] automates the generation of narrative presentation slides directly from the notebooks. Addressing the related challenge of slide creation from disorganized computational notebooks, Wang et al. [175] introduced *OutlineSpark* to generate slides through interactive outlining and computational support. Going one step further, Ouyang et al. [107] proposed *NotePlayer*, which bridges notebook cells with dynamic video segments by

integrating computational engines and LLM-generated narrations, addressing the problem of excessive manual effort traditionally required for detailed tutorial video creation.

5.4.3. Cell Header Generation

In notebooks, a cell header is typically a Markdown cell that defines section titles or headings, which helps organize and structure the notebook more effectively. Cell headers often utilize Markdown syntax (e.g., # or ##) to create hierarchical headers, enhancing the readability and navigability. Unlike standard documentation generation, which focuses on providing detailed explanations, inputs, or outputs, cell headers primarily serve to visually structure the flow of the notebook [165]. These headers act as navigational aids, dividing the notebook into logical sections to improve readability and guide users through the workflow. Venkatesh et al. [167, 166] introduced an automatic cell header generation tool *HeaderGen*, which analyzes the functions and call graphs of the notebook code to generate appropriate headers for the notebook cells. Venkatesh and Bodden [165] published an early prototype of the *HeaderGen* tool that can automatically create headers in Markdown cells using static analysis of the code cells, even without providing specific details about the Markdown text documentation. Extending beyond static analysis, Perez et al. [111] proposed a hybrid approach called *JUPYLABEL*, which incorporated both rule-based heuristics and decision-tree classifiers to classify and label notebook cells to generate headers.

Summary of documentation of notebooks

Research on notebook documentation in Jupyter notebooks addressed integrating explanatory text to enhance understandability and shareability. In documentation generation, ML models help automate the process, with tools such as *Themisto* [174] and *HAConvGNN* [91] which use ML to generate comprehensive documentation based on code structure and content. Research has explored human interaction with AI-generated text, highlighting the need for human intervention to improve the quality of generated documentation. Cell header generation focuses on automatically generating headers to improve notebook navigability and usability, exemplified by tools such as *HeaderGen* [167] and *JUPYLABEL* [111].

5.5. Testing and Debugging

Jupyter notebooks are widely used for prototyping, offering rapid iteration and immediate feedback [34]. However, testing in notebooks differs from traditional software testing, as it may require verifying code correctness at the cell level rather than treating the entire notebook as a single executable unit. Due to the lack of standardized testing practices, data scientists have adopted various approaches. For example, some embed test cases within the notebook, while others use separate test notebooks to verify the correctness [21]. This section describes studies related to the testing of notebooks, including empirical studies on bug analysis and bug detection.

5.5.1. Empirical Studies on Testing Notebooks

De Santana et al. [26] empirically analyzed the problems and challenges of identifying, diagnosing and resolving bugs in Jupyter notebooks. They presented the first comprehensive study of bugs in Jupyter Notebook projects, analyzing bugs in GitHub repositories and Stack Overflow posts, and interviews with data scientists. The study identifies eight types of bugs that are found in notebooks along with their studies. Shome et al. [147] investigated feedback mechanisms within ML-focused Jupyter notebooks and presented a taxonomy of explicit (assertions) and implicit (print statements and last cell outputs) feedback approaches. Their study also provided practical recommendations to encourage explicit testing for minimizing technical debt and improving reproducibility. Wang et al. [182] analyzed crashes in ML notebooks and categorized predominant exceptions such as *NameError* and *ValueError*. In addition, they reported that some root causes of these exceptions are related to the features of Jupyter Notebook, such as out-of-order cell executions and errors propagating from previous cells.

5.5.2. Detecting Bugs

Detecting bugs or errors in Jupyter Notebook presents unique challenges due to nonlinear workflows and hidden states, distinguishing them from traditional programming environments [132]. Xin et al. [196] developed a framework to detect anomalies and identify their root causes by combining provenance data with performance metrics. Since Jupyter notebooks do not offer a built-in debugger, the study also presents strategies for debugging Jupyter notebooks by identifying cell workflows and the root cause of errors. Another bug-detecting study introduced a tool called *Vizier*, which helps debug notebooks by maintaining a complete version history of notebooks, cells, and

Table 6: Types of bugs in Jupyter notebooks

Bug Type	**Description**	**Study**
Processing bugs	Issues in memory or computation, i.e., memory overflow, data loss, or performance degradation.	[155, 103, 197, 181, 196]
Implementation bugs	General coding errors that can cause incorrect results or runtime errors, i.e., syntax, logic, variables, or algorithms-related bugs.	[109, 12, 47, 45]
Cell defect	Bugs related to notebook cell renderings, i.e., code cells, Markdown, or output interactions.	[34]
Environments and settings bugs	Problems from misconfigured environments or dependencies that interrupt notebook execution, i.e., missing libraries or lack of dependencies.	[12]
Kernel bugs	Issues related to the Jupyter kernel, such as crashing, freezing, or failure to start that can interrupt execution.	-
Conversion bugs	Errors during notebook conversion from `.ipynb` file type to other formats, i.e., *Nbconvert* bugs.	-
Portability bugs	Failures when running notebooks in different environments, i.e., rendering problems.	-
Connection bugs	Failures connecting to external systems like databases or APIs that can block data access and interrupt workflows.	-

datasets, while tracking potential errors through fine-grained provenance [12]. This tool maintains a complete version history for each notebook and allows reproducible data manipulation through a spreadsheet mode [12].

Another method to detect potential bugs in notebooks is to focus on inconsistencies in variable assignments, output visualization, or logical operations within the notebook [109]. For example, output inconsistency occurs when the output of a notebook cell does not match the expected output. An implementation of such an inconsistency-based approach is the *nbval* notebook validation plugin [34]. *nbval* detects bugs in Jupyter notebooks by automatically executing each code cell and comparing its output with previously stored results. If there is any deviation, such as an unexpected change in numerical values, formatting differences, or execution failures, *nbval* reports a test failure. Wang et al. [181] showed that traditional static analyzers frequently fail to detect runtime tensor shape mismatch bugs in ML codes (particularly TensorFlow) within notebooks. Their study demonstrates how incorporating runtime information improves static analysis capabilities for detecting such common ML-related notebook bugs.

Another type of inconsistency bug is the name-value inconsistency, which occurs when the name of a variable does not accurately reflect its assigned value. Unlike traditional software development environments, notebooks allow for nonlinear execution and frequent variable reuse, increasing the risk

of such name-value inconsistencies [109]. For instance, naming a variable `log_file` while storing a list of all files in a directory can cause a name-value inconsistency bug, as a developer might assume that `log_file` stores a single file name, but it actually contains a list. To avoid confusion, the variable could be renamed to `log_files` or `log_file_list`. Patra and Pradel [109] developed *Nalin*, an automated tool for detecting name-value inconsistencies. This tool uses a neural model to predict whether a variable's name and its assigned value are consistent. *Nalin* employs dynamic program analysis alongside deep learning to monitor variable activities throughout the execution process.

Recently, advancements in LLMs have created promising opportunities for automated error detection and debugging. Grotov et al. [47] performed a detailed analysis of prevalent notebook error patterns and proposed an iterative, LLM-based method for detecting and dynamically resolving errors. Furthermore, Grotov et al. [45] developed an LLM-powered AI agent specifically designed to handle stateful and nonlinear notebook debugging cases, demonstrating that agent-based debugging can effectively resolve complex errors in computational notebooks. Their research also highlights the need to balance automation with human oversight, carefully manage interface complexity, and address security considerations (e.g., sandboxing).

5.5.3. Detecting Data Leakage

In data science projects, data leakage occurs when information from outside the training dataset is used to develop a model [155]. This issue can arise in Jupyter Notebook if code cells are executed in the wrong order. Although executing cells in an arbitrary sequence may seem harmless, it can lead to data leakage bugs, such as unintentionally sharing information between the training and test datasets. To provide warnings about potential data leakage, a static analysis framework has been developed that uses abstract interpretation for intracell static analysis, ensuring both efficiency and guaranteed termination [155]. Data leakage in ML can be detected early in Jupyter notebooks by turning dataframe operations across cells into a graph. Negrini et al. [103] showed that building such a graph-based model can capture dataframe operations across all notebook cells in their actual execution order, allowing them to detect data leakage effectively in Jupyter notebooks. Yang et al. [197] also tackled data leakage in data science notebooks by developing a static analysis tool that models how data flows across notebook cells. This tool tracks the relationships between datasets, transformations,

and model evaluation steps to identify common leakage patterns specific to notebooks, such as the use of test data during preprocessing, the reuse of test data for model selection (known as multi-test leakage), and overlaps between training and test datasets.

Summary of testing and debugging

Researchers have focused on developing automated tools designed explicitly for bug detection in notebooks. Those tools can deal with different types of bugs, for example, name-value inconsistencies [109], data leakage issues [197, 155], and output inconsistency [34]. Researchers explored LLM-powered solutions for debugging and resolving notebook errors [45, 47]. However, more research on notebook testing is needed to deal with other bugs such as performance and kernel-related bugs.

5.6. Visualization in Notebooks

As an important data analysis outcome, visualization is vital in Jupyter notebooks [145, 2]. It also serves software engineering purposes, as it can be used to validate the correctness of a notebook by allowing comparisons between expected and actual outcomes. Additionally, visualization is crucial for managing and enhancing communication within notebook projects.

5.6.1. Empirical Studies on Visualization in Notebooks

Visualization in notebooks can be divided into two main categories: data visualization and workflow visualization. Data visualization refers to the process of displaying the output of a code cell. One of the key features of Jupyter Notebook is the ability to visualize outputs alongside the code cells. Researchers showed that one out of four issues (1,071 out of 4,210) in the studied Jupyter Notebook projects contain at least one visualization-related issue [2]. The study also noticed that visual content is not limited to communicating user interaction design but contains various types of information, such as command-line content or code snippets. Settewong et al. [145] identified nine main reasons for visualizing data when notebook users write code. They analyzed 68 notebooks containing 821 visualizations and categorized them into nine types, showing that the most frequent use cases are for visualizing data distribution (e.g., histograms, box plots) and data frequency (e.g., bar charts, count plots) making these the core tools for understanding and communicating data insights during coding.

In addition, Wootton et al. [194] investigated the exploratory data analysis (EDA) process within Jupyter notebooks and identified distinct temporal and sequential patterns. They proposed quantitative metrics for measuring visualization usage, including revisit counts, representational diversity, and representation velocity. Wang et al. [184] showed that effective visual analytics in notebooks depends not only on what is visualized, but also on how visualizations are designed to fit the notebook environment. Their study identified four key design dimensions that shape the user experience, focusing on how visualizations connect with notebook code, where their data comes from, when they appear, and how easily they can be reused.

Another type of visualization in notebooks is workflow visualization, which helps users understand and navigate the structure of notebook code. Ramasamy et al. [127] introduced a tool called *MARG*, which visualizes data science notebooks as a graph of decisions, forks, and dead-ends instead of a linear list of cells. This approach reveals the nonlinear nature of real-world notebooks and allows users to trace the workflow more effectively. Their study showed that such visualizations significantly improve the understanding of the workflow of data science notebooks.

5.6.2. Interactive Visualization

Interactive visualizations in Jupyter Notebook refer to an interactive interface within notebooks that allows users to analyze data through interactive charts. These features greatly enhance the usability and practicality of Jupyter notebooks by facilitating real-time analysis, monitoring, and interaction with code and output [195, 77]. Interactive visualization also provides valuable insights and support informed decision-making by integrating visual analytics directly into the notebook environment. For instance, Scully-Allison et al. [144] developed a notebook-embedded interactive visualization that traces and synchronizes visual components with code cells. Furthermore, Guo et al. [50] developed *bonXAI*, which integrates interactive explainable AI (XAI) visualizations into Python-based ML workflows within notebooks.

Researchers have proposed interactive dashboards to help users explore and analyze their Jupyter notebooks. Kwon et al. [77] developed a customizable interactive dashboard to support data-centric NLP in Jupyter notebooks. Their system includes built-in text transformation operations and various visual analysis features, allowing users to create interactive dashboards seamlessly. Similarly, Wu et al. [195] introduced a flexible visualization dashboard that complements traditional code cells, offering users an

intuitive interface for data interaction. By importing the necessary library, users can generate visualizations by clicking on dataset columns or creating custom data queries, such as scatter plots. This ensures real-time updates to code cells based on interactions with the dashboard.

Recent works have also explored extending notebook interaction beyond traditional desktop interfaces. In et al. [60] investigated adapting computational notebook interfaces into virtual reality (VR) and introduced embodied gestures for efficient non-linear exploration to enhance notebook navigation.

Summary of visualization in notebooks

Visualization is essential in Jupyter Notebook for data analysis, result validation, and effective communication. Research shows that common applications include visualizing data distributions, statistical measures, and ML workflows [145, 2]. Researchers demonstrated that interactive visualizations enhance usability by enabling real-time interactions with both data and code [195, 77]. Several approaches have been proposed to trace and synchronize visual components with notebook code cells [144, 50]. Additionally, nonlinear workflow visualizations improve the comprehension of complex notebooks, and while established guidelines encourage integration, modularity to improve visual analytics [184, 127, 194].

5.7. Best Practices in Notebooks

It is essential to adhere to best practices in Jupyter notebooks to enhance code quality and collaborative efforts [149, 46]. Unlike conventional software development, Jupyter notebooks are often utilized for exploratory data analysis, often involving frequent adjustments and iterative code execution, resulting in deviations from traditional software development methodologies, such as waterfall or test-driven approaches [179]. This section underscores the importance of adhering to best practices in Jupyter notebooks, with a specific emphasis on ensuring code style consistency and addressing the unique challenges of collaborative usage, especially within the realm of ML projects.

5.7.1. Following Code Style Standards

One of the prevalent challenges in Jupyter notebooks is the inconsistency in code style arising from the absence of enforced coding standards. Re-

searchers showed that notebooks have 1.4 times more stylistic issues than traditional Python scripts [46]. Another study experimentally demonstrated that Jupyter notebooks are inundated with poor-quality code, such as violations of recommended coding practices, unused variables, and deprecated functions [179]. Considering the knowledge-sharing nature of Jupyter notebooks, these poor coding practices might be propagated to future developers.

Researchers have studied the adherence to best practices for coding from various angles. Siddik and Bezemer [149] evaluated the adherence to the PEP-8 code style guidelines in ML code in Jupyter notebooks. Their findings indicate that ML notebooks generally exhibit lower code quality than non-ML notebooks, with notable discrepancies in how packages and libraries are managed. To address this, a static analysis tool called *Pynblint* [122, 123] was implemented to identify quality issues in Jupyter notebooks. In addition, Candela et al. [15] studied the quality of Jupyter Notebook projects published by GLAM (Galleries, Libraries, Archives, and Museums) institutions within the cultural heritage sector. Their evaluation criteria focused on documentation, code readability, and metadata usage, identifying that these areas need improvement for better traceability and overall quality.

5.7.2. Best Practices for Collaborative Use

Researchers showed that it is necessary to develop and validate best practices for collaborative notebook use and the tools required to enforce these practices [119]. Quaranta et al. [121] introduced a catalog of collaboration-specific best practices for Jupyter Notebook, aiming to improve teamwork, reproducibility, and code quality in data science workflows. The catalog emphasizes practices like using version control, structuring code with modular functions, documenting with Markdown, cleaning and organizing cells, and separating exploratory from production notebooks. It also encourages open sharing and the use of self-contained environments to ensure reproducibility. To manage editing conflicts during real-time collaboration effectively, Wang et al. [170] proposed *PADLOCK*, which provides conflict-resolution mechanisms such as cell-level and variable-level access control and parallel cell groups. Their empirical evaluation demonstrated reduced editing conflicts and improved support for diverse collaboration styles. Together, these guidelines and tools support more maintainable and collaborative notebook development.

Summary of best practices in notebooks

Unlike traditional software development environments, Jupyter notebooks often prioritize data processing and exploration, which can lead to deviations from established software engineering practices. No studies have identified or explored the ML-specific best practices in notebooks. Building upon the study by Pimentel et al. [116], Grotov et al. [46], which identified good and bad coding practices, there is a need for an impact analysis of these coding practices.

5.8. Cell Execution Order

Notebook cells can be executed independently and out of order, making it challenging for developers to manage the global variables that notebooks keep persistent in memory [93]. Developers should strive to maintain a linear order of code execution in Jupyter notebooks, as researchers noticed that the nonlinear execution order is one of the main challenges while reusing notebooks [151, 93]. These non-linear executions in Jupyter notebooks create difficulties for users in understanding and tracing dependencies between code cells, which can cause confusion, errors, and challenges in reproducing results [14]. To address this, Brown et al. [14] introduced Dataflow Notebooks (*DFNBs*), a system that can locate variable definitions and track variable references as explicit data dependencies to overcome out-of-order cell executions.

Developing reproducible ML pipelines can also help address notebook execution order issues. In this context, the ML pipelines in notebooks consist of interconnected code cells where each step of the ML process (such as preprocessing, model training, and evaluation) is represented as a separate code cell [61, 99]. Jiang et al. [61] developed an approach to determine the correct execution order by identifying and extracting the underlying structure of a notebook by building a labeled dependency graph. In this graph, each cell is represented as a node labeled with a specific ML stage (e.g., data collection, training, evaluation), and edges represent data dependencies between cells. These labeled cells are then reordered to maintain the ML pipeline execution flow. A tool named *Ploomber* has been developed to manage and automate the execution sequence of notebook cells [99]. *Ploomber* transforms notebooks into a structured format, making a Jupyter notebook function like a reproducible pipeline with a single execution flow. This tool allows users to break down large notebooks into smaller, manageable tasks

that are connected through a clear execution order. This structured approach helps prevent common issues associated with out-of-order execution, such as undefined variables or incorrect data states.

Summary of cell execution order

Nonlinear cell execution in Jupyter notebooks leads to persistent memory challenges, making it difficult for developers to track variable dependencies and ensure reproducibility [93, 151]. To address this, researchers have developed methods like *DFNBs* for tracking dependencies [14], and tools like *Ploomber* [99] to automate and streamline notebook workflows. Researchers also developed solutions for reproducible ML pipelines, helping resolve notebook execution order issues [61, 99].

5.9. AI-based Coding Assistance for Notebooks

Jupyter notebooks offer a convenient way to write and execute code in a single shareable file for exploratory data analysis and insight finding [72]. However, users with limited coding experience may struggle to participate in the analysis process quickly. AI-based coding assistance for Jupyter notebooks can be a solution for these users by generating code snippets based on various inputs, such as natural language instructions or notebook contexts. Researchers have explored automatic generation of notebook codes with low-code strategies to support data analysis [22, 201]. For example, Chen et al. [22] developed a low-code interaction panel to recommend follow-up questions to guide the next steps in exploratory data analysis [22]. This approach helps users visualize the structure of their data science workflow through a tree-based representation of the notebook cells. Notebook users can generate code cells based on natural language instructions. Yin et al. [201] proposed an approach to enable users to describe their desired outcome in English and automatically generate the necessary code. However, this solution is designed specifically for tasks involving the *Pandas* library. Huang et al. [58] further expanded code generation methods by developing *DataCoder*, a dual-encoder model that generates contextualized data-wrangling code cells by encoding textual, code, and tabular data separately to improve accuracy. Users can also generate notebook code cells by interacting with appropriate UI scaffolds. For instance, Cheng et al. [23] developed *BISCUIT*, a JupyterLab extension leveraging ephemeral user interfaces generated by

LLMs, enabling interactive elements (e.g., sliders, dropdowns) to facilitate more understandable and exploratory ML coding tasks.

Beyond code generation, studies have emphasized the importance of mediated interactions and enhanced code-cell execution assistance. For educational environments, George and Dewan [38] introduced *NotebookGPT*, embedding GPT interactions within Jupyter notebooks to encourage effective prompt usage and reduce the direct copying behavior among students through programmatic mediation strategies. In parallel, efforts have also been directed toward defining new design approaches for seamlessly integrating LLM support into notebook workflows, potentially improving productivity for diverse developer groups [188]. Similarly, Brault et al. [13] demonstrated a solution for generating new notebooks by retrieving relevant past experiments based on user-specified problem configurations. Lastly, McNutt et al. [97] investigated AI-powered notebook assistants which recommend optimal code cell execution pathways by analyzing data dependencies alongside the code itself, highlighting another crucial dimension of notebook assistance beyond generating code snippets.

Summary of AI-based coding assistance for notebook

AI-based coding assistance in Jupyter notebooks helps users generate code snippets using natural language instructions or notebook contexts. Researchers have explored a range of techniques, including low-code interaction panels [22, 201], context-aware AI models [58], and interactive UI scaffolds [23] to streamline notebook code. Additionally, AI-powered assistants integrated into notebooks facilitate code-cell execution [97], workflow guidance [13], and enhanced user interactions [38] to make notebooks more intuitive for broader audiences.

5.10. Supporting other Programming Paradigms

To support the use of multiple programming languages within a notebook, researchers have proposed polyglot execution environments [104]. A polyglot notebook system allows code cells to interact directly with data structures and invoke functions or methods from different programming languages [104, 114]. This system facilitates direct object sharing and function calling across different programming languages, eliminating the need for data serialization between languages. Hence, a polyglot notebook system can improve the

interoperability of various programming languages within a single notebook environment.

Furthermore, block-based visual programming within notebooks provides a graphical interface for programming, making the structure and logic of the code visually clear [168]. Integrating block-based programming into Jupyter notebooks enhances coding practices, simplifies syntax, reduces errors, and lowers the learning curve for non-expert developers. Verano Merino et al. [168] proposed this integration to provide a user-friendly interface for programming and data science tasks, thus facilitating the adoption of computational notebooks across different domains. Similarly, Weber et al. [187] developed a multi-paradigm editor within the Jupyter ecosystem that integrates graphical and textual views with automated synchronization, demonstrating enhanced usability, lower workload, and prevention of execution order errors.

Summary of supporting other programming paradigms

Supporting multiple programming paradigms in notebooks enhances flexibility by enabling cross-language execution and intuitive visual programming interfaces. For example, polyglot execution environments enable interaction between multiple programming languages within a notebook [104, 114]. Additionally, block-based visual programming enhances accessibility by providing a graphical interface, simplifying syntax, and supporting non-expert developers in computational notebooks [168, 187].

5.11. Datasets of Notebooks

Publicly available datasets of Jupyter notebooks have been instrumental in advancing research on computational notebooks, enabling large-scale empirical studies on their usage, quality, and evolution. These datasets typically include collections of notebooks from platforms such as GitHub and Kaggle, often annotated with metadata such as commit histories, bug-related information, or workflow analysis. In our systematic literature review, we discovered 18 distinct datasets that are publicly available. Table 7 lists all the public datasets of Jupyter notebooks used in the literature, highlighting their publication years, public URLs, the description of sources, and research articles that used them. Among these, 11 were gathered from GitHub, 5 from Kaggle, and 2 from both GitHub and Kaggle.

Table 7: Datasets of Jupyter notebooks used in literature

Year	Public URL	Description	Used by
2024	https://github.com/ISE-Research/DistilKaggle	542,051 notebooks from Kaggle with 34 code quality metrics	[40, 39]
2024	https://doi.org/10.6084/m9.figshare.26372140	297,800 ML notebooks from GitHub and Kaggle to study feedback mechanisms (e.g., assertions and print)	[147]
2024	https://zenodo.org/records/11396773	138,376 notebooks from Kaggle used to study data preprocessing practices in ML development	[43]
2024	https://github.com/Jun-jie-Huang/CoCoNote	58,221 code generation examples from GitHub Jupyter notebooks to study data-wrangling	[58]
2024	https://huggingface.co/datasets/JetBrains-Research/jupyter-errors-dataset	10,000 GitHub notebooks containing at least one thrown exception	[47]
2024	https://zenodo.org/records/15114367	64,031 ML notebooks (61k GitHub, 2.7k Kaggle) with 92,542 crashes	[182]
2024	https://zenodo.org/records/13836922	113 notebooks containing 342 recommended cells	[7]
2022	https://zenodo.org/records/6383115	847,881 notebooks from GitHub to study code structure and style	[46]
2022	https://github.com/bugs-jupyter/empirical-study	105 notebooks from GitHub with 14,740 bug-related commits	[27]
2021	https://zenodo.org/records/4468523	248,761 notebooks from Kaggle (the *KGTorrent* dataset)	[120, 24, 41, 126, 40, 43]
2021	https://zenodo.org/records/5109482	267,602 notebooks from Kaggle	[121]
2021	https://ibm.biz/Bdfpk6	3,944 notebooks from Kaggle	[91]
2021	https://zenodo.org/records/7109939	470 notebooks from GitHub for analyzing the workflow of data science code	[127]
2020	https://github.com/SMAT-Lab/SnifferDog/blob/master/dataset/all.github.urls	100,000 notebook projects from GitHub, with 507 notebook projects which are executed for reproducibility	[180]
2020	https://zenodo.org/records/3836691	6,000 notebooks from GitHub with code clones	[73]
2020	https://osf.io/9q4wp	2,574 notebooks from GitHub	[125]
2020	https://figshare.com/s/4c5f96bc7d8a8116c271	200 notebooks from GitHub with meaningful commit histories	[90]
2019	https://zenodo.org/records/2592524	1.4 million notebooks from GitHub	[115, 105, 116, 177, 159]
2018	https://library.ucsd.edu/dc/object/bb2733859v	1.25 million notebooks from GitHub	[135, 109, 84]

These datasets vary significantly in size and purpose. Some datasets are intended for general research on Jupyter Notebook. For instance, the dataset from Grotov et al. [46] contains 847,881 notebooks from GitHub, supporting analyses of notebook usage trends. Similarly, the *KGTorrent* dataset includes 248,761 notebooks from Kaggle, offering insights into the structured use of

notebooks in data science competitions [120]. Additionally, large-scale collections such as the 1.4 million GitHub notebooks compiled by Pimentel et al. [115] and the 1.25 million GitHub notebooks by Rule et al. [135] provide a comprehensive view of the Jupyter Notebook platform. *DistilKaggle* [39, 40] extends KGTorrent, containing 542,051 Kaggle notebooks annotated with 34 code quality metrics.

Specialized datasets, such as the one by De Santana et al. [26], focus on bug-related commits, including 105 Jupyter notebooks from GitHub with 14,740 associated commits, making it highly relevant for debugging and maintenance studies. Similarly, the dataset from Wang et al. [180] emphasizes the reproducibility of notebooks, while the dataset by Koenzen et al. [73] specifically supports research on code clone detection. Another focused dataset from Ramasamy et al. [127] provides data science code for analyzing the workflow. Furthermore, Grotov et al. [47] built a dataset of 10,000 GitHub notebooks explicitly containing exceptions and error outputs, and Huang et al. [58] focused on the new task of contextualized data-wrangling code generation leveraging code, data, and text contexts. Aydin et al. [7] prepared a *CelRec-DB* dataset containing 342 recommended code cells worked in Jupyter notebooks. Unfortunately, one of the used datasets [63] was not accessible due to a permission issue and therefore excluded from the table.

Several recent datasets specifically target ML notebooks to address emerging research challenges and trends. For instance, Golendukhina and Felderer [43] curated a dataset of 138,376 Kaggle notebooks that contain ML-specific data preprocessing API usage based on the *KGTorrent* dataset [120]. Similarly, Wang et al. [182] collected 64,031 ML notebooks that contain execution errors (notebook crashes) to support research on debugging practices and error mitigation. Shome et al. [147] constructed a dataset containing 297,800 ML notebooks from both GitHub and Kaggle, including 89.6 thousand assertions, 1.4 million print statements, and 1 million last cell statements.

Summary of datasets of notebooks

Publicly available Jupyter Notebook datasets from GitHub and Kaggle serve various purposes, from analyzing overall notebook usage to studying reproducibility and reusability [180, 73]. Some collections, such as *KGTorrent* [120] and *DistilKaggle* [39, 40], focus on structured notebook usage, while others, like *CelRec-DB* [7], capture recommended code cells. Additionally, some datasets specifically address machine learning notebooks, supporting research on preprocessing API usage [43], execution errors [47], and debugging practices [182, 26].

6. Future Research Directions

The future directions of software engineering research on Jupyter Notebook are vast and have great potential to advance the field of data science and computational research. In this section, we outline key future research directions.

Research Direction 1: Improving Jupyter Notebook Code Search through Natural Language Summarization

As discussed in Section 5.1.2, existing code search techniques in Jupyter Notebook primarily use keyword matching or deep learning techniques to search the code directly based on natural language queries [186, 84]. However, these methods can be further improved by incorporating natural language summaries of code functionality into the search process. Future research should explore leveraging LLMs to automate the summarization of individual notebook code blocks, producing concise and descriptive explanations of their purposes. Such summaries not only enrich search accuracy, but also aid users in quickly assessing retrieved results without having to read and interpret the code directly.

Research Direction 2: Building a Broad Set of Notebook-Specific Code Refactoring Tools

In Section 5.3.1, we discuss different code refactoring tools that offer basic functionalities for notebooks, such as splitting cells [146] and reordering notebook cells [159]. Future research should expand the capabilities of these refactoring tools, addressing common patterns such as the distribution of exploratory workflows across multiple notebooks [135]. In addition, researchers

can target other refactoring features such as detecting and eliminating duplicate code, extracting reusable functions, and restructuring notebooks by merging cells to improve the readability and maintainability of notebooks.

Research Direction 3: Generating Documentation for Groups of Dependent Code Cells

Existing methods for automated documentation generation in notebooks (discussed in Section 5.4) focus primarily on the generation of documentation at the individual cell level. For example, *Cell2Doc* generates Markdown descriptions for single cells [100], and *HeaderGen* generates structural headings into the notebook [167]. However, these single-cell approaches often fail to capture broader contexts or overarching objectives spanning multiple interrelated cells that perform tasks collectively. Since notebook users intentionally cluster related operations into adjacent groups of cells to boost productivity and organization [183], future research should aim to automatically generate context-aware documentation for these clusters. Utilizing analysis techniques such as dependency graph extraction and runtime tracing, researchers can identify groups of cells and automatically produce Markdown summaries.

Research Direction 4: Designing Automated Bug Detection and Remediation Tools Tailored to Notebooks

Since bug analysis is crucial to maintaining the quality of Jupyter notebooks (see Section 5.5), future research could focus on developing automated tools specifically designed for notebook-related bugs. Although some existing approaches address issues such as name-value inconsistencies detection [109] and data leakage detection [197], further research is needed to tackle more complex bugs, such as cell defects and kernel failure. Future studies should focus on addressing notebook code cell defects, which arise from missing dependencies, incorrect execution order, or undefined variables, as well as kernel crashes, often caused by excessive resource consumption, infinite loops, or faulty package interactions [26]. Advanced techniques, including static and dynamic code analysis, execution flow tracking, and visualization-based debugging, can be leveraged to efficiently detect, categorize, and resolve these bugs.

Research Direction 5: Conducting Empirical Studies on the Impact of Best Practices for Coding

As outlined in Section 5.7, little is known about the actual influence of following best practices for coding in Jupyter notebooks. Future studies should

conduct empirical studies to quantify the effects of following best practices for coding based on categorized effective and ineffective practices [116]. Systematic studies comparing compliant and non-compliant notebooks could provide concrete empirical evidence highlighting the advantages of rigorous adherence to recommended practices. Such evidence would strongly motivate wider adoption of disciplined coding standards among notebook users.

Research Direction 6: Detecting and Resolving Duplicate Execution Numbers in Notebook Cells

In this SLR, we discuss the studies on execution order in Section 5.8, but we found no existing studies that explicitly address the issue of duplicate cell execution numbers. This issue occurs when notebook kernels are restarted or when cells are repeatedly executed without proper tracking, resulting in identical cell execution indices. Future studies should develop methods for detecting, tracking and resolving these duplicate execution numbers. Promising solutions include tools for logging, visualizing, and auditing the complete notebook execution history, incorporating timestamps and cell dependency graphs to facilitate clear identification and resolution of duplicate execution numbers.

Research Direction 7: Enhancing AI-Powered Code Generation and Workflow Assistance in Jupyter Notebook

In Section 5.9, we discuss existing solutions for generating code snippets based on natural language instructions for specific libraries like Pandas [201]. These solutions possess limited scope with a single library and fail to support broader, multi-library, multi-step data science workflow automation. To address these limitations, future research should leverage advanced AI techniques (e.g., large language models), to improve conversational AI-powered assistants for automated code generation by multi-step workflow automation. For example, interactive low-code panels that recommend follow-up analytical questions and visualize workflow structures [22] could be refined and augmented through continuous conversational AI interactions. By dynamically adapting their recommendations based on real-time user inputs and evolving contexts, these AI-powered assistants could offer more responsive and intuitive exploratory data analysis experiences for notebook users.

7. Threats to Validity

Internal validity: One potential threat to the internal validity of this study is the subjective selection of studies. While we applied clear inclusion and exclusion criteria to select the studies related to software engineering practices in Jupyter Notebook, manual selection may introduce inconsistencies. To mitigate this risk, authors independently labelled the studies and resolved disagreements through discussion. In addition, the data extraction form, which defines the information collected on software engineering practices in Jupyter Notebook, may influence validity. To mitigate this, we refined the form through collaborative discussion between the first and third authors to ensure the form aligned with the research questions and adequately covered the scope. Despite these efforts, some subjectivity is inherent, which we acknowledge as a limitation of the study.

External validity: While most studies analyzed in this review focus on Python notebooks, which may limit generalizability, this focus is justified by the fact that 93% of notebooks available on GitHub are written in Python [63]. This high percentage highlights the relevance of Python-based research within the broader context of Jupyter Notebook usage. Another concern regarding external validity is that our findings primarily apply to software engineering research on Jupyter notebooks. Future research should expand the scope by investigating other types of research on notebooks to improve the generalizability of the results.

Construct validity: One potential threat to the construct validity of our review is related to the study selection process. We conducted a comprehensive search of all academic articles listed in the DBLP bibliography, intentionally excluding non-academic sources such as blog posts. The DBLP bibliography is widely recognized and frequently used by software engineering researchers to identify relevant articles in systematic literature reviews [193, 32, 82].

Conclusion validity: Conclusion validity threats can occur due to potential biases in interpreting and synthesizing data, particularly when grouping studies under various software engineering topics. This process required careful interpretation and collaborative discussions between the first and the third authors to finalize the list of software engineering topics. To minimize these threats, we utilized a Trello board[13] to categorize the studies according

[13] `https://trello.com/`

to different software engineering topics and held regular discussions to reach a consensus on the final list.

8. Conclusion

In this systematic literature review, we provide a comprehensive overview of the research landscape of software engineering in Jupyter notebooks. Our review uncovers critical insights and trends that have shaped the field's current state. We utilized a thorough methodology involving identifying, screening, and analyzing 146 relevant research articles (supplementary data are available here[14]). The most important findings of our study are:

- Most studies were published in conference venues, indicating the field is rapidly evolving. Interestingly, the most frequent venues were HCI-related instead of core SE venues. The Conference on Human Factors in Computing Systems (CHI) published 15 studies, whereas the International Conference on Software Engineering (ICSE) published 6.

- Notebook-specific solutions for software engineering issues, such as testing, refactoring, and documentation, are relatively underexplored. For example, resolving duplicate execution numbers, refactoring inter-notebook clones, and generating group documentation for coherent code cells are future directions derived from our study.

- There is a growing need to integrate modern AI-based solutions into Jupyter notebooks to support various software engineering topics, including code search and code generation. By summarizing code functionality and narratives in accessible language, users can more effectively search for the relevance of existing code and facilitate its reuse. Additionally, incorporating advanced AI models can improve code generation and management within Jupyter notebooks.

- The majority of replication packages are hosted on GitHub, raising concerns about long-term availability and adherence to open science principles, as GitHub repositories can be volatile. In contrast, only 13 studies used permanent repositories such as Zenodo or Figshare, aligning better with open science best practices.

[14] `https://zenodo.org/records/15226809`

This systematic review focuses on recent advances in the area of software engineering applied to Jupyter notebooks. Our analysis and findings can assist researchers and practitioners in addressing challenges such as effectively integrating notebooks into production, ensuring seamless code functionality, and promoting open science principles and reproducibility. By leveraging this knowledge, researchers can work towards optimizing notebooks for greater user-friendliness, reliability, and scalability in real-world applications.

References

[1] Adams, K., Vilkomir, A., Hills, M., 2023. A Comparison of Machine Learning Code Quality in Python Scripts and Jupyter Notebooks. Journal of Computing Sciences in Colleges 39, 96–108.

[2] Agrawal, V., Lin, Y.H., Cheng, J., 2022. Understanding the Characteristics of Visual Contents in Open Source Issue Discussions: A Case Study of Jupyter Notebook, in: Proceedings of the 26th International Conference on Evaluation and Assessment in Software Engineering, pp. 249–254.

[3] Ahmad, R., Manne, N.N., Malik, T., 2022. Reproducible Notebook Containers using Application Virtualization, in: 18th International Conference on e-Science (e-Science), IEEE. pp. 1–10.

[4] Al-Gahmi, A., Zhang, Y., Valle, H., 2022. Jupyter in the Classroom: An Experience Report, in: Proceedings of the 53rd ACM Technical Symposium on Computer Science Education-Volume 1, pp. 425–431.

[5] Almugbel, R., Hung, L.H., Hu, J., Almutairy, A., Ortogero, N., Tamta, Y., Yeung, K.Y., 2018. Reproducible Bioconductor Workflows using Browser-based Interactive Notebooks and Containers. Journal of the American Medical Informatics Association 25, 4–12.

[6] Amoudi, G., Tbaishat, D., 2023. Interactive Notebooks for Achieving Learning Outcomes in a Graduate Course: A Pedagogical Approach. Education and Information Technologies 28, 16669–16704.

[7] Aydin, S., Mertens, D., Xu, O., 2024. An Automated Evaluation Approach for Jupyter Notebook Code Cell Recommender Systems, in: Lichter, H., Wild, N., Sunetnanta, T., Anwar, T. (Eds.), Proceedings

of the 12th International Workshop on Quantitative Approaches to Software Quality co-located with the 31st Asia Pacific Software Engineering Conference (APSEC 2024), pp. 4–11.

[8] Ayobi, A., Hughes, J., Duckworth, C.J., Dylag, J.J., James, S., Marshall, P., Guy, M., Kumaran, A., Chapman, A., Boniface, M., et al., 2023. Computational Notebooks as Co-Design Tools: Engaging Young Adults Living with Diabetes, Family Carers, and Clinicians with Machine Learning Models, in: Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems, pp. 1–20.

[9] Barba, L.A., Barker, L.J., Blank, D.S., Brown, J., Downey, A.B., George, T., Heagy, L.J., Mandli, K.T., Moore, J.K., Lippert, D., et al., 2019. Teaching and Learning with Jupyter. Recuperado: https://jupyter4edu. github. io/jupyter-edu-book , 1–77.

[10] Bavishi, R., Laddad, S., Yoshida, H., Prasad, M.R., Sen, K., 2021. VizSmith: Automated Visualization Synthesis by Mining Data-Science Notebooks, in: 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE. pp. 129–141.

[11] Beg, M., Taka, J., Kluyver, T., Konovalov, A., Ragan-Kelley, M., Thiéry, N.M., Fangohr, H., 2021. Using Jupyter for Reproducible Scientific Workflows. Computing in Science & Engineering 23, 36–46.

[12] Brachmann, M., Spoth, W., 2020. Your Notebook is not Crumby Enough, REPLace it, in: Conference on Innovative Data Systems Research (CIDR), pp. 1–16.

[13] Brault, Y., El Amraoui, Y., Blay-Fornarino, M., Collet, P., Jaillet, F., Precioso, F., 2023. Taming the Diversity of Computational Notebooks, in: Proceedings of the 27th ACM International Systems and Software Product Line Conference-Volume A, pp. 27–33.

[14] Brown, C., Alhoori, H., Koop, D., 2023. Facilitating Dependency Exploration in Computational Notebooks, in: Proceedings of the Workshop on Human-In-the-Loop Data Analytics, pp. 1–7.

[15] Candela, G., Chambers, S., Sherratt, T., 2023. An Approach to Assess the Quality of Jupyter Projects Published by GLAM Institutions. Journal of the Association for Information Science and Technology .

[16] Cao, P., 2024. Jupyter Notebook Attacks Taxonomy: Ransomware, Data Exfiltration, and Security Misconfiguration, in: SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE. pp. 750–754. doi:10.1109/scw63240.2024.00106.

[17] Casseau, C., Falleri, J.R., Degueule, T., Blanc, X., 2023. MOON: Assisting Students in Completing Educational Notebook Scenarios, in: Symposium on Visual Languages and Human-Centric Computing (VL/HCC), IEEE. pp. 157–167.

[18] Chandel, S., Clement, C.B., Serrato, G., Sundaresan, N., 2022. Training and Evaluating a Jupyter Notebook Data Dcience Assistant. arXiv preprint arXiv:2201.12901 .

[19] Chanson, A., El Outa, F., Labroche, N., Marcel, P., Peralta, V., Verdeaux, W., Jacquemart, L., 2022. Generating Personalized Data Narrations from EDA Notebooks, in: DOLAP, pp. 91–95.

[20] Chattopadhyay, S., Feng, Z., Arteaga, E., Au, A., Ramos, G., Barik, T., Sarma, A., 2023. Make It Make Sense! Understanding and Facilitating Sensemaking in Computational Notebooks. arXiv preprint arXiv:2312.11431 .

[21] Chattopadhyay, S., Prasad, I., Henley, A.Z., Sarma, A., Barik, T., 2020. What's wrong with computational notebooks? Pain points, needs, and design opportunities, in: Proceedings of the 2020 CHI conference on human factors in computing systems, pp. 1–12.

[22] Chen, C., Hoffswell, J., Guo, S., Rossi, R., Chan, Y.Y., Du, F., Koh, E., Liu, Z., 2023. WHATSNEXT: Guidance-enriched Exploratory Data Analysis with Interactive, Low-Code Notebooks, in: 2023 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), IEEE. pp. 209–214.

[23] Cheng, R., Barik, T., Leung, A., Hohman, F., Nichols, J., 2024. BISCUIT: Scaffolding LLM-Generated Code with Ephemeral UIs in Computational Notebooks, in: IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC '24), IEEE. pp. 13–23. doi:10.1109/vl/hcc60511.2024.00012.

[24] Choetkiertikul, M., Hoonlor, A., Ragkhitwetsagul, C., Pongpaichet, S., Sunetnanta, T., Settewong, T., Jiravatvanich, V., Kaewpichai, U., 2023. Mining the Characteristics of Jupyter Notebooks in Data Science Projects, in: 20th International Conference on Mining Software Repositories: Registered Reports (MSR-RR), IEEE.

[25] Cunha, R.L., Real, L.C.V., Souza, R., Silva, B., Netto, M.A., 2021. Context-aware Execution Migration Tool for Data Science Jupyter Notebooks on Hybrid Clouds, in: 2021 IEEE 17th International Conference on eScience (eScience), IEEE. pp. 30–39.

[26] De Santana, T.L., Neto, P.A.D.M.S., De Almeida, E.S., Ahmed, I., 2024a. Bug Analysis in Jupyter Notebook Projects: An Empirical Study. ACM Transactions on Software Engineering and Methodology 33, 1–34. doi:10.1145/3641539.

[27] De Santana, T.L., Neto, P.A.D.M.S., De Almeida, E.S., Ahmed, I., 2024b. Bug Analysis in Jupyter Notebook Projects: An Empirical Study. ACM Transactions on Software Engineering and Methodology 33, 1–34. doi:10.1145/3641539.

[28] Deo, N., Glavic, B., Kennedy, O., 2022. Runtime Provenance Refinement for Notebooks, in: Proceedings of the 14th International Workshop on the Theory and Practice of Provenance, ACM. pp. 1–4.

[29] Dong, H., 2021. A Qualitative Study of Cleaning in Jupyter Notebooks, in: 29th Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ACM. pp. 1663–1665.

[30] Dong, H., Zhou, S., Guo, J.L., Kästner, C., 2021. Splitting, Renaming, Removing: A Study of Common Cleaning Activities in Jupyter Notebooks, in: 2021 36th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW), IEEE. pp. 114–119.

[31] Duan, J., Dennis, S., 2023. Jup2kub: Algorithms and a system to translate a jupyter notebook pipeline to a fault tolerant distributed kubernetes feployment. arXiv preprint arXiv:2311.12308 .

[32] Duarte, C.H.C., 2019. The Quest for Productivity in Software Engineering: A Practitioners Systematic Literature Review, in:

IEEE/ACM International Conference on Software and System Processes (ICSSP), IEEE. pp. 145–154.

[33] Fangohr, H., Beg, M., Bergemann, M., Bondar, V., Brockhauser, S., Carinan, C., Costa, R., Fortmann, C., Marsa, D.F., Giovanetti, G., et al., 2019. Data Exploration and Analysis with Jupyter Notebooks, in: International Conference on Accelerator and Large Experimental Physics Control Systems, pp. 799–806.

[34] Fangohr, H., Fauske, V., Kluyver, T., Albert, M., Laslett, O., Cortés-Ortuño, D., Beg, M., Ragan-Kelly, M., 2020. Testing with Jupyter Notebooks: NoteBook VALidation (nbval) Plug-in for PyTest. arXiv preprint arXiv:2001.04808 .

[35] Fitzpatrick, P.C., Manning, J.R., 2023. Davos: a Python "smuggler" for Constructing Lightweight Reproducible Notebooks. SoftwareX 25, 1–13.

[36] Fruchart, M., Guinhouya, B., Pelayo, S., Vilhelm, C., Lamer, A., 2022. Jupyter Notebooks for Introducing Data Science to Novice Users, in: Challenges of Trustable AI and Added-Value on Health. IOS Press, pp. 823–824.

[37] Gadhave, K., Cutler, Z., Lex, A., 2024. Persist: Persistent and Reusable Interactions in Computational Notebooks. Computer Graphics Forum 43. doi:10.1111/cgf.15092.

[38] George, S.D., Dewan, P., 2024. NotebookGPT – Facilitating and Monitoring Explicit Lightweight Student GPT Help Requests During Programming Exercises, in: Companion Proceedings of the 29th International Conference on Intelligent User Interfaces, ACM. pp. 62–65. doi:10.1145/3640544.3645234.

[39] Ghahfarokhi, M.M., Asadi, A., Asgari, A., Mohammadi, B., Rizi, M.B., Heydarnoori, A., 2024a. Predicting the Understandability of Computational Notebooks through Code Metrics Analysis. arXiv preprint arXiv:2406.10989 .

[40] Ghahfarokhi, M.M., Asgari, A., Abolnejadian, M., Heydarnoori, A., 2024b. DistilKaggle: A Distilled Dataset of Kaggle Jupyter Notebooks, in: Proceedings of the 21st International Conference on Mining

Software Repositories (MSR '24), ACM. pp. 647–651. doi:10.1145/3643991.3644882.

[41] Ghahfarokhi, M.M., Khademian, A., Kianiangolafshani, S., Asadi, A., Jahantigh, H., Heydarnoori, A., 2024c. Beyond Syntax: Unleashing the Power of Computational Notebooks Code Metrics in Documentation Generation, in: Proceedings of the IEEE/ACM 3rd International Conference on AI Engineering - Software Engineering for AI (CAIN '24), ACM. pp. 278–279. doi:10.1145/3644815.3644979.

[42] Gharehyazie, M., Ray, B., Keshani, M., Zavosht, M.S., Heydarnoori, A., Filkov, V., 2019. Cross-project Code Clones in GitHub. Empirical Software Engineering 24, 1538–1573.

[43] Golendukhina, V., Felderer, M., 2024. Unveiling Data Preprocessing Patterns in Computational Notebooks, in: 50th Euromicro Conference on Software Engineering and Advanced Applications (SEAA '24), IEEE. pp. 114–121. doi:10.1109/seaa64295.2024.00025.

[44] González-Albo, B., Bordons, M., 2011. Articles vs. Proceedings Papers: Do they Differ in Research Relevance and Impact? A Case Study in the Library and Information Science Field. Journal of Informetrics 5, 369–381.

[45] Grotov, K., Borzilov, A., Krivobok, M., Bryksin, T., Zharov, Y., 2024a. Debug Smarter, Not Harder: AI Agents for Error Resolution in Computational Notebooks. arXiv preprint arXiv:2410.14393 .

[46] Grotov, K., Titov, S., Sotnikov, V., Golubev, Y., Bryksin, T., 2022. A Large-Scale Comparison of Python Code in Jupyter Notebooks and Scripts, in: 19th International Conference on Mining Software Repositories (MSR), pp. 353–364.

[47] Grotov, K., Titov, S., Zharov, Y., Bryksin, T., 2024b. Untangling Knots: Leveraging LLM for Error Resolution in Computational Notebooks. arXiv preprint arXiv:2405.01559 .

[48] Grüning, B.A., Rasche, E., Rebolledo-Jaramillo, B., Eberhard, C., Houwaart, T., Chilton, J., Coraor, N., Backofen, R., Taylor, J., Nekrutenko, A., 2017. Jupyter and Galaxy: Easing Entry Barriers

into Complex Data Analyses for Biomedical Researchers. PLoS computational biology 13, e1005425.

[49] Guo, D., Lu, S., Duan, N., Wang, Y., Zhou, M., Yin, J., 2022. UniXcoder: Unified cross-modal pre-training for code representation, in: Muresan, S., Nakov, P., Villavicencio, A. (Eds.), Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL '22), Association for Computational Linguistics, Dublin, Ireland. pp. 7212–7225. doi:10.18653/v1/2022.acl-long.499.

[50] Guo, G., Arendt, D., Endert, A., 2024. Explainability in JupyterLab and Beyond: Interactive XAI Systems for Integrated and Collaborative Workflows. arXiv preprint arXiv:2404.02081 .

[51] Haedrich, C., Petras, V., Petrasova, A., Blumentrath, S., Mitasova, H., 2023. Integrating GRASS GIS and Jupyter Notebooks to Facilitate Advanced Geospatial Modeling Education. Transactions in GIS 27, 686–702.

[52] Hamrick, J.B., 2016. Creating and Grading IPython/Jupyter Notebook Assignments with NbGrader, in: Proceedings of the 47th ACM Technical Symposium on Computing Science Education, pp. 242–242.

[53] Hao, B., Sun, W., Yu, Y., Xie, G., 2017. Developing Healthcare Data Analytics APPs with Open Data Science Tools, in: Informatics for Health: Connected Citizen-Led Wellness and Population Health. IOS Press, pp. 176–180.

[54] Harrison, G., Bryson, K., Bamba, A.E.B., Dovichi, L., Binion, A.H., Borem, A., Ur, B., 2024. JupyterLab in Retrograde: Contextual Notifications That Highlight Fairness and Bias Issues for Data Scientists, in: Proceedings of the CHI Conference on Human Factors in Computing Systems, ACM. pp. 1–19. doi:10.1145/3613904.3642755.

[55] Head, A., Hohman, F., Barik, T., Drucker, S.M., DeLine, R., 2019. Managing Messes in Computational Notebooks, in: CHI Conference on Human Factors in Computing Systems, ACM. pp. 1–12.

[56] Horiuchi, M., Sasaki, Y., Xiao, C., Onizuka, M., 2022a. JupySim: Jupyter Notebook Similarity Search System, in: 25th International Conference on Extending Database Technology (EDBT), pp. 554–557.

[57] Horiuchi, M., Sasaki, Y., Xiao, C., Onizuka, M., 2022b. Similarity Search on Computational Notebooks. arXiv preprint arXiv:2201.12786 .

[58] Huang, J., Guo, D., Wang, C., Gu, J., Lu, S., Inala, J.P., Yan, C., Gao, J., Duan, N., Lyu, M.R., 2024a. Contextualized Data-Wrangling Code Generation in Computational Notebooks, in: Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering, ACM. pp. 1282–1294. doi:10.1145/3691620.3695503.

[59] Huang, Y., Luo, J., Yu, Y., Zhang, Y., Lei, F., Wei, Y., He, S., Huang, L., Liu, X., Zhao, J., et al., 2024b. DA-Code: Agent Data Science Code Generation Benchmark for Large Language Models. arXiv preprint arXiv:2410.07331 .

[60] In, S., Krokos, E., Whitley, K., North, C., Yang, Y., 2024. Evaluating Navigation and Comparison Performance of Computational Notebooks on Desktop and in Virtual Reality, in: Proceedings of the CHI Conference on Human Factors in Computing Systems, ACM. pp. 1–15. doi:10.1145/3613904.3642932.

[61] Jiang, Y., Kästner, C., Zhou, S., 2022. Elevating Jupyter Notebook Maintenance Tooling by Identifying and Extracting Notebook Structures, in: International Conference on Software Maintenance and Evolution (ICSME), IEEE. pp. 399–403.

[62] Juneau, S., Olsen, K., Nikutta, R., Jacques, A., Bailey, S., 2021. Jupyter-Enabled Astrophysical Analysis Using Data-Proximate Computing Platforms. Computing in Science & Engineering 23, 15–25.

[63] Källén, M., Sigvardsson, U., Wrigstad, T., 2021. Jupyter Notebooks on GitHub: Characteristics and Code Clones. The Art, Science, and Engineering of Programming 5.

[64] Kang, D., Ho, T., Marquardt, N., Mutlu, B., Bianchi, A., 2021. Toonnote: Improving Communication in Computational Notebooks using Interactive Data Comics, in: CHI Conference on Human Factors in Computing Systems, ACM. pp. 1–14.

[65] Kastner, M., Franzkeit, J., Lainé, A., 2020. Teaching Machine Learning and Data Literacy to Students of Logistics using Jupyter Notebooks, in: DELFI 2020 – Die 18. Fachtagung Bildungstechnologien der Gesellschaft für Informatik e.V.. Gesellschaft für Informatik e.V., Bonn, pp. 365–366.

[66] Kery, M.B., Myers, B.A., 2018. Interactions for Untangling Messy History in a Computational Notebook, in: 2018 IEEE symposium on visual languages and human-centric computing (VL/HCC), IEEE. pp. 147–155.

[67] Kery, M.B., Radensky, M., Arya, M., John, B.E., Myers, B.A., 2018. The story in the Notebook: Exploratory Data Science using a Literate Programming Tool, in: Proceedings of the CHI Conference on Human Factors in Computing Systems, pp. 1–11.

[68] Kerzel, D., König-Ries, B., Sheeba, S., 2023. MLProvLab: Provenance Management for Data Science Notebooks, in: BTW 2023. Gesellschaft für Informatik e.V., Bonn, pp. 965–980.

[69] Kerzel, D., Samuel, S., König-Ries, B., 2021. Towards Tracking Provenance from Machine Learning Notebooks, in: KDIR, pp. 274–281.

[70] Kinanen, O., Muñoz-Moller, A.D., Stirbu, V., Mikkonen, T., 2024. Improving Quantum Developer Experience with Kubernetes and Jupyter Notebooks, in: 2024 IEEE International Conference on Quantum Computing and Engineering (QCE), IEEE. pp. 245–250. doi:10.1109/qce60285.2024.10286.

[71] Kitchenham, B., 2004. Procedures for Performing Systematic Reviews. Keele, UK, Keele University 33, 1–26.

[72] Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B.E., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J.B., Grout, J., Corlay, S., et al., 2016. Jupyter Notebooks-a Publishing Format for Reproducible Computational Workflows. Elpub 2016, 87–90.

[73] Koenzen, A.P., Ernst, N.A., Storey, M.A.D., 2020. Code Duplication and Reuse in Jupyter Notebooks, in: Symposium on Visual Languages and Human-Centric Computing (VL/HCC), IEEE. pp. 1–9.

[74] Koop, D., 2021. Notebook Archaeology: Inferring Provenance from Computational Notebooks, in: Provenance and Annotation of Data and Processes: 8th and 9th International Provenance and Annotation Workshop, IPAW 2020+ IPAW 2021, Springer. pp. 109–126.

[75] Koop, D., Patel, J., 2017. Dataflow Notebooks: Encoding and Tracking Dependencies of Cells, in: 9th USENIX Workshop on the Theory and Practice of Provenance (TaPP).

[76] Kuramitsu, K., Obara, Y., Sato, M., Obara, M., 2023. KOGI: A Seamless Integration of ChatGPT into Jupyter Environments for Programming Education, in: Proceedings of the 2023 ACM SIGPLAN International Symposium on SPLASH-E, Association for Computing Machinery, New York, NY, USA. pp. 50–59. doi:10.1145/3622780.3623648.

[77] Kwon, N., Kim, H., Rahman, S., Zhang, D., Hruschka, E., 2023. Weedle: Composable Dashboard for Data-Centric NLP in Computational Notebooks, in: Companion Proceedings of the ACM Web Conference 2023, pp. 132–135.

[78] Lau, S., Drosos, I., Markel, J.M., Guo, P.J., 2020. The Design Space of Computational Notebooks: An Analysis of 60 Systems in Academia and Industry, in: Symposium on Visual Languages and Human-Centric Computing (VL/HCC), IEEE. pp. 1–11.

[79] Launet, L., Wang, Y., Colomer, A., Igual, J., Pulgarín-Ospina, C., Koulouzis, S., Bianchi, R., Mosquera-Zamudio, A., Monteagudo, C., Naranjo, V., et al., 2023. Federating Medical Deep Learning Models from Private Jupyter Notebooks to Distributed Institutions. Applied Sciences 13, 919.

[80] Ley, M., 2002. The DBLP Computer Science Bibliography: Evolution, Research Issues, Perspectives, in: International symposium on string processing and information retrieval, Springer. pp. 1–10.

[81] Li, H., Ying, L., Zhang, H., Wu, Y., Qu, H., Wang, Y., 2023a. Notable: On-the-fly Assistant for Data Storytelling in Computational Notebooks, in: Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems, pp. 1–16.

[82] Li, L., Bissyandé, T.F., Papadakis, M., Rasthofer, S., Bartel, A., Octeau, D., Klein, J., Traon, L., 2017. Static Analysis of Android Apps: A Systematic Literature Review. Information and Software Technology 88, 67–95.

[83] Li, L., Lv, J., 2024. Unlocking Insights: Semantic Search in Jupyter Notebooks. arXiv preprint arXiv:2402.13234 .

[84] Li, X., Wang, Y., Wang, H., Wang, Y., Zhao, J., 2021. Nbsearch: Semantic Search and Visual Exploration of Computational Notebooks, in: CHI Conference on Human Factors in Computing Systems, ACM. pp. 1–14.

[85] Li, X., Zhang, Y., Leung, J., Sun, C., Zhao, J., 2023b. EDAssistant: Supporting Exploratory Data Analysis in Computational Notebooks with In Situ Code Search and Recommendation. ACM Transactions on Interactive Intelligent Systems 13, 1–27.

[86] Li, Z., Chockchowwat, S., Fang, H., Sahu, R., Thakurdesai, S., Pridaphatrakun, K., Park, Y., 2024a. Demonstration of ElasticNotebook: Migrating Live Computational Notebook States, in: Companion of the 2024 International Conference on Management of Data, ACM. pp. 540–543. doi:10.1145/3626246.3654752.

[87] Li, Z., Chockchowwat, S., Sahu, R., Sheth, A., Park, Y., 2024b. Kishu: Time-Traveling for Computational Notebooks. arXiv preprint arXiv:2406.13856 doi:https://doi.org/10.14778/3717755.3717759.

[88] Li, Z., Gor, P., Prabhu, R., Yu, H., Mao, Y., Park, Y., 2023c. ElasticNotebook: Enabling Live Migration for Computational Notebooks. Proc. VLDB Endow. 17, 119–133.

[89] Lin, Y., Li, H., Yang, L., Wu, A., Qu, H., 2023. InkSight: Leveraging Sketch Interaction for Documenting Chart Findings in Computational Notebooks. IEEE Transactions on Visualization and Computer Graphics , 1–11doi:10.1109/tvcg.2023.3327170.

[90] Liu, E.S., Lukes, D.A., Griswold, W.G., 2023. Refactoring in Computational Notebooks. ACM Transactions on Software Engineering and Methodology 32, 1–24.

[91] Liu, X., Wang, D., Wang, A., Hou, Y., Wu, L., 2021. HAConvGNN: Hierarchical Attention Based Convolutional Graph Neural Network for Code Documentation Generation in Jupyter Notebooks, in: Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics. pp. 4473–4485.

[92] Lu, H.m., Kwong, A., Unpingco, J., 2020. Securing Your Collaborative Jupyter Notebooks in the Cloud using Container and Load Balancing Services, in: SciPy, pp. 2–10.

[93] Macke, S., 2021. Automating State Management in Computational Notebooks, in: 11th Annual Conference on Innovative Data Systems Research (CIDR).

[94] Malone, M., Wang, Y., Monrose, F., 2023. Securely Autograding Cybersecurity Exercises Using Web Accessible Jupyter Notebooks, in: Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1, pp. 165–171.

[95] Manzoor, H., Naik, A., Shaffer, C.A., North, C., Edwards, S.H., 2020. Auto-grading jupyter notebooks, in: Proceedings of the 51st ACM Technical Symposium on Computer Science Education, pp. 1139–1144.

[96] McHugh, M.L., 2012. Interrater Reliability: the Kappa Statistic. Biochemia medica 22, 276–282.

[97] McNutt, A.M., Wang, C., Deline, R.A., Drucker, S.M., 2023. On the Design of AI-powered Code Assistants for Notebooks, in: Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems, pp. 1–16.

[98] Mendez, D., Graziotin, D., Wagner, S., Seibold, H., 2020. Open Science in Software Engineering. Springer International Publishing. pp. 477–501.

[99] Michael, I., 2022. Keeping your Jupyter Notebook Code Quality Bar High and Production Ready with Ploomber, in: SciPy, pp. 121–124.

[100] Mondal, T., Barnett, S., Lal, A., Vedurada, J., 2023. Cell2Doc: ML Pipeline for Generating Documentation in Computational Notebooks,

in: 2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE. pp. 384–396.

[101] Muller, M.J., Wang, A.Y., Ross, S.I., Weisz, J.D., Agarwal, M., Talamadupula, K., Houde, S., Martinez, F., Richards, J.T., Drozdal, J., et al., 2021. How Data Scientists Improve Generated Code Documentation in Jupyter Notebooks, in: 26th Conference on Intelligent User Interfaces (ACM IUI), ACM.

[102] Murta, L., Braganholo, V., Chirigati, F., Koop, D., Freire, J., 2015. noWorkflow: Capturing and Analyzing Provenance of Scripts, in: Provenance and Annotation of Data and Processes: 5th International Provenance and Annotation Workshop, (IPAW), Springer. pp. 71–83.

[103] Negrini, L., Shabadi, G., Urban, C., 2023. Static Analysis of Data Transformations in Jupyter Notebooks, in: Proceedings of the 12th ACM SIGPLAN International Workshop on the State Of the Art in Program Analysis, pp. 8–13.

[104] Niephaus, F., Krebs, E., Flach, C., Lincke, J., Hirschfeld, R., 2019. PolyJuS: a Squeak/Smalltalk-based Polyglot Notebook System for the GraalVM, in: Companion Proceedings of the 3rd International Conference on the Art, Science, and Engineering of Programming, pp. 1–6.

[105] Oli, P., Banjade, R., Tamang, L.J., Rus, V., 2021. Automated Assessment of Quality of Jupyter Notebooks Using Artificial Intelligence and Big Code, in: The International FLAIRS Conference Proceedings.

[106] Ono, J.P., Freire, J., Silva, C.T., 2021. Interactive Data Visualization in Jupyter Notebooks. Computing in Science & Engineering 23, 99–106.

[107] Ouyang, Y., Shen, L., Wang, Y., Li, Q., 2024. NotePlayer: Engaging Computational Notebooks for Dynamic Presentation of Analytical Processes, in: Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology, Association for Computing Machinery, New York, NY, USA. pp. 1–20. doi:10.1145/3654777.3676410.

[108] Owusu, C., Snigdha, N.J., Martin, M.T., Kalyanapu, A.J., 2022. PyGEE-SWToolbox: A Python Jupyter Notebook Toolbox for Interactive Surface Water Mapping and Analysis Using Google Earth Engine. Sustainability 14, 2557.

[109] Patra, J., Pradel, M., 2022. Nalin: Learning from Runtime Behavior to Find Name-Value Inconsistencies in Jupyter Notebooks, in: Proceedings of the ACM/IEEE 44th International Conference on Software Engineering, pp. 1469–1481.

[110] Peñuela, A., Hutton, C., Pianosi, F., 2021. An open-source package with interactive Jupyter Notebooks to enhance the accessibility of reservoir operations simulation and optimisation. Environmental Modelling & Software 145, 105188.

[111] Perez, M., Aydin, S., Lichter, H., 2024. A Flexible Cell Classification for ML Projects in Jupyter Notebooks. arXiv preprint arXiv:2403.07562 .

[112] Perkel, J.M., 2018. Why Jupyter is Data Scientists' Computational Notebook of Choice. Nature 563, 145–147.

[113] Petersohn, M., Schöbel, K., et al., 2023. Kopplung von Jupyter Notebooks mit externen E-Assessment-Systemen am Beispiel des Data Management Testers. Lecture Notes in Informatics 21, 1–6.

[114] Petricek, T., Geddes, J., Sutton, C., 2018. Wrattler: Reproducible, Live and Polyglot Notebooks, in: 10th USENIX workshop on the theory and practice of provenance (TaPP).

[115] Pimentel, J.F., Murta, L., Braganholo, V., Freire, J., 2019. A Large-Scale Study About Quality and Reproducibility of Jupyter Notebooks, in: 16th international conference on mining software repositories (MSR), IEEE. pp. 507–517.

[116] Pimentel, J.F., Murta, L., Braganholo, V., Freire, J., 2021. Understanding and Improving the Quality and Reproducibility of Jupyter Notebooks. Empirical Software Engineering 26, 65.

[117] Pimentel, J.F.N., Braganholo, V., Murta, L., Freire, J., 2015. Collecting and Analyzing Provenance on Interactive Notebooks: When {IPython} Meets {noWorkflow}, in: 7th USENIX workshop on the theory and practice of provenance (TaPP 15).

[118] Psallidas, F., Zhu, Y., Karlas, B., Henkel, J., Interlandi, M., Krishnan, S., Kroth, B., Emani, V., Wu, W., Zhang, C., et al., 2022. Data Science

Through the Looking Glass: Analysis of Millions of GitHub Notebooks and ML. NET Pipelines. ACM SIGMOD Record 51, 30–37.

[119] Quaranta, L., 2022. Assessing the Quality of Computational Notebooks for a Frictionless Transition from Exploration to Production, in: Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings, pp. 256–260.

[120] Quaranta, L., Calefato, F., Lanubile, F., 2021. KGTorrent: A Dataset of Python Jupyter Notebooks from Kaggle, in: 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR), IEEE. pp. 550–554.

[121] Quaranta, L., Calefato, F., Lanubile, F., 2022a. Eliciting Best Practices for Collaboration with Computational Notebooks. Proceedings of the ACM on Human-Computer Interaction 6, 1–41.

[122] Quaranta, L., Calefato, F., Lanubile, F., 2022b. Pynblint: a Static Analyzer for Python Jupyter Notebooks, in: Proceedings of the 1st International Conference on AI Engineering: Software Engineering for AI, pp. 48–49.

[123] Quaranta, L., Calefato, F., Lanubile, F., 2024. Pynblint: A quality assurance tool to improve the quality of Python Jupyter notebooks. SoftwareX 28, 101959. doi:10.1016/j.softx.2024.101959.

[124] Raghunandan, D., Elmqvist, N., Battle, L., 2023a. Measuring How Data Science Notebooks Evolve Over Time. Interactions 30, 17–18.

[125] Raghunandan, D., Roy, A., Shi, S., Elmqvist, N., Battle, L., 2023b. Code code evolution: Understanding how people change data science notebooks over time, in: Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems, pp. 1–12.

[126] Ragkhitwetsagul, C., Prasertpol, V., Ritta, N., Sae-Wong, P., Noraset, T., Choetkiertikul, M., 2024. Typhon: Automatic Recommendation of Relevant Code Cells in Jupyter Notebooks, in: 2024 21st International Joint Conference on Computer Science and Software Engineering (JCSSE), IEEE. pp. 662–669. doi:10.1109/jcsse61278.2024.10613645.

[127] Ramasamy, D., Sarasua, C., Bacchelli, A., Bernstein, A., 2023. Visualising Data Science Workflows to Support Third-party Notebook Comprehension: An Empirical Study. Empirical Software Engineering 28, 58.

[128] Ramsingh, A., Verma, P., 2024. Understanding & Mitigating the Challenges of Securing Jupyter Notebooks Online, in: 2024 IEEE International Conference on Cyber Security and Resilience (CSR), IEEE. pp. 01–07. doi:10.1109/csr61664.2024.10679378.

[129] Reades, J., 2020. Teaching on Jupyter. Region 7, 21–34.

[130] Rehman, M.S., 2019. Towards Understanding Data Analysis Workflows using a Large Notebook Corpus, in: International Conference on Management of Data, ACM. pp. 1841–1843.

[131] Ritta, N., Settewong, T., Kula, R.G., Ragkhitwetsagul, C., Sunetnanta, T., Matsumoto, K., 2022. Reusing My Own Code: Preliminary Results for Competitive Coding in Jupyter Notebooks, in: 29th Asia-Pacific Software Engineering Conference (APSEC), IEEE. pp. 457–461.

[132] Robinson, D., Ernst, N.A., Vargas, E.L., Storey, M.A.D., 2022. Error Identification Strategies for Python Jupyter Notebooks, in: Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension, pp. 253–263.

[133] Rule, A., Birmingham, A., Zuniga, C., Altintas, I., Huang, S.C., Knight, R., Moshiri, N., Nguyen, M.H., Rosenthal, S.B., Pérez, F., et al., 2018a. Ten Simple Rules for Reproducible Research in Jupyter Notebooks. PLoS Computational Biology 15.

[134] Rule, A., Drosos, I., Tabard, A., Hollan, J.D., 2018b. Aiding Collaborative Reuse of Computational Notebooks with Annotated Cell Folding. Proceedings of the ACM on Human-Computer Interaction 2, 1–12.

[135] Rule, A., Tabard, A., Hollan, J.D., 2018c. Exploration and Explanation in Computational Notebooks, in: CHI Conference on Human Factors in Computing Systems, ACM. pp. 1–12.

[136] Rule, A.C., 2018. Design and Use of Computational Notebooks. University of California, San Diego.

[137] Samuel, S., König-Ries, B., 2018. ProvBook: Provenance-based Semantic Enrichment of Interactive Notebooks for Reproducibility, in: ISWC (P&D/Industry/BlueSky).

[138] Samuel, S., König-Ries, B., 2021. Reproducemegit: A Visualization Tool for Analyzing Reproducibility of Jupyter Notebooks, in: 9th International Provenance and Annotation Workshop (IPAW), Springer. pp. 201–206.

[139] Samuel, S., Mietchen, D., 2023. Computational Reproducibility of Jupyter Notebooks from Biomedical Publications. GigaScience 13, giad113.

[140] Sato, F., Ikegami, A., Ishio, T., Shimari, K., Matsumoto, K., 2022. Comparing Execution Traces of Jupyter Notebook for Checking Correctness of Refactoring, in: 16th International Workshop on Software Clones (IWSC), IEEE. pp. 62–68.

[141] Sato, S., Nakamaru, T., 2024. Multiverse Notebook: Shifting Data Scientists to Time Travelers. Proceedings of the ACM on Programming Languages 8, 754–783. doi:10.1145/3649838.

[142] Savira, P., Marrinan, T., Papka, M.E., 2021. Writing, Running, and Analyzing Large-scale Scientific Simulations with Jupyter Notebooks, in: 2021 IEEE 11th Symposium on Large Data Analysis and Visualization (LDAV), IEEE. pp. 90–91.

[143] Schröder, M., Krüger, F., Spors, S., 2019. Reproducible Research is more than Publishing Research Artefacts: A Systematic Analysis of Jupyter Notebooks from Research Articles. arXiv preprint arXiv:1905.00092 .

[144] Scully-Allison, C., Lumsden, I., Williams, K., Bartels, J., Taufer, M., Brink, S., Bhatele, A., Pearce, O., Isaacs, K.E., 2024. Design Concerns for Integrated Scripting and Interactive Visualization in Notebook Environments. IEEE Transactions on Visualization and Computer Graphics 30, 6572–6585. doi:10.1109/tvcg.2024.3354561.

[145] Settewong, T., Ritta, N., Kula, R.G., Ragkhitwetsagul, C., Sunetnanta, T., Matsumoto, K., 2022. Why Visualize Data When Coding? Prelimi-

nary Categories for Coding in Jupyter Notebooks, in: 29th Asia-Pacific Software Engineering Conference (APSEC), IEEE. pp. 462–466.

[146] Shankar, S., Macke, S., Chasins, S., Head, A., Parameswaran, A., 2022. Bolt-on, Compact, and Rapid Program Slicing for Notebooks. VLDB Endowment 15, 4038–4047.

[147] Shome, A., Cruz, L., Spinellis, D., van Deursen, A., 2024. Understanding Feedback Mechanisms in Machine Learning Jupyter Notebooks. arXiv preprint arXiv:2408.00153 .

[148] Showkat, D., Baumer, E.P., 2021. Where Do Stories Come From? Examining the Exploration Process in Investigative Data Journalism. Proceedings of the ACM on Human-Computer Interaction 5, 1–31.

[149] Siddik, M.S., Bezemer, C.P., 2023. Do Code Quality and Style Issues Differ Across (Non-) Machine Learning Notebooks? Yes!, in: 2023 IEEE 23rd International Working Conference on Source Code Analysis and Manipulation (SCAM), IEEE. pp. 72–83.

[150] Simmons, A.J., Barnett, S., Rivera-Villicana, J., Bajaj, A., Vasa, R., 2020. A Large-scale Comparative Analysis of Coding Standard Conformance in Open-source Data Science Projects, in: Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 1–11.

[151] Singer, J., 2020. Notes on Notebooks: Is Jupyter the Bringer of Jollity?, in: ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software, pp. 180–186.

[152] Speicher, D., Dong, T., Cremers, O., Bauckhage, C., Cremers, A.B., 2019. Notes on the Code Quality Culture on Jupyter (Notebooks), in: Softwaretechnik-Trends Band 39, Heft 2, Gesellschaft für Informatik eV. pp. 17–18.

[153] Stark, J.A., Diakopoulos, N., 2016. Towards Editorial Transparency in Computational Journalism, in: Computation+ Journalism Symposium.

[154] Studtmann, L., Aydin, S., Lichter, H., 2023. Histree: A Tree-Based Experiment History Tracking Tool for Jupyter Notebooks, in: 2023 30th Asia-Pacific Software Engineering Conference (APSEC), pp. 299–308. doi:10.1109/APSEC60848.2023.00040.

[155] Subotić, P., Milikić, L., Stojić, M., 2022. A Static Analysis Framework for Data Science Notebooks, in: 2022 IEEE/ACM 44th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), IEEE. pp. 13–22.

[156] Subramanian, K., Hamdan, N., Borchers, J., 2020. Casual Notebooks and Rigid Scripts: Understanding Data Science Programming, in: 2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), IEEE. pp. 1–5.

[157] Tan, C.R., 2021. The Nascent Case for Adopting Jupyter Notebooks as a Pedagogical Tool for Interdisciplinary Humanities, Social Science, and Arts Education. Computing in Science & Engineering 23, 107–113.

[158] Terlych, N.A., Rodriges Zalipynis, R.A., 2021. Jupyter Lab Based System for Geospatial Environmental Data Processing, in: Proceedings of the Future Technologies Conference (FTC) 2020, Volume 2, Springer. pp. 627–638.

[159] Titov, S., Golubev, Y., Bryksin, T., 2022. ReSplit: Improving the Structure of Jupyter Notebooks by Re-Splitting Their Cells, in: International Conference on Software Analysis, Evolution and Reengineering (SANER), IEEE. pp. 492–496.

[160] Titov, S., Grotov, K., Prasad S. Venkatesh, A., 2024. Hidden Gems in the Rough: Computational Notebooks as an Uncharted Oasis for IDEs, in: Proceedings of the 1st ACM/IEEE Workshop on Integrated Development Environments, Association for Computing Machinery, New York, NY, USA. pp. 107–109. doi:10.1145/3643796.3648465.

[161] Tran O'Leary, J., Benabdallah, G., Peek, N., 2023. Imprimer: Computational Notebooks for CNC Milling, in: Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems, pp. 1–15.

[162] Valentine, D., Zaslavsky, I., Richard, S., Meier, O., Hudman, G., Peucker-Ehrenbrink, B., Stocks, K., 2021. EarthCube Data Discovery

Studio: A Gateway into Geoscience Data Discovery and Exploration with Jupyter Notebooks. Concurrency and computation: practice and experience 33, e6086.

[163] Van Dusen, E., 2020. Jupyter for Teaching Data Science, in: Proceedings of the 51st ACM Technical Symposium on Computer Science Education, pp. 1399–1399.

[164] Vandewalle, R., Kang, J.Y., Yin, D., Wang, S., 2019. Integrating CyberGIS-Jupyter and Spatial Agent-based Modelling to Evaluate Emergency Evacuation Time, in: proceedings of the 2nd ACM SIGSPATIAL international workshop on GeoSpatial simulation, pp. 28–31.

[165] Venkatesh, A.P.S., Bodden, E., 2021. Automated Cell Header Generator for Jupyter Notebooks, in: Proceedings of the 1st ACM International Workshop on AI and Software Testing/Analysis, ACM. pp. 17–20.

[166] Venkatesh, A.P.S., Sabu, S., Chekkapalli, M., Wang, J., Li, L., Bodden, E., 2024. Static analysis driven enhancements for comprehension in machine learning notebooks. Empirical Software Engineering 29. doi:10.1007/s10664-024-10525-w.

[167] Venkatesh, A.P.S., Wang, J., Li, L., Bodden, E., 2023. Enhancing Comprehension and Navigation in Jupyter Notebooks with Static Analysis, in: 2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), IEEE. pp. 391–401.

[168] Verano Merino, M., Sáenz, J.P., Díaz Castillo, A.M., 2022. Suppose You Had Blocks within a Notebook, in: Proceedings of the 1st ACM SIGPLAN International Workshop on Programming Abstractions and Interactive Notations, Tools, and Environments, pp. 57–62.

[169] Wang, A., Wang, D., Liu, X., Wu, L., 2021a. Graph-Augmented Code Summarization in Computational Notebooks, in: 13th International Joint Conference on Artificial Intelligence, IJCAI, pp. 5020–5023.

[170] Wang, A., Wu, Z., Brooks, C., Oney, S., 2024a. Don't step on my toes: resolving editing conflicts in real-time collaboration in computa-

tional notebooks, in: Proceedings of the 1st ACM/IEEE Workshop on Integrated Development Environments, pp. 47–52.

[171] Wang, A.Y., Mittal, A., Brooks, C., Oney, S., 2019. How Data Scientists Use Computational Notebooks for Real-Time Collaboration. Proceedings of the ACM on Human-Computer Interaction 3, 1–30.

[172] Wang, A.Y., Wang, D., Drozdal, J., Liu, X., Park, S., Oney, S., Brooks, C., 2021b. What makes a well-documented notebook? a case study of data scientists' documentation practices in kaggle, in: CHI Conference on Human Factors in Computing Systems, pp. 1–7.

[173] Wang, A.Y., Wang, D., Drozdal, J., Muller, M., Park, S., Weisz, J.D., Liu, X., Wu, L., Dugan, C., 2021c. Themisto: Towards Automated Documentation Generation in Computational Notebooks. arXiv preprint arXiv:2102.12592 .

[174] Wang, A.Y., Wang, D., Drozdal, J., Muller, M., Park, S., Weisz, J.D., Liu, X., Wu, L., Dugan, C., 2022a. Documentation Matters: Human-centered AI System to Assist Data Science Code Documentation in Computational Notebooks. ACM Transactions on Computer-Human Interaction 29, 1–33.

[175] Wang, F., Lin, Y., Yang, L., Li, H., Gu, M., Zhu, M., Qu, H., 2024b. OutlineSpark: Igniting AI-powered Presentation Slides Creation from Computational Notebooks through Outlines, in: Proceedings of the CHI Conference on Human Factors in Computing Systems, ACM. pp. 1–16. doi:10.1145/3613904.3642865.

[176] Wang, F., Liu, X., Liu, O., Neshati, A., Ma, T., Zhu, M., Zhao, J., 2023. Slide4N: Creating Presentation Slides from Computational Notebooks with Human-AI Collaboration, in: Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems, pp. 1–18.

[177] Wang, J., Kuo, T.y., Li, L., Zeller, A., 2020a. Assessing and Restoring Reproducibility of Jupyter Notebooks, in: 35th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 138–149.

[178] Wang, J., Kuo, T.y., Li, L., Zeller, A., 2020b. Restoring Reproducibility of Jupyter Notebooks, in: Proceedings of the ACM/IEEE 42nd

International Conference on Software Engineering (ICSE): Companion proceedings, pp. 288–289.

[179] Wang, J., Li, L., Zeller, A., 2020c. Better Code, Better Sharing:On the Need of Analyzing Jupyter Notebooks, in: ACM/IEEE 42nd international conference on software engineering: new ideas and emerging results, pp. 53–56.

[180] Wang, J., Li, L., Zeller, A., 2021d. Restoring Execution Environments of Jupyter Notebooks, in: 43rd International Conference on Software Engineering (ICSE), IEEE. pp. 1622–1633.

[181] Wang, Y., López, J.A.H., Nilsson, U., Varro, D., 2024c. Using Run-Time Information to Enhance Static Analysis of Machine Learning Code in Notebooks, in: Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering, pp. 497–501.

[182] Wang, Y., Meijer, W., López, J.A.H., Nilsson, U., Varró, D., 2024d. Why do Machine Learning Notebooks Crash? arXiv preprint arXiv:2411.16795 .

[183] Wang, Z.J., Dai, K., Edwards, W.K., 2022b. Stickyland: Breaking the Linear Presentation of Computational Notebooks, in: CHI Conference on Human Factors in Computing Systems, pp. 1–7.

[184] Wang, Z.J., Munechika, D., Lee, S., Chau, D.H., 2024e. SuperNOVA: Design Strategies and Opportunities for Interactive Visualization in Computational Notebooks, in: Extended Abstracts of the CHI Conference on Human Factors in Computing Systems, pp. 1–17.

[185] Wannipurage, D., Marru, S., Pierce, M., 2022. A Framework to Capture and Reproduce the Absolute State of Jupyter Notebooks, in: Practice and Experience in Advanced Research Computing. ACM, pp. 1–8.

[186] Watson, A., Bateman, S., Ray, S., 2019. PySnippet: Accelerating Exploratory Data Analysis in Jupyter Notebook through Facilitated Access to Example Code, in: 22nd International Conference on Extending Database Technology (EDBT)/ICDT Workshops.

[187] Weber, T., Ehe, J., Mayer, S., 2024. Extending Jupyter with Multi-Paradigm Editors. Proceedings of the ACM on Human-Computer Interaction 8, 1–22. doi:10.1145/3660247.

[188] Weber, T., Mayer, S., 2024. From Computational to Conversational Notebooks, in: Proceedings of the 1st CHI Workshop on Human-Notebook Interactions, pp. 1–6.

[189] Weinman, N., Drucker, S.M., Barik, T., DeLine, R., 2021. Fork it: Supporting Stateful Alternatives in Computational Notebooks, in: Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems, pp. 1–12.

[190] Werner, E., Manjunath, L., Frenzel, J., Torge, S., 2021. Bridging between Data Science and Performance Analysis: Tracing of Jupyter Notebooks, in: Proceedings of the First International Conference on AI-ML Systems, pp. 1–7.

[191] Werner, E., Rygin, A., Gocht-Zech, A., Döbel, S., Lieber, M., 2024. JUmPER: Performance Data Monitoring, Instrumentation and Visualization for Jupyter Notebooks, in: SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE. pp. 2003–2011. doi:10.1109/scw63240.2024.00250.

[192] Wilsdorf, P., Kirchhübel, A.W., Uhrmacher, A.M., 2023. NBSIMGEN: Jupyter Notebook Extension for Generating Simulation Experiments, in: 2023 Winter Simulation Conference (WSC), IEEE. pp. 2884–2895.

[193] Wong, T., Wagner, M., Treude, C., 2022. Self-Adaptive Systems: A Systematic Literature Review Across Categories and Domains. Information and Software Technology 148, 106934.

[194] Wootton, D., Fox, A.R., Peck, E., Satyanarayan, A., 2024. Charting EDA: Characterizing Interactive Visualization Use in Computational Notebooks with a Mixed-Methods Formalism. arXiv preprint arXiv:2409.10450 .

[195] Wu, Y., Hellerstein, J.M., Satyanarayan, A., 2020. B2: Bridging Code and Interactive Visualization in Computational Notebooks, in: Pro-

ceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology, pp. 152–165.

[196] Xin, R., Stallinga, S., Liu, H., Chen, P., Zhao, Z., 2022. Provenance-enhanced Root Cause Analysis for Jupyter Notebooks, in: 15th International Conference on Utility and Cloud Computing (UCC), IEEE. pp. 327–333.

[197] Yang, C., Brower-Sinning, R.A., Lewis, G., Kästner, C., 2022. Data Leakage in Notebooks: Static Detection and Better Processes, in: 37th IEEE/ACM International Conference on Automated Software Engineering (ICSE), pp. 1–12.

[198] Yang, M., Zhou, Y., Li, B., Tang, Y., 2023. On Code Reuse from StackOverflow: An Exploratory Study on Jupyter Notebook. arXiv preprint arXiv:2302.11732 .

[199] Yin, D., Liu, Y., Hu, H., Terstriep, J., Hong, X., Padmanabhan, A., Wang, S., 2019. CyberGIS-Jupyter for Reproducible and Scalable Geospatial Analytics. Concurrency and Computation: Practice and Experience 31, e5040.

[200] Yin, D., Liu, Y., Padmanabhan, A., Terstriep, J., Rush, J., Wang, S., 2017. A CyberGIS-Jupyter Framework for Geospatial Analytics at Scale, in: Proceedings of the practice and experience in advanced research computing 2017 on sustainability, success and impact, ACM. pp. 1–8.

[201] Yin, P., Li, W.D., Xiao, K., Rao, A., Wen, Y., Shi, K., Howland, J., Bailey, P., Catasta, M., Michalewski, H., et al., 2023. Natural Language to Code Generation in Interactive Data Science Notebooks, in: Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics. pp. 126–173.

[202] Zheng, C., Wang, D., Wang, A.Y., Ma, X., 2022. Telling Stories from Computational Notebooks: AI-Assisted Presentation Slides Creation for Presenting Data Science Work, in: Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems, pp. 1–20.

[203] Zhu, C., Saha, R.K., Prasad, M.R., Khurshid, S., 2021. Restoring the Executability of Jupyter Notebooks by Automatic Upgrade of Deprecated APIs, in: 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), IEEE. pp. 240–252.

[204] Zhu, J.S., Zhang, Z., Zhao, J., 2024. Facilitating Mixed-Methods Analysis with Computational Notebooks. arXiv preprint arXiv:2405.19580 .

[205] Zou, Y., Shan, X., Tan, S., Zhou, S., 2024. Can We Do Better with What We Have Done? Unveiling the Potential of ML Pipeline in Notebooks, in: 2024 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE. pp. 499–511. doi:10.1109/icsme58944.2024.00052.