The Design Space of Computational Notebooks: An Analysis of 60 Systems in Academia and Industry

Sam Lau, Ian Drosos, Julia M. Markel, Philip J. Guo
UC San Diego
La Jolla, CA, USA

***Abstract*—Computational notebooks such as Jupyter are now used by millions of data scientists, machine learning engineers, and computational researchers to do exploratory and end-user programming. In recent years, dozens of different notebook systems have been developed across academia and industry. However, we still lack an understanding of how their individual designs relate to one another and what their tradeoffs are. To provide a holistic view of this rapidly-emerging landscape, we performed, to our knowledge, the first comprehensive design analysis of dozens of notebook systems. We analyzed 60 notebooks (16 academic papers, 29 industry products, and 15 experimental/R&D projects) and formulated a design space that succinctly captures variations in system features. Our design space covers 10 dimensions that include diverse ways of importing data, editing code and prose, running code, and publishing notebook outputs. We conclude by suggesting ways for researchers to push future projects beyond the current bounds of this space.**

***Index Terms*—computational notebooks, literate programming**

I. Introduction

Over the past decade, computational notebooks such as Jupyter [1] have become popular programming environments for data scientists, machine learning engineers, computational researchers, and business analysts [2]. As modern embodiments of Knuth's *literate programming* vision [3], notebooks enable programmers to rapidly prototype and share explorations by interweaving expository prose, executable code, and rich program outputs (e.g., data tables, images, and interactive widgets). In addition, modern notebook systems are situated within complex end-to-end workflows that involve data ingest, computing environments, collaboration, and distribution.

Computational notebooks embody several themes that are of interest to the VL/HCC community: end-user programming [4], exploratory programming [5], live programming [6], and literate computing [7]. Both academic researchers [8]–[11] and industry practitioners [12], [13] have studied how people use notebooks in their work. These findings have informed the design of new systems that reduce common user frustrations: For instance, Verdant [14], [15] helps users manage the abundance of code and output versions created within notebooks, and Stitch Fix's Nodebook [16] eliminates out-of-order cell execution to improve reproducibility of results.

At present, dozens of such notebook systems have been developed in both academia and industry, but most are one-off designs meant to address a specific user need or, for academic projects, to prototype a novel user interaction technique. Despite this recent proliferation of notebook systems and the gradual maturing of this field, we still lack a systematic understanding of how these dozens of individual designs relate to one another and what their tradeoffs are. To provide this holistic overview, we developed the first comprehensive design space of computational notebooks by analyzing dozens of notebook systems across academia and industry.

In HCI research, a *design space* succinctly captures multiple dimensions of variation in possible system features within a given domain; each individual system covers a specific range in its design space. Researchers have mapped out design spaces for systems in domains such as end-user programming [17], [18], information visualization [19], [20], and tangible user interfaces [21]. In this paper, we extend this analysis technique to the domain of computational notebooks.

Figure 1 shows the design space we created by analyzing the features of 60 notebook systems (16 academic papers, 29 industry products, and 15 experimental/R&D projects). We grouped our 10 design space dimensions into four major stages of a data science workflow: importing data into notebooks, editing code and prose (editor style, supported programming languages, versioning, collaboration), running code to generate outputs (cell execution order, liveness [6], execution environment, and cell outputs), and publishing notebook outputs.

We found that industry products often prioritize widespread adoption, so their designs do not deviate much from that of the widely-used Jupyter Notebook format. In contrast, academic and R&D projects can afford to experiment more with different kinds of cell execution orders, live programming, and interactive outputs. That said, most designs are still restricted by people's current data science workflows, which involve interfacing with desktop computers using keyboards and mice. To innovate beyond this frontier, we suggest supplementing future notebook systems with non-desktop computing devices such as mobile and ubiquitous computing devices. Another way to push beyond current designs is to focus on user groups other than professional data scientists, such as educators, artists, or those in low-resource computing environments.

In sum, this paper's contributions are:

- A definition of "computational notebook" (Section IIIA)
- The first comprehensive design analysis of computational notebooks, identifying 60 notebook systems across three categories: academic, product, and experimental/R&D.
- A design space that summarizes notebook features, and ideas for how researchers can innovate beyond its bounds.

978-1-7281-6901-9/20/$31.00

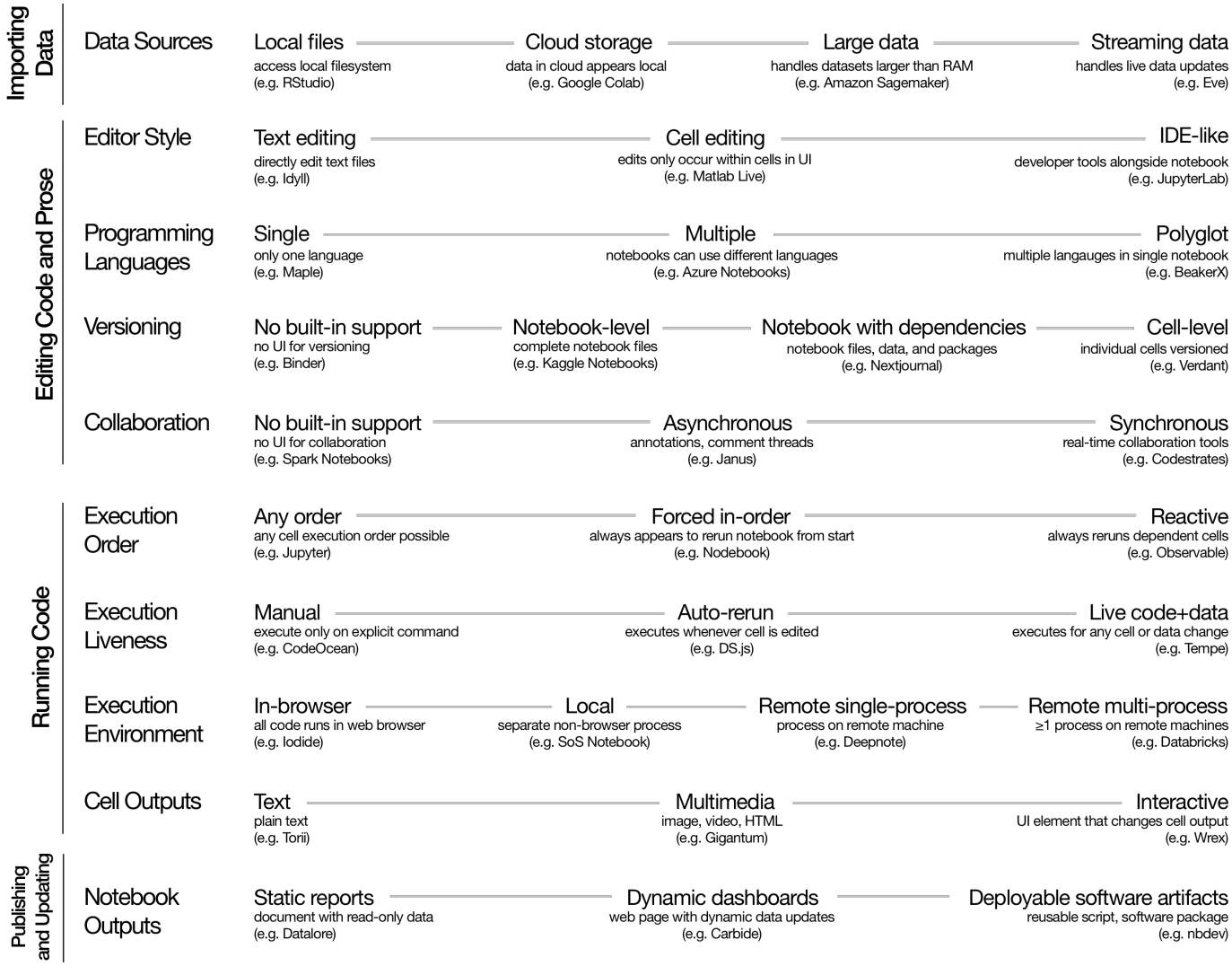

Fig. 1. The design space of computational notebooks, which we formulated by analyzing the features of 60 notebook projects across academia and industry. As Table I shows, each individual project can occupy multiple points along each dimension of this design space.

II. Related Work

Computational notebooks trace their lineage back to the vision of literate programming that Knuth first articulated in the early 1980s [3]. Knuth envisioned software being written like literature, with code and expository text interwoven into a single document to facilitate a natural human reading order. To implement this vision, he created the WEB system to interweave Pascal code with TeX-formatted exposition. The Mathematica scientific environment extended these ideas by creating the first computational notebook in 1988 [22], followed by a similar feature in Maple in 1989 [23]. Because early notebooks were embedded within proprietary environments, they remained niche products within the scientific community for the next two decades. As free and open-source web technologies matured in the 2000s, Perez and Granger developed the IPython Notebook in 2011 [24], which evolved into the popular Jupyter Notebook in 2015 [1].

The notebook systems we analyze embody several facets of programming research that are of interest to the VL/HCC community: end-user programming, live programming [6], and literate computing. Notebooks are widely-used environments for exploratory [5] and end-user programming, which Ko et al. define as coding as a means to an end (e.g., to produce data science insights or research findings) rather than to create artifacts for broader public use [4]. That said, some systems in Table I have affordances for creating public artifacts such as web dashboards or reproducible software packages. Also, modern notebooks embody the spirit of literate *computing*, which is a generalization of Knuth's literate programming vision that mixes code with both exposition and rich outputs such as images, videos, and interactive widgets [7].

Aside from publishing academic papers on new notebook systems (summarized in Section IV), researchers have also studied how data scientists use notebooks and what obstacles they face. Rule et al. discovered a pervasive tension between Jupyter notebooks used for exploration (i.e., prototyping anal-

yses) and explanation (i.e., sharing research results) [10]. Kery et al. found that data scientists often cleaned up exploratory notebooks into explanatory ones by using ad-hoc methods like alternately expanding and consolidating code cells [9]. A broader study discovered pain points for notebook users along the entire workflow spectrum ranging from setup to exploration to sharing [8]; the 60 systems that we analyze in this paper are often attempts to address specific pain points along that spectrum. Lastly, CSCW researchers investigated how data science teams collaborate using notebooks and discovered the limitations of current notebook systems for both asynchronous and synchronous collaboration [11], [25], [26], which we also include as a dimension in our design space.

The closest prior work to ours was a PPIG workshop paper [7] where the authors surveyed 12 literate computing projects, 7 of which were notebooks (which we also included in our study). Similarly, Merino et al. surveyed 12 notebooks (which we also included in our study) and 4 other systems as a formative study to inform the design of their Bacatá system, which generates notebook UIs for DSLs [27]. Both of these analyses were much smaller in scope and did not focus on mapping a design space of technical notebook features. To our knowledge, ours is the first comprehensive design study of dozens of notebook systems across academia and industry.

III. Methods

We performed a qualitative analysis of 60 computational notebook systems across academia and industry.

A. Defining the Term "Computational Notebook"

Our first task was to define "computational notebook" to determine what projects to include in our study. On one hand, we did not want to restrict ourselves to only variants of Jupyter Notebooks, which is now the most popular format [1]. On the other hand, we did not want to expand our definition too much to include projects that were too distant. Based on prior studies of notebook use [8]–[11], [25], [26], [28] and our own experiences as notebook researchers, ***we define a computational notebook as a system that supports literate programming [3] using a text-based programming language (e.g., Python, R, JavaScript, or a DSL [27]) while interweaving expository text and program outputs into a single document.***

This definition excludes "non-computational" notebooks such as electronic lab notebooks [29] used by scientists to track their daily work, general notetaking tools such as OneNote and Evernote, and modern notetaking apps with spreadsheet-like computational capabilities (e.g., Airtable [30], Coda [31], Notion [32]). It also excludes projects like Distill.pub [33] and Explorable Explanations [34], which provide widgets for users to interactively tune parameters but which do *not* support writing text-based programming languages in the notebook.

Since many notebooks are implemented as Jupyter extensions, it can be hard to tell what counts as a standalone system. In general, we counted academic contributions that are published papers and industry contributions that are marketed as either standalone products (both open-source and proprietary) or substantive R&D efforts. We excluded smaller extensions like postprocessing scripts that only transform notebook outputs (e.g., nbinteract [35], ThebeLab [36], Voila [37]).

B. Finding Notebook Systems Across Academia and Industry

We sought to be comprehensive in finding all publicly-known notebook systems that fit our definition. For academic projects, we performed a literature search starting with research papers at venues including VL/HCC, CHI, UIST, and CSCW, and then branched outward via bibliography crawls and Google Scholar searches for notebook-related terms (e.g., "computational notebook", "literate programming"). For industry projects, we consulted with data scientist colleagues, performed web searches for relevant terms, watched talk videos at practitioner conferences such as JupyterCon [38], and searched through discussion forums and blogs. We iterated on our list by showing drafts to other notebook researchers and practitioners, including core members of the Jupyter project team [39], to receive additional suggestions to include.

C. Data Overview and Analysis

We found 60 total systems (circa Feb 2020), summarized in Table I. For each, multiple members of our research team independently enumerated its features by reading relevant papers and user guides/documentation, watching demo and talk videos, and trying out notebooks that were publicly available. We focused our qualitative content analysis purely on technical features rather than business-oriented features such as pricing, licensing models, or target markets. The research team met multiple times to merge our notes, categorize them together into themes using an inductive analysis approach [40], and iterate until we could not find additional features. To formulate a design space from these features, we followed a similar methodology as Segel and Heer [19], who made a design space of narrative visualizations from content analysis of 58 visualization webpages. Specifically, we sought to distill generalizable concepts from specific notebook features and find what concepts multiple notebooks shared in common. For instance, whether a notebook supports Python or R is an implementation detail, but the fact that a single or multiple languages can coexist within a notebook is a more generalizable design concept. We made several iterations as a team before finalizing our 10 dimensions, which are grounded in the phases of a data science workflow. We originally considered a larger number of more specific dimensions (e.g., notebook distribution) but merged them into four main workflow phases.

D. Study Design Limitations

Although we sought to be comprehensive, there is no guarantee that we found all relevant computational notebook systems; we do not have access to internal company projects nor to unpublished academic projects. As the first study of this scope, though, we believe our set of 60 is sufficient for surveying trends and forming the basis for future analyses.

Our goal was to use these 60 systems to map out a design space, so we do not describe *all* features of each one. Since

we organized findings around the 10 design space dimensions rather than describing each notebook, we omit specific feature details such as the programming languages supported by each. Also, there is subjectivity in how we selected our dimensions, so other researchers may have picked different dimensions.

Since Jupyter is now the most popular platform, systems built on top of it are overrepresented in our data, accounting for around half of our 60 projects. That said, we included other ecosystems such as R, Mathematica, and Spark; we also found experimental and academic research projects that built their own bespoke notebook interfaces separate from Jupyter.

IV. Results Overview: Sources of Notebooks

We identified three sources of notebooks, summarized in Table I: academic, product, and experimental/R&D.

Academic: The first source includes 16 research projects that have been published as academic papers. Most of these are from university labs, but MessyNotebooks [41], Tempe [42], [43], and Wrex [44] are from Microsoft Research.

Each academic project usually makes one distinct and precise innovation. Some are implemented as Jupyter extensions: Wrex [44] adds programming-by-example to Jupyter by enabling users to interact with data tables and having the system synthesize data wrangling code; Callisto [45] adds inline chat and deictic references to facilitate anchored discussions around notebook cells; Janus [46] adds cell folding and annotations; Verdant [14], [15] adds automated cell-level micro-versioning and development history visualizations; MessyNotebooks [41] generates slices of notebook cells that produce a given output; Bacatá [27] synthesizes Jupyter UIs for DSLs; Dataflow [47] and SoS [48] notebooks track provenance for reproducibility.

Many academic projects are not tied to Jupyter since their goal is to demonstrate novel ideas, not necessarily to gain wide adoption. Three such systems are built atop the Webstrates [49] platform: Codestrates [50], [51], Vistrates [52], and Labbook [53]. Also, PolyJuS [54], Tempe [42], [43], and Torii [55] implement their own custom UIs, Idyll [56] provides a literate programming [3] markup format, and DS.js [57] turns existing data-rich webpages into computational notebooks.

Product: In contrast to precisely-targeted innovations of academic research, notebook products are monolithic solutions aimed at broad adoption. We define *product* broadly to mean any system intended to serve a sizable user population; by this definition, products can be free or paid, open-source or closed-source, and maintained by for-profit companies or non-profits. Many of these are built on top of Jupyter. Note that since Jupyter is a platform (and accompanying nonprofit organization) rather than a specific product, in Table I we separately show the two official products maintained by this organization: the original Jupyter Notebook UI and the newer JupyterLab IDE [1], [58]; for these two, we count only their features from default installations without any extensions.

Many products are companies hosting Jupyter in the cloud to provide "Jupyter-as-a-service." They add features such as access to large-scale datastores, fast compute engines, real-time collaboration, or integration with other cloud services. General-purpose systems include Binder [59], Databricks [60], Gigantum [61], IBM Watson Studio [62], Kaggle Notebooks [63], Microsoft Azure Notebooks [64], and Mode [65]. Some are specialized for collaboration: Google Colab [66], Datalore [67], Deepnote [68], and Kyso [69]. Others have a domain-specific focus, such as CoCalc [70] (mathematics), Kogence [71] (supercomputing), Quantopian [72] (finance), and CodeOcean [73] and Nextjournal [74] (reproducibility).

Besides systems built atop Jupyter Notebooks, there are several other major ecosystems: RStudio [75] provides an open-source IDE for R where users can write RMarkdown Notebooks [76] and make interactive dashboards with the Shiny framework [77]. Mathematica (now renamed to Wolfram) has maintained their own proprietary notebook format since 1988 [22], [78]; Maple [79] and MATLAB [80] have similar embedded notebooks. Spark Notebooks [81] and Zeppelin [82] provide notebook interfaces to access the Apache Spark big data ecosystem using Scala and SQL. Spyder [83] provides a MATLAB-style IDE for Python with notebook support. Finally, startups such as Observable [84], RunKit [85], and Streamlit [86] have created their own notebook-based environments for rapid prototyping of data-driven web apps.

Experimental / R&D: This final source contains 15 systems that fall in between academic and product types. They differ from academic research in that they are not formally evaluated or published as papers, and they differ from industry products in that they are less polished. Note that the line between experimental/R&D and industry products may be blurry. One distinction is that experimental/R&D projects do not feel as "standalone" as products do, and their websites are often just a GitHub code repository with some technical documentation.

Nonprofits have created experimental systems such as Iodide [87] from Mozilla, nbdev [88] from fast.ai, Livebook [89] from Ink & Switch, and the nteract notebook UI [90] and Papermill [2], [91] production scheduler from NumFOCUS. Independent creators have started projects such as Carbide [92], Leisure [93], and Eve (a startup that was in R&D phase) [94].

Some companies also release their R&D projects as open-source software: Stitch Fix altered Jupyter with forced in-order cell execution in their Nodebook [16] system. Stripe shared their reproducible production notebook workflows [95]. Finance firm Two Sigma extended Jupyter with polyglot JVM and Spark support in BeakerX [96], which is similar to Netflix's Polynote [97] project. Another finance firm, Capital Fund Management, released Jupytext [98] to provide a Markdown-like text editing experience for notebooks. Several companies also developed plug-ins to integrate notebooks into traditional IDEs, such as Hydrogen [99] for the Atom IDE and Microsoft's Jupyter mode for Visual Studio Code [100].

V. The Design Space of Computational Notebooks

Table I summarizes how all 60 notebooks we analyzed fit into our design space from Figure 1. We grouped our 10 design space dimensions by their typical order in a computational workflow: importing data into notebooks, editing code and

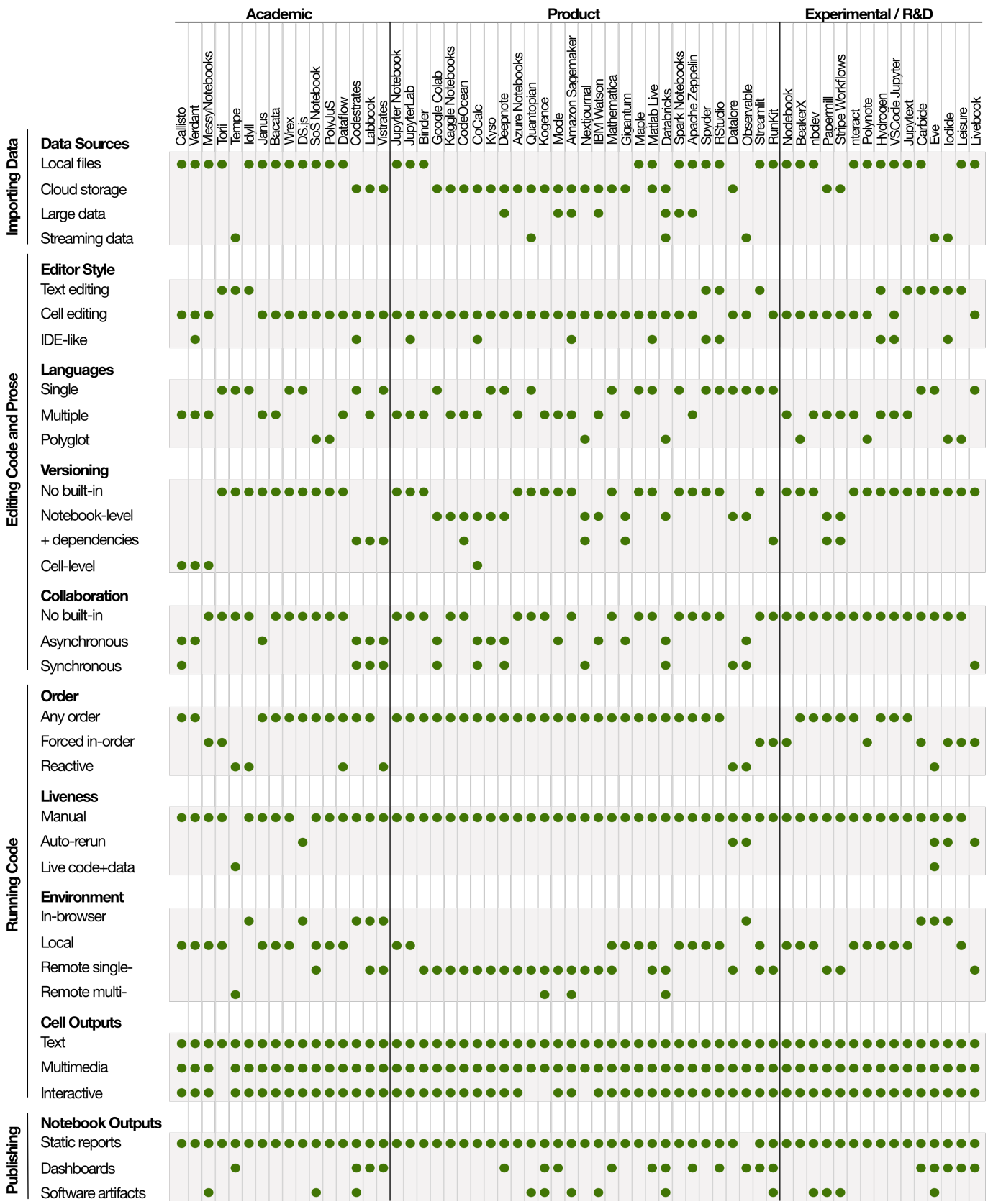

TABLE I

HOW THE 60 NOTEBOOK SYSTEMS IN OUR STUDY (COLUMNS) FIT INTO THE DESIGN SPACE DIMENSIONS (ROWS) WE ILLUSTRATED IN FIGURE 1.

prose, running code to generate cell outputs, and publishing notebook outputs. Note that our design space covers more than just the core UI of the notebook itself; it captures the entire end-to-end *system* that the notebook resides within.

A. Importing Data into Notebooks

1) Data Sources: The baseline is accessing only *local files*, which means that it is up to the user to manage their data sets and import them using a programming language and libraries (e.g., the Python CSV reader module). Default Jupyter Notebooks and most non-cloud variants fall into this category.

Cloud storage means that the system exposes data sets stored in a cloud service as though they were local files. For example, Google Colab [66] lets users mount a Google Drive folder and then access its files in the notebook.

Large data means that the system has built-in support for accessing data that is too large to fit into RAM; these datasets cannot be fully imported into a running notebook session. For instance, Databricks [60] and Spark Notebooks [81] allow users to write SQL queries within notebook cells to access selected slices of terabyte-scale data within a session.

Finally, *streaming data* means the notebook allows users to both connect to real-time streaming data sources *and* automatically update its computed outputs as new data arrives.

B. Editing Code and Prose

Four design space dimensions relate to editing interfaces:

2) Editor Style: At one end, systems like Iodide [87] and Streamlit [86] let users write code and prose in an enhanced text editor; those systems compile text into interactive notebook-like webpages. In the middle, most notebooks provide a more structured cell-based editor where users can independently edit, run, and rearrange cells within a notebook file. At the other extreme are fully-featured IDEs (e.g., JupyterLab, RStudio) that embed notebooks alongside data inspectors, debuggers, terminals, and other developer tools. Codestrates [50] is the most unique since it allows users to write JavaScript to customize its own UI; although it starts as a basic cell-based editor, it can be reprogrammed into an IDE.

3) Programming Languages: The baseline here is support for coding in a single language (e.g., JavaScript for Codestrates [50], a JavaScript variant for Observable [84], a Datalog-like DSL for Eve [94]). Next is support for multiple languages, but each notebook file can only run a single language; default Jupyter works this way. Polyglot notebooks such as Polynote [97] and PolyJuS [54] enable users to natively mix multiple language within the same notebook, which is useful when the most convenient libraries for different stages of a data analysis are in different languages (e.g., web scrapers in Perl, machine learning in Python, statistics in R).

4) Versioning: Studies found that notebooks are often used for exploratory workflows [5] involving lots of trial-and-error [8]–[10] and that users struggle to keep track of many versions of analysis code and outputs. The baseline here is no built-in support, which means that it is up to users to track their own versions by, say, committing notebooks to Git. Next is *notebook-level versioning*, where the system automatically saves versions of entire notebook files so users do not need to learn about version control. Next is *notebook+dependency versioning*, where systems like Nextjournal [74] save not only raw notebook files but also accompanying data sets and environment dependencies (e.g., 3rd-party libraries and packages). This enables notebooks to be reliably re-run to reproduce the same results, which is important for replicability of scientific results. At the most extreme is *cell-level versioning*, where each code and output cell can be separately auto-versioned. For instance, Verdant keeps track of individual cell edits at the code AST level so that "each syntactically meaningful span of text in the code can be recorded with its own versions" [14].

5) Collaboration: Studies also found that notebooks are often used in collaborative data science workflows [11], [25], [26]. The baseline here is no built-in collaboration support, so users must coordinate via a mix of third-party tools (e.g., workplace chat with Slack, code review and issue commenting with GitHub's web UI). Next is built-in support for asynchronous collaboration, most commonly via annotations and comment threads next to notebook cells. Finally, some support synchronous collaboration by embedding real-time chat and allowing multiple users to concurrently edit the same notebook cells, akin to Google Docs. (Note that Google Colab was originally designed for synchronous collaboration but its real-time sync API is currently down [101].)

C. Running Code to Generate Cell Outputs

We identified four design space dimensions related to code execution: order, liveness, environment, and cell outputs.

6) Execution Order: Notebooks contain a series of code cells, each of which can be run independently to produce outputs. Most allow cell execution in *any order*, which means users can run cells out of order and multiple times as they iterate. However, both academic studies [8]–[10] and industry reports [12], [13] show that any-order execution is a major source of frustration for notebook users; specifically, it is hard to tell which exact series of cell executions led to the notebook's current state or how to reproduce that state. A recommended best practice is for users to clean up their cells and run them all in-order to create a final shareable notebook.

To reduce these frustrations, some notebooks implement *forced in-order execution*. This means when the user clicks "Run" on a particular cell, the notebook will reset its global state and then run all cells from the top until that cell. (That is the user's mental model, but in reality these systems such as Nodebook [16] track dataflow dependencies so that not all cells need to be re-run each time.) This design eliminates the problems of out-of-order cell execution, of cells being run multiple times and altering global state in unexpected ways, and of cells being deleted after being run but their outputs still remaining in the notebook's global state. But the tradeoff is lack of flexibility, since some users appreciate rapid prototyping by running cells in any order. As a middle ground,

systems like MessyNotebooks *retroactively* turn an existing notebook into a forced in-order one by backward-slicing only the cells that lead to a certain desired output [41].

Finally, some notebooks implement a *reactive* execution model, much like spreadsheets do. In a reactive model, running a cell triggers the notebook to automatically re-run all other cells that depend on values defined or altered in that cell. This model gives users the flexibility to write their code in any notebook cell regardless of order, but run-time value dependencies are automatically tracked by the notebook so there is still a predictable (albeit non-linear) execution order.

7) Execution Liveness: Most notebooks run a cell only when the user clicks "Run" or uses a keyboard shortcut. Some have simple forms of live execution where a cell is automatically re-run whenever the user momentarily pauses editing or moves their cursor to another cell; this type of liveness (level-3 in Tanimoto's taxonomy [6]) provides immediate visual feedback, which can speed up the iteration and debugging cycles. Systems like Tempe implement a more sophisticated form of level-4 liveness [43] by updating the output live not only after the user edits a cell but also in real time when the data stream that the cell accesses has new data arrive in it.

8) Execution Environment: Notebook systems also vary in *where* they execute code. JavaScript-based notebooks such as Codestrates [50] and Observable [84] run code directly in the user's browser, which is a convenient way to eliminate complex installation and setup issues; Iodide [87] compiles a Python environment to WebAssembly to run directly in the browser. Next, most desktop Jupyter variants run code locally on the user's computer in a separate non-browser process that communicates with the notebook's browser-based UI. Cloud-hosted Jupyter systems run notebook code in a single process on a remote machine with access to more computational power and GPUs than the user's machine. Finally, systems like Databricks [60] and Kogence [71] automatically set up notebooks to run in parallel and distributed computing environments involving *multiple* remote machines at once.

9) Cell Outputs: When each cell is executed, it produces output directly underneath or beside it. This output can range from plain text (akin to a terminal or REPL) to multimedia (e.g., data tables, images, sound clips, videos) all the way to interactive widgets. These widgets allow users to set parameter values and re-run cells in order to quickly prototype variants of data analyses. Note that the systems in Table I that are built atop Jupyter support all types of outputs since Jupyter comes bundled with an interactive widget library. Finally, some research systems go even further by allowing interactive widgets to alter the code in the accompanying cell: Carbide [92] exposes sliders whose values are synchronized with numeric literals in code cells, and Wrex allows users to do programming-by-example [44] by entering example values in output tables and having the system automatically synthesize Python data wrangling code to insert into nearby code cells.

D. Publishing and Updating Notebook Outputs

Notebooks are often used for personal exploration and end-user programming workflows [5], [10], but users sometimes intend to share notebooks with others. They do so by publishing it in various formats, which have different levels of support for updating in response to post-publication feedback.

10) Notebook Outputs: The baseline here is for notebooks to generate *static reports* that are read-only documents. For instance, Jupyter Notebooks can be exported as static HTML files, but those might contain JavaScript-powered interactive visualizations (e.g., using D3 [102] or Vega [103]). Next, some can be exported as *dynamic dashboards* that (unlike static HTML) automatically update and recompute cell outputs as new streaming data arrives. Both static and dynamic formats facilitate common explanation use cases [10] for notebooks.

Lastly, some have stretched notebooks beyond data science use cases to turn them into *deployable software artifacts*. For instance, nbdev [88] exports notebooks as self-contained Python libraries that can be imported into other software projects; IBM Watson Studio [62] deploys notebooks as machine learning models with live monitoring; Codestrates [50] and Eve [94] allow users to create standalone web applications.

VI. Discussion

High-Level Trends: Industry products are geared toward scalability and adoption, so their designs tend to be standard Jupyter Notebooks integrated into a cloud platform (e.g., for data sources, automatic versioning, execution environments, and distribution). Although one could in theory emulate these features by installing open-source software and manually provisioning cloud resources, these products provide a level of convenience that fosters widespread adoption.

In contrast, academic and R&D projects are more experimental, adopting less conventional designs for execution order, liveness, and interactive outputs. They also contain properties that are hard to fully capture in our design space: For instance, Codestrates [50] and systems built on it emphasize malleability where the entire notebook programming interface can be adjusted on-the-fly by editing its code. Systems like DS.js [57], Idyll [56], and Torii [55] target educational use cases rather than being for professional data scientists, so their interfaces differ from conventional programming-oriented notebooks.

Design Tradeoffs: Each dimension in Figure 1 has associated tradeoffs. 1) Data sources: moving to the cloud may sacrifice privacy and impose more friction for working with legally-protected data (e.g., COPPA, HIPAA), 2,3) Editor style and languages: IDE-like interfaces and polyglot notebooks make the UI more complex for novices to learn, 4) Versioning: finer-grained versioning again requires greater UI complexity to let users organize and sift through versions, 5) Collaboration: integrated collaboration may not always be better since users may prefer to use their own organization's external tools (e.g., Slack), 6) Execution order: in-order is simplest but lacks flexibility; reactive can be hard for novices since it is not clear which cells execute when a particular one is edited

(*hidden dependencies* in Green's cognitive dimensions [104]), 7) Liveness: again it may not be clear what code executes if the user does not initiate those actions, 8) Environment: remote execution is higher-performance than local but can be harder to debug, 9,10) Cell/notebook outputs: the more interactive, the less of a *closeness-of-mapping* (in Green's cognitive dimensions [104]) there is between code and outputs.

There are also design tradeoffs *across* dimensions: For instance, an in-browser JavaScript notebook may have a hard time supporting polyglot programming (though not impossible as more languages now compile to WebAssembly); and notebooks that rely on remote multi-process execution may be hard to package up as portable/versioned software artifacts, since they often depend on the specifics of their execution environment (e.g., supercomputer infrastructure from Kogence [71]).

Notebook Toolkits: As a thought experiment, what about a "kitchen-sink" approach where a hypothetical system covers the *entire design space* of Figure 1 by providing *all* these features as options? While that might seem appealing since it would foster customization, from a usability perspective its interface would get overly complex with too many modes. And from a software engineering perspective it would be hard to get such a monolithic system working robustly. Perhaps the most practical path along this direction is to take a toolkit approach like D3 [102] by turning each dimension of Figure 1 into a well-scoped software module and then allowing developers to compose them together using a "grammar of notebook systems" akin to the grammar of graphics implemented by the Vega [103] ecosystem. That way, software developers can easily build bespoke notebooks for their intended user group.

Charting the Future of Computational Notebook Research: Our study provides a map of the current state of the notebook world, as summarized in Table I. In the short term, industry products will continue to fill in more points along our design space. Meanwhile, researchers in both academia and industry can expand the boundaries of this map into uncharted territory. What are possible ways forward? Here are a few directions:

There are still many opportunities for needs-based research of the sort embodied by some of the academic projects in Table I. The HCI approach of understanding the needs of a user group, building a novel notebook system to address those needs, and evaluating it on that user group will continue to yield research innovations. To stretch the bounds of our current design space, we encourage researchers to broaden out to target user groups beyond professional data scientists, such as educators, artists, designers, journalists, digital humanities scholars, young children, older adults, people with accessibility needs, or those in low-resource computing environments.

Though important, we believe that needs-based research will mostly continue to fill in our design space but not expand much beyond it. That is because people's needs are still tied to their existing workflows; for computational notebooks that means using keyboards to write textual code on desktop or laptop computers. Thus, one way to go beyond the current design space is to think about the abundance of ***non-desktop computing devices*** available to us in our daily lives and how we could use those for data-oriented work. For the price of a laptop computer we can now buy several low-cost tablet- and phone-sized devices. Weiser's ubiquitous computing vision is already here since we have plentiful access to pads (e.g., iPads) and tabs (e.g., phones/watches) [105] but are not yet taking full advantage of them for programming. Practically, we do not envision these devices replacing keyboard+mouse+monitors but rather *supplementing* them with auxiliary displays, dynamic magic lenses [106] (e.g., hovering a phone over a monitor to peek into code or data), and touch and voice-based inputs. Beyond pads and tabs, what about other ubicomp devices such as wearables, smart glasses, augmented/mixed-reality interfaces, portable projectors with 3D depth cameras, and large-scale tabletop and wall displays? Specifically, Table I shows that there is not much variation in cell outputs; a way to innovate along this dimension is by adopting different sorts of displays beyond ordinary computer screens. Similarly, many systems do not support synchronous collaboration, and when they do, it is often simple forms of multi-user text editing; once again ubiquitous computing techniques can point the way toward more expressive forms of synchronous collaboration.

The above was a technology-centric approach (start with novel devices and prototype outward), so a complementary approach is to be task-centric. What do people actually want to do with notebooks? Examples include deriving business insights, conducting scientific research, scholarly communication, collaboration, education, and personal creative expression. What future interfaces that mix code, data, and multimedia might aid these tasks, regardless of whether they "look" like the cell-based notebooks of today? We encourage researchers not to have their thinking restricted by current notebook formats simply due to the zeitgeist of recent academic publication trends. One metric for success here is how *different* a new project looks from all the systems in Table I.

A more radical way to expand beyond current notebook designs is to ban the term "computational notebook" altogether and generalize it into interfaces for literate computing (i.e., literate programming + interactive media) [7]. One could argue that Figure 1 is actually a design space of literate computing, of which notebooks are a subset. This approach may mean reviving classic lines of work embodied by Smalltalk, Boxer, and HyperCard [7], which Bret Victor et al. are doing at Dynamicland [107]. One challenge of such ambitiously generalizable research is to balance expressiveness with usability/learnability in order to avoid the Turing tar-pit, "in which everything is possible but nothing of interest is easy" [108].

VII. Conclusion

We identified three main sources of computational notebooks (academic, industry products, and experimental/R&D) and presented the first comprehensive study of 60 notebook projects, which resulted in a ten-dimensional design space that spans their technical features. In closing, we encourage researchers to stretch the bounds of this space by pursuing project ideas that go beyond fulfilling current user needs.

ACKNOWLEDGMENTS

Huge thanks to Brian Granger, Jim Hollan, Paul Ivanov, Clemens Nylandsted Klokmose, Adam Rule, Carol Willing, and Xiong Zhang for their feedback. This material is based upon work supported by the National Science Foundation under Grant No. NSF IIS-1845900.

REFERENCES

[1] J. Perkel, "Why jupyter is data scientists' computational notebook of choice," *Nature*, vol. 563, pp. 145–146, 11 2018.

[2] M. Ufford, M. Pacer, M. Seal, and K. Kelley, "Beyond Interactive: Notebook Innovation at Netflix," https://netflixtechblog.com/notebook-innovation-591ee3221233, accessed: 2020-02-01.

[3] D. E. Knuth, "Literate programming," *Comput. J.*, vol. 27, no. 2, pp. 97–111, May 1984. [Online]. Available: http://dx.doi.org/10.1093/comjnl/27.2.97

[4] A. J. Ko, R. Abraham, L. Beckwith, A. Blackwell, M. Burnett, M. Erwig, C. Scaffidi, J. Lawrance, H. Lieberman, B. Myers, M. B. Rosson, G. Rothermel, M. Shaw, and S. Wiedenbeck, "The state of the art in end-user software engineering," *ACM Comput. Surv.*, vol. 43, no. 3, pp. 21:1–21:44, Apr. 2011. [Online]. Available: http://doi.acm.org/10.1145/1922649.1922658

[5] M. B. Kery and B. A. Myers, "Exploring exploratory programming," in *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, Oct 2017, pp. 25–29.

[6] S. L. Tanimoto, "A perspective on the evolution of live programming," in *2013 1st International Workshop on Live Programming (LIVE)*, May 2013, pp. 31–34.

[7] B. Fog and C. N. Klokmose, "Mapping the landscape of literate computing," in *Proceedings of the 30th Annual Workshop of the Psychology of Programming Interest Group*, ser. PPIG, 2019.

[8] S. Chattopadhyay, I. Prasad, A. Z. Henley, A. Sarma, and T. Barik, "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*, ser. CHI '20. New York, NY, USA: ACM, 2020.

[9] M. B. Kery, M. Radensky, M. Arya, B. E. John, and B. A. Myers, "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*, ser. CHI '18. New York, NY, USA: ACM, 2018, pp. 174:1–174:11. [Online]. Available: http://doi.acm.org/10.1145/3173574.3173748

[10] A. Rule, A. Tabard, and J. D. Hollan, "Exploration and explanation in computational notebooks," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, ser. CHI '18. New York, NY, USA: ACM, 2018, pp. 32:1–32:12. [Online]. Available: http://doi.acm.org/10.1145/3173574.3173606

[11] A. Y. Wang, A. Mittal, C. Brooks, and S. Oney, "How data scientists use computational notebooks for real-time collaboration," *Proc. ACM Hum.-Comput. Interact.*, vol. 3, no. CSCW, Nov. 2019. [Online]. Available: https://doi.org/10.1145/3359141

[12] J. Grus, "I Don't Like Notebooks - Joel Grus - #JupyterCon 2018," https://docs.google.com/presentation/d/1n2RlMdmv1p25Xy5thJUhkKGvjtV-dkAIsUXP-AL4ffI/edit?urp=gmail_link&usp=embed_facebook, 2018, accessed: 2020-02-01.

[13] "Jupyter Notebook 2015 UX Survey Results," https://github.com/jupyter/surveys/blob/master/surveys/2015-12-notebook-ux/analysis/report_dashboard.ipynb, accessed: 2020-02-01.

[14] M. Kery and B. Myers, "Interactions for untangling messy history in a computational notebook," ser. VL/HCC '18, 10 2018, pp. 147–155.

[15] M. B. Kery, B. E. John, P. O'Flaherty, A. Horvath, and B. A. Myers, "Towards effective foraging by data scientists to find past analysis choices," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, ser. CHI '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3290605.3300322

[16] "Nodebook from Stitch Fix," https://github.com/stitchfix/nodebook, accessed: 2020-02-01.

[17] B. Hartmann, L. Yu, A. Allison, Y. Yang, and S. R. Klemmer, "Design as exploration: Creating interface alternatives through parallel authoring and runtime tuning," in *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 91–100. [Online]. Available: https://doi.org/10.1145/1449715.1449732

[18] J. Brandt, M. Dontcheva, M. Weskamp, and S. R. Klemmer, "Example-centric programming: Integrating web search into the development environment," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '10. New York, NY, USA: ACM, 2010, pp. 513–522. [Online]. Available: http://doi.acm.org/10.1145/1753326.1753402

[19] E. Segel and J. Heer, "Narrative visualization: Telling stories with data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 6, pp. 1139–1148, Nov 2010.

[20] T. Horak, A. Mathisen, C. N. Klokmose, R. Dachselt, and N. Elmqvist, "Vistribute: Distributing interactive visualizations in dynamic multi-device setups," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, ser. CHI '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3290605.3300846

[21] G. W. Fitzmaurice, H. Ishii, and W. A. S. Buxton, "Bricks: Laying the foundations for graspable user interfaces," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '95. USA: ACM Press/Addison-Wesley Publishing Co., 1995, p. 442–449. [Online]. Available: https://doi.org/10.1145/223904.223964

[22] B. Hayes, *Thoughts on Mathematica*. Pixel, Jan/Feb 1990.

[23] "Wikipedia: Notebook interface," https://en.wikipedia.org/wiki/Notebook interface, accessed: 2020-02-01.

[24] "The IPython notebook: a historical retrospective," http://blog.fperez.org/2012/01/ipython-notebook-historical.html, accessed: 2020-02-01.

[25] M. Muller, I. Lange, D. Wang, D. Piorkowski, J. Tsay, Q. V. Liao, C. Dugan, and T. Erickson, "How data science workers work with data: Discovery, capture, curation, design, creation," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, ser. CHI '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3290605.3300356

[26] A. X. Zhang, M. Muller, and D. Wang, "How do data science workers collaborate? roles, workflows, and tools," vol. 1, no. CSCW. New York, NY, USA: Association for Computing Machinery, Jan. 2020.

[27] M. V. Merino, J. Vinju, and T. van der Storm, "Bacatá: Notebooks for DSLs, Almost for Free," in *Proceedings of the Conference Companion of the 4th International Conference on Art, Science, and Engineering of Programming*, ser. Programming '20. New York, NY, USA: Association for Computing Machinery, 2020.

[28] 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 writing and sharing computational analyses in jupyter notebooks," *PLOS Computational Biology*, vol. 15, no. 7, pp. 1–8, 07 2019. [Online]. Available: https://doi.org/10.1371/journal.pcbi.1007007

[29] P. J. Guo and M. Seltzer, "Burrito: Wrapping your lab notebook in computational infrastructure," in *Proceedings of the 4th USENIX Workshop on the Theory and Practice of Provenance*, ser. TaPP'12. Berkeley, CA, USA: USENIX Association, 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2342875.2342882

[30] "Airtable: Part spreadsheet, part database, and entirely flexible, teams use Airtable to organize their work, their way," https://airtable.com/, accessed: 2020-02-01.

[31] "Coda — A new doc for teams." https://coda.io/welcome, accessed: 2020-02-01.

[32] "Notion: all-in-one workspace," https://www.notion.so/, accessed: 2020-02-01.

[33] "Machine learning research should be clear, dynamic and vivid. distill is here to help," https://distill.pub/about/, accessed: 2020-02-01.

[34] "Explorable explanations, a hub for learning through play!" https://explorabl.es/, accessed: 2020-02-01.

[35] S. Lau and J. Hug, "Nbinteract: Generate interactive web pages from Jupyter notebooks," Master's Thesis, Master's thesis, EECS Department, University of California, Berkeley, 2018.

[36] "ThebeLab: turning static html pages into live documents," https://github.com/minrk/thebelab, accessed: 2020-02-01.

[37] "Voila," https://github.com/voila-dashboards/voila, accessed: 2020-02-01.

[38] "JupyterCon: The Official Jupyter Conference," https://conferences.oreilly.com/jupyter/jup-ny, accessed: 2020-02-01.

[39] "Jupyter receives the ACM Software System Award," https://blog.jupyter.org/jupyter-receives-the-acm-software-system-award-d433b0dfe3a2, May 2018, accessed: 2020-02-01.

[40] J. M. Corbin and A. L. Strauss, *Basics of qualitative research: techniques and procedures for developing grounded theory*. SAGE Publications, Inc., 2008.

[41] A. Head, F. Hohman, T. Barik, S. M. Drucker, and R. DeLine, "Managing messes in computational notebooks," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, ser. CHI '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3290605.3300500

[42] D. Fisher, B. Chandramouli, R. DeLine, J. Goldstein, A. Aron, M. Barnett, J. Platt, J. Terwilliger, and J. Wernsing, "Tempe: An interactive data science environment for exploration of temporal and streaming data," Tech. Rep., November 2014.

[43] R. DeLine, D. Fisher, B. Chandramouli, J. Goldstein, M. Barnett, J. Terwilliger, and J. Wernsing, "Tempe: Live scripting for live data," ser. VL/HCC '15, 10 2015.

[44] I. Drosos, T. Barik, P. J. Guo, R. DeLine, and S. Gulwani, "Wrex: A unifed programming-by-example interaction for synthesizing readable code for data scientists," in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, ser. CHI '20. New York, NY, USA: ACM, 2020.

[45] A. Y. Wang, Z. Wu, C. Brooks, and S. Oney, "Callisto: Capturing the "why" by connecting conversations with computational narratives," in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, ser. CHI '20. New York, NY, USA: ACM, 2020.

[46] A. Rule, I. Drosos, A. Tabard, and J. D. Hollan, "Aiding collaborative reuse of computational notebooks with annotated cell folding," *Proc. ACM Hum.-Comput. Interact.*, vol. 2, no. CSCW, Nov. 2018. [Online]. Available: https://doi.org/10.1145/3274419

[47] D. Koop and J. Patel, "Dataflow notebooks: Encoding and tracking dependencies of cells," in *Proceedings of the 9th USENIX Conference on Theory and Practice of Provenance*, ser. TaPP'17. USA: USENIX Association, 2017, p. 17.

[48] B. Peng, G. Wang, J. Ma, M. C. Leong, C. Wakefield, J. Melott, Y. Chiu, D. Du, and J. N. Weinstein, "SoS Notebook: an interactive multi-language data analysis environment," *Bioinformatics*, vol. 34, no. 21, pp. 3768–3770, 05 2018. [Online]. Available: https://doi.org/10.1093/bioinformatics/bty405

[49] C. N. Klokmose, J. R. Eagan, S. Baader, W. Mackay, and M. Beaudouin-Lafon, "Webstrates: Shareable dynamic media," in *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, ser. UIST '15. New York, NY, USA: ACM, 2015, pp. 280–290. [Online]. Available: http://doi.acm.org/10.1145/2807442.2807446

[50] R. Rädle, M. Nouwens, K. Antonsen, J. R. Eagan, and C. N. Klokmose, "Codestrates: Literate computing with webstrates," in *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '17. New York, NY, USA: ACM, 2017, pp. 715–725. [Online]. Available: http://doi.acm.org/10.1145/3126594.3126642

[51] M. Borowski, R. Rädle, and C. N. Klokmose, "Codestrate packages: An alternative to "one-size-fits-all" software," in *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems*, ser. CHI EA '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3170427.3188563

[52] S. K. Badam, A. Mathisen, R. Rädle, C. N. Klokmose, and N. Elmqvist, "Vistrates: A component model for ubiquitous analytics," *IEEE Trans. Vis. Comput. Graph.*, vol. 25, no. 1, pp. 586–596, 2019. [Online]. Available: https://doi.org/10.1109/TVCG.2018.2865144

[53] M. Nouwens, M. Borowski, B. Fog, and C. N. Klokmose, "Between scripts and applications: Computational media for the frontier of nanoscience," in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, ser. CHI '20. New York, NY, USA: ACM, 2020.

[54] F. Niephaus, E. Krebs, C. Flach, J. Lincke, and R. Hirschfeld, "PolyJuS: A Squeak/Smalltalk-Based Polyglot Notebook System for the GraalVM," in *Proceedings of the Conference Companion of the 3rd International Conference on Art, Science, and Engineering of Programming*, ser. Programming '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3328433.3328434

[55] A. Head, J. Jiang, J. Smith, M. A. Hearst, and B. Hartmann, "Composing flexibly-organized step-by-step tutorials from linked source code, snippets, and outputs," in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, ser. CHI '20. New York, NY, USA: ACM, 2020.

[56] M. Conlen and J. Heer, "Idyll: A markup language for authoring and publishing interactive articles on the web," in *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 977–989. [Online]. Available: https://doi.org/10.1145/3242587.3242600

[57] X. Zhang and P. J. Guo, "Ds.js: Turn any webpage into an example-centric live programming environment for learning data science," in *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '17. New York, NY, USA: ACM, 2017, pp. 691–702. [Online]. Available: http://doi.acm.org/10.1145/3126594.3126663

[58] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, and C. Willing, "Jupyter notebooks – a publishing format for reproducible computational workflows," in *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, F. Loizides and B. Schmidt, Eds. IOS Press, 2016, pp. 87 – 90.

[59] "The Binder Project," https://mybinder.org/, accessed: 2020-02-01.

[60] "Databricks Collaborative Notebooks," https://databricks.com/product/collaborative-notebooks, accessed: 2020-02-01.

[61] "Gigantum - Build it. Move it. Share it." https://gigantum.com/, accessed: 2020-02-01.

[62] "IBM Watson Studio," https://www.ibm.com/uk-en/cloud/watson-studio, accessed: 2020-02-01.

[63] "Kaggle Kernels," https://www.kaggle.com/kernels, accessed: 2020-02-01.

[64] "Microsoft Azure Notebooks," https://notebooks.azure.com/, accessed: 2020-02-01.

[65] "Mode Notebooks," https://mode.com/notebooks/, accessed: 2020-02-01.

[66] "Google Colaboratory," https://colab.research.google.com, accessed: 2020-02-01.

[67] "Datalore," https://datalore.io/, accessed: 2020-02-01.

[68] "Deepnote - Data science notebook for teams," https://www.deepnote.com/, accessed: 2020-02-01.

[69] "Kyso — Data Analytics Knowledge Hub," https://kyso.io/, accessed: 2020-02-01.

[70] "CoCalc - Collaborative Calculation and Data Science," https://cocalc.com/, accessed: 2020-02-01.

[71] "Kogence," https://kogence.com/app/docs/JupyterNotebook, accessed: 2020-02-01.

[72] "Quantopian: The Place For Learning Quant Finance," https://www.quantopian.com/, accessed: 2020-02-01.

[73] "Code Ocean — Professional tools for researchers." https://codeocean.com/, accessed: 2020-02-01.

[74] "The notebook for reproducible research — Nextjournal," https://nextjournal.com/, accessed: 2020-02-01.

[75] "RStudio is an integrated development environment (IDE) for R," https://rstudio.com/products/rstudio/, accessed: 2020-02-01.

[76] "RMarkdown," https://rmarkdown.rstudio.com/, accessed: 2020-02-01.

[77] "Shiny is an R package that makes it easy to build interactive web apps straight from R," https://shiny.rstudio.com/, accessed: 2020-02-01.

[78] "Wolfram Notebooks: Environment for Technical Workflows," http://www.wolfram.com/notebooks/, accessed: 2020-02-01.

[79] "Maple: The essential tool for mathematics," https://www.maplesoft.com/products/maple/, accessed: 2020-02-01.

[80] "MATLAB Live Editor: Create scripts that combine code, output, and formatted text in an executable notebook," https://www.mathworks.com/products/matlab/live-editor.html, accessed: 2020-02-01.

[81] "Spark Notebook," https://github.com/spark-notebook/spark-notebook, accessed: 2020-02-01.

[82] "Zeppelin," https://zeppelin.apache.org/, accessed: 2020-02-01.
[83] "Spyder Website," https://www.spyder-ide.org/, accessed: 2020-02-01.
[84] "Observable," https://observablehq.com/, accessed: 2020-02-01.
[85] "RunKit is a Node playground in your browser," https://runkit.com/, accessed: 2020-05-01.
[86] "Streamlit — The fastest way to build custom ML tools," https://www.streamlit.io/, accessed: 2020-02-01.
[87] "Iodide," https://alpha.iodide.io/, accessed: 2020-02-01.
[88] "Nbdev," http://nbdev.fast.ai/, accessed: 2020-02-01.
[89] "Livebook," https://github.com/inkandswitch/livebook, accessed: 2020-02-01.
[90] "nteract: building the future of interactive computing," https://nteract.io/, accessed: 2020-02-01.
[91] M. Seal, K. Kelley, and M. Ufford, "Scheduling Notebooks at Netflix," accessed: 2020-02-01.
[92] "Carbide Alpha — Buggy But Live!" https://alpha.trycarbide.com/, accessed: 2020-02-01.
[93] "Leisure," https://github.com/zot/Leisure, accessed: 2020-02-01.
[94] "Eve," http://witheve.com/, accessed: 2020-02-01.
[95] D. Frank, "Reproducible research: Stripe's approach to data science," https://stripe.com/blog/reproducible-research, accessed: 2020-02-01.
[96] "BeakerX," http://beakerx.com, accessed: 2020-02-01.
[97] "Polynote — The polyglot Scala notebook," https://polynote.org/, accessed: 2020-02-01.
[98] "jupytext: Jupyter Notebooks as Markdown Documents, Julia, Python or R scripts," https://github.com/mwouts/jupytext, accessed: 2020-02-01.
[99] "Hydrogen: run code interactively, inspect data, and plot. all the power of jupyter kernels, inside your favorite text editor," https://atom.io/packages/hydrogen, accessed: 2020-02-01.
[100] "Working with Jupyter Notebooks in Visual Studio Code," https://code.visualstudio.com/docs/python/jupyter-support, accessed: 2020-02-01.
[101] "Google Realtime API Deprecation," https://developers.google.com/realtime/deprecation, accessed: 2020-02-01.
[102] M. Bostock, V. Ogievetsky, and J. Heer, "D3 data-driven documents," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 12, pp. 2301–2309, Dec. 2011. [Online]. Available: http://dx.doi.org/10.1109/TVCG.2011.185
[103] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer, "Vega-Lite: A grammar of interactive graphics," *IEEE Transactions on Visualization and Computer Graphics*, vol. 23, no. 1, pp. 341–350, Jan. 2017. [Online]. Available: https://doi.org/10.1109/TVCG.2016.2599030
[104] T. R. G. Green, "Cognitive dimensions of notations," in *Proceedings of the Fifth Conference of the British Computer Society, Human-Computer Interaction Specialist Group on People and Computers V*. USA: Cambridge University Press, 1990, p. 443–460.
[105] M. Weiser, "The computer for the 21st century," *Scientific American*, vol. 265, no. 3, pp. 66–75, January 1991. [Online]. Available: http://www.ubiq.com/hypertext/weiser/SciAmDraft3.html
[106] E. A. Bier, M. C. Stone, K. Pier, W. Buxton, and T. D. DeRose, "Toolglass and magic lenses: The see-through interface," in *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '93. New York, NY, USA: Association for Computing Machinery, 1993, p. 73–80. [Online]. Available: https://doi.org/10.1145/166117.166126
[107] "Dynamicland: Our mission is to incubate a humane dynamic medium whose full power is accessible to all people." https://dynamicland.org/, accessed: 2020-02-01.
[108] A. J. Perlis, "Special feature: Epigrams on programming," *SIGPLAN Not.*, vol. 17, no. 9, p. 7–13, Sep. 1982. [Online]. Available: https://doi.org/10.1145/947955.1083808