Explainability in JupyterLab and Beyond: Interactive XAI Systems for Integrated and Collaborative Workflows

GRACE GUO, Georgia Institute of Technology, USA
DUSTIN ARENDT, Pacific Northwest National Laboratory, USA
ALEX ENDERT, Georgia Institute of Technology, USA

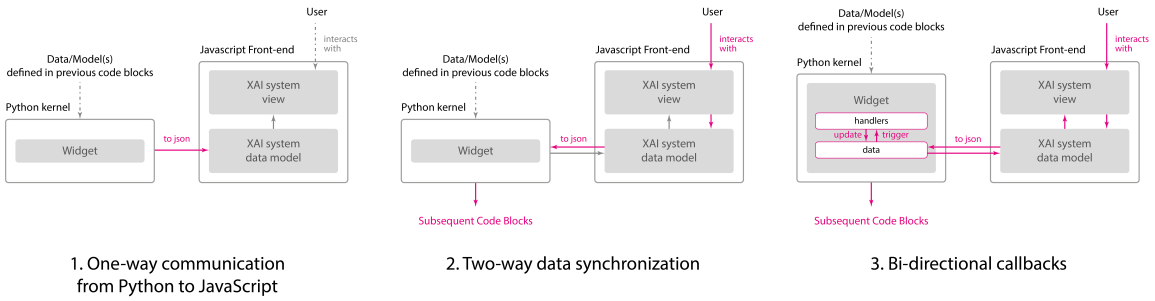

Fig. 1. Three design patterns for embedding XAI systems into JupyterLab notebooks.

Explainable AI (XAI) tools represent a turn to more human-centered and human-in-the-loop AI approaches that emphasize user needs and perspectives in machine learning model development workflows. However, while the majority of ML resources available today are developed for Python computational environments such as JupyterLab and Jupyter Notebook, the same has not been true of interactive XAI systems, which are often still implemented as standalone interfaces. In this paper, we address this mismatch by identifying three design patterns for embedding front-end XAI interfaces into Jupyter, namely: 1) One-way communication from Python to JavaScript, 2) Two-way data synchronization, and 3) Bi-directional callbacks. We also provide an open-source toolkit, bonXAI, that demonstrates how each design pattern might be used to build interactive XAI tools for a Pytorch text classification workflow. Finally, we conclude with a discussion of best practices and open questions. Our aims for this paper are to discuss how interactive XAI tools might be developed for computational notebooks, and how they can better integrate into existing model development workflows to support more collaborative, human-centered AI.

CCS Concepts: • **Human-centered computing** → **Visualization systems and tools**; **Visual analytics**.

Additional Key Words and Phrases: Jupyter, Computational Notebooks, Literate Programming, Explainable AI, Human-centered AI

ACM Reference Format:
Grace Guo, Dustin Arendt, and Alex Endert. 2024. Explainability in JupyterLab and Beyond: Interactive XAI Systems for Integrated and Collaborative Workflows. In *Proceedings of CHI Workshop on Human-Notebook Interactions (CHI '24)*. ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The recent advancements in machine learning (ML) and artificial intelligence (AI) have led to applications of AI models in diverse contexts ranging from medicine and healthcare [73, 76, 82] to education and fraud detection [1, 17, 53]. But while these AI models can mimic or exceed human proficiencies, they often lack transparency and can be inscrutable to

Manuscript submitted to ACM

ML developers and end-users. This lack of transparency may lead to unintended harms that persist even when the models themselves are removed [3, 20, 23, 24, 33, 90]. To mitigate these consequences, various explainable AI (XAI) tools and systems have thus been developed that aim to open the "black box" of AI such that developers, end-users, and stakeholders can better inspect, interpret, and refine model performance [7, 21, 22, 25, 26]. By enhancing the interpretability and transparency of models, XAI represents a turn to more human-centered and human-in-the-loop AI approaches that better address user needs and perspectives.

Within XAI, interactivity has emerged as an important factor for effective explanations [8, 52, 64]. Interactivity helps support human-centered and human-in-the-loop AI approaches by facilitating conversation between user and model [4], and the use of such interactive XAI systems has been found to improve prediction outcomes by allowing end-users to refine models based on their domain expertise or contextual knowledge [11]. To date, these XAI tools are typically implemented as in-browser applications, largely due to the wide variety of JavaScript toolkits and libraries that support richly interactive browser-based interfaces and visualizations.

In laying out his vision for literate programming, Donald Knuth proposed that *"Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do"* [46]. This aligns closely with the goals of many human-centered AI and XAI approaches, yet there is a mismatch in how XAI tools and ML models are built and used. While JavaScript browser applications are common for XAI, the core of resources available for ML are developed for Python – from Pytorch [66] and Tensorflow [2] to platforms such as Hugging Face[1] that provide a central repository of ML resources and pipelines. The dominance of these tools means that Python programming environments, such as JupyterLab and Jupyter Notebook[2], have also become key components of ML workflows. However, this mismatch in usage environments between XAI tools and ML models means that model explanations often cannot be presented *in context* of how the model is built and used. Developers who wish to adopt XAI in their work typically need two distributed processes running on the same machine, and must individually manage the challenges of data communication and state synchronization between model and explanation.

Solutions to these challenges have been influenced by available tools and their ease of use. One typical approach combines a Python server back-end with a browser front-end written in JavaScript [5, 48, 78]. However, this solution does not support the integration of XAI tools into notebook environments. Other solutions – such as IPyWidgets [94], NOVA [87] and anywidget [59] – have been proposed that aim to embed XAI interfaces directly into notebook cells. However, some of these frameworks are limited to one-directional data communication. NOVA, for example, does not propagate user inputs back to the Python kernel, which creates challenges for incorporating user feedback into model development workflows.

In our prior work [31, 32], we have addressed these challenges by adapting the existing IPyWidgets framework to build interactive XAI systems with two-way communication such that data/states are synchronized between the front-end interface and back-end Python kernel. This implementation meant that user inputs could be accessed in subsequent notebook cells and used to trigger callback functions to run data analytics and model pipelines. For data analysts and model developers, these embedded XAI systems facilitated better communication of the model development process by presenting explanations in context of the code. The explanations also helped them engage end-users in tasks of data selection, outcome evaluation, and iterative model refinement that would otherwise require programming expertise.

[1]https://huggingface.co/
[2]https://jupyter.org/

In this paper, we reflect on our prior work to identify three design patterns for how front-end XAI interfaces can be embedded into Jupyter, namely: 1) One-way communication from Python to JavaScript, 2) Two-way data synchronization, and 3) Bi-directional callbacks (Fig. 1). We also provide an open-source toolkit, bonXAI, that provides examples of how each design pattern might be used to build interactive XAI systems for a Pytorch text classification workflow. Finally, we outline some best practices, design guidelines, and open questions for future research. In sum, our aims for this paper are to discuss how interactive systems can be developed for computational notebooks, and how XAI tools can better integrate into existing model development workflows to support collaborative, human-centered AI.

2 RELATED WORK

In this section, we provide an overview of interactive XAI approaches and systems developed in prior work. This overview highlights how interactivity is both crucial to the effectiveness of AI explanations, while, at the same time, many explanations are still presented independently of the model being explained. Due to the expansive landscape of current research, we cannot enumerate all XAI systems here. For recent surveys refer to [4] and [11].

This section will also provide a summary of libraries for building interactive widgets in computational notebooks, particularly for the Jupyter notebook environments. To date, there are two notebook versions offered by Project Jupyter: the more recent JupyterLab interface, and the classic Jupyter Notebook[3]. In the rest of this paper and unless otherwise stated, we use Jupyter to refer to both environments. As our own prior works [31, 32] have primarily used Python and Jupyter, they are also the focus of this overview. However, other programming languages and computational environments exist, such as Google Colab[4] (an in-browser version of Jupyter), that have similarly seen widespread adoption by developers. We discuss them in relation to embedded XAI systems in Section 5.3.

2.1 Interactive XAI Systems

AI models have often been described as "black boxes" due to their lack of transparency and interpretability. Explainability tools, also referred to as XAI systems, have thus been developed to help open these "black boxes" and explain how AI models work. In recent years, interactivity has emerged as one crucial factor influencing the effectiveness of these explanations. In a notable 2018 survey, Adadi and Berrada propose that "explainability can only happen [through] interaction between human and machine" [4]. This has since been supported by work from other researchers arguing from psychological, philosophical and social perspectives that interactivity is an important factor for good explanations [8, 52, 64]. And in a 2023 paper looking at 48 evaluations of interactive XAI systems, Bertrand et al. found empirical evidence that "interactive explanations improve perceived usefulness and performance of the human+AI team" [11].

Beyond improving outcomes, prior studies have also found that the use of XAI systems can better support human-centered AI approaches. For ML model developers, a wide range of tools exist to support tasks from data pre-processing to model evaluation and refinement [7, 8, 79]. In contrast, far fewer XAI systems have been developed for or evaluated with end-users who are affected by the models [12, 35, 55, 79]. For end-users, explanations tend to focus on providing insight into model outcomes in specific usage domains, such as healthcare [13, 14] and education [29, 42]. However, in a recent study of end-user needs and expectations by Kim et al., the authors found that end-users, like model developers, wanted XAI that do more than explain model outcomes. Instead, participants in the study expressed a desire for tools that engage them in model development, helping them better calibrate trust, improve task skills, supply better inputs,

[3]Project Jupyter recommends using JupyterLab since the classic Notebook is being deprecated. However, both environments remain in use, and our work supports both.
[4]https://colab.research.google.com/

and give constructive feedback to developers [43]. The authors went on to propose an set of recommendations to guide the implementation of interactive XAI tools for end-users to actively participate in model development workflows, but how these recommendations should be realized remains open to study.

Taken together, these prior works lay out a compelling case for the effectiveness of interactive XAI tools, as well as their potential as a means to engage end-users without ML expertise in the development and application of ML models. In this paper, we explore how interactive XAI systems can be *embedded into computational environments* as a means for AI model development workflows to support collaboration between users of different backgrounds [21, 43, 56], and seamlessly integrate user feedback and judgements into model development processes.

2.2 Computational Notebooks and the IPyWidgets Framework

Donald Knuth first laid out the vision of "literate programming" in his 1984 paper, which posits that programs are works of literature for *humans* rather than machines and should be documented as such [46]. This vision has since been developed and extended into a range of computational notebook environments that allow developers to display executable code with code outputs, multimedia text, images, video, audio, and interactive interfaces in a single document. These notebook environments help communicate the work of a program for human audiences, supporting collaboration, reproducibility, sharing, and communication, particularly in data science workflows [44, 74, 77]. The benefits of working in notebook environments has since led to widespread adoption [38], and has motivated research into how they might be extended into contexts of non-linear analysis [86, 91, 96], workflow recommendation [69], automatic narrative generation [39, 40, 54, 85, 98], and fluid transitions between code and graphical interfaces [41].

In line with many of these prior studies, our work looks at the Jupyter computational environments, and specifically the IPyWidgets framework [94]. The framework was designed to help embed interactive front-end interfaces into notebook cells by providing a communications layer that sends data in JSON format between the front-end interface and the back-end Python kernel. This communications layer ensures that the data (or states) are synchronized within a Jupyter *widget*. On widget instantiation, the front-end and the back-end store copies of the same data/state. When users interact with the interface, any changes to the data can be automatically propagated back to the kernel using *traitlets*, allowing the new data to be accessible in the notebook. Handlers can also be registered to these data attributes such that any change triggers a callback function that can be used to automatically run some data analysis or, in the ML context, a pre-specified model. In this paper, we leverage this IPyWidgets framework to define three design patterns for how interactive XAI systems can be embedded into Jupyter workflows.

2.3 Interactive XAI Widgets for Jupyter

Python libraries and computational environments make up the vast majority of current AI and ML tools [38]. From libraries such as scikit-learn [67], Pytorch [66], Keras [15] and Tensorflow [2], to platforms such as Hugging Face, these resources have built up a robust Python ecosystem for model development and use. Consequently, Python computational environments, such as Jupyter and Google Colab, have also become some of the most common environments for machine learning. All libraries listed above, for example, include some mention of Jupyter, Colab, or "notebooks" in their examples and tutorials.

However, while Python and Juptyer have become the dominant tools for machine learning development, this has not been the case for XAI systems, particularly interactive XAI systems. Consider the tools surveyed by Bertrand et al. [11]. One typical architecture implements the system interface using a JavaScript front-end while a Python server back-end provides the ML model being explained [5, 48, 78]. This approach is useful, but requires additional work to set

up and connect the server and the interface. Other implementations have sought to run the XAI system interface and the ML models from a single environment. For example, Ross et al. were able to implement models directly in-browser by first converting them to Tensorflow.js [2, 72], while Spinner et al. build their XAI visualizations as Tensorboard[5] plugins that can be used by any developer designing and training a TensorFlow model [80]. In the latter example, the adoption of Tensorboard helps provide a more seamless integration of XAI into existing model development processes for users who are already working with the Tensorflow library. However, since Tensorboard visualizations are hosted in a separate browser window, this set-up still requires frequent context switching between the coding environment and the in-browser explanations. This can also make it challenging to obtain an end-to-end overview of model development with the XAI systems inserted in an ordered, linear narrative. More critically, when we look at the 48 papers included in Bertrand et al.'s survey [11], most do not provide any detail about how the interactive XAI systems are implemented and used. System screenshots also typically exclude details about computational environments or the expected usage context of the tool, which can result in barriers to adoption when ML developers are not shown how they can quickly incorporate these XAI systems into their work.

More commonly, toolkits built for Jupyter can be used to explain ML models. These include general purpose visualization libraries – such as matplotlib [36], seaborn [89], and Altair [83] – and specialized XAI tools – such as Captum [47], Class Activation Mapping methods [30], ELI5 [27], InterpretML [65], LIME [70] and SHAP [57]. Of these, the majority are static visualizations. While some libraries include interactivity (e.g. Altair), they tend to be visualization-centric and are not designed to support specific XAI tasks. To address these limitations, frameworks, such as NOVA [87], have been built to help embed interactive JavaScript interfaces into notebooks. These frameworks simplify the process of wrapping complex XAI systems into notebook widgets, allowing ML developers to rapidly add explanations to their existing workflows and obtain insights about the data or model. However, these widgets typically only support the one-way communication from the notebook to the JavaScript interface, and changes to the data/state caused by user interactions with the interface can neither be propagated back to the Python kernel nor accessed in subsequent notebook cells. This is a limitation for interactive XAI systems (such as the ones discussed in Section 2.1) where user feedback may be necessary to evaluate, refine or update a model. In particular, two-way data communication and state synchronization can be essential in collaborative situations where end-users without programming expertise may benefit from inspecting models and providing input through an interactive front-end interface.

In this paper, we build on existing widget frameworks [59, 87, 94] and our own prior work [31, 32] to define three design patterns of how interactive XAI systems can be embedded into ML development workflows in Jupyter. These design patterns provide a basic template for displaying the interfaces, synchronizing data, automatically propagating user input back to the kernel, and triggering callback functions. Taken together, this paper aims to provide an exploration of how interactive XAI systems can be more tightly integrated into model development environments to better support human-centered and human-in-the-loop AI approaches.

3 THREE DESIGN PATTERNS FOR XAI IN JUPYTER

We use the IPyWidgets framework (see Section 2.2) to embed interactive XAI systems into Jupyter. A widget can be instantiated in a notebook code block with relevant arguments (parameters) in the manner of `widget(*args)`. On instantiation, the JavaScript front-end of the widget will be rendered in an output notebook cell. From existing XAI

[5] https://www.tensorflow.org/tensorboard

systems and our own prior work, we identified three design patterns for the IPyWidgets framework: 1) One-way communication from Python to JavaScript, 2) Two-way data synchronization, and 3) Bi-directional callbacks (Fig. 1).

To demonstrate how each design pattern might be implemented, we also provide a gallery of example XAI widgets in our supplemental toolkit, bonXAI[6]. This toolkit includes three widgets: 1) DataExplorer, 2) DataSelector, and 3) InferenceExplorer that correspond to the three design patterns. They are developed for the Hugging Face PyTorch text classification workflow [28, 66], but can be generalized to other textual data exploration, selection and inference tasks. BonXAI uses React[7] and the IPyWidgets framework, but readers may consider using other implementations when building XAI tools for Jupyter.

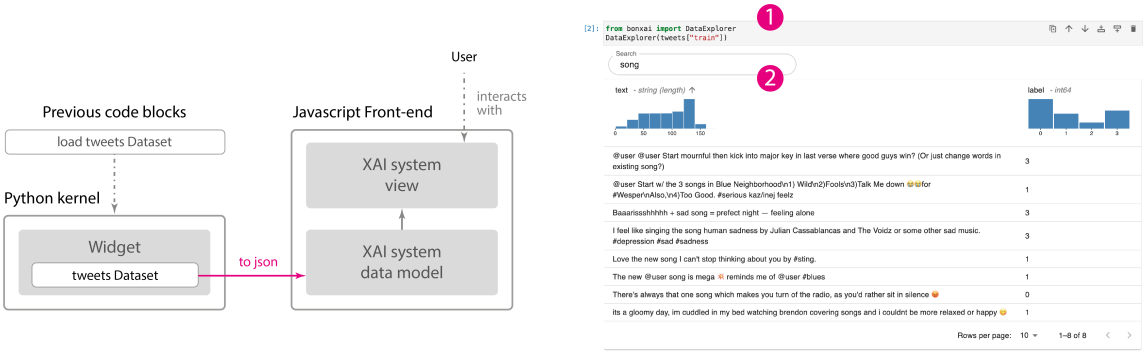

Fig. 2. The DataExplorer widget is an example of design pattern 1. This type of XAI system provides a display based on a given a model and/or dataset as input. ① The DataExplorer widget is instantiated with some data that is converted to JSON and sent to the front-end. ② While users can interact with the display, their interactions do not change data attributes on the back-end Python kernel.

3.1 Design Pattern 1: One-way communication from Python to JavaScript

The first design pattern for embedded XAI systems involves basic JavaScript displays in notebook cells. Such systems are typically instantiated by passing in some data that is then displayed in the front-end view of the widget. This design pattern is widely used in many XAI toolkits for Jupyter – such as Captum [47], Class Activation Mapping methods [30], ELI5 [27], InterpretML [65], LIME [70] and SHAP [57]. However, the majority of these explanations are static displays. Even in cases where the user may be able to interact and explore the data/model further, the results of their interactions are not propagated back to the Python kernel [6, 31]. A key characteristic of this design pattern is this one-way communication of information from notebook to XAI system. User interactions (if any) are contained in the front-end interface, and cannot be accessed in subsequent code blocks.

In our demonstration library, we provide an example of this design pattern with the DataExplorer widget (Fig. 2). We can pass a Hugging Face text Dataset to this widget, which is displayed in an interactive table for exploration. Users can search for substrings, or filter by string length and other variables. However, these interactions do not change any attributes on the Python back-end, and the filtered subset is not accessible in subsequent notebook code blocks.

To date, this design pattern is, by and large, the most common approach for embedding XAI systems into Jupyter. A variety of frameworks, such as NOVA [87], have also been developed to simplify implementation by automating the

[6]https://github.com/gracegsy/bonXAI
[7]https://reactjs.org/

process of wrapping front-end JavaScript into widgets. The resulting toolkits are well suited for post-hoc exploring, understanding, and communicating details about a dataset, model, or training process. However, since interactions are not propagated back to the kernel, these XAI systems may be insufficient for human-in-the-loop tasks that rely on user input.

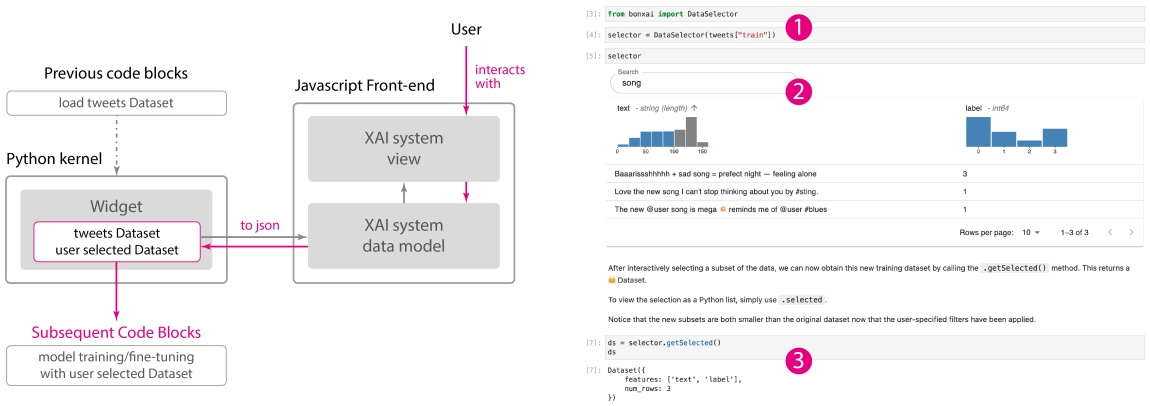

Fig. 3. The DataSelector widget is an example of design pattern 2, which synchronizes the data/states between the front-end and back-end, allowing user inputs to be propagated back to the notebook. ① As before, the DataSelector widget is instantiated with some data that is converted to JSON and sent to the front-end. ② Users can interactively explore the data set by searching or applying filters. ③ The filtered data set from user interactions can be accessed in subsequent notebook cells/code blocks.

3.2 Design Pattern 2: Two-way data synchronization

The second design pattern supports the two-way communication of data. In such XAI systems, when users interact with front-end input components, the results of their interactions are propagated back to the kernel to update the relevant data structures and/or variables. In this way, data/states are synchronized between the interface and the kernel. User input can thus be accessed in subsequent notebook code blocks and applied back to the data analysis process. To the best of our knowledge, this two-way communication must be manually implemented using a framework, such as IPyWidgets or anywidget, and no libraries or tools have been developed to automate the process as yet.

Increasingly, researchers have called for the engagement of domain experts and end-users in order to develop machine learning methods that better incorporate domain knowledge and user judgements [19]. However, extracting this knowledge can be challenging since domain experts and end-users may not be familiar enough with ML or programming to input their feedback directly in code. In such situations, a front-end system embedded into Jupyter can potentially better facilitate collaborative efforts between developers and experts. The front-end interface is an intuitive way for domain experts and end-users to interactively provide their domain knowledge, while the propagation of this input back to the notebook means that developers can quickly incorporate updates into subsequent code blocks and ML pipelines.

In one of our prior studies [32], we applied this design pattern in three embedded interactive visualizations for causal inference analysis. Using these visualizations, causal inference analysts and domain experts can collaboratively edit the outputs of causal discovery algorithms, discuss the appropriate variables to use in analysis, inspect the data instances to include or exclude, and explore subgroup differences in estimated treatment outcomes. We provide another

demonstration of this design pattern with the DataSelector module in our example library (Fig. 3). This module is similar to the DataExplorer, however, in addition to allowing users to interactively explore their data, the filtered subset is also sent back to the Python kernel and can be accessed in subsequent notebook code blocks as a Hugging Face Dataset. Like any other Dataset provided by the Hugging Face Datasets library, this filtered subset can also be used for model training and fine-tuning.

More generally, we see potential for embedded XAI systems of this design pattern to be adapted for "data explanations" [8] and other data-centric tasks. These are tasks that require collaboration or expert judgements, such as data labeling and cleaning [49, 50, 71, 92, 93] or curating human feedback for model fine-tuning using RLHF [16, 45, 51, 58, 88, 95].

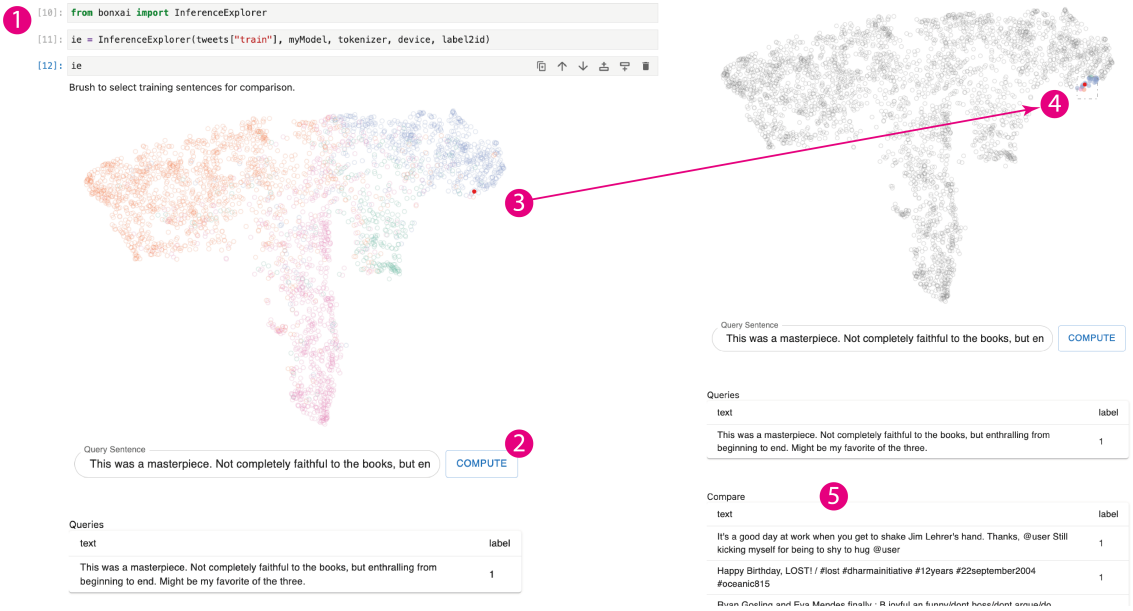

Fig. 4. The InferenceExplorer module demonstrates how Design Pattern 3 might be applied. ① Like in previous examples, data is converted to JSON and sent to the front-end. In this widget, we also provide a pre-trained model. Since a model cannot be converted to JSON format, the model is simply registered as a widget attribute in the Python back-end. The widget computes and visualizes the UMAP embedding of all text in the data set. ② Users can interactively input some new text for inference. Its label, predicted by the pre-trained model on the Python back-end, is displayed in the table below. ③ The position of this new text is also computed and added to the UMAP embedding visualization as a red dot. ④ To *explain* why their input was given a particular label, users can brush and select its nearest neighbors in the visualization. ⑤ These selected neighbors are listed in a separate table. We can see that tonally, they resemble the user input text, and have the same label.

3.3 Design Pattern 3: Bi-directional callbacks

The third design pattern of embedded XAI systems extends the two-way communication such that changes to the data/state also trigger callback functions in the kernel that execute additional data processing tasks. The results of these callback functions are passed back to the front-end interface and used to update the display, thus ensuring that any new data or user input is dynamically reflected in the XAI tool. This design pattern most closely realizes the goals of human-in-the-loop AI approaches by allowing end-users to engage in tasks that directly affect model development and use, such as prompt tuning, inference validation, model refinement and others.

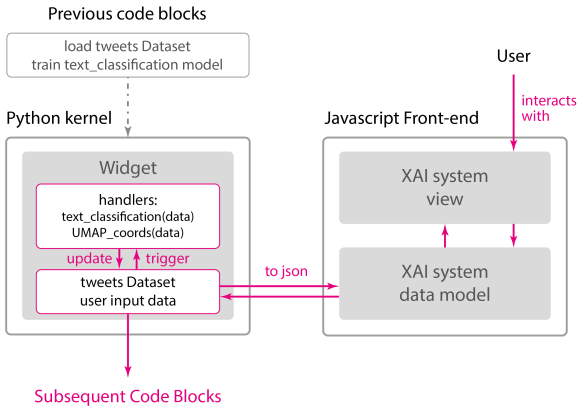

Fig. 5. The InferenceExplorer module applies the third design pattern to provide bi-directional callbacks. When users interact with input components, their interactions update the data/state on the Python kernel back-end. These changes automatically trigger callback functions for data processing, model inferencing, and updating the front-end display.

In our demonstration library, we have provided an example of how one such XAI system can be embedded into Jupyter (Fig. 4 and 5). Similar to the previous examples, the InferenceExplorer widget is instantiated with a training data set. For each line of text in the data set, the widget first gets its embedding, then performs a UMAP dimensionality reduction [62] to obtain 2D-coordinates for all instances. These coordinates are visualized to provide an overview of the data set, and can be used to determine how well the model distinguishes the different labels.

This InferenceExplorer widget also applies the third design pattern to support bi-directional callbacks. One of the parameters required to instantiate this widget is a pre-trained model. When users interact with the front-end interface, they can choose to input new sentences for inference. This input is propagated back to the widget, updating an internal data attribute. In addition to this two-way data communication and synchronization, a *handler* automatically triggers a callback function that uses the pre-trained model to get the predicted label for the new sentence input. This handler function also computes 2D coordinates for the sentence. Finally, the text, predicted label, and umap coordinates are sent back to the front-end to be added to the visualization as a red dot.

By adopting bi-directional callbacks in our InferenceExplorer widget example, the XAI system closes the loop between user input, model inference, and visual explanation. Users can now interactively explore how a model performs on new text samples, compare predicted labels to training data, as well as reason about *why* certain predictions were made by inspecting the nearest neighbors. This iterative back-and-forth allows for the tighter integration of user inputs into model development, usage, and explanation, which in turn supports a more human-centered approach to AI that better incorporates end-user feedback, tasks, and perspectives.

4 BEST PRACTICES AND DESIGN GUIDELINES

So far, we have described three design patterns for how XAI systems can be embedded into Jupyter. A summary of the design patterns is provided in Table 1. In this section, we reflect on our experiences developing such systems and the lessons learned. We distill these into a set of best practices and design guidelines that can be used to determine which

	data communication	callbacks
DP 1	one-way Python to JS	none
DP 2	two-way synchronized	none
DP 3	two-way synchronized	changes in data trigger callback handlers in Python kernel

Table 1. Summary of features in the three design patterns.

design pattern of XAI system to develop for different usage scenarios. We structure the following sections around four main considerations: *When* should we build and embed XAI systems into Jupyter? *Who* are the target users of these XAI systems? *What* should be taken into account? *How*, and with what methodologies, should these XAI systems be designed, implemented, and evaluated?

4.1 Support existing development environments

We know from existing studies into interaction design that minimizing interruptions and context switching can enhance focus, reduce cognitive load, and improve task performance [10, 60, 68]. As such, embedding XAI systems into notebooks should primarily be done when Jupyter is already the main computational environment of target users. This is particularly useful when existing workflows are highly iterative, since repeatedly switching to a standalone XAI system can be more disruptive due to frequent task interruptions. At the same time, the inverse is also true. While Python libraries and computational notebooks are the dominant site of ML development today, increasingly, other tools such as TensorFlow.js [2] and WebLLM[8] are being adopted that allow models to run directly in-browser. In this scenario, designing and embedding XAI systems into Jupyter may be less appropriate since it would require users to relocate their workflow to entirely new languages and environments, which might increase barriers to adoption instead.

4.2 Support existing workflows

Another factor to consider when building embedded XAI systems is the typical model development workflow and associated user tasks. Recent taxonomies have proposed that different tasks require different explanatory tools [8, 55], and understanding the typical or existing user workflow is crucial for determining the appropriate places where interactive XAI widgets are needed. For example, some widgets can help with data exploration and pre-processing, while other widgets are more appropriate for inspecting feature weights after training. In cases where multiple XAI systems are needed to support user input at different steps of the workflow, it may even be necessary for multiple widgets to be implemented in a single library to account for varied or iterative user feedback [32].

4.3 Select design patterns based on user tasks

We also know from prior studies that different audiences require different explanations throughout the AI life cycle [18]. As such, the types of interactive XAI tools that are needed, the users they support, and the associated explainability tasks are all key considerations for deciding which design pattern should be adopted when building explainability tools for Jupyter environments.

For example, if an XAI system is designed for ML developers who need to refine a model iteratively, it may be more effective to use design patterns 2 and 3 to integrate explanation and inspection into the process of model development.

[8] https://llm.mlc.ai/

Similarly, if a model requires in-depth or lengthy collaboration with domain-experts or end-users without ML/data science experience, design patterns 2 and 3 may also be well-suited to help them interactively specify their requirements and domain knowledge. In both of these cases, the automatic propagation of inputs to the Python widget means that any feedback can be immediately incorporated into the workflow, allowing developers and end-users to collaboratively discuss and iterate on ML models rapidly. On the other hand, if user tasks are oriented around simply communicating or understanding the data or model, then the self-contained XAI systems of design pattern 1 may be sufficient.

4.4 Design for relevant data types and libraries

Since embedded XAI systems are meant for Python/Jupyter environments, there is also a need to account for the data types and data-processing libraries that are typically available. In our prior work on causal inference [32], for example, data analysts we collaborated with wanted the XAI tools to accept directed acyclic graph (DAG) specifications in both `.json` and NetworkX[9]. This latter requirement arose because our collaborators wanted to use the CausalNex library [9], which outputs NetworkX graphs, in conjunction with the interactive visualizations we provided. Similarly, in our DataSelector example above (Section 3.2), since the widget is developed for Hugging Face models, the training data input and the selected output of the widget are formatted as Hugging Face Dataset classes. In both cases, while it is possible for users to manually convert data into the required data types, we can support more seamless integration of XAI systems by automatically processing the data to match available libraries and workflows.

4.5 Incorporate computational environment throughout XAI development

The prior sections have highlighted how, like all human-centered computing systems, interactive XAI should be implemented with a clear understanding of users and user tasks. In visualization research, many existing methodologies are well-suited for this purpose [61, 63, 75, 81]. More broadly, participatory design and participatory action research approaches [34, 84] that emphasize designing *with* users have also been proposed as effective methods for human-centered XAI [22].

At the same time, developing interactive XAI systems for Jupyter requires existing methodologies to be extended on account of the particularities of this computational environment. In our prior works, we have utilized an approach that makes Jupyter part of the design process from the very beginning. For instance, during initial prototyping, we included notebook cells as screenshots to help users visualize the intended usage context. This was effective for eliciting requests about the data types and file formats the embedded XAI system should support. Similarly, in the evaluation studies, we presented users with notebooks that demonstrate an end-to-end workflow with the XAI systems embedded. This gave users a richer understanding of what the integrated workflow looks like and how the XAI systems could be added to their existing projects. It also helped elicit feedback about how the XAI systems could facilitate collaborative discussions and analyses with other stakeholders.

5 OPEN QUESTIONS

While the examples in this paper and in our supplemental toolkit have demonstrated the possibilities of embedded XAI systems in Jupyter, as well as how they might realize integrated and collaborative ML workflows, there remain challenges to the development of these systems. In the following sections, we discuss some of these open questions and outline potential avenues for future research.

[9]https://networkx.org/

5.1 Data Size Limitations

Since all communication and synchronization between the front-end interface and back-end kernel must be done in JSON format, this places a limitation on the types of information that can be transferred. As such, while most data file types can be easily JSON serialized, it would not be possible to do the same for the models themselves. Furthermore, there also exists a limit on the amount of data that can be transferred this way. In our example toolkit, the XAI widgets are designed for a model fine-tuning workflow, which requires only relatively small amounts of text data. However, other usage scenarios may include greater amounts of data or larger file types (such as image and video files) that cannot be easily transferred and displayed simultaneously. In our prior work, we have addressed this by processing data on-demand, allowing users to interactively choose which subset of the data to serialize, communicate, and display. This ensures that the JSON created is of a manageable size, however, it also limits the ability of users to gain an overview of the entire data set at once. More work is thus needed to explore other solutions for the communication of large data sets, particularly in usage contexts that need also handle dynamic user interactions and interface updates.

5.2 Alternate Design Patterns

The design patterns included in Section 3 describe the current state of embedded XAI systems in Jupyter, both in terms of existing libraries and what we have been able to achieve in our own work. However, this is by no means an exhaustive list. There are other approaches to implementing how JavaScript front-end interfaces can be incorporated into Python workflows. For example, libraries such as OmniXAI [97], Plotly Dash [37] and Shapash[10] allow users to initialize an XAI system as a web app from their notebook. This web app is hosted in a new browser window, and include interactive visualizations that help users inspect properties of their model. While this requires users to switch between browser windows, their interactions with the front-end XAI system can still be used to communicate user-input data back to the notebook. Future work can look at how such implementations compare to the design patterns discussed in this paper, and how they support end-user engagement in ML workflows. More broadly, we also see potential for the synthesis of a more comprehensive taxonomy of design patterns for different types of XAI tools for computational notebooks, as well as the systematic exploration of the different usage scenarios and tasks that can be supported by each.

5.3 Beyond Jupyter

Finally, while this paper and related examples have all been developed for Jupyter, more work is needed before the design patterns discussed here can be applied to other computational environments. Of these, Google Colab (an in-browser version of JupyterLab) is widely used by ML developers [38] for quick sharing and reproduction of ML workflows. However, due to security reasons, Colab and other computational environments do not yet support the two-way communication and synchronization of data. This thus limits the applicability of design patterns 2 and 3, which are only usable in local versions of Jupyter for now.

6 CONCLUSION

XAI tools are effective methods of explaining and reasoning about how ML models are developed and used. However, unlike many other ML libraries and resources, most interactive XAI tools are not built for Python computational environments such as Jupyter. In this paper, we addressed this mismatch by proposing three design patterns for how front-end XAI interfaces can be embedded into notebooks, namely: 1) One-way communication from Python to

[10] https://shapash.readthedocs.io

JavaScript, 2) Two-way data synchronization, and 3) Bi-directional callbacks. We also provided an open-source toolkit, bonXAI, that provides examples of how each design pattern might be used to build interactive XAI systems that fit into a text classification model development workflow. Taken together, these examples demonstrate how the design patterns proposed can support the better communication of explanations of ML processes, as well as methods for seamlessly incorporating end-user inputs into model development workflows.

ACKNOWLEDGMENTS

The authors would like to thank Raj Shah for improving the InferenceExplorer embedding space. This work is supported in part by NSF IIS-1750474 and the IBM PhD Fellowship.

REFERENCES

[1] Roger A. Leite, Theresia Gschwandtner, Silvia Miksch, Erich Gstrein, and Johannes Kuntner. 2020. Neva: Visual analytics to identify fraudulent networks. In *Computer Graphics Forum*, Vol. 39. Wiley Online Library, 344–359.

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.

[3] Mark S Ackerman. 2000. The intellectual challenge of CSCW: the gap between social requirements and technical feasibility. *Human–Computer Interaction* 15, 2-3 (2000), 179–203.

[4] Amina Adadi and Mohammed Berrada. 2018. Peeking inside the black-box: a survey on explainable artificial intelligence (XAI). *IEEE access* 6 (2018), 52138–52160.

[5] Yongsu Ahn, Muheng Yan, Yu-Ru Lin, Wen-Ting Chung, and Rebecca Hwa. 2022. Tribe or not? Critical inspection of group differences using TribalGram. *ACM Transactions on Interactive Intelligent Systems (TiiS)* 12, 1 (2022), 1–34.

[6] J Alammar. 2021. Ecco: An Open Source Library for the Explainability of Transformer Language Models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, 249–257.

[7] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-López, Daniel Molina, Richard Benjamins, et al. 2020. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information fusion* 58 (2020), 82–115.

[8] Vijay Arya, Rachel KE Bellamy, Pin-Yu Chen, Amit Dhurandhar, Michael Hind, Samuel C Hoffman, Stephanie Houde, Q Vera Liao, Ronny Luss, Aleksandra Mojsilović, et al. 2019. One explanation does not fit all: A toolkit and taxonomy of ai explainability techniques. *arXiv preprint arXiv:1909.03012* (2019).

[9] Paul Beaumont, Ben Horsburgh, Philip Pilgerstorfer, Angel Droth, Richard Oentaryo, Steven Ler, Hiep Nguyen, Gabriel Azevedo Ferreira, Zain Patel, and Wesley Leong. 2021. *CausalNex*. https://github.com/quantumblacklabs/causalnex

[10] Benjamin B Bederson. 2004. Interfaces for staying in the flow. *Ubiquity* 5, 27 (2004), 1.

[11] Astrid Bertrand, Tiphaine Viard, Rafik Belloum, James R Eagan, and Winston Maxwell. 2023. On Selective, Mutable and Dialogic XAI: a Review of What Users Say about Different Types of Interactive Explanations. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–21.

[12] Umang Bhatt, Alice Xiang, Shubham Sharma, Adrian Weller, Ankur Taly, Yunhan Jia, Joydeep Ghosh, Ruchir Puri, José MF Moura, and Peter Eckersley. 2020. Explainable machine learning in deployment. In *Proceedings of the 2020 conference on fairness, accountability, and transparency*. 648–657.

[13] Carrie J Cai, Emily Reif, Narayan Hegde, Jason Hipp, Been Kim, Daniel Smilkov, Martin Wattenberg, Fernanda Viegas, Greg S Corrado, Martin C Stumpe, et al. 2019. Human-centered tools for coping with imperfect algorithms during medical decision-making. In *Proceedings of the 2019 chi conference on human factors in computing systems*. 1–14.

[14] Carrie J Cai, Samantha Winter, David Steiner, Lauren Wilcox, and Michael Terry. 2019. " Hello AI": uncovering the onboarding needs of medical practitioners for human-AI collaborative decision-making. *Proceedings of the ACM on Human-computer Interaction* 3, CSCW (2019), 1–24.

[15] François Chollet et al. 2015. Keras. https://keras.io.

[16] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. Deep reinforcement learning from human preferences. *Advances in neural information processing systems* 30 (2017).

[17] Albert T Corbett, Kenneth R Koedinger, and John R Anderson. 1997. Intelligent tutoring systems. In *Handbook of human-computer interaction*. Elsevier, 849–874.

[18] Shipi Dhanorkar, Christine T Wolf, Kun Qian, Anbang Xu, Lucian Popa, and Yunyao Li. 2021. Who needs to know what, when?: Broadening the Explainable AI (XAI) Design Space by Looking at Explanations Across the AI Lifecycle. In *Designing Interactive Systems Conference 2021*. 1591–1602.

[19] Finale Doshi-Velez and Been Kim. 2017. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608* (2017).

[20] Upol Ehsan, Q Vera Liao, Michael Muller, Mark O Riedl, and Justin D Weisz. 2021. Expanding explainability: Towards social transparency in ai systems. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–19.

[21] Upol Ehsan, Samir Passi, Q Vera Liao, Larry Chan, I Lee, Michael Muller, Mark O Riedl, et al. 2021. The who in explainable ai: How ai background shapes perceptions of ai explanations. *arXiv preprint arXiv:2107.13509* (2021).

[22] Upol Ehsan and Mark O Riedl. 2020. Human-centered explainable ai: Towards a reflective sociotechnical approach. In *HCI International 2020-Late Breaking Papers: Multimodality and Intelligence: 22nd HCI International Conference, HCII 2020, Copenhagen, Denmark, July 19–24, 2020, Proceedings 22*. Springer, 449–466.

[23] Upol Ehsan, Koustuv Saha, Munmun De Choudhury, and Mark O Riedl. 2023. Charting the Sociotechnical Gap in Explainable AI: A Framework to Address the Gap in XAI. *Proceedings of the ACM on Human-Computer Interaction* 7, CSCW1 (2023), 1–32.

[24] Upol Ehsan, Ranjit Singh, Jacob Metcalf, and Mark Riedl. 2022. The algorithmic imprint. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*. 1305–1317.

[25] Upol Ehsan, Philipp Wintersberger, Q Vera Liao, Martina Mara, Marc Streit, Sandra Wachter, Andreas Riener, and Mark O Riedl. 2021. Operationalizing human-centered perspectives in explainable AI. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–6.

[26] Upol Ehsan, Philipp Wintersberger, Q Vera Liao, Elizabeth Anne Watkins, Carina Manger, Hal Daumé III, Andreas Riener, and Mark O Riedl. 2022. Human-Centered Explainable AI (HCXAI): beyond opening the black-box of AI. In *CHI conference on human factors in computing systems extended abstracts*. 1–7.

[27] eli5 community. 2021. Welcome to ELI5's documentation! https://eli5.readthedocs.io/en/latest/ Accessed Feb 20, 2024.

[28] Hugging Face. [n. d.]. Text classification. https://huggingface.co/docs/transformers/en/tasks/sequence_classification Accessed Feb 20, 2024.

[29] Krzysztof Fiok, Farzad V Farahani, Waldemar Karwowski, and Tareq Ahram. 2022. Explainable artificial intelligence for education and training. *The Journal of Defense Modeling and Simulation* 19, 2 (2022), 133–144.

[30] Jacob Gildenblat and contributors. 2021. PyTorch library for CAM methods. https://github.com/jacobgil/pytorch-grad-cam.

[31] Grace Guo, Maria Glenski, ZhuanYi Shaw, Emily Saldanha, Alex Endert, Svitlana Volkova, and Dustin Arendt. 2021. Vaine: Visualization and ai for natural experiments. In *2021 IEEE Visualization Conference (VIS)*. IEEE, 21–25.

[32] Grace Guo, Ehud Karavani, Alex Endert, and Bum Chul Kwon. 2023. Causalvis: Visualizations for Causal Inference. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–20.

[33] Alexa Hagerty and Igor Rubinov. 2019. Global AI ethics: a review of the social impacts and ethical implications of artificial intelligence. *arXiv preprint arXiv:1907.07892* (2019).

[34] Gillian R Hayes. 2011. The relationship of action research to human-computer interaction. *ACM Transactions on Computer-Human Interaction (TOCHI)* 18, 3 (2011), 1–20.

[35] Sungsoo Ray Hong, Jessica Hullman, and Enrico Bertini. 2020. Human factors in model interpretability: Industry practices, challenges, and needs. *Proceedings of the ACM on Human-Computer Interaction* 4, CSCW1 (2020), 1–26.

[36] J. D. Hunter. 2007. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering* 9, 3 (2007), 90–95. https://doi.org/10.1109/MCSE.2007.55

[37] Plotly Technologies Inc. 2015. *Collaborative data science*. Montreal, QC. https://plot.ly

[38] Kaggle. 2022. State of Data Science and Machine Learning 2022. https://www.kaggle.com/kaggle-survey-2022 Accessed Feb 20, 2024.

[39] DaYe Kang, Tony Ho, Nicolai Marquardt, Bilge Mutlu, and Andrea Bianchi. 2021. Toonnote: Improving communication in computational notebooks using interactive data comics. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–14.

[40] 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.

[41] Mary Beth Kery, Donghao Ren, Fred Hohman, Dominik Moritz, Kanit Wongsuphasawat, and Kayur Patel. 2020. mage: Fluid moves between code and graphical work in computational notebooks. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. 140–151.

[42] Hassan Khosravi, Simon Buckingham Shum, Guanliang Chen, Cristina Conati, Yi-Shan Tsai, Judy Kay, Simon Knight, Roberto Martinez-Maldonado, Shazia Sadiq, and Dragan Gašević. 2022. Explainable artificial intelligence in education. *Computers and Education: Artificial Intelligence* 3 (2022), 100074.

[43] Sunnie SY Kim, Elizabeth Anne Watkins, Olga Russakovsky, Ruth Fong, and Andrés Monroy-Hernández. 2023. " Help Me Help the AI": Understanding How Explainability Can Support Human-AI Interaction. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–17.

[44] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian E Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica B Hamrick, Jason Grout, Sylvain Corlay, et al. 2016. Jupyter Notebooks-a publishing format for reproducible computational workflows. *Elpub* 2016 (2016), 87–90.

[45] W Bradley Knox and Peter Stone. 2008. Tamer: Training an agent manually via evaluative reinforcement. In *2008 7th IEEE international conference on development and learning*. IEEE, 292–297.

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

[47] Narine Kokhlikyan, Vivek Miglani, Miguel Martin, Edward Wang, Bilal Alsallakh, Jonathan Reynolds, Alexander Melnikov, Natalia Kliushkina, Carlos Araya, Siqi Yan, and Orion Reblitz-Richardson. 2020. Captum: A unified and generic model interpretability library for PyTorch. arXiv:2009.07896 [cs.LG]

[48] Maria Kouvela, Ilias Dimitriadis, and Athena Vakali. 2020. Bot-Detective: An explainable Twitter bot detection service with crowdsourcing functionalities. In *Proceedings of the 12th International Conference on Management of Digital EcoSystems*. 55–63.

[49] Sanjay Krishnan, Daniel Haas, Michael J Franklin, and Eugene Wu. 2016. Towards reliable interactive data cleaning: A user survey and recommendations. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*. 1–5.

[50] Sanjay Krishnan, Jiannan Wang, Eugene Wu, Michael J Franklin, and Ken Goldberg. 2016. Activeclean: Interactive data cleaning for statistical modeling. *Proceedings of the VLDB Endowment* 9, 12 (2016), 948–959.

[51] Nathan Lambert, Louis Castricato, Leandro von Werra, and Alex Havrilla. 2022. Illustrating Reinforcement Learning from Human Feedback (RLHF). *Hugging Face Blog* (2022). https://huggingface.co/blog/rlhf.

[52] Markus Langer, Daniel Oster, Timo Speith, Holger Hermanns, Lena Kästner, Eva Schmidt, Andreas Sesing, and Kevin Baum. 2021. What do we want from Explainable Artificial Intelligence (XAI)?–A stakeholder perspective on XAI and a conceptual model guiding interdisciplinary XAI research. *Artificial Intelligence* 296 (2021), 103473.

[53] Roger A Leite, Theresia Gschwandtner, Silvia Miksch, Simone Kriglstein, Margit Pohl, Erich Gstrein, and Johannes Kuntner. 2017. Eva: Visual analytics to identify fraudulent events. *IEEE transactions on visualization and computer graphics* 24, 1 (2017), 330–339.

[54] Haotian Li, Lu Ying, Haidong Zhang, Yingcai Wu, Huamin Qu, and Yun Wang. 2023. Notable: On-the-fly Assistant for Data Storytelling in Computational Notebooks. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–16.

[55] Q Vera Liao, Daniel Gruen, and Sarah Miller. 2020. Questioning the AI: informing design practices for explainable AI user experiences. In *Proceedings of the 2020 CHI conference on human factors in computing systems*. 1–15.

[56] Q Vera Liao and Kush R Varshney. 2021. Human-centered explainable ai (xai): From algorithms to user experiences. *arXiv preprint arXiv:2110.10790* (2021).

[57] Scott M Lundberg and Su-In Lee. 2017. A Unified Approach to Interpreting Model Predictions. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 4765–4774. http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf

[58] James MacGlashan, Mark K Ho, Robert Loftin, Bei Peng, Guan Wang, David L Roberts, Matthew E Taylor, and Michael L Littman. 2017. Interactive learning from policy-dependent human feedback. In *International conference on machine learning*. PMLR, 2285–2294.

[59] Trevor Manz. [n. d.]. *anywidget*. https://github.com/manzt/anywidget

[60] Gloria Mark, Daniela Gudith, and Ulrich Klocke. 2008. The cost of interrupted work: more speed and stress. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. 107–110.

[61] Nina McCurdy, Jason Dykes, and Miriah Meyer. 2016. Action design research and visualization design. In *Proceedings of the Sixth Workshop on Beyond Time and Errors on Novel Evaluation Methods for Visualization*. 10–18.

[62] Leland McInnes, John Healy, Nathaniel Saul, and Lukas Grossberger. 2018. UMAP: Uniform Manifold Approximation and Projection. *The Journal of Open Source Software* 3, 29 (2018), 861.

[63] Sean McKenna, Dominika Mazur, James Agutter, and Miriah Meyer. 2014. Design activity framework for visualization design. *IEEE Transactions on Visualization and Computer Graphics* 20, 12 (2014), 2191–2200.

[64] Tim Miller. 2019. Explanation in artificial intelligence: Insights from the social sciences. *Artificial intelligence* 267 (2019), 1–38.

[65] Harsha Nori, Samuel Jenkins, Paul Koch, and Rich Caruana. 2019. InterpretML: A Unified Framework for Machine Learning Interpretability. *arXiv preprint arXiv:1909.09223* (2019).

[66] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[67] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[68] William A Pike, John Stasko, Remco Chang, and Theresa A O'connell. 2009. The science of interaction. *Information visualization* 8, 4 (2009), 263–274.

[69] Deepthi Raghunandan, Zhe Cui, Kartik Krishnan, Segen Tirfe, Shenzhi Shi, Tejaswi Darshan Shrestha, Leilani Battle, and Niklas Elmqvist. 2024. Lodestar: Supporting rapid prototyping of data science workflows through data-driven analysis recommendations. *Information Visualization* 23, 1 (2024), 21–39.

[70] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*. 1135–1144.

[71] Yuji Roh, Geon Heo, and Steven Euijong Whang. 2019. A survey on data collection for machine learning: a big data-ai integration perspective. *IEEE Transactions on Knowledge and Data Engineering* 33, 4 (2019), 1328–1347.

[72] Andrew Ross, Nina Chen, Elisa Zhao Hang, Elena L Glassman, and Finale Doshi-Velez. 2021. Evaluating the interpretability of generative models by interactive reconstruction. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–15.

[73] Pouria Rouzrokh, Bardia Khosravi, Sanaz Vahdati, Mana Moassefi, Shahriar Faghani, Elham Mahmoudi, Hamid Chalian, and Bradley J. Erickson. 2023. Machine Learning in Cardiovascular Imaging: A Scoping Review of Published Literature. *Current Radiology Reports* 11, 2 (Feb. 2023), 34–45. https://doi.org/10.1007/s40134-022-00407-8

[74] Matthias Schwab, N Karrenbach, and Jon Claerbout. 2000. Making scientific computations reproducible. *Computing in Science & Engineering* 2, 6 (2000), 61–67.

[75] Michael Sedlmair, Miriah Meyer, and Tamara Munzner. 2012. Design study methodology: Reflections from the trenches and the stacks. *IEEE transactions on visualization and computer graphics* 18, 12 (2012), 2431–2440.

[76] Maxime Sermesant, Hervé Delingette, Hubert Cochet, Pierre Jaïs, and Nicholas Ayache. 2021. Applications of artificial intelligence in cardiovascular imaging. *Nature Reviews Cardiology* 18, 8 (Aug. 2021), 600–609. https://doi.org/10.1038/s41569-021-00527-2

[77] Helen Shen. 2014. Interactive notebooks: Sharing the code. *Nature* 515, 7525 (2014), 152–152.

[78] Francesco Sovrano and Fabio Vitali. 2021. From Philosophy to Interfaces: An Explanatory Method and a Tool Inspired by Achinstein's Theory of Explanation. In *26th International Conference on Intelligent User Interfaces*. 81–91.

[79] Fabian Sperrle, Mennatallah El-Assady, Grace Guo, Rita Borgo, D Horng Chau, Alex Endert, and Daniel Keim. 2021. A Survey of Human-Centered Evaluations in Human-Centered Machine Learning. In *Computer Graphics Forum*, Vol. 40. Wiley Online Library, 543–568.

[80] Thilo Spinner, Udo Schlegel, Hanna Schäfer, and Mennatallah El-Assady. 2019. explAIner: A visual analytics framework for interactive and explainable machine learning. *IEEE transactions on visualization and computer graphics* 26, 1 (2019), 1064–1074.

[81] Uzma Haque Syeda, Prasanth Murali, Lisa Roe, Becca Berkey, and Michelle A Borkin. 2020. Design Study" Lite" Methodology: Expediting Design Studies and Enabling the Synergy of Visualization Pedagogy and Social Good. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–13.

[82] Marly van Assen, Alexander C Razavi, Seamus P Whelton, and Carlo N De Cecco. 2023. Artificial intelligence in cardiac imaging: where we are and what we want. *European Heart Journal* 44, 7 (Feb. 2023), 541–543. https://doi.org/10.1093/eurheartj/ehac700

[83] Jacob VanderPlas, Brian Granger, Jeffrey Heer, Dominik Moritz, Kanit Wongsuphasawat, Arvind Satyanarayan, Eitan Lees, Ilia Timofeev, Ben Welsh, and Scott Sievert. 2018. Altair: Interactive Statistical Visualizations for Python. *Journal of Open Source Software* 3, 32 (2018), 1057. https://doi.org/10.21105/joss.01057

[84] Yoland Wadsworth. 1993. *What is participatory action research?* Action Research Issues Association.

[85] Fengjie Wang, Xuye Liu, Oujing Liu, Ali Neshati, Tengfei Ma, Min Zhu, and Jian Zhao. 2023. Slide4N: Creating Presentation Slides from Computational Notebooks with Human-AI Collaboration. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–18.

[86] Zijie J Wang, Katie Dai, and W Keith Edwards. 2022. Stickyland: Breaking the linear presentation of computational notebooks. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts*. 1–7.

[87] Zijie J Wang, David Munechika, Seongmin Lee, and Duen Horng Chau. 2022. Nova: A practical method for creating notebook-ready visual analytics. *arXiv preprint arXiv:2205.03963* (2022).

[88] Garrett Warnell, Nicholas Waytowich, Vernon Lawhern, and Peter Stone. 2018. Deep tamer: Interactive agent shaping in high-dimensional state spaces. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.

[89] Michael L. Waskom. 2021. seaborn: statistical data visualization. *Journal of Open Source Software* 6, 60 (2021), 3021. https://doi.org/10.21105/joss.03021

[90] Laura Weidinger, John Mellor, Maribeth Rauh, Conor Griffin, Jonathan Uesato, Po-Sen Huang, Myra Cheng, Mia Glaese, Borja Balle, Atoosa Kasirzadeh, et al. 2021. Ethical and social risks of harm from language models. *arXiv preprint arXiv:2112.04359* (2021).

[91] 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*. 1–12.

[92] Steven Euijong Whang and Jae-Gil Lee. 2020. Data collection and quality challenges for deep learning. *Proceedings of the VLDB Endowment* 13, 12 (2020), 3429–3432.

[93] Steven Euijong Whang, Yuji Roh, Hwanjun Song, and Jae-Gil Lee. 2023. Data collection and quality challenges in deep learning: A data-centric ai perspective. *The VLDB Journal* 32, 4 (2023), 791–813.

[94] Jupyter widgets community. 2023. Jupyter Widgets. https://ipywidgets.readthedocs.io/en/stable/ Accessed Feb 20, 2024.

[95] Christian Wirth, Riad Akrour, Gerhard Neumann, Johannes Fürnkranz, et al. 2017. A survey of preference-based reinforcement learning methods. *Journal of Machine Learning Research* 18, 136 (2017), 1–46.

[96] Yifan Wu, Joseph M Hellerstein, and Arvind Satyanarayan. 2020. B2: Bridging code and interactive visualization in computational notebooks. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. 152–165.

[97] Wenzhuo Yang, Hung Le, Silvio Savarese, and Steven Hoi. 2022. OmniXAI: A Library for Explainable AI. (2022). https://doi.org/10.48550/ARXIV. 2206.01612 arXiv:206.01612

[98] Chengbo Zheng, Dakuo Wang, April Yi Wang, and Xiaojuan Ma. 2022. Telling stories from computational notebooks: Ai-assisted presentation slides creation for presenting data science work. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. 1–20.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009