This preprint is a preliminary version. The final version will be published at KDA-AI 2026.

CellFlow: A Tool For Automatic Jupyter Notebook Workflow Visualization

1st Selin Coban
Research Group Software Construction
RWTH Aachen University
Aachen, Germany
coban@swc.rwth-aachen.de

2nd David Kierdorf
Research Group Software Construction
RWTH Aachen University
Aachen, Germany
david.kierdorf@rwth-aachen.de

3rd Çağatay Akpınar
Research Group Software Construction
RWTH Aachen University
Aachen, Germany
cagatay.akpinar@rwth-aachen.de

4th Horst Lichter
Research Group Software Construction
RWTH Aachen University
Aachen, Germany
lichter@swc.rwth-aachen.de

***Abstract*—Jupyter Notebooks are widely used in machine learning (ML) and data science, yet their linear, cell-based representation makes it difficult for developers to quickly form accurate mental models, especially when notebooks contain branching logic or alternative solution paths.**

This paper presents CELLFLOW, a JupyterLab extension that automatically generates workflow diagrams from notebooks. CELLFLOW statically analyzes code cells using abstract syntax trees to identify variable definition–use dependencies and combines this analysis with LLM–generated semantic labels and descriptions. The resulting diagrams, named cell-flow diagrams, explicitly visualize computational activities and the flow of data artifacts between them, providing an architectural view of notebook workflows.

As a demonstration, we show how CELLFLOW integrates with JupyterLab and supports notebook comprehension. Initial user study results indicate that the generated diagrams help developers understand previously unseen notebooks more quickly and are perceived as useful and easy to use.

***Index Terms*—Jupyter Notebook, Comprehension, Navigation**

I. INTRODUCTION AND PROBLEM STATEMENT

Jupyter Notebooks (notebooks short) are widely used in machine learning (ML) development to explore solutions through interactive experimentation [1]. Given their central role in ML development, ensuring the comprehensibility of notebooks has become increasingly important [2], [3].

Prior research highlights the importance of developers forming accurate mental models of software systems to reason effectively about their behavior [4]. Developers familiar with established ML process models, such as CRISP-DM [5], often adopt a top-down approach, forming hypotheses about the workflow encoded in a notebook and subsequently validating them through manual inspection. This approach, however, requires time and excludes stakeholders who are not proficient with the underlying programming language.

Furthermore, the current notebook paradigm limits comprehensibility [6]. Notebooks provide only a linear representation of cells and offer no explicit support for perceiving data flow between them, making it difficult for developers to grasp the underlying workflow quickly. This limitation becomes particularly apparent when multiple solution approaches are implemented within a single notebook, introducing branching workflows that are not adequately reflected by the linear cell-based representation.

To address these challenges, we present CELLFLOW, a JupyterLab extension that automatically derives cell-flow graphs from notebooks and visualizes them as diagrams. CELLFLOW makes data dependencies between notebook cells explicit by exposing variable usage and data flow, thereby supporting developers in understanding, analyzing, and communicating ML notebook architectures.

All associated artifacts, including the implementation, a demonstration video, and evaluation artifacts, are made publicly available [7].

This paper is structured as follows. Section II reviews related work on improving notebook comprehension. Section III presents our approach for automatic workflow diagram generation. Section IV introduces CELLFLOW, a JupyterLab extension implementing the approach. Section V reports initial results of a user study evaluating CELLFLOW. Section VI concludes the paper and outlines directions for future research.

II. RELATED WORK

Only a limited number of approaches and tools have been proposed to improve notebook comprehension and navigation.

Ramasamy et al. [8] introduce a visualization method for notebook workflows to empirically demonstrate that notebook comprehension can be significantly improved through visual guidance. However, their study remained largely conceptual and lacked a fully integrated implementation.

Venkatesh et al. [9] introduce HEADERGEN, a tool that automatically generates Markdown headers for code cells from a fixed set of task labels and subsequently produces a table of contents to facilitate navigation within notebooks.

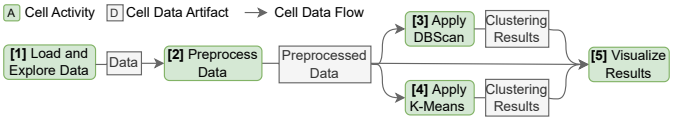

Fig. 1. An example cell-flow graph depicting cell activities, cell data artifacts, and cell data flows.

To extract and analyze the structure of notebooks, Jiang et al. [10] propose an algorithm for generating data-dependency graphs, in which nodes are labeled with a fixed set of high-level tasks, such as "training" or "evaluation." As future work, they suggest using these data-dependency graphs for navigation support, such as enabling developers to jump directly to "training" code cells.

Overall, these approaches demonstrate the benefits of structural and visual abstractions for improving notebook comprehension. However, they all rely on fixed task taxonomies or are limited in the integration into development environments.

III. Cell-Flow Graphs

To provide an architectural view of the workflow within a notebook, we introduce *cell-flow graphs*. They model the semantic activities performed in the notebook and the flow of data artifacts between them. These graphs should support developers in comprehending both *what* activities are performed and *how* they are structurally organized.

We define the elements of cell-flow graphs as follows:

- A *cell activity* is a semantic unit representing an action performed in a single notebook code cell.
- A *cell data artifact* is any information that is defined or processed by a cell activity. Examples are variables or datasets.
- An *input cell data artifact* is a cell data artifact that is not defined but modified by a cell activity.
- An *output cell data artifact* is a cell data artifact that is defined or processed by a cell activity.
- A *cell data flow* is a directed relation from a cell activity A to a cell activity B, where A has an output data artifact that is an input data artifact of B.
- Each cell activity has at least one input or output data artifact, or both.

Figure 1 visualizes a cell-flow graph of a simple notebook on a clustering problem. In this notebook, a dataset is first loaded, and its characteristics are explored (cell activity 1). Next, the data is preprocessed (cell activity 2) for the application of clustering algorithms. Two different algorithms (K-Means & DBScan) are applied in separate cells (cell activities 3 and 4). The results of both clustering algorithms are then visualized along with the preprocessed data (cell activity 5).

A. Generation of Cell-Flow Graphs

To generate cell-flow graphs, we assume that each notebook cell corresponds to a semantic activity, a widely adopted best practice [11].

In general, a cell-flow graph is generated by transforming each code cell into a corresponding cell activity and linking these activities according to variable definition–use dependencies between cells. Abstract syntax trees (ASTs) are used to derive definition–use dependencies because they provide a structured representation of code in which variable definitions, scopes, and usages are explicitly encoded.

To capture the meaning of a cell activity, we enrich cell activities with additional information. First, we use labels to concisely describe each cell activity. These labels are generated by translating the contained code into natural language descriptions. For this task, large language models (LLMs) have proven to be reliable [12]. Second, we generate more detailed descriptions of cell activities to optionally provide additional detail.

To elaborate further, we generate cell-flow diagrams from notebooks as follows (see Figure 2):

1) First, a *notebook preprocessor* extracts the code cells. Magic and shell commands are removed to facilitate later parsing of code cells into ASTs.
2) The code cells are sent along with two prompts, one for the cell activity label and one for the cell activity description, to the *LLM*. We restrict the length of each cell activity label to 1-5 words, and the length of each description to 110 tokens (based on experiments). Further, the cleaned code cells are transformed into ASTs by the *AST parser*.
3) The *cell-flow graph generator* iterates through the ASTs to analyze code cell dependencies based on variable definitions and uses. This information is also used to derive the input and output data artifacts of a code cell. Using the identified data artifacts, along with the generated labels and descriptions of code cells, the cell-flow graph is constructed.
4) The generated cell-flow graph is transformed into a diagram by the *cell-flow diagram generator* to be presented to the developer.

IV. Implementation

We implemented cell-flow graph generation and its visualization via a diagram in the JupyterLab extension CellFlow. We used Mixtral-8x22B as the LLM, given its strong performance on code-reasoning tasks [13]. The cell-flow diagrams are rendered using React Flow [14]. Because React Flow does not compute element positions automatically, CellFlow recalculates positions whenever a new cell activity is inserted into the cell-flow graph. Currently, the generation has to be triggered by clicking a button. A screenshot of a generated cell-flow diagram is provided in Figure 3. By hovering over a cell activity, developers can view its input and output data artifacts, along with a short description of the actions performed in the corresponding notebook cell.

V. Initial Results of User Experiment

To evaluate the *usefulness* and *usability* of CellFlow we conducted user experiments with 10 participants from our

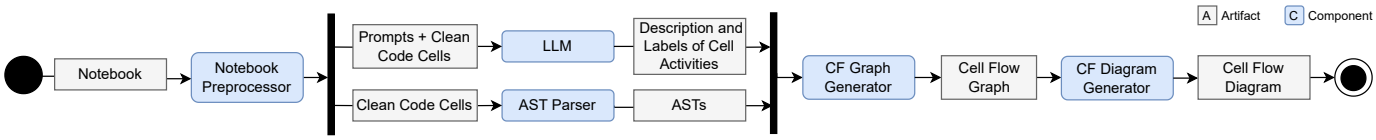

Fig. 2. Cell-Flow Diagram Generation Process

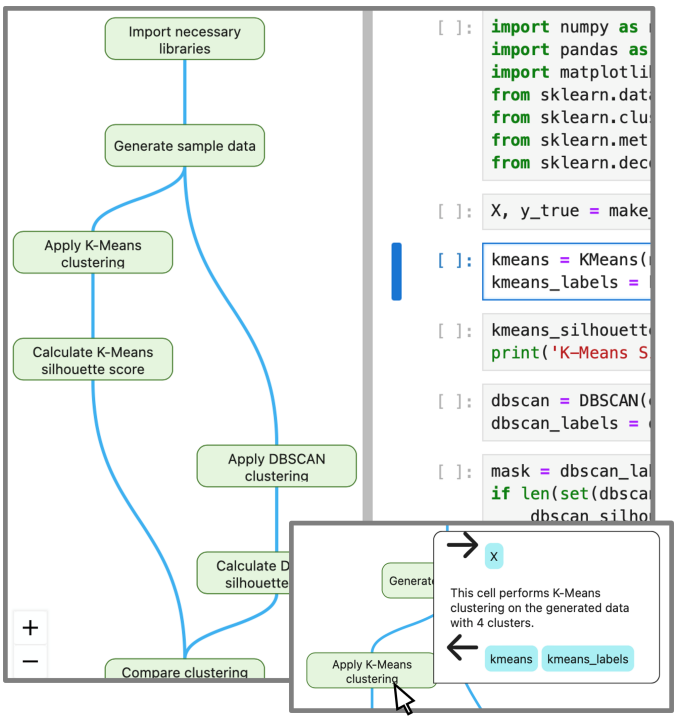

Fig. 3. Workflow visualization of a clustering notebook using CELLFLOW. Hovering over a cell activity node displays a detailed explanation and the corresponding input and output data artifacts. By clicking on a cell activity node, the corresponding notebook cell is marked.

TABLE I
PARTICIPANT EXPERTISE LEVELS (1=NOVICE, 5=EXPERT)

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
ML	1	3	3	3	2	2	5	2	4	4
Python	1	4	3	3	3	3	4	3	5	4

personal and professional networks, all of whom were students with a background in Computer Science or a related field. The participants had varying levels of experience with Python and ML. An overview of the demographic is shown in Table I.

A. Task and Procedure

To conduct the study, we deployed a dockerized JupyterLab environment integrated with CELLFLOW on our server. We uploaded two Python notebooks (n_a and n_b) with comparable complexity levels and incorporating branching workflows, each addressing a different ML problem. Notebook n_a addressed a classification task in which multiple ML models were trained and evaluated, and the best-performing model was subsequently hyper-tuned. Notebook n_b focused on a clustering task and included more elaborate visualization activities.

First, participants were informed about the study and how their data would be used. We obtained their consent to use their data and to record session audio. After requesting demographic information, the participants were asked to explain notebook n_a in their own words. Next, participants were presented with a diagram generated by CELLFLOW for notebook n_a and introduced to the tool. Afterward, they opened notebook n_b and, when asked again to explain the notebook in their own words, were allowed to use CELLFLOW. Finally, participants completed a questionnaire assessing usefulness and usability; usability was measured using the System Usability Scale (SUS). The questionnaire included the following statements, which participants rated on a Likert scale from 1 to 5, where 1 = *strongly disagree* and 5 = *strongly agree*:

Usefulness:

- S1 The generated diagram is easily understandable.
- S2 The generated diagram helps me build a mental model of the notebook.
- S3 The generated diagram displays the right amount of information.
- S4 CellFlow supports me in understanding the notebook.

Usability:

- S5 I think that I would like to use this system frequently.
- S6 I found the system unnecessarily complex.
- S7 I thought the system was easy to use.
- S8 I think that I would need the support of a technical person to be able to use this system.
- S9 I found the various functions in this system were well integrated.
- S10 I thought there was too much inconsistency in this system.
- S11 I would imagine that most people would learn to use this system very quickly.
- S12 I found the system very cumbersome to use.
- S13 I felt very confident using the system.
- S14 I needed to learn a lot of things before I could get going with this system.

Finally, they are encouraged to provide further comments or suggest improvements to CELLFLOW's features.

Throughout the session, a think-aloud protocol was applied. To lessen the influence of the notebook content itself on the results, the notebooks n_a and n_b were interchanged across sessions.

B. Data Collection & Analysis

All sessions were conducted and recorded via Zoom, with an average duration of 19.6 minutes. Notes were taken to document key statements from participants before comprehensive analysis. The questionnaire was created in Google Forms and shared with the participants.

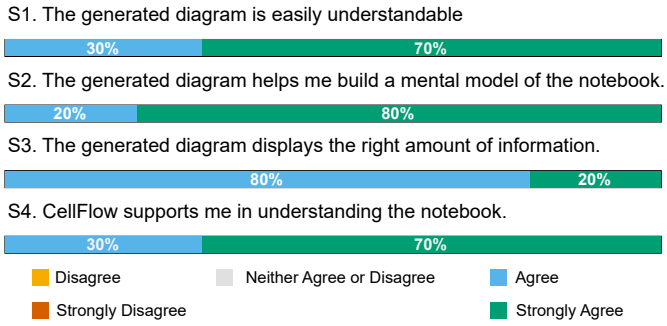

Fig. 4. Participant Agreement Levels with Statements on Usefulness

For the analysis of quantitative data, we recorded all responses as Likert scores in an Excel spreadsheet and applied descriptive statistics. For the SUS model questions, we computed the SUS score accordingly. For the analysis of qualitative data, we transcribed all user experiment sessions and applied thematic open coding to identify recurring themes.

C. Initial Results

While thematic analysis is still ongoing, we present the results from the quantitative data and notes taken during the user experiment sessions. The ratings for the statements on the usefulness of CELLFLOW (S1–S4) are shown in Figure 4. In general, CELLFLOW is perceived as very useful and usable across all participants. According to the SUS model, CELLFLOW receives an SUS score of 91.75, which corresponds to *excellent* usability. In the following, we describe the identified strengths and weaknesses of CELLFLOW, as well as mentioned feature improvements.

Strengths: In general, participants highlighted the strength of the cell-flow graph in capturing the notebook's branching structure, rather than presenting it as a linear sequence. During notebook comprehension tasks, we observed that participants relied on the explanations provided in the cell activity nodes whenever they were unfamiliar with specific lines of code. All participants stated that they would use CELLFLOW in the future. In particular, P3 noted that CELLFLOW would be especially helpful when exploring notebooks they had not authored.

Weakness: P1, who has the least ML expertise, noted that they would appreciate more explanation of the ML algorithms mentioned in the cell-activity descriptions, as they are not familiar with terms such as "DBSCAN". P10, a highly skilled participant, mentioned that they would like the option to receive more detailed explanations regarding variable usage, particularly their composition.

Improvements: Two participants mentioned it would be helpful if by clicking on a cell activity node, one can jump directly to the corresponding code cell in the notebook. This would also help in notebook navigation. In response, we implemented this feature.

VI. CONCLUSION AND FUTURE WORK

In this paper, we present CELLFLOW, a tool that supports developers in understanding and navigating Jupyter notebooks. To achieve this, we introduce the concept of cell-flow diagrams and describe a process for automatically generating them. Results from user experiments with CELLFLOW show strong positive feedback, particularly highlighting its effectiveness in supporting comprehension of previously unseen notebooks. Furthermore, CELLFLOW can serve as a navigation tool within notebooks.

A promising future research direction is the analysis and mining of notebook workflows using cell-flow diagrams generated from large notebook collections.

REFERENCES

[1] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay *et al.*, "Jupyter Notebooks–a Publishing Format for Reproducible Computational Workflows," in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. IOS press, 2016, pp. 87–90.

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

[3] N. Nahar, S. Zhou, G. Lewis, and C. Kästner, "Collaboration Challenges in Building ML-Enabled Systems: Communication, Documentation, Engineering, and Process," in *Proceedings of the 44th International Conference on Software Engineering*, ser. ICSE '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 413–425.

[4] M.-A. Storey, "Theories, Methods and Tools in Program Comprehension: Past, Present and Future," in *13th International Workshop on Program Comprehension (IWPC'05)*, 2005, pp. 181–191.

[5] R. Wirth and J. Hipp, "CRISP-DM: Towards a Standard Process Model for Data Mining," in *Proceedings of the 4th International Conference on the Practical Applications of Knowledge Discovery and Data Mining*, vol. 1. Manchester, 2000, pp. 29–39.

[6] R. L. Huang, S. Ravi, M. He, B. Tian, S. Lerner, and M. Coblenz, "How Scientists Use Jupyter Notebooks: Goals, Quality Attributes, and Opportunities," in *Proceedings of the IEEE/ACM 47th International Conference on Software Engineering*, ser. ICSE '25. IEEE Press, 2025, p. 1243–1255.

[7] S. Coban, D. Kierdorf, C. Akpinar, and H. Lichter, "Artifacts of the Paper "CellFlow: A Tool For Automatic Jupyter Notebook Workflow Visualization" ," Feb. 2026. [Online]. Available: https://doi.org/10.5281/zenodo.18545917

[8] D. Ramasamy, C. Sarasua, A. Bacchelli, and A. Bernstein, "Visualising Data Science Workflows to Support Third-Party Notebook Comprehension: An Empirical Study," *Empirical Software Engineering*, vol. 28, no. 3, p. 58, 2023.

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

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

[11] A. Rule, A. Birmingham, C. Zuniga, I. Altintas, S.-C. Huang, R. Knight, N. Moshiri, M. H. Nguyen, S. B. Rosenthal, F. Pérez, and P. W. Rose, "Ten Simple Rules for Reproducible Research in Jupyter Notebooks," *Computing Research Repository (CoRR)*, vol. 1810, 2018.

[12] D. Nam, A. Macvean, V. Hellendoorn, B. Vasilescu, and B. Myers, "Using an LLM to Help With Code Understanding," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024, pp. 1–13.

[13] Mistral AI Team, "Cheaper, Better, Faster, Stronger," https://mistral.ai/news/mixtral-8x22b, 2025, accessed: 2026-02-06.

[14] XY Flow Team, "Node-Based UIs in React," https://reactflow.dev/, 2025, accessed: 2026-02-06.