Integrating Model Structure Representation and History Retrieval in Computational Notebooks

Submitted April 2024, in partial fulfilment of the conditions for the award of the Bachelor degree.

20320978

School of Computer Science
University of Nottingham

Jianwen Lyu

Date: April 18, 2024

Abstract

Data analysts often find it difficult to conclude model structure or workflow from code or documentations in computational notebooks and to retrieve history data, such as previous experiments they have performed. Currently, there are many successful approaches focusing on one of these two problems. However, as observed from user interviews targeting at both data analysts and machine learning model users, where each participant was required to finish several model comprehension and model parameter tuning tasks, the author noticed that there is a need for integrating model overview and history retrieval together. Based on this overall objective, the author developed a Jupyterlab extension for integrating these two functionalities in a single extension interface, which can be displayed in parallel with user's computational notebook area. Through qualitative analysis, the extension was proved to offer additional convenience for the data exploring and machine learning process. This Machine Learning Helper can be found at the project repository in GitHub.

Table of Contents

1. Introduction

Computational Notebooks, such as Jupyter Notebook, have been widely used by data scientists as tools for multiple types of data-based analysis or machine learning tasks. These coding platforms are especially suitable for data-relevant tasks, for user can execute single code cells instead of whole project, which tends to be flexible and convenient for explorative data analysis, helping to discover data patterns and evaluate machine learning model performances. Combining multiple medias (code, text, plot, etc.) together, computational notebooks also enable users to obtain more instant feedback or evaluations comparing to traditional code editors.

However, in the process of exploring machine learning problems, data analysts may still find two main pain points:

1. It is hard to understand model structures from other's code.
2. It is difficult to tackle with repeated experiments with high similarity in modelling and tunning process.

For pain point 1, Since data analysts or researchers may not have specific knowledge of the coding mechanics of machine learning models, when analysing model behaviours, these people may be confused and unable to extract model structure from code contents in the notebook. Researchers have also found that the flexibility of computational notebooks can often lead to disordered writing style of machine learning models, making the code messy, harder to read and explain , pushing this point to an even more irritating level (Head et al. 2019). For pain point 2, as demonstrated by one study, even in computational notebooks, which take supporting iterative exploration as a core purpose, scientists still must go through same process manually to do even simple modelling tasks (Chattopadhyay et al. 2020). Since in this process, newer code and its execution results will overwrite former ones, when tunning or evaluating model parameters, the user may find it difficult to retrieve previous experiments for summarizing patterns from model performance histories. As stated by (E. S. Liu, Lukes, and Griswold 2023), in the process of explorative coding, as experiments and decisions are rapidly developed, abandoned and replaced, data scientist's ability to continually make quick and correct changes will be undermined. This situation can fit well in the model tunning procedures of machine learning, where continuous changes to current parameters should be made based on previous model performances. Hence, in the machine learning process, experiment histories are important for data scientists to make correct decisions.

Current solutions for data analysts to solve these two pain points at once are limited. Though various approaches are employed to tackle these two pain points, each of these approaches has encountered different challenges in corresponding process. For pain point 1, it may be necessary for data analysts to track the markdown descriptions in notebook contents to infer model structures, since these markdown contents can serve as code documentations to improve understanding. This brings up the challenge that as current machine learning models are gradually increasing in scale, it often takes more time and efforts for these users to find and track multiple markdown cells which are often located far away from each other, especially in exploratory tasks, where corresponding documentations in notebook contents also tend to be personal, exploratory and messy (Rule, Tabard, and Hollan 2018), which in several cases means markdown cells scattered at discrete locations around notebook contents, hard for user to focus on their relations and integrated semantics. In addition, even if the user has successfully collected all markdown cells indicating model structure, from fragmented information in these individual cells (even documented in detail), the user may still feel difficult to integrate them and build a full picture of the model structure. For example, an analyst may somehow gather all markdown contents indicating detailed information of each different model components, but they may find it difficult to link these model components together to form a workflow of the entire model, especially when the model complexity increases. For the second pain point, it is often essential for analysts to document the details of their previous experiments, encompassing data, model parameters, and performance metrics. In documenting these historical records, analysts frequently resort to tools or methods external to the computational notebook itself, such as version control systems or the direct copying and pasting of data (Kery, John, et al. 2019). This practice, while sometimes necessary to users, can complicate the management of information, making it challenging for users to gain a comprehensive understanding of the experimental context. It will be a better approach if the history recording tools can be integrated within the computational notebooks, which saves user from oscillating between notebook contents and their own history records.

These challenges motivate the need for a tool to aid with both machine learning model structure comprehension and former experiments recording for data analysts or machine learning model evaluators. Based on the pain points mentioned above, the objectives of this tool will be:

1. to provide an intuitive visualization of model structure.
2. to enable user to record past experiments at any time of the exploring process within the computational notebook.

Enlightened by these objectives, a Machine Learning Helper, with model structure visualizations and experiment history recording functionalities, is proposed in this article as a Jupyterlab extension. This extension can provide assistance for users to achieve both the two objectives, and therefore release corresponding pain points in exploring and experimenting process. To achieve objective 1, the extension enables a model structure view automatically extracted from the markdown documentations of the notebook in forms of both table-of-contents and graphics for each notebook file. In order to support extra flexibility, the extension also allows user to add more components in the table-of-contents view to extend original documentations. To achieve objective 2, the extension enables user to append notes under any of the model components shown on the table-of-contents view of specified notebook file, including past experiment results or user's own understanding of specific model components, these records will be stored in user's local storage space, and will be loaded under the same model component at next time user returns to this notebook file. Comparing to previous works in model structure visualization, or history retrieving, this Machine Learning Helper extension integrates these functionalities together with more intuitive graphics view of model structures, making it more convenient for users to both easily comprehend model structures and review past experiments during data exploring or model tunning in machine learning process, without seeking help outside the computational notebook.

2. Related Works

2.1. Exploratory Programming in Computational Notebooks

Data scientists encountered difficulties in performing machine learning tasks through traditional programming approaches. As a typical type of data-driven problems, one unique difficulty in machine learning, as discovered by researchers, is that iterations in code seem to be inevitable because users cannot fully trust the accuracy of the predictions of the machine learning system and cannot have a direct way to prove its correctness (Hill et al. 2016). To support the need of trying out different hypothesis, which is difficult by following traditional programming patterns or strategies, the concept of exploratory programming was introduced. Exploratory programming tends to be a programming workflow of exploring different approaches to obtain insights from data, and extend current goals based on these insights (Subramanian et al. 2019). As discussed by Kery et al., when iterated experimenting must appear in coding for achieving hard tasks, or in scenarios where user must explore different possibilities of the program, exploratory programming often plays an important role for its attributes: flexibility, discovery and innovation (Beth Kery and B. A. Myers 2017). Under this context, as exploratory programming, or exploratory data analysis in data science context, tends to be the solution, and has quickly become an important part of modern data science workflow (Batch and Elmqvist 2018), data scientists need a coding platform which enables highly iterative data exploration, as traditional code IDEs may not fully support this programming method.

In response to this need, computational notebooks have been welcomed by data scientists. Commonly used computational notebooks, such as Jupyter Notebook, often consist of multiple individual cells for code or markdown documentations, users can run these cells individually and the execution of code in one cell will not be affected by contents in another cell. Also, computational notebooks contain instant feedback for the code cell which user just run, including plot visualizations, which even makes process of exploration more intuitive. These attributes enable users to rapidly prototype and share exploration results, making the computational notebook a suitable and widely preferred environment for exploratory programming (Lau et al. 2020). Moreover, by combining documentations in markdown cells, code in code cells and executed outputs (in forms of plot, tables, etc.) in one file, computational notebooks naturally provide expressive and interactive assistance for exploratory data analysis, and with the support of literate programming such as Python, the user can also integrate the exploration process into production pipelines (Li et al. 2023). By using computational notebooks, data scientists can perform exploratory data analysis with more ease and convenience, and the lightweight and shareability of these notebooks enables data scientists to exchange ideas or hypothesis in open communities such as GitHub or Kaggle. However, as pointed by (A. Y. Wang, D. Wang, Drozdal, Muller, et al. 2022), even with computational notebooks, the exploration still tends to be hindered by the strict demand of exploratory data analysis: active, rapid and diverse explorations. While supporting rapid iterative exploration, on the other side, proper documentations or explanations in computational notebooks are shown to be tedious, and tracking previous experiments seems to be a huge workload for data scientists. These difficulties can obviously compromise the efficiency of exploration ideas sharing, and data pattern discovering in exploration process.

In this work, the Jupyterlab extension project aims to release these difficulties for data scientists by providing them an overview or explanation of model structures in notebook contents and enable them to record information about their past experiments in the data exploring process.

2.2. Model Structure Explanation in Computational Notebooks

Under the context that experiment results and exploration process sharing is now common between data scientists using computational notebooks, it is important to improve the ability of a data scientist to understand the exploration of other data scientists. In machine learning, this means to assist the readers of notebook documents to understand model structures and workflow. One obvious approach is to read the markdown documentations left by model developers, as the author of the notebook usually has most complete view and understanding of the model or their findings when exploring data. Well documented notebooks can provide most significant help for readers, for they can present the author's discoveries to the readers in direct and intuitive ways, increasing the efficiency of data exploration sharing.

However, potential problems still exist. As demonstrated by Wang et al., in notebooks for exploratory data analysis, as hypothesis and alternative solutions change frequently, documentations usually contain a broad range of topics and purposes (A. Y. Wang, D. Wang, Drozdal, X. Liu, et al. 2021). In a case study surrounding documentations of notebooks in Kaggle, they found that except for describing the contents of adjacent code cells, named Process, which occupies 58.65% of total content of the notebook, 32.36% of the content were used for organizing layout, specifying headlines of each part for navigation, and 19.19% of the markdown cells were used for describing and explaining results of code execution, some markdown cells were also used for explaining reasons of specific decisions, while some cells were mainly used for building to-do lists or summaries, explaining certain concepts, or providing references or meta-data (A. Y. Wang, D. Wang, Drozdal, X. Liu, et al. 2021). Given this number of potential topics of documentation, which may be separated in different markdown cell in different locations of the notebook, user experience will probably be compromised while scrolling all along the notebook to track these cells to find information they desire for, especially in modern machine learning tasks, if without some tools to help them.

To solve this problem, model structure visualizations tools were presented by many previous works. For example, Jupyter presented official Table of Contents functionality (jupyter toc), which will automatically generate a table of contents for user while launching the notebook by taking all user's headings in markdown cells. InsideInsights, a web-based data-driven reporting system, aiming to help data explorers to express their exploration process, can provide a hierarchical structured data-flow view on an interactive canvas (Mathisen et al. 2019). Albireo, developed by Wenskovitch et al., can build a graphic cell-level overview of the notebook, resolving relationships and similarities between different cells, and provide variable level navigations (Wenskovitch et al. 2019). ToonNote enables user to convert their cells in notebook to an interactable "comic view", where selected notebook contents will be summarized in a form of data comic in the extension panel (Kang et al. 2021). These works, with algorithmic or programming approaches, provide users with clear visualizations of notebook structure based on user's cell structure. The Machine Learning Helper extension in this article absorbed these advantages as part of the main functionalities. The extension has an interactive list-view of model components for user to dynamically navigate to corresponding cells, as well as adding new components for own designs. Also, an edible graphic interface is presented to users in a separate tab page of extension panel, providing more intuitive overview of model structure and overall dataflow.

2.3. History Recording and Retrieving in Computational Notebooks

Despite the difficulties surrounding documentations while exploring data patterns with computational notebooks in machine learning process, data scientists found themselves in a situation where new alternatives rapidly replacing old ones, and when they wanted to find what they used to think of or experiment with, they often had no idea because there were not enough effective approaches to retrieve their old experiment records. It is useful for data scientists to record sufficient details for interrelated steps happened to produce the results, which can assist data scientists to track their exploring process (Sandve et al. 2013), and as mentioned by Abdul et al., data scientists, especially when applying machine learning models, should be able to explain how they achieve current steps in analysis and this can help to better control the decisions made by themselves or by the algorithm (Abdul et al. 2018). By being able to retrieve their own experiment histories, data analysts can get better knowledge of how they came over to current state, and better avoid potential mistakes they have made before. Meanwhile, by having more understanding of patterns emerged from these records, analysts can also control their model parameters, tunning them to achieve improvements of performance.

Though supporting history recording and retrieving is difficult in computational notebooks since one characteristic of computational notebooks is fast code executing and replacing, there are many previous works indicating to solve this problem. At first came the version control systems helping users to backup their previous notebooks, one of the examples is Variolite, where user can create and manage different versions of code blocks by drawing “Variolite boxes” around the desired code clock (Kery, Horvath, and B. Myers 2017). Besides, one similar example from outside of computational notebooks is Azurite, a timeline-based code versioning plugin of Eclipse to help perform code backtracking, which means to help user going back to an earlier version of the code contents by removing later inserted code or restoring removed code (Yoon, B. A. Myers, and Koo 2013). To complement some inabilities of version control, including completeness and intuitiveness of history information, Verdant, a system enabling user to find each past performed choices, was presented. This system can provide powerful abilities to forage past notebook actions, inspect detailed history for partial and retrieve full notebooks in past versions (Kery, John, et al. 2019). In addition, Diff in the Loop, developed by Wang et al., supports visualizations of code and data differences between snapshot of past data versions dynamically during the data exploring process, which added another layer of flexibility while monitoring and controlling data exploration procedure (A. Y. Wang, Epperson, et al. 2022). These tools, by tracking past notebook actions through version control or history model approaches, can clearly improve relevant user experience.

However, while applying these automatic history tracking tools, it is always better to leave some space for users to manually record former experiments as notes. This can help user to cope with several emergent conditions and leave other messages for explanation except pure history information. In the Machine Learning Helper extension, user can take notes under model components in the list view of extension panel at any time of the exploration process, these notes will be saved for next visit, and user can delete them as like. Taking advantage of the time flexibility, this approach can supplement the disadvantage of some history tracking systems, since these systems usually wait for a check point to record history actions (Kery and B. A. Myers 2018), which on some occasions cannot fully perform their functionalities.

3. User Requirements

The user requirements were collected by remotely interviewing with participants. Through the interviews, general problems while using computational notebooks (commonly Jupyter Notebook or Jupyterlab) were discovered and expected functionalities from users were integrated into specifications. The interview questions first included several general questions about use experiences of computational notebooks, and pain points when using these notebooks in machine learning tasks. Also, during the interview, participants were requested to complete comprehension tasks of notebooks from GitHub, as well as parameter tuning tasks related to specific model written in these notebooks. There were totally 4 people attending the interview. One of the participants was a PhD developing reinforcement learning models, who regularly uses computational notebooks such as Jupyter Notebook for data analysing and model training. Two undergraduate students studying Computer Science at University of Nottingham were also counted as participants, since they were both attending a project surrounding computer vision model training. In addition, an undergraduate in University of Nottingham, Ningbo, China was interviewed, for he has extensive experience in building and training machine learning models using Jupyter Notebook, which was believed to provide valuable insights for the author.

Before the user interview procedure, the participants were all introduced to a machine learning sample solution project for an image classification problem with Kaggle's Cats vs Dogs image set collected from GitHub. The classifier model was encapsulated into one notebook file by the project author, where participants can perform hyper-parameter tuning in self-training process. The first task was to observe solution code provided for the notebook author, and write a short summary about the model workflow. In this task, participants were encouraged not seeking help for notebook author's documentations, and check these markdown text for help only when they feel confusing. The difference of participant's understanding before and after having access to markdown explanations were recorded, in order to research for the role of markdown explanations in machine learning process. The following task was to retrain the model from scratch, and tune the hyper-parameters for optimal model performance. Considering that this process may take relatively long time, approximately one day was left for all participants. The interview occurred after all participants submitted their model summaries and training results, during the interview, the participants were requested to answer several interview questions about difficulties encountered in the training, especially parameter tuning process, along with problems in understanding model structures, and other general questions of user experiences surrounding computational notebooks.

The results of interviews tended to show that some participants thought that in machine learning tasks, it is hard to tell a clear workflow of models from the code left by both others and earlier-selves. Some participants stated that even when building machine learning models from scratch, he or she can also forget what they done several hours earlier and must take more time to remember these lost steps in order to understand the current state of data or model performance. When performing given tasks about reading notebooks from GitHub, most participants feel confused as they had to scroll around to check different descriptions of different code parts, which were usually distractive and tiring procedures. In the model tunning tasks given in the interview, most participants took long time when there were no proper tools to help them remember their performed steps. They expressed that parameter tunning is a process where you need to make decisions based on your experience, records about the past experiments are important since you need to know under which set of parameters did the model perform better or worse.

After integrating and analysing the user requirements extracted from interviews, the project in this article aims to develop a machine learning assisting system in form of a Jupyterlab extension, which can support data analysts using computational notebooks to:

1. Have a comprehensive overview of notebook contents based on author's markdown documentations, in both table-of-contents and graphic views.
2. Quick access to the markdown cells from table-of-contents view.
3. Add user's own documentations or understandings for extra flexibility.
4. Be able to record experiment histories, or other supplement information such as hyper-parameter values in the extension.

Absorbing experiences from previous works, the extension developed in this work supports the functionality of model structure viewing and past experiment information recording. While what's different from previous approaches is that this extension integrates these functionalities together, making it more convenient to work with data exploring tasks for data analysts within the computational notebook.

4. Methodology

4.1. Research Process

Prior to initiating the development works of this project, it is crucial to determine the functionalities that the Machine Learning Helper tool will implement, as well as the optimal approaches to achieve these functionalities. In this section, the detailed method used for obtaining user requirements, process of analysing these user requirements, and possible approaches for implementation stage will be discussed. In addition, during the research procedure, several hypothesis for functionalities and implementations were obtained,

4.1.1. Analysing User Requirements

For each of the participants in user interview, a task including machine learning model training and parameter tuning was proceeded for user requirement collection. A raw solution for the Kaggle's Cats vs Dogs image classification problem, which is a classical machine learning task that requires model tuning and data analysing, written in Jupyter Notebook, was attributed to each participant. The participant was required to comprehend the notebook contents through model code left by the notebook author, and perform some parameter tuning to obtain several experiment results (model performance). After participants finished the tasks, they were introduced to several questions mainly about 3 aspects: general frustrating steps in the task, code clarity of the notebook, and data memorizing in parameter tuning process. These questions were based on the hypothesis that **notebook content understanding and history retrieving are origins leading multiple difficulties in machine learning process with computational notebooks**, which turned out to be a pattern discovered from related works. With the interview questions being answered, the author believe that this hypothesis will be proved during the user interview.

As shown by their replies, code clarity and data memorizing tended to be the general frustrating steps (as shown in User Requirements section). In the interview results, 3 of 4 participants complained about the code clarity, stated that the model code in the notebook forced them to take a long time to understand, and lacked detailed explanations for usages and purposes of code chunks. Also, in the model tuning process, all participants mentioned that it was a tedious process to memorize previous experiment details. Common steps taken by participants were to user csv files to record their past experiments. In normal python projects, stated by one participant, he can use libraries such as tqdm for data tracking, but in such iterative model training, which requires him to quickly think and abandon new hypothesis, switching from tools outside the notebook to notebook contents becomes much more frequent, thus made him exhausted. These observations supported the hypothesis that two main aspects, notebook comprehension and history retrieval, form the general difficulties in data exploring in computational notebooks, and based on these two aspects, main objectives for the implementation stage were established:

1. enable intuitive model overview for users
2. enable navigation to cells in notebook contents
3. enable quick notes for users to record experiment histories

The first objective was enlightened by common complaint from participants about code clarity, while the second objective was derived from a phenomenon happened to several participants where they found themselves scrolling everywhere in the notebook to find correct code or markdown cells. The third objectives was also obtained because all participants pointed out that there will be much convenient for them if they can record their experiments in somewhere in the notebook panel, without switching their screens to elsewhere. In addition, with correspondence to the user requirements mentioned in the previous section, additional functionalities such as adding custom documentations were considered within the scope of implementations, if these is still time left after achieving the main objectives.

4.1.2. Possible Approaches for Implementation Stage

Before the implementation stage, the hypothesis that **a Jupyterlab extension will be a more effective way to realize the objectives** was established. This hypothesis was based on the requirements to integrate cell navigation and notes taking into the notebook panel. Jupyterlab is a commonly used platform suitable for explorative coding, data analysis and machine learning with an embedded computational notebook interface (the notebook panel), while a Jupyterlab extension serves as extended part of the Jupyterlab system which can have its own user interfaces to form a parallel view along with the computatinal notebook interface, as detailed in the Jupyterlab extension documentation. This characteristic of JupyterLab extensions aligns seamlessly with the aforementioned development objectives. However, to validate the hypothesis, further exploration into relevant software methodologies for constructing a JupyterLab extension is required.

Through researches on software approaches to build a Jupyterlab extension with functionalities to achieve the implementation objectives, multiple types of software libraries including Jupyterlab's official API, Lumino and yFiles, as will be mentioned in details in the next section, were found to be useful in the implementation process. With methods and classes provided by these software libraries serving as basic components, the extension will be relatively easy to build in the implementation stage comparing to normal desktop-based software or web applications. This is because that it will be much easier to realize the objectives that are all required to be embedded in user's computational notebook interface in the user interface of a Jupyterlab extension, which can already be shown together with the notebook panels. Based on these findings and derivations, the hypothesis was believed to be well supported. Consequently, it was resolved that the Machine Learning Helper tool would be developed as a JupyterLab extension on the belief that such a format would facilitate a simpler and more efficient design and implementation process.

4.2. Software Libraries

4.2.1. Jupyterlab

Developed as an extended coding and data analysing platform for Jupyter Notebook users, Jupyterlab contains additional functionalities, including Jupyter Notebook panel, code terminals, text editors, etc (@jupyterlab documentaion). To enable more flexibility for user to manipulate their own workspaces, Jupyterlab allows user to build and share their own extensions, and provides a set of official classes and interfaces for adding functions or inserting extra panels into the default lab layout. These classes and interfaces are encapsuled in several sub-namespaces under the @jupyterlab namespace.

In this work, JupyterFrontEnd, as well as JupyterFrontEndPlugin class in the application namespace were used to launch the extension in Jupyterlab's environment and link its functionalities with user actions. ICommandPalette, InputDialog and several Toolbar related classes under apputil namespace were implemented to enable user interactions through both commands and user interfaces. Several Notebook related classes under notebook namespace, enabling notebook changes tracking and notebook contents accessing functions, were deployed to realize markdown components extracting, sorting and navigating. Also, ReactWidget class, which can allow React-based approaches in extension development, were used for graphic interfaces.

4.2.2. Lumino

The software library Lumino is a set of JavaScript packages written in TypeScript (Lumino GitHub page). This library provides a rich toolkit of widgets, layouts, events and data structures, enabling developers to build web applications for highly interactive and reproducible computing tasks, such as exploratory data analysis or machine learning. The structure of Lumino applications are typically consists of different types of panels as containers, and widgets within the containers can encapsule original HTML components or self-defined functionalities. By combining these widgets in flexible ways, developers can build application interfaces with high performance.

In this work, Lumino UI components, such as Panel and Widget, were used for constructing the basic framework of extension user interfaces. Text views extracted from notebook information, or functions for background processing were encapsulated in Widget objects of Lumino, forming the basic building blocks of this extension. These blocks will be sorted following specific hierarchy and will be inserted into the list view panel of extension. In addition, some non-UI-based functions, such as notes storing and loading functions, still need information from these building blocks. By using the powerful methods to link Widgets and its containers in Lumino, the above considerations tended to be implemented with great ease and convenience in the development stage of this project. Taking advantage of exploratory task supports provided by Lumino components, it is believed that this extension will also bring convenience in machine learning process of its users.

4.2.3. yFiles

As one of the solutions to generate structured graphic layout with automatic arranging algorithms within its nodes and edges (Eichelberger 2003), the software library yFiles can provide help for visualizing notebook contents into intuitive graphic views. There are multiple libraries as branches of yFiles library, and yFile for HTML, which is designed for web-based projects, was used in this work as the visualizing tool (yFiles for HTML page).

The yFiles graph in the graphics view of this work consists a start point, meaning the start of model workflow, and nodes containing texts extracted from markdown cells of user's notebook. Edges were created between specific nodes depending on default or user's personal settings and the user can set one markdown component block to several displaying modes and consequently change the edges layout of graphics view at any time. yFiles was proved to be right choice for achieving this flexibility, for it provides a large set of automatic layout generating algorithms, such as hierarchical layout, which is implemented in this project, tree layout or circle layout. These layout algorithms can save the workload of attributing spaces and locations for each nodes in the graph, which tended to be a savor when handling complex model structures in the extension. Also, yFiles provides functions for dynamically modifying node styles and edge distributions, making it well supported for user to change graph layout when using the extension.

5. Design

5.1. User Interface Design

As the direct bridge between user and functionalities, user interfaces play important role when judging the overall quality of a system. During the research process of user interface designing paradigms, it was found that classic overview+detail interfaces, introduced by Cockburn and colleagues, are suitable for the context of building Jupyterlab extensions, since such computational

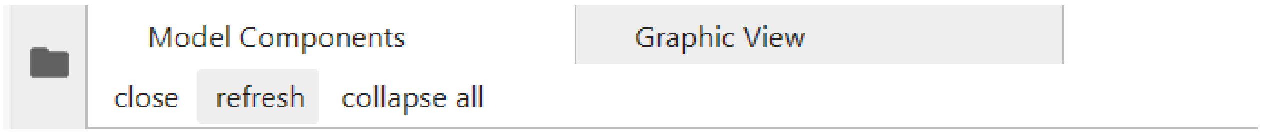


Figure 1: Refresh

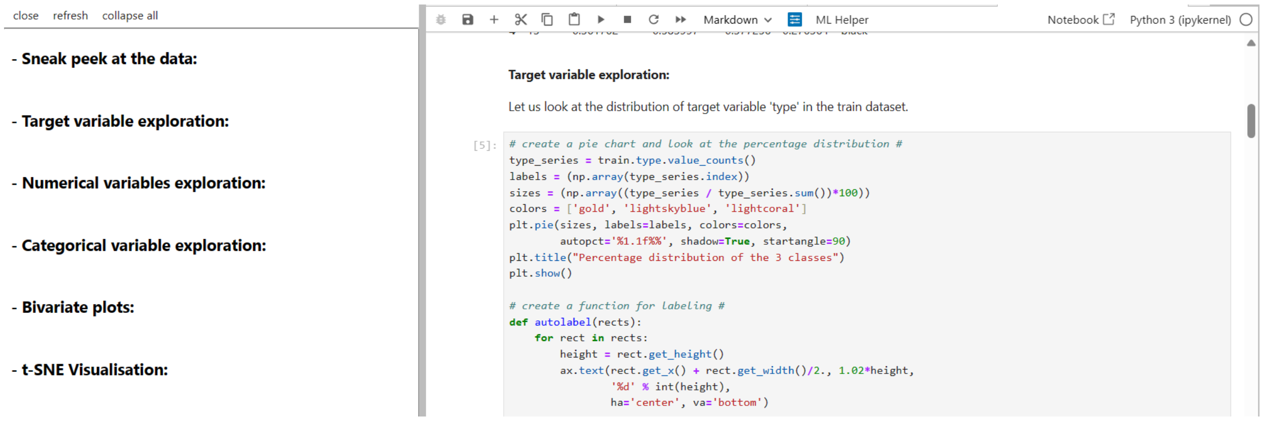
notebooks provide natural detailed model information in markdown documentation cells and overview can be placed separately on side bars by extensions, fitting the Cockburn's idea that overview and detail with spatial separation (each in distinct presentation space) can avoid information in these two views alternating with each other, thereby to efficiently help user manage focused and contextual information to finish works with improved speed (Cockburn, Karlson, and Bederson 2009). The user interface of this work applied this overview+detail conceptual design prototype, where panels for model structure representation were placed as side panels individual to main notebook contents, and this design turned out to be clean and intuitive for users.

In addition, to enable both interactive Table-of-Contents to conclude model structures in a more classic form and graph interface to provide users more intuitive recognition of model workflow, the main panel of extension is separated to tabs of Table-of-Contents view and Graphics view. From one of the two views user can switch to another, and from Table-of-Contents view user can manipulate displaying structure of the Graphic view, as will be discussed in next few sections. Both these views have a "refresh" functionality capsuled in a button (Figure 1), which can refresh the extension panel to enable users to instantly observe the changes on user interface caused by their interactions, such as adding a new model component, or adding some explanations as notes, etc. These model overview interfaces and corresponding functionalities are believed to help users obtain a clear understanding of model structure and general model workflow, which can fulfill the first objective from user requirements.

5.1.1. Table-of-Contents View

Among the two views available to users, the Table-of-Contents view is served as a container for "Component Blocks", which is a concept of components forming the structure of machine learning model or other project code in notebook contents. These component blocks, as shown in Figure 2, contain their individual text descriptions, which are integrated with hierarchy in the extension panel to compose the overview of model structure. Also, these Component Blocks encapsulate several functionalities based on user interaction, such as cell-level navigating and notes recording. Since model structure overview in the extension is based on markdown documentations in the notebook, text descriptions of the Component Block were designed to be captured from markdown headings left by notebook authors, bolded text occupying individual lines are also taken into consideration, since in some scenarios, such as in several notebooks found in Kaggle, bolded text lines can also play a role of guidelines to overall model structures.

Though for most computer programs the execution order is from up to bottom, in computational notebooks, since code cells can be executed individually, model developers often create code structures including both sequentially executed steps and individual modules for data representing, outcome



- **Sneak peek at the data:**
- **Target variable exploration:**
- **Numerical variables exploration:**
- **Categorical variable exploration:**
- **Bivariate plots:**
- **t-SNE Visualisation:**

Target variable exploration:

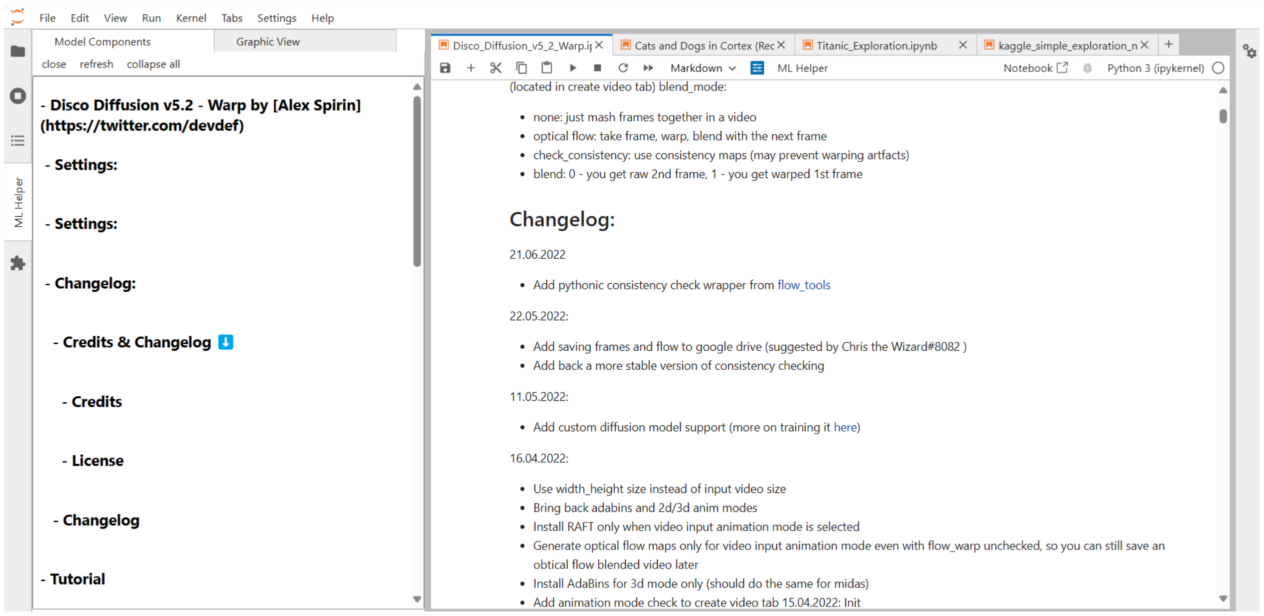
Let us look at the distribution of target variable 'type' in the train dataset.

```
[5]: # create a pie chart and look at the percentage distribution #
type_series = train.type.value_counts()
labels = (np.array(type_series.index))
sizes = (np.array((type_series / type_series.sum())*100))
colors = ['gold', 'lightskyblue', 'lightcoral']
plt.pie(sizes, labels=labels, colors=colors,
        autopct='%1.1f%%', shadow=True, startangle=90)
plt.title("Percentage distribution of the 3 classes")
plt.show()

# create a function for labeling #
def autolabel(rects):
    for rect in rects:
        height = rect.get_height()
        ax.text(rect.get_x() + rect.get_width()/2., 1.02*height,
                '%d' % int(height),
                ha='center', va='bottom')
```

Figure 2: Component Blocks (with bold text in consideration)

analysing, etc, making the machine learning model structure in the notebooks more complicated. To simulate these structures, after the step of capturing all these model components from notebook and build corresponding Component Blocks, a proper hierarchy of these Component Blocks should also be determined. Considering markdown has heading levels as natural indicators for structure, the Component Blocks take these heading levels as their component levels (while bold text lines are considered as the lowest level). Based on these component levels, all Component Blocks will be inserted in a tree-like hierarchy indicating model structure by a hierarchy generating algorithm, the algorithm starts at the first markdown heading detected, perform a Deep First Search to find its potential children (Component Blocks with lower component levels), once no children can be found, the parent Component Block and all its children will be rendered to the Table-of-Contents panel. In order to show the hierarchy more intuitively, text descriptions of low-level Component Blocks are designed to have fixed length of margins depending on their levels, distinguished with higher-level ones.

As for the encapsuled functionalities of Component Blocks, a menu can be invoked by clicking one specific Component Block. This menu includes several available user interactions known as user options, as shown by Figure 4. One potentially useful option is Component Block collapsing and expanding, which means that parent Component Blocks can hide or show its children following user's preference. Except expanding and collapsing function for single Component Block, a "collapse all" button is provided to users to only show Component Blocks obtained from first level headings in markdown parts of the notebook, which may probably be better for user to quickly get a global understanding of the notebook contents. It is proved that in the main panel of computational notebooks, cell folding has a clear positive effect on both original analysis and later reuse, encouraging clearer communication, freer sharing and deeper engagement with complex data analysis (Rule, Drosos, et al. 2018). From this observed pattern it is supported that for the structure overview part of the extension, enable collapsing and expanding, which is similar to cell folding, can also help users to better understand the model structure since through this approach user can first have more general and global view, thereby aid the proceeding details organization and navigation, especially useful in circumstances when large number of model components dividing into different branches of the structure.


File Edit View Run Kernel Tabs Settings Help
Model Components
Graphic View
close refresh collapse all
Disco_Diffusion_v5_2_Warp.i
Cats and Dogs in Cortex (Rec
Titanic_Exploration.ipynb
kaggle_simple_exploration_n
Markdown
ML Helper
Notebook
Python 3 (ipykernel)
ML Helper

- Disco Diffusion v5.2 - Warp by [Alex Spirin](https://twitter.com/devdef)
- Settings:
- Settings:
- Changelog:
- Credits & Changelog ⬇️
- Credits
- License
- Changelog
- Tutorial

(located in create video tab) blend_mode:

- none: just mash frames together in a video
- optical flow: take frame, warp, blend with the next frame
- check_consistency: use consistency maps (may prevent warping artfacts)
- blend: 0 - you get raw 2nd frame, 1 - you get warped 1st frame

Changelog:

21.06.2022

- Add pythonic consistency check wrapper from flow_tools

22.05.2022:

- Add saving frames and flow to google drive (suggested by Chris the Wizard#8082)
- Add back a more stable version of consistency checking

11.05.2022:

- Add custom diffusion model support (more on training it here)

16.04.2022:

- Use width_height size instead of input video size
- Bring back adabins and 2d/3d anim modes
- Install RAFT only when video input animation mode is selected
- Generate optical flow maps only for video input animation mode even with flow_warp unchecked, so you can still save an obtical flow blended video later
- Install AdaBins for 3d mode only (should do the same for midas)
- Add animation mode check to create video tab 15.04.2022: Init

Figure 3: Component Blocks Hierarchy

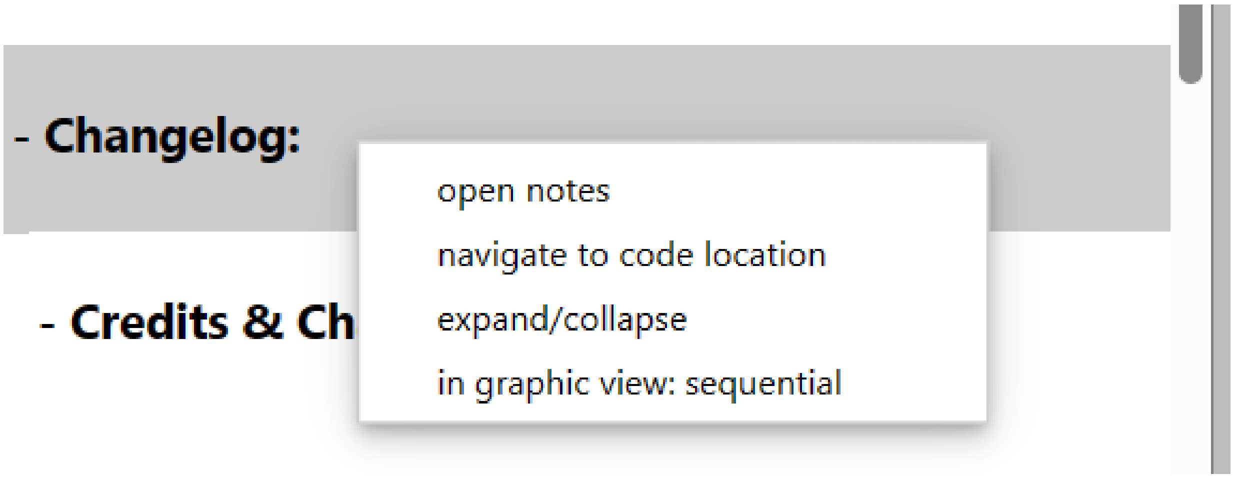
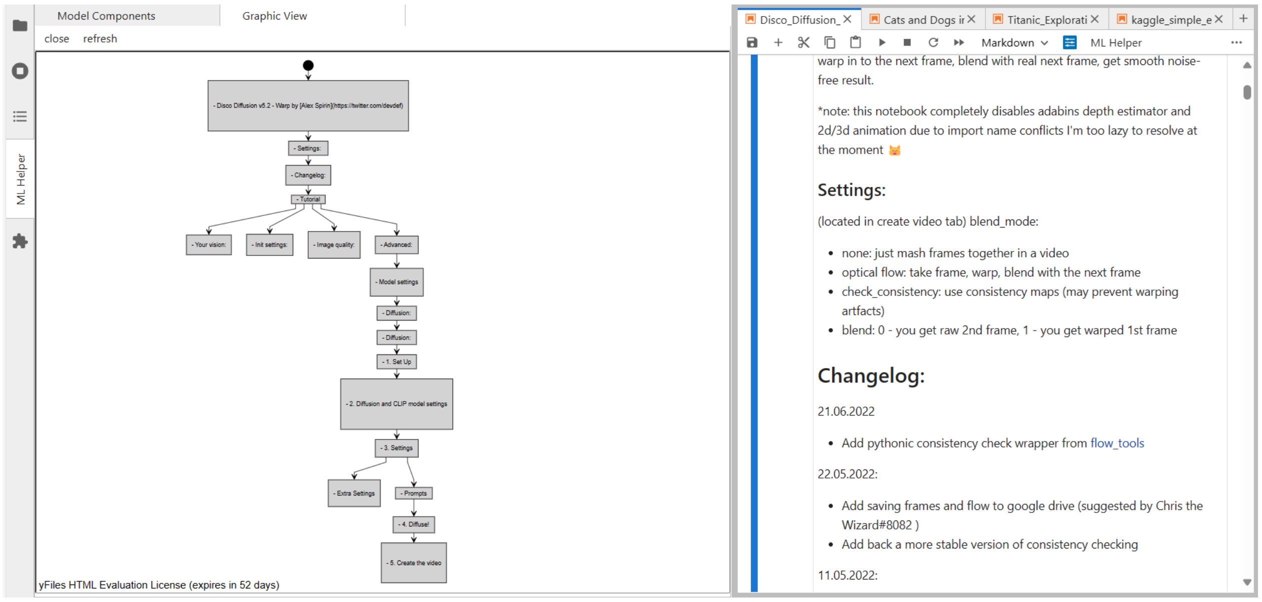


Figure 4: Option Menu



warp in to the next frame, blend with real next frame, get smooth noise-free result.

*note: this notebook completely disables adabins depth estimator and 2d/3d animation due to import name conflicts I'm too lazy to resolve at the moment 😺

Settings:

(located in create video tab) blend_mode:

- none: just mash frames together in a video
- optical flow: take frame, warp, blend with the next frame
- check_consistency: use consistency maps (may prevent warping artfacts)
- blend: 0 - you get raw 2nd frame, 1 - you get warped 1st frame

Changelog:

21.06.2022

- Add pythonic consistency check wrapper from flow_tools

22.05.2022:

- Add saving frames and flow to google drive (suggested by Chris the Wizard#8082)
- Add back a more stable version of consistency checking

11.05.2022:

Figure 5: Graphics Panel

5.1.2. Graphics View

As stated by Kuhn et al., proper software visualization can be great use of comprehending and exploring the system in an intuitive manner (Kuhn, Erni, and Nierstrasz 2010). As a web-based interactive system containing multiple interactive user interfaces, the Machine Learning Helper extension should also contain an intuitive graphic visualization of general machine learning workflow to improve comprehension and exploration. Therefore, the extension is equipped with a graphic view as an individual panel alongside with Table-of-Contents view, as shown in Figure 5. Same as the Table-of-Contents view, the graphics view is also based on Component Blocks obtained from the markdown cells in notebook. For each Component Block, a corresponding node will be shown on the graphics interface with edges connecting each other, indicating sequential or parent/child relations. In order to represent multiple types of model structures, each node has two display modes in the graph, "Sequential" and "Parallel", which will be mentioned in later sections. The default display mode for each node is "Sequential", since for most machine learning related notebooks, components are explained with sequential order. However, as mentioned in previous sections, some machine learning models may contain both sequential steps where each step relies on the product of previous steps and individual code blocks whose executions often serve as optional settings or code effect presentations. One example is Google Colab's DiscoDiffusion notebook, where many steps of the main model workflow include extra settings or explanatory code blocks, such as custom model settings, partial saves, changelogs or other advanced configuration options. To handle these complex model structures, the extension also allows user to change the display mode of nodes in the option menu of each Component Block in Table-of-Contents view.

5.2. General Workflow

From the user requirements, the main objectives this extension system will achieve has already known. However, apart from simply achieving these functionalities, it requires careful consideration

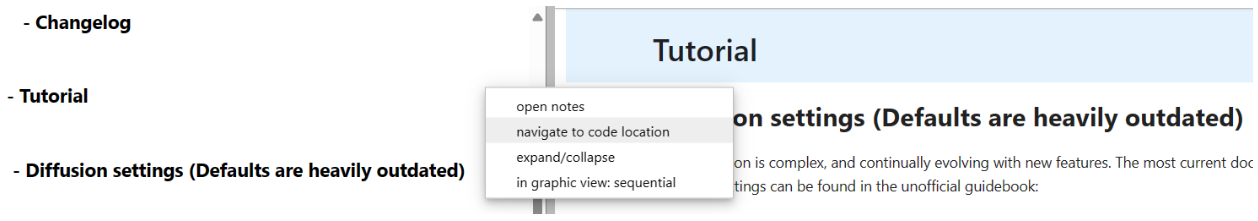

- Changelog
- Tutorial
- Diffusion settings (Defaults are heavily outdated)
Tutorial
open notes
navigate to code location
expand/collapse
in graphic view: sequential
on settings (Defaults are heavily outdated)
on is complex, and continually evolving with new features. The most current doc
tings can be found in the unofficial guidebook:

Figure 6: Navigation to Model Components

of how to combine these functionalities into the available scope of one single system while ensuring the user can have a clear consciousness of the correct workflow when using it. In this work, the general workflow of the extension is designed toward a set of simple and direct user operations which basically contains all available ways of interaction. These interactions should not require much extra comprehension or exploration of “How to use the system”. Based on this consideration, expected user operations on this extension (table-of-contents view and graphics view) are mainly divided to 4 User Actions.

5.2.1. User Action 1: Navigating to Notebook Components

The most commonly needed model structure related user option for computational notebooks is usually proper navigation, since one study has demonstrated that in the programming process, users generally spend 35 percent of total working time navigating between different code blocks (Ko, Aung, and B. A. Myers 2005). In this extension, the term of ”component-level navigating”, which tends to be a slightly different feature from common cell-level navigation in previous works, is implemented. Combining experience from cell-level navigation in these works with Component Block structure of this work, the extension enables user to navigate to corresponding cells containing explanations of the selected Component Block. The core difference between component-level navigation and cell-level navigation is that multiple Component Blocks can be navigated to the same markdown cell. The reason of this design is that many notebooks found in GitHub prefer to combine descriptions of different steps that may be useful for users, such as different stages in model parameter settings, into one single markdown cell, making it difficult to summarize the whole meaning of the cell for some targeted navigation. In comparison, component-level navigation in this extension, which based on Component Blocks instead of whole markdown cells, tends to be a clearer way for user to know complete information within the documentations in single cells. To perform this component-level navigation, while opening the option list of a Component Block in the Table-of-Content view, user can observe a “navigate to code location” option, which can bring them to the corresponding notebook cell containing the original markdown titles or text used to build this Component Block, as shown in Figure 6. The navigating functions in the extension achieve the second objective of user requirements, and can probably help user find desired parts in the notebook more rapidly.

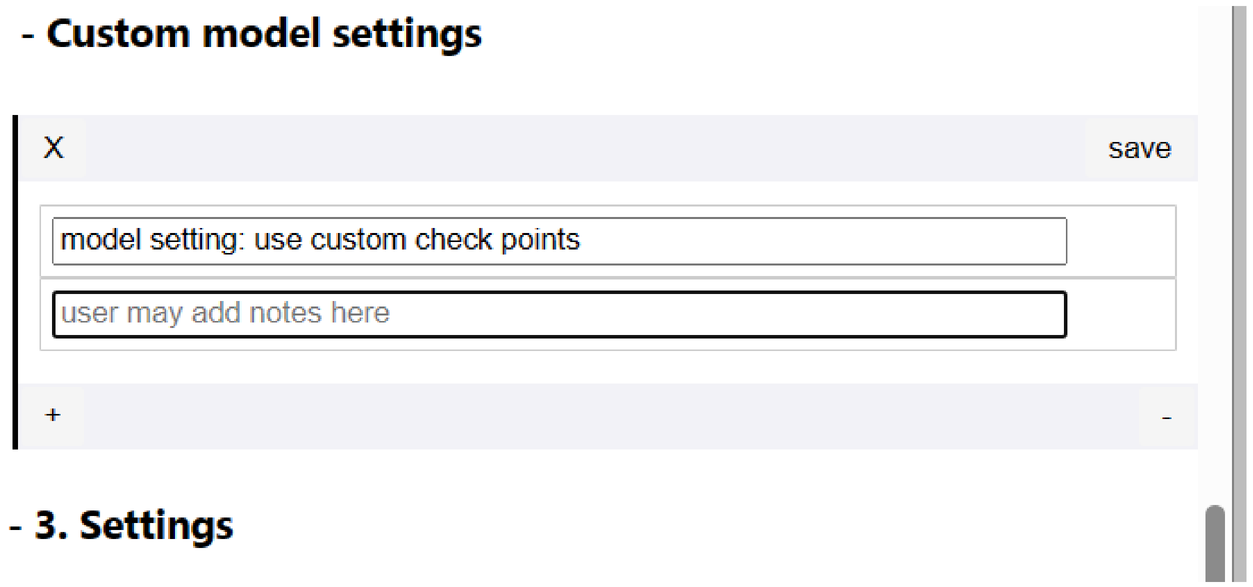
5.2.2. User Action 2: Taking Notes of Notebook Components

Previous sections have been introducing model structure overviewing functionalities and relevant user interface designs. However, according to the user requirements, one significant problem for exploratory data analysis should also be solved by this extension, which is recording and retrieving previous notebook actions, including past experiment configurations, experiment results and model performance, etc. There are multiple types of approaches applied by other computational notebook history recording systems. For example, Verdant implemented a version control model called litGit, which was developed based on Git but also employed an Abstract Syntax Tree (AST) for storing smaller artifacts down to code snippets to prevent data duplication (Kery, John, et al. 2019). The Verdant system listens important notebook UI events triggered by user interactions, such as new notebook loading, content changing and code executing, thereby record related data as history record in the AST structure mentioned above (Kery, John, et al. 2019). Besides, Azurite implements a timeline view which can automatically show changes in code performed by users, allowing users to observe current code edits in real-time and access historical code edits (Yoon, B. A. Myers, and Koo 2013). There are still other existing examples of history tracking based on different approaches, while the Machine Learning Helper extension in this work is designed to enable manual noting functionalities as a supplementary approach to these existing extensions.

In the Machine Learning Helper extension, user can open a noting area under selected Component Block, as shown in Figure 7. The noting area contains note clips in a list view, where each note clip indicates a history record, or simply an explanatory information authored by the user. These notes can be created or checked at any time during the data exploring process, after the creation of a note clip, data in this clip will be stored in the user's local storage space in form of a JSON string, and when user open the noting area under same Component Block, the corresponding notes will be loaded and shown to the user. To help user identify noted components from others, after users take a note under some Component Block, the title of component will be added a text identifier shown as "Original Title (Notes)", as shown in Figure 8. The extension also provides a way to delete existing notes if user found themselves wrongly recorded some data, or created some empty notes by mistakes. By clicking the "subtract" icon in bottom bar of the noting area, the user can delete the note clip they just created and after the deleting operation, the storage of this note clip will also be erased. This noting system, though without algorithmic approaches for automation, still provides user essential abilities for history recording within the computational notebook interface, which can make data exploring more convenient along with the model structure overview functionality mentioned in previous sections, and satisfies the third system objective derived from user requirements.

5.2.3. User Action 3: Adding / Deleting Custom Component Blocks

To produce a high-performance interactive computing environment, the system should allow dynamic introspection of users, which means to enable users to directly access to code and objects existing in the environment (Perez and Granger 2007). Previous works surrounding exploratory data analysis in computational notebooks did satisfy this design principle by providing intuitive notebook content overviews and automatic navigations. However, notebook overviews merely depending on notebook contents tend to provide less user accessibility in some scenarios. For example, when using current available overview approaches such as Jupyter's Table-pf-Contents, if user want some direct and rapid ways to access some normal code cells with no markdown descriptions, it may still go through a tedious manual navigating process since current cell-level navigations normally target at markdown contents. Taking this into consideration, the Machine Learning Helper extension allows user to create custom Component Blocks for any notebook cells, including markdown cells and code cells. To ensure the hierarchy of original Component Blocks, this functionality will not be evoked when launching the extension, instead, user can create custom Component Blocks in the context menu of selected notebook cell by right-clicking (Figure 9).


- Custom model settings

X save

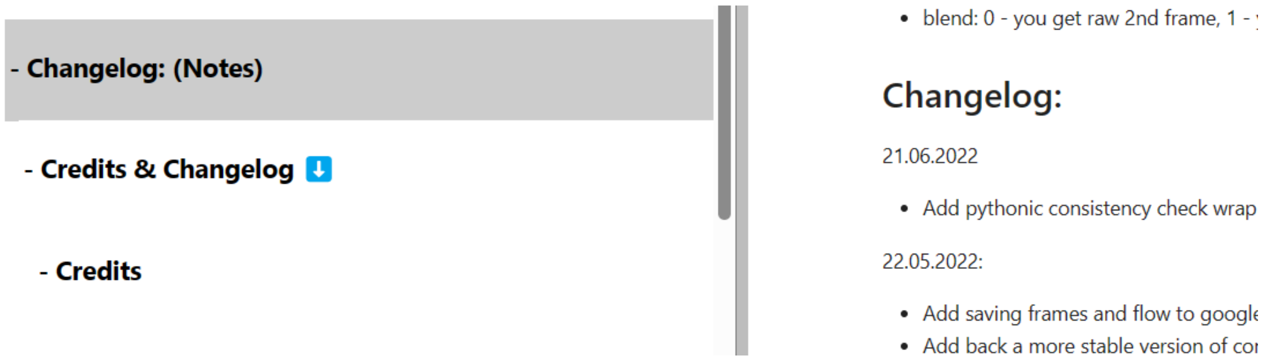
model setting: use custom check points

user may add notes here

+ -

- 3. Settings

Figure 7: User Notes


- Changelog: (Notes)

- Credits & Changelog ⬇️

- Credits

- blend: 0 - you get raw 2nd frame, 1 -

Changelog:

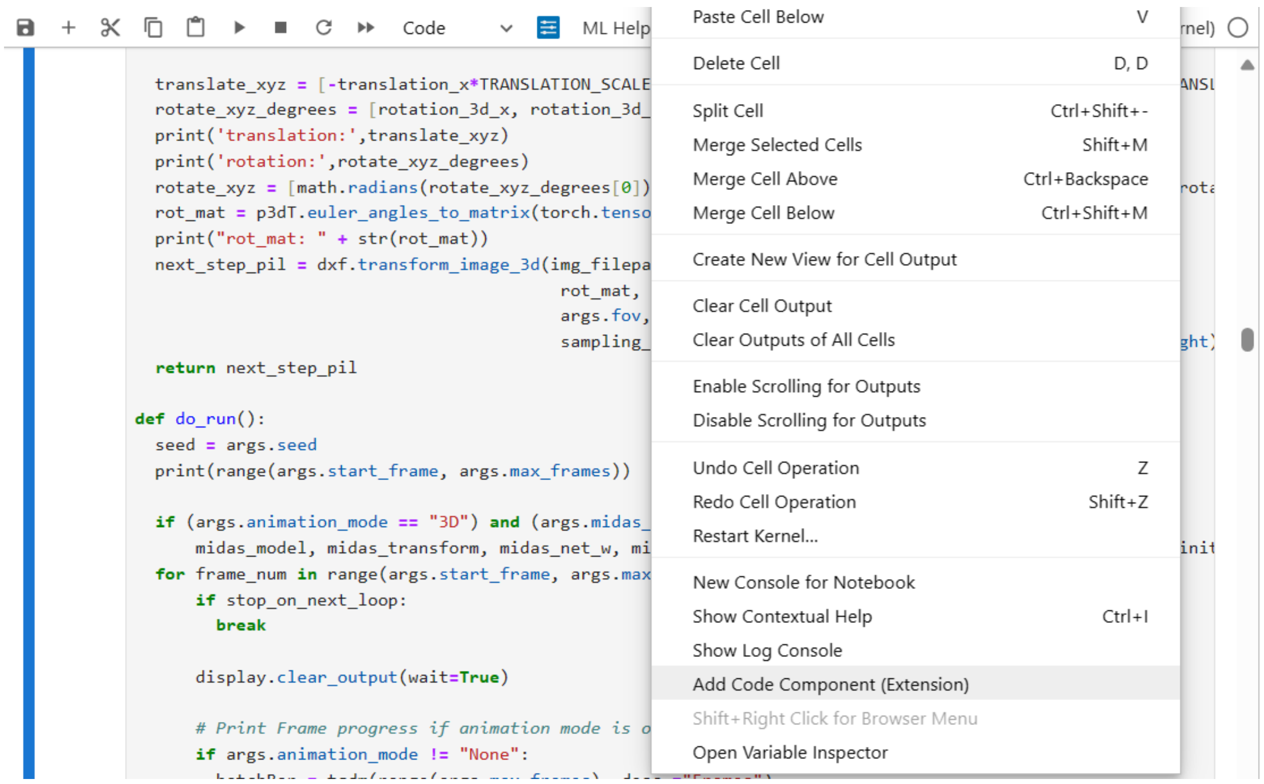
21.06.2022

- Add pythonic consistency check wrap

22.05.2022:

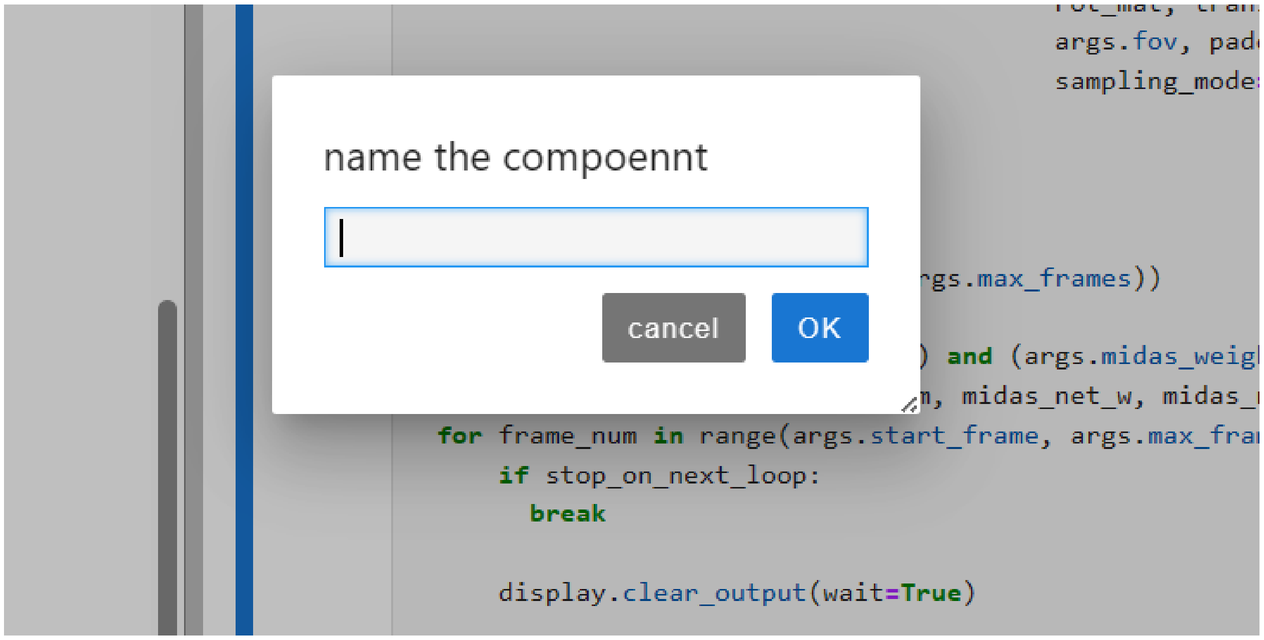
- Add saving frames and flow to googl
- Add back a more stable version of co

Figure 8: Title of Component Blocks with Notes


Code
ML Help
rnel)
Paste Cell Below V
Delete Cell D, D
Split Cell Ctrl+Shift+-
Merge Selected Cells Shift+M
Merge Cell Above Ctrl+Backspace
Merge Cell Below Ctrl+Shift+M
Create New View for Cell Output
Clear Cell Output
Clear Outputs of All Cells
Enable Scrolling for Outputs
Disable Scrolling for Outputs
Undo Cell Operation Z
Redo Cell Operation Shift+Z
Restart Kernel...
New Console for Notebook
Show Contextual Help Ctrl+I
Show Log Console
Add Code Component (Extension)
Shift+Right Click for Browser Menu
Open Variable Inspector

```
    translate_xyz = [-translation_x*TRANSLATION_SCALE
    rotate_xyz_degrees = [rotation_3d_x, rotation_3d_
    print('translation:',translate_xyz)
    print('rotation:',rotate_xyz_degrees)
    rotate_xyz = [math.radians(rotate_xyz_degrees[0])
    rot_mat = p3dT.euler_angles_to_matrix(torch.tenso
    print("rot_mat: " + str(rot_mat))
    next_step_pil = dxf.transform_image_3d(img_filepa
                                        rot_mat,
                                        args.fov,
                                        sampling_
    return next_step_pil

def do_run():
    seed = args.seed
    print(range(args.start_frame, args.max_frames))

    if (args.animation_mode == "3D") and (args.midas_
        midas_model, midas_transform, midas_net_w, mi
    for frame_num in range(args.start_frame, args.max
        if stop_on_next_loop:
          break

        display.clear_output(wait=True)

        # Print Frame progress if animation mode is o
        if args.animation_mode != "None":
```

Figure 9: Adding Custom Component Blocks

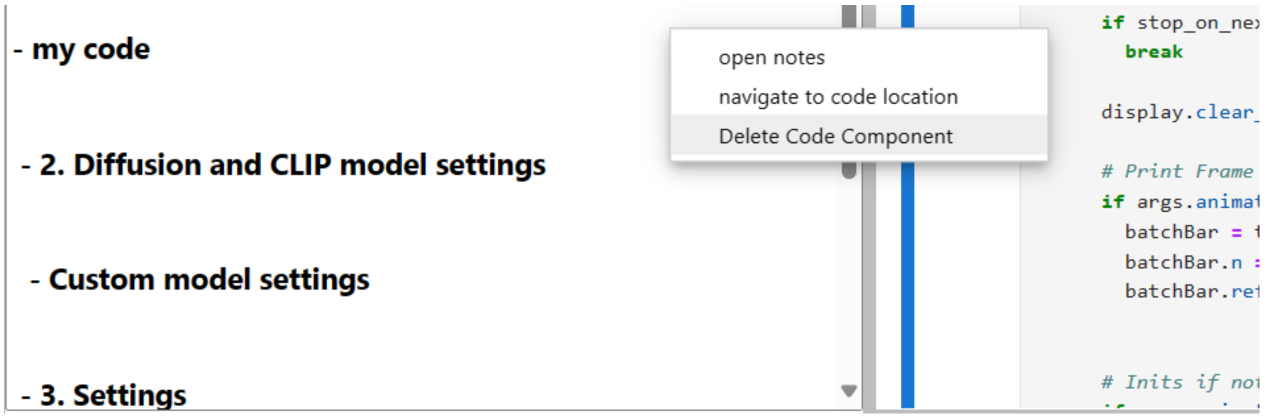


```
                                        rot_mat, tran
                                        args.fov, pad
                                        sampling_mode
                                    gs.max_frames))

                                    ) and (args.midas_weig
                                    n, midas_net_w, midas_
    for frame_num in range(args.start_frame, args.max_fra
        if stop_on_next_loop:
          break

        display.clear_output(wait=True)
```

Figure 10: Component Block Naming

The custom Component Block creation process can be split to 3 steps: custom Component Block naming, hierarchy restoring, and data saving. In the naming step, on query window will pop up and ask user to name the Component Block they just created (Figure 10). Once the user confirmed their custom name, this name will be set to the title of the custom block, and stored to the user's local storage space for next visit. After the custom block is created, it will be inserted to the model structure overview panel in Table-of-Contents view, but as mentioned in previous parts, since notebook has its own cell structure, and the Table-of-Contents view is the simulation to this cell structure. Simply placing the new Component Block after old blocks will break the hierarchical structure of existing Component Blocks. In order to solve this problem, the next step is to restore the hierarchy of these blocks, including the newly created one. The method in this step is to insert the new block into proper location comparing to the original notebook cell structure, without modifying the order of existing Component Blocks. Starting at the user selected cell, which is about to be a new Component Block, the extension will first find nearest markdown heading (include bold text) prior to this cell, for in most cases the selected code cell will belong to the documentation scope of the closest prior markdown text. With the nearest prior markdown heading in the notebook, the extension will search for the same heading in existing Component Blocks and insert the newly created block as a child under that block. After restoring the hierarchy, relevant data containing in the custom Component Block should be saved, including its title, its location and its ID for tracking possible notes user created after its creation. These data will be saved to user's local storage space, along with information of other Component Blocks.

Though through adding custom Component Block, the user can have access to more notebook contents through the Table-of-Contents panel of the extension. There might also be a use case that some users create custom Component Blocks mistakenly, or left improper name for the newly created custom block. During the design process, the developer of Machine Learning Helper extension also considered these scenarios and custom component deleting functionality is enabled in this extension


- my code

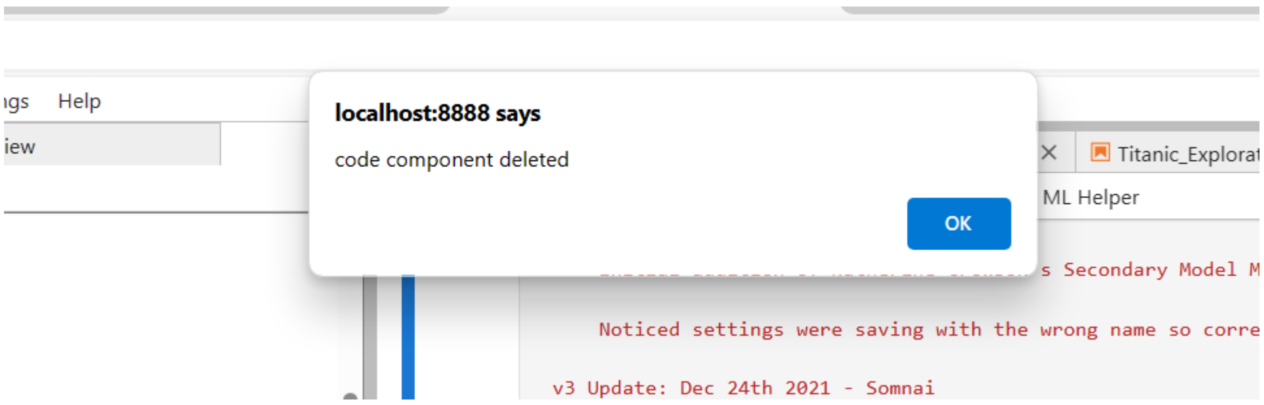
- 2. Diffusion and CLIP model settings

- Custom model settings

- 3. Settings

```
if stop_on_nex
  break

display.clear_

# Print Frame
if args.animat
  batchBar = t
  batchBar.n =
  batchBar.ref


# Inits if not
```

Figure 11: Deleting Custom Component Block



localhost:8888 says

code component deleted

```
s Secondary Model M

Noticed settings were saving with the wrong name so corre

v3 Update: Dec 24th 2021 - Somnai
```

Figure 12: Message for User after Deletion

in response. To achieve this functionality, the option menu of user created custom Component Blocks are designed to be different to default Component Blocks from markdown headings. The custom block option menu contains a “delete component” option, which will delete all relevant data of the selected custom component and remove it from the Table-of-Contents view (Figure 11). After user deleted a custom Component Block, the system will send a message to inform the user action as well (Figure 12). The deleting option for users provides a way to recall their interaction of creating new Component Blocks, thus enables more flexibility during user’s usage of the extension.

5.2.4. User Action 4: Changing Display Mode of Component Blocks

As mentioned in previous sections, the graphic view of Machine Learning Helper extension is formed with several nodes corresponding to Component Blocks in Table-of-Contents panel. It is not easy to form graph which can clearly indicated notebook structures especially those notebooks with machine learning problem solving procedures. For machine learning related notebooks, the workflow may be sequential, such as the problem-solving workflow in the study conducted by Jajoo and colleagues, where Preprocessing, Dataset Segmentation, Model Training and Application steps all formed a straight-line process, with each step depending on the results of the previous one (Jajoo, Jain, and Jangir 2023). Also, in the process of developing Chinese character recognizing model held by Xie et al., the essential steps, including character location and segmentation, dataset generation, AlexNet network construction and character recognition were permuted as a linear procedure which contains inter-step dependence (Xie, Yang, and Nie 2020). However, in some circumstances where model structures are required to be more convoluted, such as modern Diffusion models, scientists often perform complex machine learning procedures including both individual steps and sequential steps. The Controllable Light Enhancement Diffusion Model, developed by Yin et al., contains specific Brightness Control Module, which need to be embedded in the U-Net part of regular diffusion process (Yin et al. 2023). Also, even in regular machine learning process, there may be parallel steps for user to choose, such as different evaluation matrices, or different model configurations. Such modules indicating researching products or user options will likely be placed as an individual section in notebook and in this case, sequential arrangement will not be suitable.

To handle this problem, as mentioned before, the nodes in graph view of the Machine Learning Helper extension are equipped with two types of display modes, "Parallel" and "Sequential" to fit the user's need for a Component Block to be in a sequence with others, or an individual module standing alone. If one node is in "Sequential" mode, it will be linked with nearest node which is also "Sequential", otherwise, if the node is in "Parallel" mode, it will be linked to its parent node, which has already been determined while generating the initial hierarchy of component blocks in table-of-contents view happening in the earliest steps after activation of the extension. The default display mode of each node is set to "Sequential", but option menu under each Component Block in the Table-of-Contents view can provide user the ability to change display mode. Through this interaction, user can build their own preferred graph view for the model structure, thus achieve their needs to modify model structure following their own comprehensions.

6. Implementation

Except the time for preliminary user requirement collecting and prototype designing, the development process of this Machine Learning Helper extension spent approximately two months. The language used for developing was TypeScript, which can be comprehended as syntax-extended JavaScript supporting tighter integration with code editors. As introduced in Methodology part, React and yFiles were also used for building the graphics view of the extension. The developing process was not always in smooth progress, many coding-level problems were encountered by the developer and were managed to be solved through multiple approaches. The detailed implementation code could be found at the project's GitHub repository.

6.1. Storing and Loading Component Data

Data storage plays an important role in multiple web-based systems, especially those in computational notebooks, since tasks performed on computational notebooks are likely to be data-based, where large number of relevant information need to be stored to support complex data processing and analysis. In the Machine Learning Helper extension, to enable user to access their notes left before or navigate to specific cells, it is necessary to store some essential data within each Component Block, such as user's notes. Considering that this type of data usually occupies less amount of memory, in this extension, the data storing functionality is achieved through attributing data into user's local storage space for long-term storage. The local storage space has similar behaviour as hash table, where data is stored in terms of key and value pairs, in which both key and value should be string.

Generally, data contained in each Component Block includes component title, corresponding cell identifier of the component, and user notes, but the data need to be stored tend to vary among different types of Component Blocks. For default Component Blocks that generated from user's markdown cells, the essential data only includes notes and cell-related data under this Component Block because other information, such as component title or corresponding cell identifier, can all be directly obtained from the notebook contents hence storing these information strings can be waste of memory. For the custom Component Block created by users, on the other hand, component title and corresponding cell identifier should also be stored, since the component name is defined by user, which is unlikely to be found in original notebook contents.

In both of these two cases, the general steps of data storing in the extension are:

1. Create a space in local storage with identifier of specific Component Block as key.
2. Store relevant data into this space as value.

To effectively determine which information should be attributed to the storage space of which Component Block, a component ID was also assigned as an attribute of each Component Block, as will be discussed later in this section.

In the process of developing data storage functionality, challenges arose from an inherent characteristic of the local storage space if without proper setting of Component Block identifiers (keys of local storage), such as taking component titles as identifiers in the early stage of extension development. Each key in the local storage is uniquely associated with a single value to ensure the effectiveness of this data structure, but also turned out to be a limitation that may impede the development process. This is because in some scenarios where data collisions are likely to occur, this potentially of one-to-one correspondence may lead to data integrity issues. The developer considered two potential scenarios, first, when there are same markdown titles occurring in one file, it will likely to be navigation errors or notes attribution errors taking the premise of component titles as keys because with two or more Component Blocks with same title, the saved data of previous one will immediately be overwritten by the other one's data with same key. To solve this problem, the developer introduced component ID as an attribute to each Component Block, which is composed of component title and the component's location on Table-of-Content panel, which can provide unique and stable keys for component data. However, with this component ID as keys in local storage, another issue emerged in the development process: what if there are two Component Blocks with same title, but in different notebook files? There will still be collisions, as different notebook files still share same storage space, and the result will be relevant data from one Component Block obtained from one notebook turns out to show on a different block with same name on a different notebook file. To prevent such occurrences, the component ID of each Component Blocks was modified to the form of "component title" plus "component location of Table of Contents" plus "file name of this component's origin notebook". Through this modification, each component will find its unique correspondence in local storage data without collisions.

6.2. Alignment Between Component Blocks and Real Cells

As mentioned in the previous section, the data encapsulated within each Component Block contains cell-related information to enable cell navigation functionalities. To make these data useful for cell navigating, a procedure called components-cells-alignment should take place in the background processes of the extension, where some identifier of a specific cell corresponding to one Component Block is stored into local storage and will be used as reference to find the cell when user need to be navigated to corresponding cell of this Component Block. Finding the proper identifier for this alignment process tended to be a challenge in the implementation stage of this project. At the first stage, cell id provided in the notebook, which can be obtained from Jupyterlab's official API, was used to align a cell to its corresponding component. However, after testing it was found that the id of cells in Jupyter Notebook were generated dynamically after each launch of the notebook, which means the cell id will be different after each time user opens the notebook interface. This did not affect cell tracking of default Component Blocks much, because their data will also be reobtained after the launch of notebook. However, the scenario differed largely with respect to user-created custom components, as the corresponding cell data for these components were specified by the user and could not be derived directly from the raw notebook metadata. Consequently, it is imperative that the cell information should be meticulously stored alongside the component title, ID, and other relevant details. With cell id as the identifier of corresponding cell, these custom cells cannot find their corresponding cells anymore at next time because once the notebook rebooted, the id of all cells will dynamically change, making last time's cell id useless for navigation. To solve this problem, the developer aligns the contents of corresponding cells with Component Blocks instead of dynamic cell ids. This feature tended to be unique since there is unlikely that two cell have exact same contents, and to make this feature also stable, for risks still occur because user may change the cell contents at any time, an extra data saving step was added to the closing stage of the extension to ensure that the local storage retains a record of the most recently updated contents of the cell. To save user's memory space, the cell related information for custom components is stored along with other data such as component title and user notes, with component ID as the key to ensure the uniqueness and stability of each Component Block in the data saving process.

7. Evaluation

To assess the usability and functionality of our JupyterLab extension, a comprehensive user evaluation was conducted. This extension, designed to assist data exploration process within JupyterLab notebooks, was evaluated through qualitative methods to gather insights on its performance, ease of use, and overall impact on the user workflow. The evaluation was targeted at totally three participants, one of them (participant A) is a master student currently working on transition learning models in University of Shandong, China, and the others (participant B and C) are computer science majored undergraduate students in University of Nottingham, who were working on their own projects surrounding data analysis and machine learning. The strategy applied by the user evaluation was similar to questionnaires, each of the participants received totally eleven questions about user experience, feedback of specific functionalities, and space of improvement after performing a given machine learning task with this extension.

For the analysis stage of user evaluation, both Content Analysis and Thematic Analysis were implemented to find common patterns, frequent keywords and main themes of evaluation results from different participant. From the Content Analysis, for example, the keyword "notes" tended to be frequent in the favorable assessments from all participants, indicating the history retrieving through user notes indeed provide additional convenience for data analysis. However, the keyword "graphics", "style" and "diverse" occurred in negative comments of 2 to 3 of the participants, which may demonstrate that a large space for potential improvement. In the Thematic Analysis, the user evaluation went through several levels of encoding in order to find implicit patterns or indications under different comments from each participant. One common theme found in evaluations from all participant was the concept of "compaction of functionalities", which means to integrate functionalities of different fields into one system. From the user evaluations, participants met an agreement about "compaction of functionalities" can bring extra convenience for them, which also aligned with the contribution of the Machine Learning Helper extension. Through these data analysis strategies, the author can effectively obtain user preference, potential contributions of the project, and space for improvement of this extension, as will be demonstrated later.

To conclude from the analysis, all three evaluators found this Machine Learning Helper useful in both machine learning and general data analysis tasks. For example, by integrating model overview and history recording functionalities together, as said by participant A, the extension helped him to constraint almost all essential information in data processing in the notebook panels, saved much time and effort for him to seek help from outer tools. All three participants found the Table-of-Contents view of model structure overview clear and intuitive, and the navigation functions helped them to correctly track notebook cells they needed through finding the relevant Component Block. The extension helped him to quickly identify which part of the notebook contains which part of code cell or text explanations, and scroll to that content with the navigation function, as stated by participant B. Besides, by collecting headings in notebook content together, also stated by participant B, he can rapidly understand the whole model training procedure by just observing steps shown on the Table-of-Contents panel, instead of manually finding markdown headings which may scatter anywhere in the notebook. The noting function was also reflected as useful during the evaluation. According to participant C, adding notes tended to be especially helpful in iterative model parameters tuning process, where he must remember past parameter and model performances to find better parameter scopes. By keeping a note inside the Jupyterlab interface he can effectively track the patterns appeared from experiment history, which successfully increased his working efficiency.

From these user evaluations, potential limitations of the system were also discovered. From the perspective of participant A, for example, though the graphics view of the extension increased his understanding of the total workflow of the model his working on, it cannot provide much additional help to model comprehension other than the Table-of-Contents view. In order to further improve the intuitiveness of the graphics view, pointed by participant A, several detailed design options of the graphics display should be adopted, such as introducing different node shapes for different display mode. Also, more space should be allocated for user interactions on the graphics panel, such as allowing user to manipulate edges and position of nodes. Participant B also suggested that the nodes in graphics view should be supplemented with some side comments or descriptions for users to accelerate understanding process. Some correlations between the nodes in graphics view and notebook contents may be helpful, as participant B stated further, for example, to enable navigation to notebook cells by clicking the nodes, or even enable users to directly check and add notes under these graph nodes along with the notes made under Table-of-Contents view.

Other potential limitations reflected from participants were mainly about details of user interface. As mentioned by participant B, the left-side margins of Component Block titles tended to be less clear to represent hierarchical structures, it will be better, as he suggested, to use some visual symbols as level identifiers, or use different font size of the component title, or user different colours to represent parent-child relations instead of left margins. Also, participant C pointed that the identifier of an already-noted Component Block can be confusing in some circumstances. ”When I added notes to each heading, the title's text had been modified by adding (Note) word to the end of the original headings. I think this may cause some confusion because the original title has been changed. ”, he reflected under the relevant question in evaluation. He suggested adding a special symbol (like red *) to the headings after adding the notes instead of text identifiers, and it may be a better approach to track the event of users putting their mouse to that obvious symbol, and handle this event by making the Plugin to show a prompt saying that “This section contains a note”. Also, the expand and collapse functionality can be extended, as stated by participant A, to enable user to choose which level of headings to hide, instead of just keeping components obtained from first level headings.

8. Reflections

During the research and implementation of the dissertation project, the author has obtained considerable amount of valuable software development experiences. Hence, reflections on the whole project procedure is significant, for this is not only an opportunity to evaluate the achievements and limitations of the project but also a crucial exercise in self-evaluating how each project phase contributes to the author's professional growth and understanding. The following reflection sections considers a critical appraisal of the project, focusing on both the strategic decisions made and the implementation processes, the personal progresses and areas for improvements of the author, and also the broader context of the project, particularly focusing on the legal, social, ethical, and professional dimensions, and discusses their significance to the work conducted.

8.1. Project Managing

The project procedure was managed using effective timeline-based approaches. A Gantt chart was created from the initial stage of the project, containing initial plans for research and implementation stages, approximately 1 week was planned to be left blank for final checking and handling unexpected problems. Meanwhile, several milestones, such as user requirements collections and analysis, project prototype design, and code-level implementations were set as deadlines to ensure the progress of project. This time scheduling strategy tended to be effective for project proceeding in the initial stage. However, as some challenges emerged, the timeline schedule were forced to be modified.

The main challenge encountered by the author was that during the research process, the project topic was forced to change for totally twice. The first change of topic was from ”AI-powered Artwork Generation Assistant” to ”General Machine Learning Assistant”, due to the compatibility of the author's local device to large generative models, and lack of surrounding potential users of large generative models. After the change of project direction, as shown in the timeline (Figure 13), user interviews was collected again to fit the new research topic, and new hypothesis was established based on these new interview results. The second modification of project topic tended to be a refinement based on original topic without changing basic research and implementation directions, thus user requirements obtained in previous interviews can be reused to obtain a new set of hypothesis and design and implementation objectives, which were determined to be the objectives for the Machine Learning Helper extension introduced in this paper.

With the final dissertation topic determined, design and implementation process were launched with efficiency. The software libraries found in previous researches tended to be useful for Jupyterlab extensions as well, for developing Jupyterlab-based applications was included in initial plans for this project. The prototyping works were accomplished with the tool Figma, which is known to be a professional designing application for producing intuitive prototypes for software projects. With these prototypes as references, the implementation steps took about one month and a half, which were less than expected. The implementation steps were clearly divided into implemention of different functionalities, which helped organizing then author's pace of development and accelerate total progress. Before user evaluations in the last project stage, a test targeted at different ways of user interactions was performed to the extension to ensure the stability of system. Overall, except for unforeseen contingencies that led to managerial oversights, the scheduling of the entire plan was sufficient to ensure the timely completion of the project, while also allowing adequate space for testing and user evaluation.

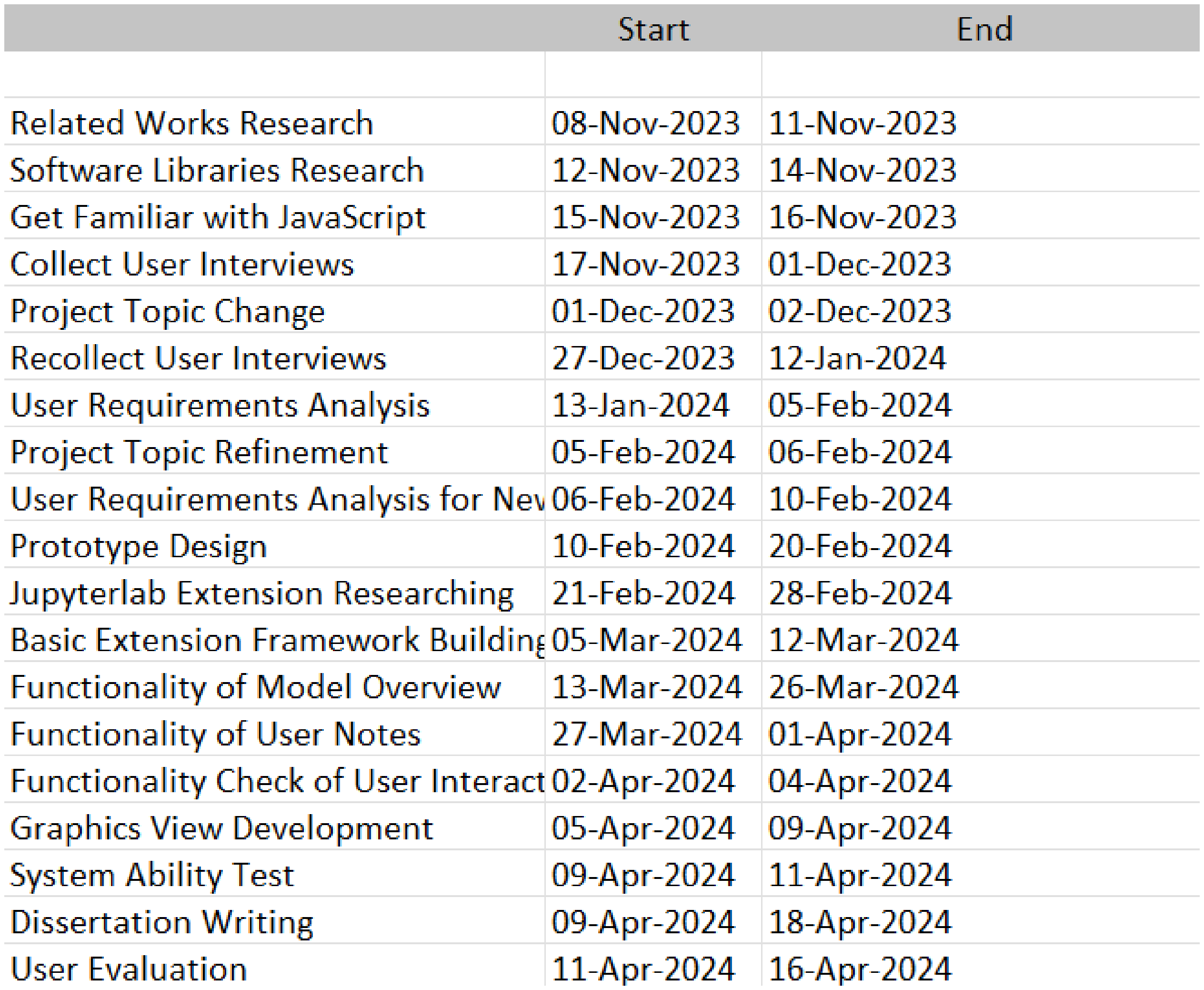
8.2. Contributions and Personal Reflections

The project, as an effective approach for assisting data analysts or machine learning scientists in iterative data exploring works, integrates both model overviews and experiment histories retrieval in single extension system. This feature was scarcely observed in prior studies, and has been demonstrated through user evaluations to offer enhanced convenience for targeted users.

Apart from the contributions, the project offered the author valuable software development experiences. From both achievements obtained and challenged encountered in the research and implementation process, the author established several personal reflections:

1. The significance of clear project ideas.
2. The significance of scheduling in coding process.

For personal reflections, the author first found that it is important to have a clear project idea prior to the project planning process. With a clearer self-defined project idea, the developer of the project can consider implementation steps with more details, thus find possible pain points, unexpected conditions or challenges in the following stages of the project. When planning the project timeline, the author of this paper had few own project ideas and had to schedule for unfamiliar topics with limited understanding, which led to the topic modifications in the later stages. Besides, steps scheduling can play significant role in coding process. During the coding stage of the project, the author first started to code basic framework of the extension, and gradually fulfill different



	Start	End
Related Works Research	08-Nov-2023	11-Nov-2023
Software Libraries Research	12-Nov-2023	14-Nov-2023
Get Familiar with JavaScript	15-Nov-2023	16-Nov-2023
Collect User Interviews	17-Nov-2023	01-Dec-2023
Project Topic Change	01-Dec-2023	02-Dec-2023
Recollect User Interviews	27-Dec-2023	12-Jan-2024
User Requirements Analysis	13-Jan-2024	05-Feb-2024
Project Topic Refinement	05-Feb-2024	06-Feb-2024
User Requirements Analysis for Nev	06-Feb-2024	10-Feb-2024
Prototype Design	10-Feb-2024	20-Feb-2024
Jupyterlab Extension Researching	21-Feb-2024	28-Feb-2024
Basic Extension Framework Building	05-Mar-2024	12-Mar-2024
Functionality of Model Overview	13-Mar-2024	26-Mar-2024
Functionality of User Notes	27-Mar-2024	01-Apr-2024
Functionality Check of User Interact	02-Apr-2024	04-Apr-2024
Graphics View Development	05-Apr-2024	09-Apr-2024
System Ability Test	09-Apr-2024	11-Apr-2024
Dissertation Writing	09-Apr-2024	18-Apr-2024
User Evaluation	11-Apr-2024	16-Apr-2024

Figure 13: Project Timeline

embedded functionalities, which tended to be efficient approaches. However, for the graphics view of the extension, the author stated coding for this functionality after the implementation of all user interactions was finished, which hindered the graphics layout building because several previously built code for user interactions must be changed to fit the demands for new graphics view. These experiences turned out to be of great significance to the author's future development in the field of software development.

8.3. LSEPI Considerations

In this dissertation project, the consideration of Laws, Social, Ethical, and Professional Issues (LSEPI) is important not only for ensuring compliance with legal standards but also for fostering trust and safety among users. In this project, since data collections from user questionnaires were employed, adherence to relevant laws, such as data protection regulations and intellectual property rights is fundamental to safeguard user interests. Furthermore, by addressing and avoiding social concerns such as possible uncomfortable comments for questions in user interviews, the developer can enhance the user accessibility of the extension, thereby reaching a wider and more diverse user base for further evaluation stage. Ethical considerations are less critical in this project, since there are no potential AI-based parts in this extension and the user interactions does not contain inadvertently harmful affects or consequences. Lastly, following professional standards can promote integrity and accountability in development practices of this work. Based on these considerations, integrating LSEPI into the the development procedure is essential for the sustainability and completeness of this project, ensuring the project to be socially responsible and ethically grounded.

In the research process of this work, user interviews and user evaluations were ensured to adhere relevant data protection laws, with strict protective approaches to avoid data and private information leakage. The user interviews and evaluations were based on pure machine learning based questions containing no contents with potential discrimination or other malicious comments, therefore had equal accessibility to all participants. All participants were informed about the contents of the questionnaire in advance and voluntarily participated in the data collection process after fully understanding the research purpose and procedures. In addition, during the analysis of data from user interviews and evaluations, the developer were subjected to stringent inspections from the participants to ensure the accuracy and fairness of data analysis, thereby preventing any potential manipulation of the data.

9. Summary

In summary, this dissertation presents a Jupyterlab extension for assisting iterative data exploration tasks, such as machine learning. Three main objectives are illustrated from analysis of user requirements: enabling intuitive model overview in computational notebook, enabling cell navigation for users in computational notebook, and enabling history recording functionalities in computational notebooks. These objectives are achieved through detailed implementation process, which employed professional software libraries such as Jupyterlab API, Lumino and yFiles. Also, additional functions were developed for extra user flexibility, such as creation of user custom model components, and graphics view, etc. User evaluation reflected the convenience this extension can bring to user's data exploration process, as well as some potential limitations and improvement space for further development. In conclusion, by integrating these functionalities into an extension panel that directly displays alongside the notebook panel, the extension has been demonstrated to effectively help address various challenges encountered by users during data exploration.

Bibliography

Abdul, Ashraf et al. (2018). “Trends and Trajectories for Explainable, Accountable and Intelligible Systems: An HCI Research Agenda”. In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. CHI ’18. New York, NY, USA: Association for Computing Machinery, pp. 1–18. ISBN: 9781450356206. DOI: 10.1145/3173574.3174156. URL: https://doi.org/10.1145/3173574.3174156.

Batch, Andrea and Niklas Elmqvist (2018). “The Interactive Visualization Gap in Initial Exploratory Data Analysis”. In: *IEEE Transactions on Visualization and Computer Graphics* 24.1, pp. 278–287. DOI: 10.1109/TVCG.2017.2743990.

Beth Kery, Mary and Brad A. Myers (2017). “Exploring exploratory programming”. In: *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pp. 25–29. DOI: 10.1109/VLHCC.2017.8103446.

Chattopadhyay, Souti et al. (2020). “What’s Wrong with Computational Notebooks? Pain Points, Needs, and Design Opportunities”. In: *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. CHI ’20. New York, NY, USA: Association for Computing Machinery, pp. 1–12. ISBN: 9781450367080. DOI: 10.1145/3313831.3376729. URL: https://doi.org/10.1145/3313831.3376729.

Cockburn, Andy, Amy Karlson, and Benjamin B. Bederson (Jan. 2009). “A review of overview+detail, zooming, and focus+context interfaces”. In: *ACM Comput. Surv.* 41.1. ISSN: 0360-0300. DOI: 10.1145/1456650.1456652. URL: https://doi.org/10.1145/1456650.1456652.

Eichelberger, Holger (2003). “Nice class diagrams admit good design?” In: *Proceedings of the 2003 ACM Symposium on Software Visualization*. SoftVis ’03. New York, NY, USA: Association for Computing Machinery, 159–ff. ISBN: 1581136420. DOI: 10.1145/774833.774857. URL: https://doi.org/10.1145/774833.774857.

Head, Andrew et al. (2019). “Managing Messes in Computational Notebooks”. In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. CHI ’19. New York, NY, USA: Association for Computing Machinery, pp. 1–12. ISBN: 9781450359702. DOI: 10.1145/3290605.3300500. URL: https://doi.org/10.1145/3290605.3300500.

Hill, Charles et al. (2016). “Trials and tribulations of developers of intelligent systems: A field study”. In: *2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pp. 162–170. DOI: 10.1109/VLHCC.2016.7739680.

Jajoo, Palika, Mayank Kumar Jain, and Sarla Jangir (2023). “Plant Disease Detection Over Multiple Datasets Using AlexNet”. In: *Proceedings of the 4th International Conference on Information Management & Machine Intelligence*. ICIMMI ’22. New York, NY, USA: Association for Computing Machinery. ISBN: 9781450399937. DOI: 10.1145/3590837.3590838. URL: https://doi.org/10.1145/3590837.3590838.

Kang, DaYe et al. (2021). “ToonNote: Improving Communication in Computational Notebooks Using Interactive Data Comics”. In: *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. CHI ’21. New York, NY, USA: Association for Computing Machinery. ISBN: 9781450380966. DOI: 10.1145/3411764.3445434. URL: https://doi.org/10.1145/3411764.3445434.

Kery, Mary Beth, Amber Horvath, and Brad Myers (2017). “Variolite: Supporting Exploratory Programming by Data Scientists”. In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. CHI ’17. New York, NY, USA: Association for Computing Machinery, pp. 1265–1276. ISBN: 9781450346559. DOI: 10.1145/3025453.3025626. URL: https://doi.org/10.1145/3025453.3025626.

Kery, Mary Beth, Bonnie E. John, et al. (2019). “Towards Effective Foraging by Data Scientists to Find Past Analysis Choices”. In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. CHI ’19. New York, NY, USA: Association for Computing Machinery, pp. 1–13. ISBN: 9781450359702. DOI: 10.1145/3290605.3300322. URL: https://doi.org/10.1145/3290605.3300322.

Kery, Mary Beth and Brad A. Myers (2018). “Interactions for Untangling Messy History in a Computational Notebook”. In: *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pp. 147–155. DOI: 10.1109/VLHCC.2018.8506576.

Ko, Amy J., Htet Aung, and Brad A. Myers (2005). “Eliciting design requirements for maintenance-oriented IDEs: a detailed study of corrective and perfective maintenance tasks”. In: *Proceedings of the 27th International Conference on Software Engineering*. ICSE ’05. New York, NY, USA: Association for Computing Machinery, pp. 126–135. ISBN: 1581139632. DOI: 10.1145/1062455.1062492. URL: https://doi.org/10.1145/1062455.1062492.

Kuhn, Adrian, David Erni, and Oscar Nierstrasz (2010). “Embedding spatial software visualization in the IDE: an exploratory study”. In: *Proceedings of the 5th International Symposium on Software Visualization*. SOFTVIS ’10. New York, NY, USA: Association for Computing Machinery, pp. 113–122. ISBN: 9781450300285. DOI: 10.1145/1879211.1879229. URL: https://doi.org/10.1145/1879211.1879229.

Lau, Sam et al. (2020). “The Design Space of Computational Notebooks: An Analysis of 60 Systems in Academia and Industry”. In: *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pp. 1–11. DOI: 10.1109/VL/HCC50065.2020.9127201.

Li, Xingjun et al. (Mar. 2023). “EDAssistant: Supporting Exploratory Data Analysis in Computational Notebooks with In Situ Code Search and Recommendation”. In: *ACM Trans. Interact. Intell. Syst.* 13.1. ISSN: 2160-6455. DOI: 10.1145/3545995. URL: https://doi.org/10.1145/3545995.

Liu, Eric S., Dylan A. Lukes, and William G. Griswold (Apr. 2023). “Refactoring in Computational Notebooks”. In: *ACM Trans. Softw. Eng. Methodol.* 32.3. ISSN: 1049-331X. DOI: 10.1145/3576036. URL: https://doi.org/10.1145/3576036.

Mathisen, Andreas et al. (2019). "InsideInsights: Integrating data-driven reporting in collaborative visual analytics". In: *Computer Graphics Forum.* Vol. 38. 3. Wiley Online Library, pp. 649–661.

Perez, Fernando and Brian E. Granger (2007). "IPython: A System for Interactive Scientific Computing". In: *Computing in Science Engineering* 9.3, pp. 21–29. DOI: 10.1109/MCSE.2007.53.

Rule, Adam, Ian Drosos, et al. (Nov. 2018). "Aiding Collaborative Reuse of Computational Notebooks with Annotated Cell Folding". In: *Proc. ACM Hum.-Comput. Interact.* 2.CSCW. DOI: 10.1145/3274419. URL: https://doi.org/10.1145/3274419.

Rule, Adam, Aurélien Tabard, and James D. Hollan (2018). "Exploration and Explanation in Computational Notebooks". In: CHI '18. New York, NY, USA: Association for Computing Machinery, pp. 1–12. ISBN: 9781450356206. DOI: 10.1145/3173574.3173606. URL: https://doi.org/10.1145/3173574.3173606.

Sandve, Geir Kjetil et al. (Oct. 2013). "Ten Simple Rules for Reproducible Computational Research". In: *PLOS Computational Biology* 9.10, pp. 1–4. DOI: 10.1371/journal.pcbi.1003285. URL: https://doi.org/10.1371/journal.pcbi.1003285.

Subramanian, Krishna et al. (2019). "Supporting Data Workers To Perform Exploratory Programming". In: *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems.* CHI EA '19. New York, NY, USA: Association for Computing Machinery, pp. 1–6. ISBN: 9781450359719. DOI: 10.1145/3290607.3313027. URL: https://doi.org/10.1145/3290607.3313027.

Wang, April Yi, Will Epperson, et al. (2022). "Diff in the Loop: Supporting Data Comparison in Exploratory Data Analysis". In: *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems.* CHI '22. New York, NY, USA: Association for Computing Machinery. ISBN: 9781450391573. DOI: 10.1145/3491102.3502123. URL: https://doi.org/10.1145/3491102.3502123.

Wang, April Yi, Dakuo Wang, Jaimie Drozdal, Xuye Liu, et al. (2021). "What Makes a Well-Documented Notebook? A Case Study of Data Scientists' Documentation Practices in Kaggle". In: *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems.* CHI EA '21. New York, NY, USA: Association for Computing Machinery. ISBN: 9781450380959. DOI: 10.1145/3411763.3451617. URL: https://doi.org/10.1145/3411763.3451617.

Wang, April Yi, Dakuo Wang, Jaimie Drozdal, Michael Muller, et al. (Jan. 2022). "Documentation Matters: Human-Centered AI System to Assist Data Science Code Documentation in Computational Notebooks". In: *ACM Trans. Comput.-Hum. Interact.* 29.2. ISSN: 1073-0516. DOI: 10.1145/3489465. URL: https://doi.org/10.1145/3489465.

Wenskovitch, John et al. (2019). "Albireo: An Interactive Tool for Visually Summarizing Computational Notebook Structure". In: *2019 IEEE Visualization in Data Science (VDS)*, pp. 1–10. DOI: 10.1109/VDS48975.2019.8973385.

Xie, Songhua, Hailiang Yang, and Hui Nie (2020). "Design of Chinese Character Recognition Based on AlexNet Convolution Neural Network". In: *Proceedings of the 2020 3rd International Conference on Artificial Intelligence and Pattern Recognition.* AIPR '20. New York, NY, USA:

Association for Computing Machinery, pp. 68–73. ISBN: 9781450375511. DOI: `10.1145/3430199.3430230`. URL: `https://doi.org/10.1145/3430199.3430230`.

Yin, Yuyang et al. (2023). "CLE Diffusion: Controllable Light Enhancement Diffusion Model". In: *Proceedings of the 31st ACM International Conference on Multimedia*. MM '23. New York, NY, USA: Association for Computing Machinery, pp. 8145–8156. ISBN: 9798400701085. DOI: `10.1145/3581783.3612145`. URL: `https://doi.org/10.1145/3581783.3612145`.

Yoon, YoungSeok, Brad A. Myers, and Sebon Koo (2013). "Visualization of fine-grained code change history". In: *2013 IEEE Symposium on Visual Languages and Human Centric Computing*, pp. 119–126. DOI: `10.1109/VLHCC.2013.6645254`.

Appendices

User Interview Questions

1. Do you prefer using an editor like PyCharm or Jupyter Notebook for machine learning tasks?
2. What challenges do you find when using Jupyter Notebook for machine learning tasks, and what additional features do you think Jupyter should have?
3. What steps do you generally follow in the machine learning process, and are there any steps that particularly bother you?
4. What specific characteristics of the mentioned frustrating steps make them bothersome to you?
5. When working on machine learning tasks, do you prefer modifying existing code found online or starting from scratch? What are your reasons?
6. When you come across machine learning code in Jupyter notebooks written by others or when someone modifies your code, do you find it challenging to understand new workflows? What specifically makes it difficult?
7. If you have experience writing code for machine learning tasks, do you regularly record or save data in your notebooks or code (e.g., code progress, run results)? If yes, what data do you typically save, and if not, does the absence of this data bother you?
8. Consider the Cats and Dogs classification you were asked to solve with the given solution notebook file. Are there any parts of the code mentioned above that you find confusing, or do you think the author needs more explanation? If so, what specific information would you like clarification on (e.g., package functionality, code segment purpose, cell interactions)?
9. Consider the Cats and Dogs classification you were asked to solve with the given solution notebook file. Are there any problems in the hyper-parameter tuning process? What do you think is the reason causing these problems?
10. In your machine learning process using Jupyter, what additional help or features do you wish Jupyter could provide?

User Manual

1. Installing the extension

   The Machine Learning Helper is a Jupyterlan extension to assist data analyst's exploring process by both respresenting an intuitive model structure overview and providing exploration history retrieval. To install the extension into your Jupyterlab, the first step is to enter your Anaconda Prompt, navigate to where the extension folder is, and install the extension by running "pip install myextension".

If above appraches doesn't work, you can access through developer mode by user "jlpm", "jlpm build" and "pip install -e ." commands.

2. Open extension in Jupyterlab

   After the installation, you can launch your Jupyterlab and start using the extension. The extension will be launched by two ways. First, you can enter the command palette of Jupyterlab in "View" menu, and type "ML Helper Panel", the extension will work after you execute this inner command. Also, you can click the "ML Helper" button on the toolbar of notebook view. After clicking this button, an extension panel will appear for further user operations.

3. Switching between different notebook files

   If you just switched your focus on another notebook file. To let the extension show model overview of the new file, you should click the "refresh" button to update the extension panel.

4. Navigating to Notebook Cells

   Through left-clicking one of the titles (Component Blocks) in the list view of extension panel, a menu will pop up to inform you next available actions. One of these options is to navigate to the notebook cell containing this component you chose. Click the "navigate to code location" and the extension will scroll the notebook to the location of the corresponding cell.

5. Taking Notes under Component Block

   One option in the pop-up menu is to open the notes area of selected Component Block. By clicking this option, you can take quick notes to record history data such as previous performed experiments.

   By clicking "+" button you can add a new note in current noting area. By clicking "-" button you can delete the newest note you have added. By clicking "save" button you can save your notes into your local memory. If you click "X" button for closing the noting area, the notes you made will still be saved.

   After you have created a note for the Component Block, the title of the block will be added with an identifier "(Noted)", indicating this block contained your notes.

6. Collapsing and Expanding Component Blocks

   In the option menu, on option allows you to expand the Component Block to show other Component Block that corresponding to all markdown titles with lower heading levels under the selected Component Block.

   The initial states of the Component Blocks are all set to "expanded", and you can collapse them, which means to hide all Component Blocks corresponding lower-level headings under selected block.

Also, you can collapse all components by clicking the "collapse all" button on toolbar area, which will only show components with level-1 markdown headings.

7. Graphic Views and Adjusting Display Mode

   In parallel with the list-like table of contents view, another graphics view is used to show model structure.

   The graphics view is composed of all Component Blocks originated from markdown titles with their own display modes. You can find these components in the table of contents view and change their display mode to what you prefer.

   "Parallel" mode means this component will be shown individually as a module in the graphics view, linking with its parent node (closest previous component that has higher heading level in markdown). "Sequential" mode means this component will be shown as a part of main model workflow, linking with closest components to form a sequential layout. Trough clicking this option of changing display mode, you can switch between these two displaying styles.

8. Adding Custom Model Components

   By right-clicking the notebook cell you choose, you can add a custom Component Block to the table of contents panel through "Add Code Component (Extension)".

   After choosing the option, a query will come out for the name of new component.After naming the component you created, the custom component will be shown on the table of contents view, under the location of its nearest Markdown Component.

   The newly added custom component can also enable the navigation and noting area functionalities, etc. Also, you can delete the newly added custom Component Block in case of mistakenly adding components you don't want, by finding corresponding option in the pop-up menu.

   After the deletion of custom component, you will receive a message.

User Evaluation Questions

1. To what extent do you find the model structure overview clear and intuitive? (rate from 1 - 5)
2. To what extent do you find the graphics view making model structure easier to understand? (rate from 1 - 5)
3. To what extent do you find the navigating function useful? (rate from 1 - 5)
4. To what extent do you find the noting function useful? (rate from 1 - 5)

5. To what extent do you find the display mode changing function for graphics view useful? (rate from 1 - 5)
6. Is there any space for improvement to the conciseness or intuitiveness of model structure representing of the extension?
7. Are there any advice for the UI design of this extension?
8. Is there any space for improvement to the displaying style of graphics view?
9. When using the extension to complete the task, was the extension making your process faster, or making you feel more comfortable to complete the task?
10. Overall, how much do you think this extension will help you in performing iterative data exploring tasks such as machine learning?
11. Do you have any suggestions for future refinement and maintenance works of this extension?