SHORT-PAPER

Demo of Kishu: Time-Traveling for Computational Notebooks

ZHAOHENG LI, University of Illinois Urbana-Champaign, Urbana, IL, United States

SUPAWIT CHOCKCHOWWAT, University of Illinois Urbana-Champaign, Urbana, IL, United States

HANXI FANG, University of Illinois Urbana-Champaign, Urbana, IL, United States

YONGJOO PARK, University of Illinois Urbana-Champaign, Urbana, IL, United States

Published: 22 June 2025

SIGMOD/PODS '25: International Conference on Management of Data
June 22 - 27, 2025
Berlin, Germany

Conference Sponsors:
SIGMOD

https://doi.org/10.1145/3722212.3725095
ISBN: 9798400715648

Demo of Kishu: Time-Traveling for Computational Notebooks

Zhaoheng Li
Univ. of Illinois Urbana-Champaign
zl20@illinois.edu

Supawit Chockchowwat
Univ. of Illinois Urbana-Champaign
supawit2@illinois.edu

Hanxi Fang
Univ. of Illinois Urbana-Champaign
hanxif2@illinois.edu

Yongjoo Park
Univ. of Illinois Urbana-Champaign
yongjoo@illinois.edu

Abstract

Computational notebooks (e.g., Jupyter, Google Colab) enable the interactive computing model of iteratively executing *cells* (i.e., a set of statements) and observing the result (e.g., model or plot), and are widely used by data scientists. However, existing notebook systems do not offer *time-traveling to past states*, hence certain cell executions can *irreversably modify* the session state (e.g., accidentally overwriting a variable, leading to a loss of work progress.

In this demo, we introduce Kishu, a new notebook system facilitating *time-traveling* with lightweight incremental checkpoint and restore. Kishu's frontend allows users to create manual and automatic *commits* for session state snapshots at various points in time, and perform *undos* or *checkouts* to any past state. Kishu's backend transparently monitors user behavior (e.g., data operations) to create efficient incremental checkpoints, which support the frontend's (also incremental) *sub-second* checkouts to past states. Through this demo, we will show Kishu's ability to mitigate two common pain points of notebook usage through time-traveling: undoing cell executions and facilitating path-based exploration.

CCS Concepts

• **Information systems** → *Data provenance*; *Data analytics.*

Keywords

Computational Notebooks, State Replication

ACM Reference Format:
Zhaoheng Li, Supawit Chockchowwat, Hanxi Fang, and Yongjoo Park. 2025. Demo of Kishu: Time-Traveling for Computational Notebooks. In *Companion of the 2025 International Conference on Management of Data (SIGMOD-Companion '25), June 22–27, 2025, Berlin, Germany*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3722212.3725095

1 Introduction

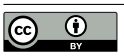
Computational notebooks (e.g., Jupyter [13, 31], Rstudio [26]) are widely used by data scientists [21, 25]. Notebooks' statefulness enable iterative workflows: users start *sessions* to run code cells, and the notebook would track *execution results* in the *session state* in the form of user-defined variables (e.g., loaded datasets, drawn plots). This iterative workflow alternating between code writing and result observation is commonly seen in data exploration [1, 8, 9, 11, 12, 14, 23, 30], visualization [7, 22], and model tuning [2, 24].

Cell Executions Cannot be Undone in Notebooks. While notebooks track cell execution results with the session state, current notebook systems do not support *undoing cell executions* to return to a past state—cell executions can *irreversably* modify the session state: for example, if an existing variable pointing to a ML model `model = lm.fit(...)` is overwritten via a re-declaration, restoring the overwritten model is not possible as it was not saved by the notebook system. This can cause significant loss of work progress, especially when such an irreversible operation is *unintentionally* performed, e.g., accidentally overwriting a variable declared in a prior, off-screen cell [15]. There exists flawed approaches for notebook users to restore overwritten and/or deleted (e.g., dropped dataframe columns) data in the session state: JupyterLab features the `%checkpoint` and `%history` commands [32] for helping users identify and version cells for rerunning to restore specific data, but reruns can be costly (e.g., re-running a model training cell). Alternatively, checkpointing tools such as ElasticNotebook [16, 18] or ForkIt [35] can be used to periodically snapshot the session state to disk (e.g., after every cell execution), which can be loaded for restoration; however, neither of these existing tools checkpoint (or restore) incrementally, causing potentially prohibitive storage costs (e.g., via repeatedly storing large, unchanged dataframes) and/or long restoration times from loading the full snapshot [17].

Kishu: Time-Traveling Notebook. In this demo, we present Kishu, a new notebook system that offers time-traveling to and from arbitrary session states with lightweight incremental checkpointing/restore. Kishu features both a Git-like interface and hotkeys for creating manual and/or automatic (e.g., per-cell execution) commits containing incremental snapshots of session state data, which can be incrementally restored to (via partial data loading) at *sub-second* latency via checkout and/or undo operations. Kishu's backend non-intrusively profiles cell executions to capture per-cell state changes (e.g., data accesses/creations/modifications) and efficiently computes how to perform incremental checkpointing and checkouts accordingly. This demo will show users how Kishu's time-traveling enhances data science workflows by (1) undoing cell executions and (2) managing branched session states. We have open-sourced Kishu [6], and have a report [4, 17] which details the technical aspects of our system. A video demo of Kishu is available at [5].

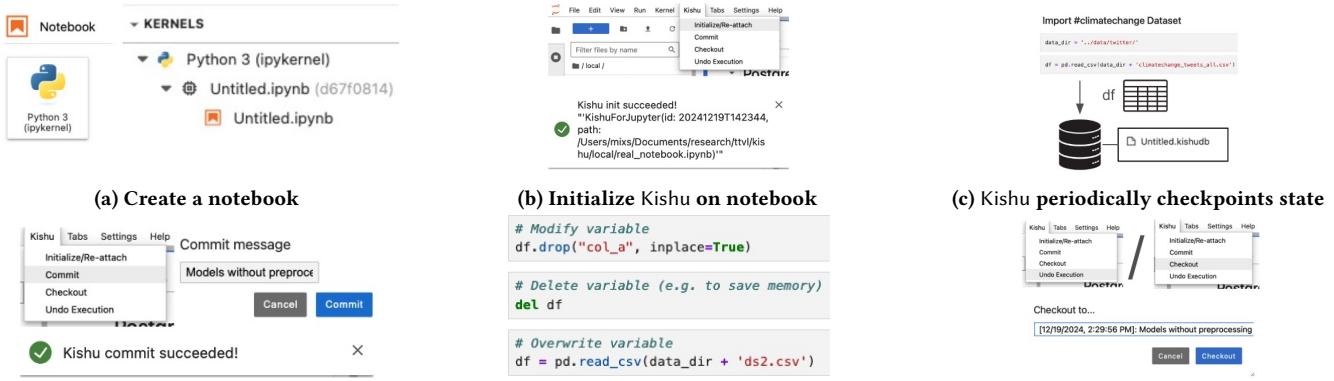
2 Kishu Overview

This section overviews Kishu's functionality. We describe how users interact with Kishu's frontend in a Jupyter-based platform



SIGMOD-Companion '25, Berlin, Germany

ACM ISBN 979-8-4007-1564-8/2025/06
https://doi.org/10.1145/3722212.3725095



(a) Create a notebook **(b) Initialize Kishu on notebook** **(c) Kishu periodically checkpoints state**

(d) Assign names to checkpoints **(e) Execute (potentially irreversible) code** **(f) Undo/Checkout to past session states**

Figure 1: Users can use Kishu to perform time-traveling to and from arbitrary session states.

(e.g., JupyterLab) in §2.1, the operations Kishu's backend performs in §2.2, and how Kishu achieves efficient time-traveling in §2.3.

2.1 Kishu Frontend and Functionality

Kishu's frontend is implemented as a Jupyter Server extension installable as a Python package (e.g., via pip). Users interact with Kishu via a drop-down menu in the Jupyter Server interface (fig 1).

Initializing Kishu's Tracking. When the user creates a new notebook file (Fig 1a), they can *initialize* Kishu onto it (Fig 1b); then, Kishu will start tracking cell executions ran in any notebook session started on this notebook file. Kishu uses the tracking results to compute what data in the session state to incrementally checkpoint (described shortly) into the notebook's *database file* (§2.2).

Transparent Checkpointing. Kishu periodically checkpoints the session state after each cell execution in attached notebooks (Fig 1c). Kishu checkpointing *incrementally*, writing *only* data that was modified by the cell execution to the database file—for example, the dataframe `df` in each depicted cell in Fig 1e. This ensures transparent checkpointing that does not incur noticeable latency (up to only 15% [17]) from the user perspective. Users can also assign names to important checkpoints (e.g., after data loading and before data cleaning) to convieniently return to later in the workflow (Fig 1d).

Time-Traveling. Kishu features two modes of time-traveling—*undo* and *checkout*: undo (Fig 1f) rolls back the session state to that before the latest cell execution, for example, to restore changes to the dataframe `df` made by cells depicted in Fig 1e, while *checkout* returns the session state to a state corresponding to a named checkpoint (Fig 1c). For both undo and checkout, Kishu incrementally reconstructs the target session state via *partial data loading* (§2.2).

2.2 Kishu Backend

Kishu's backend communicates with the frontend via API hooks in attached notebooks triggered on events such as post-cell executions. It captures cell executions' state changes, writes and manages incremental checkpoints, and performs incremental checkouts.

Low-Overhead State Change Monitoring. Kishu's backend monitors the session state during notebook usage to capture each cell execution's state changes (i.e., accessed, created, and modified variables) with live analysis. Compared to other methods utilizing live analysis (e.g., IPyFlow [29] or watchpoints [10]) which can incur high overhead, Kishu achieves low-overhead monitoring (~2% of cell runtime [17]) at a novel *Co-variable* granularity—sets of variables that share underlying references, allowing it to limit scope of monitoring: the variable names referred to within a cell execution tells Kishu exactly which data / Co-variables (which are isolated reference-wise) could have possibly been modified, hence it can skip checking data that was *definitely not* modified.

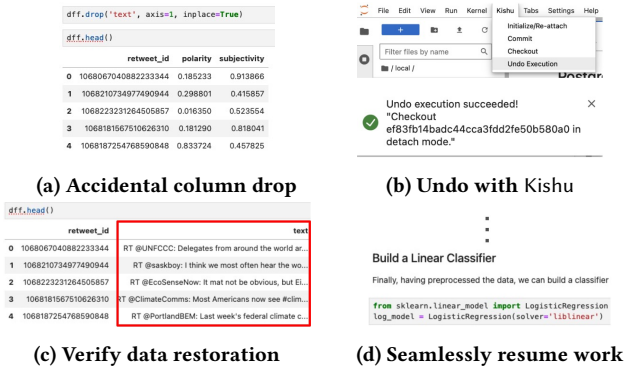
Incremental Checkpointing. Kishu uses monitoring results to write incremental checkpoints containing only changed data of each cell execution into the database file with the Pickle protocol [27], the de-facto standard observed by almost all Python libraries (e.g., NumPy, PyTorch). Hence, Kishu is compatible with almost all notebook workloads (empirically verified on 146 popular data science classes [17]). Kishu versions checkpoint data in MVCC-like [3] fashion (e.g., dataframe `df` at timestamp t), allowing it to easily identify and read (only) relevant data for checkouts (described shortly).

Incremental Checkout. When checkout is requested, Kishu's backend will compute the difference between current and target states, then only load the differing data (e.g., the dataframe with a dropped column in Fig 1c) for checkout. Checkouts are typically *sub-second* [17] when undoing single cell executions, hence can be seamlessly integrated into user workflows. Kishu's checkout is also fault-tolerant: if required data is missing from the database file (or fails to load) during checkout, Kishu can recompute it via *cell replay* (§2.3).

2.3 Correct and Robust Time-Traveling

This section derscribes how Kishu performs time-traveling *correctly* and *robustly*: it ensures that target session states that the user time-travels to are restored as is, regardless of any unexpected faults that Kishu (e.g., missing data) may encounter during checkout.

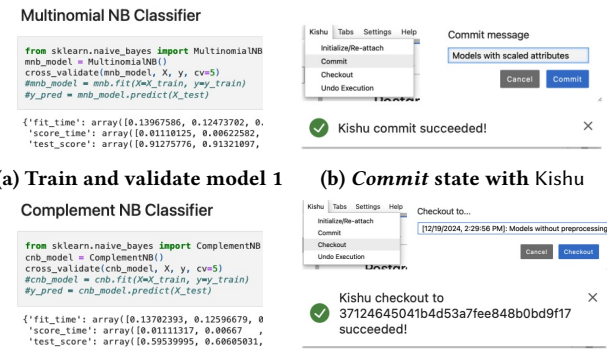
Correct Time-Traveling. Users often create *shared references* (e.g. putting lists representing *documents* into a 2D list to form a *corpus* [15]) where the same data can be reached (via referencing) from multiple variables. Users will expect these shared references



(a) Accidental column drop (b) Undo with Kishu

(c) Verify data restoration (d) Seamlessly resume work

Figure 2: Undoing cell executions: Kishu enables the user to rollback accidental data overwrites to avoid progress loss.



(a) Train and validate model 1 (b) *Commit* state with Kishu

(c) Train and validate model 2 (d) Compare models via *checkout*

Figure 3: Kishu's Branching exploration: users conveniently explore branching cell executions with *commit* and *checkout*.

to be preserved during checkouts—following the previous example, expect that modifications to a document in the corpus will also be reflected in the corpus. When Kishu performs time-traveling, it avoids the pitfall of breaking shared references encountered in other applicable checkpointing tools which store data variable-by-variable (e.g., Jupyter's %store [33]) by writing and loading data at the aforementioned Co-variable granularity (§2.2): sets of variables sharing references (e.g., documents and the corpus) will be treated as a single entity and written/loaded together during checkpointing and checkouts, ensuring Kishu to not break shared references.

Robust Time-Traveling. While Kishu uses the Pickle protocol to write/load data (§2.2), some objects such as generators [28] are incompatible with Pickle. Kishu can still time-travel to states with these objects via *cell replay*; it determines with the backend's monitoring results the session's *data lineage* (i.e., accessed/modified data of each cell), with which it accurately identifies and reruns the necessary cell executions for reconstructing the non-writeable/readable data that comprise the target state in a manner similar to Ray's operation replay [20]. Kishu also uses this technique when data for checkout is *missing* from the database (e.g., corrupted or deleted).

3 Demo Proposal

This demo will put the user in the shoes of a data scientist performing sentiment analysis with Twitter dataset [34] containing 10,000 tweets[1] on climate change-related topics: they aim to identify positive and negative tweets by first inspecting the data and performing data cleaning, then fitting various classifier models on the cleaned data. During this process, the user will be using Kishu in two cases where they would otherwise have lost significant progress.

3.1 Undo Cell Executions

To start, users will start the Jupyter server and open a prepared notebook `sklearn.ipynb`[2] containing cell code for required steps.

Notebook Preparation. The user will first initialize Kishu on the tutorial notebook by selecting the "Initialize/Re-attach" from the "Kishu" drop-down menu. Next, they will begin running the prepared cells: the cells will first load the appropriate libraries (e.g., NumPy, scikit-learn), load the dataset `df` into memory, and inspect the schema of the dataset via `df.head`. The user will then create a second copy of the dataset `df`—`dff`, which they will clean by removing duplicate tweets from `df` based on the column `retweet_id`.

Accidental Data Dropping. The user will start exploring sentiment analysis methods on the newly cleaned `dff`. They apply regex transformations (e.g., space trimming) on the `text` column containing raw tweet text, call the TextBlob sentiment analyzer [19] on the `text` column to obtain and assign per-tweet sentiment scores to a new column `subjectivity`, and finally, drop the `text` column to save memory. Unfortunately, the user realises that they forgot to compute a second `polarity` score for each tweet, but the `text` column was already dropped by the previous cell execution (Fig 2a).

'Un-dropping' a Dataframe Column. Fortunately, Kishu has been transparently checkpointing the session states to the local `sklearn.db` file after each cell execution. The user selects Kishu's "Undo Execution" option, and after a sub-second (0.4s) delay, Kishu restores the state to that before the `text` column was dropped (Fig 2b). The user can verify that the undoing was successful by running `dff.head()` to see that the `text` column is back (Fig 2c), and also the cell execution count beside the cell with `dff.drop('text')` disappearing. They can continue the workflow by computing the `polarity` scores, then build classifiers on the scores (Fig 2d) for the downstream tasks explored in the second scenario (§3.2).

Benefit: Efficient Undos. Had Kishu not been available, the user would have had to restore the `dff` dataframe with the `text` column intact by re-running two expensive cells—declaring `dff` by removing duplicate tweets, then applying regex transformations (~15s). Kishu'un-drops' `dff`'s column more efficiently by (only) loading it from the database file while the original dataframe `df` is unaltered.

Benefit: Transparent Checkpointing. During notebook usage, Kishu intelligently incrementally checkpoints only changed data of each cell execution; for example, it saves *nothing* after the static `df.head` cell which did not alter the state, and *only* the small `dff` in cells where it was modified. This, along with Kishu's low-overhead monitoring for computing incremental checkpoints (§2.2), makes Kishu's checkpointing (almost) unnoticeable latency-wise.

[1]We sample the original 450,000-tweet dataset due to the demo's time constraints.
[2]Based on Cornell's tutorial notebook, which uses the scikit-learn library [34]

3.2 Branching Data Exploration

After the user obtains the polarity and subjectivity scores with the TextBlob library, they will use scikit-learn's Vectorizer to apply Word2Vec on tweet texts and aim to fit a model between resulting features and Textblob's polarity and subjectivity scores (Fig 3a).

Checkpointing an Exploration Path. After applying Word2Vec on the `text` column and storing the results in a `features` array, the user creates a train-test split on the `features` array and the scores to predict. They will fit scikit-learn's Gaussian NB classifier on the `features` and the scores, and verify via cross-validation that the model achieves 91% accuracy (Fig 3a). The user will proceed to train and evaluate other models. However, noting that other model training cells all assign the model and cross-validation results to the same variables `model` and `res` (to mimic a common habit of data scientists assigning items to the same names [15]), they select Kishu's "Commit" option to manually create a checkpoint named "Model with scaled attributes" (Fig 3b), so they can always return to the Gaussian NB classifier and its scores for comparisons.

Exploring Alternative Models. The user repeats this training, evaluating, and checkpointing process for a different Multinomial NB classifier (Fig 3c). To compare the models, the user will use Kishu's "checkout" option to iteratively re-visit prior models and validation results. Kishu will exactly restore each model into the session state, and the user can proceed to compare the results and select the most suitable one for downstream tasks (Fig 3d).

Benefit: Fast Branching. Prior to Kishu, data scientists performed branching exploration by manually annotating each series of cells intended to be in the same branch (e.g., split → training → validation), and each cell series would have to be re-executed every time the user wishes to switch branches [15]. Kishu incremental commits and checkouts enables more convienient and efficient branching—the datasets, `df`, `dff` which are shared between the branches are untouched during checkouts; only the models and validation scores differing between the branches are loaded from the database file.

Benefit: Deterministic Restoration. For this demo, the user would not have been able to replicate validation scores for any model—the random seed was not set for cross-validations. Had they not checkpointed the states with Kishu, attempting to restore exact validation results through rerunning cells corresponding to the branches would fail as this process is non-deterministic. However, as Kishu stored the validation results, it can deterministically revert the state and grant users access to exact prior results they observed.

References

[1] Johes Bater, Yongjoo Park, Xi He, Xiao Wang, and Jennie Rogers. 2020. Saqe: practical privacy-preserving approximate query processing for data federations. *Proceedings of the VLDB Endowment* 13, 12 (2020), 2691–2705.

[2] James Bergstra and Yoshua Bengio. 2012. Random Search for Hyper-Parameter Optimization. *J. Mach. Learn. Res.* 13, null (feb 2012), 281–305.

[3] Philip A Bernstein and Nathan Goodman. 1983. Multiversion concurrency control—theory and algorithms. *ACM Transactions on Database Systems (TODS)* 8, 4 (1983), 465–483.

[4] Supawit Chockchowwat, Zhaoheng Li, and Yongjoo Park. 2023. Transactional Python for Durable Machine Learning: Vision, Challenges, and Feasibility. In *Proceedings of the Seventh Workshop on Data Management for End-to-End Machine Learning* (Seattle, WA, USA) *(DEEM '23)*. Association for Computing Machinery, New York, NY, USA, Article 5, 5 pages. doi:10.1145/3595360.3595855

[5] CreateLab. 2024. Kishu Demo. https://youtu.be/LXg-q0yMCiw.

[6] CreateLab. 2024. Kishu repository. github.com/illinoisdata/kishu.

[7] Philipp Eichmann, Emanuel Zgraggen, Carsten Binnig, and Tim Kraska. 2020. IDEBench: A Benchmark for Interactive Data Exploration. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 1555–1569. doi:10.1145/3318464.3380574

[8] Hanxi Fang, Supawit Chockchowwat, Hari Sundaram, and Yongjoo Park. 2025. Enhancing Computational Notebooks with Code+Data Space Versioning. In *CHI Conference on Human Factors in Computing Systems (CHI'25)*.

[9] Hanxi Fang, Supawit Chockchowwat, Hari Sundaram, and Yongjoo Park. 2025. Large-scale Evaluation of Notebook Checkpointing with AI Agents. In *Late-breaking work in CHI Conference on Human Factors in Computing Systems (CHI'25)*.

[10] Tian Gao. 2020. Python Watchpoints. pypi.org/project/watchpoints/.

[11] Ipoom Jeong, Jinghan Huang, Chuxuan Hu, Dohyun Park, Jaeyoung Kang, Nam Sung Kim, and Yongjoo Park. 2025. UPP: Universal Predicate Pushdown to Smart Storage. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*.

[12] Jeremiah W. Johnson. 2020. Benefits and Pitfalls of Jupyter Notebooks in the Classroom. In *Proceedings of the 21st Annual Conference on Information Technology Education* (Virtual Event, USA) *(SIGITE '20)*. Association for Computing Machinery, New York, NY, USA, 32–37. doi:10.1145/3368308.3415397

[13] Project Jupyter. 2023. Jupyter Notebook. jupyter.org/.

[14] Jaeyoung Kang, Qirong Xia, Ipoom Jeong, Yongjoo Park, and Nam Sung Kim. 2025. Intel® In-Memory Analytics Accelerator: Performance Characterization and Guidelines. In *IEEE international symposium on performance analysis of systems and software (ISPASS)*.

[15] Mary Beth Kery, Marissa Radensky, Mahima Arya, Bonnie E John, and Brad A Myers. 2018. The story in the notebook: Exploratory data science using a literate programming tool. In *Proceedings of the 2018 CHI conference on human factors in computing systems*. 1–11.

[16] Zhaoheng Li, Supawit Chockchowwat, Hanxi Fang, Ribhav Sahu, Sumay Thakurdesai, Kantanat Pridaphatrakun, and Yongjoo Park. 2024. Demonstration of ElasticNotebook: Migrating Live Computational Notebook States. In *Companion of the 2024 International Conference on Management of Data*. 540–543.

[17] Zhaoheng Li, Supawit Chockchowwat, Ribhav Sahu, Areet Sheth, and Yongjoo Park. 2024. Kishu: Time-Traveling for Computational Notebooks. *Proceedings of the VLDB Endowment* 18, 4 (2024), 970 – 985.

[18] Zhaoheng Li, Pranav Gor, Rahul Prabhu, Hui Yu, Yuzhou Mao, and Yongjoo Park. 2023. ElasticNotebook: Enabling Live Migration for Computational Notebooks. *Proc. VLDB Endow.* 17, 2 (2023), 119–133.

[19] Steven Loria. 2024. TextBlob. https://textblob.readthedocs.io/en/dev/.

[20] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. 2018. Ray: A Distributed Framework for Emerging AI Applications. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*. USENIX Association, USA, 561–577.

[21] Jim Ormond. 2018. ACM Recognizes Innovators Who Have Shaped the Digital Revolution.

[22] Yongjoo Park, Michael Cafarella, and Barzan Mozafari. 2016. Visualization-aware sampling for very large databases. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. IEEE, 755–766.

[23] Yongjoo Park, Barzan Mozafari, Joseph Sorenson, and Junhao Wang. 2018. Verdictdb: Universalizing approximate query processing. In *Proceedings of the 2018 International Conference on Management of Data*. 1461–1476.

[24] Yongjoo Park, Jingyi Qing, Xiaoyang Shen, and Barzan Mozafari. 2019. BlinkML: Efficient maximum likelihood estimation with probabilistic guarantees. In *Proceedings of the 2019 international conference on management of data*. 1135–1152.

[25] Jeffrey M Perkel. 2018. Why Jupyter is data scientists' computational notebook of choice. *Nature* 563, 7732 (2018), 145–147.

[26] PBC Posit Software, PBC formerly RStudio. 2023. Posit RStudio. posit.co/.

[27] Python. 2023. Python - Pickle. docs.python.org/3/library/pickle.html.

[28] Python. 2023. Python - Generators. wiki.python.org/moin/Generators.

[29] Shreya Shankar, Stephen Macke, Sarah Chasins, Andrew Head, and Aditya Parameswaran. 2022. Bolt-on, compact, and rapid program slicing for notebooks. *Proceedings of the VLDB Endowment* 15, 13 (2022), 4038–4047.

[30] Nikhil Sheoran, Supawit Chockchowwat, Arav Chheda, Suwen Wang, Riya Verma, and Yongjoo Park. 2023. A step toward deep online aggregation. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 1–28.

[31] The IPython Team. 2023. IPython Interactive Computing. ipython.org/.

[32] The IPython Team. 2023. Jupyter checkpoint. jupyter-server.readthedocs.io/en/latest/developers/contents.html.

[33] The IPython Team. 2023. Jupyter store magic. ipython.readthedocs.io/en/stable/config/extensions/storemagic.html.

[34] Cornell University. 2021. SKLearn Tweet Classification. github.com/CornellCAC/CVW_PyDataSci2/blob/master/code/sklearn_tweet_classification.ipynb.

[35] Nathaniel Weinman, Steven M Drucker, Titus Barik, and Robert DeLine. 2021. Fork it: Supporting stateful alternatives in computational notebooks. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*.