Computational Experiment Comprehension using Provenance Summarization

Nichole Boufford
University of British Columbia
Vancouver, British Columbia, Canada
ncbouf@student.ubc.ca

Joseph Wonsil
University of British Columbia
Vancouver, British Columbia, Canada
jwonsil@student.ubc.ca

Adam Pocock
Oracle Labs
Burlington, Massachusetts, USA
adam.pocock@oracle.com

Jack Sullivan
Oracle Labs
Burlington, Massachusetts, USA
jack.t.sullivan@oracle.com

Margo Seltzer
University of British Columbia
Vancouver, British Columbia, Canada
mseltzer@cs.ubc.ca

Thomas Pasquier
University of British Columbia
Vancouver, British Columbia, Canada
tfjmp@cs.ubc.ca

ABSTRACT

Scientists use complex multistep workflows to analyze data. However, reproducing computational experiments is often difficult as scientists' software engineering practices are geared towards the science, not the programming. In particular, reproducing a scientific workflow frequently requires information about its execution. This information includes the precise versions of packages and libraries used, the particular processor used to perform floating point computation, and the language runtime used. This can be extracted from data provenance, the formal record of what happened during an experiment. However, data provenance is inherently graph-structured and often large, which makes interpretation challenging. Rather than exposing data provenance through its graphical representation, we propose a textual one and use a large language model to generate it. We develop techniques for prompting large language models to automatically generate textual summaries of provenance data. We conduct a user study to compare the effectiveness of these summaries to the more common node-link diagram representation. Study participants are able to extract useful information from both the textual summaries and node-link diagrams. The textual summaries were particularly beneficial for scientists with low computational expertise. We discuss the qualitative results from our study to motivate future designs for reproducibility tools.

CCS CONCEPTS

• **Human-centered computing → User studies**; *Interaction paradigms*; • **Computing methodologies → Natural language generation**; • **Information systems** → *Data provenance.*

KEYWORDS

Provenance, Reproducibility, User Study, Text Generation

ACM Reference Format:
Nichole Boufford, Joseph Wonsil, Adam Pocock, Jack Sullivan, Margo Seltzer, and Thomas Pasquier. 2024. Computational Experiment Comprehension using Provenance Summarization. In *Proceedings of 2023 ACM Conference on Reproducibility and Replicability (ACM REP '24).* ACM, New York, NY, USA, 19 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Historically, researchers and scientists have recorded experimental procedures in lab notebooks. These notebooks should contain enough detail to allow another researcher to reproduce an experiment. However, this paper-based approach does not translate well to today's computational world. Computational lab notebooks (e.g., Jupyter [23]) seem an obvious solution. They automate experiment recording, making them vastly superior to paper-based approaches. However, computational lab notebooks still fall short of supporting automated reproducibility [42]. Even with access to all the experiment materials such as analysis scripts and data sources, it is non-trivial to reproduce or even understand computational experiments [42, 53, 54]. This is problematic as scientists cannot build upon prior work without a solid understanding of the computational experiment they are trying to reproduce. These challenges contribute to what is known as the reproducibility crisis in science [4].

Often, researchers reproduce experiments to establish a baseline from which they can extend existing research. Researchers use many tools that help with mechanical experiment reproducibility such as code repositories [10], dataset repositories [29, 31] and computational environment trackers [14, 43]. However, just because a researcher can run an experiment does not mean they have enough knowledge to build upon it. Conversely, if a researcher cannot immediately reproduce an experiment result, they will need a better understanding of the experiment to determine what is missing or not working correctly. The information they need to develop this understanding also might not exist in the code and data.

Data provenance addresses the issue of incomplete experiment tracking [40]. Provenance is metadata describing the history of data objects [9]. Although data provenance addresses the mechanical problem in reproducibility by recording the missing pieces a user needs to run an experiment, it falls short on helping the user understand the pipeline they are reproducing. Tools that use data provenance typically present it visually as a directed acyclic graph that represents the relationships between data objects. These objects are often scripts and datasets, whose attributes might contain version information, and the relationships describe dependence or causality. Unfortunately, since provenance graphs are large and complicated, even experienced software engineers find interpreting these graphs challenging [6].

Provenance systems typically store provenance data in a machine-readable format that must be transformed for presentation to users. In practically all prior work, the transformation produced a node-link diagram (Fig. 1) [5, 21, 25, 30, 41, 46]. Large graphs are difficult for humans to understand [33]. They often contain more information than a human can store in their working memory and thus impose too high a cognitive load.

Imagine a new graduate student who is starting to work on a problem that a previous student had worked on. To get started, they need to reproduce the student's experiment, and then they can begin to extend the work. The previous student used version control, a data repository, and an environment manager, and still, the new student cannot seem to get the same results. Although their scripts run without errors, the new student is not sure if they ran the scripts in the correct order or if the code has changed since the previous student reported their results. Even if the student was able to reproduce the correct result through trial and error, they are in no position to build upon this work, because they do not have a good understanding of the workflow.

Fortunately, the previous student collected provenance during their last experiment execution. However, the provenance is not immediately helpful since it is only machine-readable. They use a graph database tool to view the provenance, but there are hundreds of nodes! Since they cannot understand the provenance data, they are no closer to understanding the workflow than before.

This scenario is all too common and shows that data provenance, on its own, is not a complete solution. Although the answer to "Which data preprocessing script did I use to create the input data for this trained model?" is in the provenance data, interpreting the data to find this information is nontrivial. Some researchers evaluated different summarization and presentation techniques [6, 44], but these studies all assume that the right solution requires exposing the graph-structured representation of the provenance to the user. We question that assumption; our goal is to facilitate a user's understanding of the experiment. We hypothesize that explaining what a provenance graph represents is a better approach to achieving that goal.

We present a text-based provenance summarization technique and evaluate it with a user study. Our text-based provenance summarization technique is based on the observation that while experiment development is an iterative and complicated process, the ultimate experiment execution follows a relatively simple and logical procedure. As such, we prefer to directly express this sequence rather than illustrating it with a complicated graph structure. Traditional lab notebooks describe experimental control flow using natural language; we do the same. For our user study, we use a large language model (LLM) to generate a natural language summary of a provenance graph. We find that users with less computational expertise prefer the text-based explanation as it is more familiar and less overwhelming than the node-link diagram. Our qualitative findings suggest several areas of future work to improve provenance summarizations in the service of experimental reproducibility.

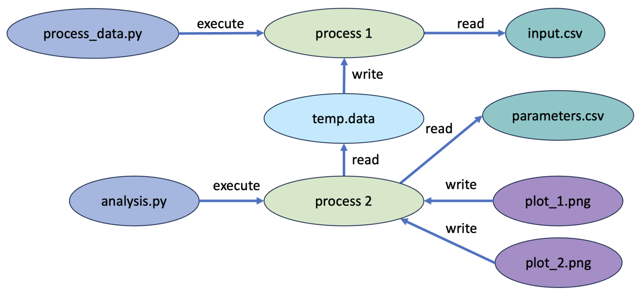


Figure 1: Simple provenance graph displayed as a node link diagram. Process 1 executes the process_data.py script. This script reads input.csv and writes to temp.data. Process 2 executes the second script, analysis.py, which reads temp.data and parameters.csv and writes to two image files, plot_1.png and plot_2.png.

Contributions

- We develop techniques for using large language models to produce summaries from provenance graphs.
- We demonstrate that these techniques produce high quality summaries.
- We conduct a user study to compare text-based provenance summaries and more typical node-link representations.
- We show that participants can extract useful information from both the textual summaries and node-link diagrams.

Our code, datasets, and study materials are publicly available (further details in Appendix A).

2 BACKGROUND

We develop and evaluate textual provenance summarizations in the context of reproducibility. We use the definition of reproducibility from the National Academies of Science, Engineering and Medicine: "Reproducibility is obtaining consistent results using the same input data, computational steps, methods, code, and conditions of analysis" [34].

2.1 Provenance

Data provenance describes when data objects were created, when they were modified and their history of ownership [9]. In experimental workflow tracking, this translates into recording the details about dependencies, creation, and modification of experiment code, data, and outputs. While many tools designed for software engineering capture version information, data provenance augments such tools by capturing an execution record and the relationships among different objects involved in a workflow. Prior work uses provenance data in experimental tracking systems to recreate computational environments [13, 41] and record interactions between applications and file systems [18].

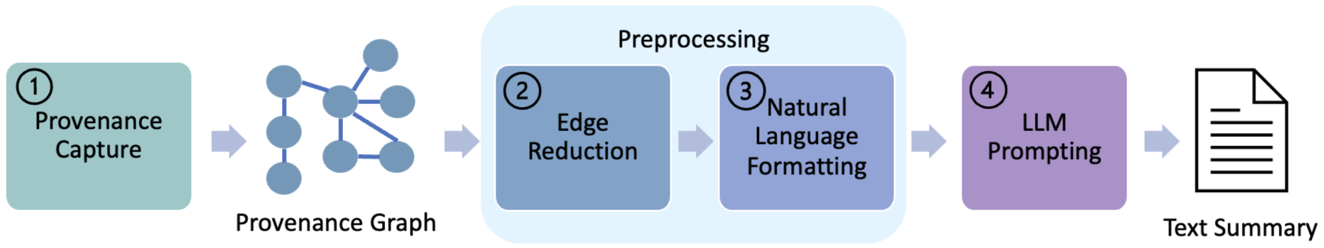


Figure 2: Steps to create a text summary of a computational experiment.

Most provenance applications, including those used for experiment and code management, present provenance as a node-link diagram [5, 21, 25, 30, 41, 46]. However, provenance graphs can contain hundreds of elements, even for small tasks such as running a computational notebook. Research shows that graphs containing more than 50 to 100 elements are hard to interpret [59] as they cannot fit in a human's working memory [33]. Alternative provenance visualizations [6, 44] have failed to see meaningful adoption (see §6 for further discussion). Given that large graphs are hard to understand, we propose natural language text summaries as an alternative. Our intuition is that scientific experiments follow a logical control flow that we can describe using natural language. We know that scientists frequently read papers, lab reports and procedural documents, so we hypothesize that they might find a written format easier to understand and a better way to explain how to reproduce a computational experiment. While it was previously impractical to generate these text summaries manually, we now can generate them automatically using large language models.

2.2 Summarization using Generative AI

Recent work in generative artificial intelligence shows that large language models (LLMs) are able to effectively summarize large quantities of text [17, 62]. Users interact with LLMs using a prompting interface where they use natural language to instruct the model to answer a question or complete a task [63]. The input to an LLM is a natural language expression, called a *prompt*. The model outputs a response to that prompt, also in natural language. If the task is summarization, the user also provides the document as part of the prompt.

Many prior works uncover limitations of LLM summarization [22, 28, 47, 50]. It requires careful prompting to generate useful responses [60]. LLM responses are sometimes verbose, redundant, and unclearly organized. Additionally, with current generative AI models, we cannot guarantee response correctness [8]. Lastly, LLMs can process a limited amount of text at one time. The *context window* is the maximum amount of text a model can process. The context consists of one or more prompt and model response pairs, similar to a conversation. Since our prompt contains an instruction and a provenance log, our instruction, provenance log and the model response together must be smaller than the context window. The context window is measured in *tokens*; for GPT models, a token is approximately equal to 4 characters. At the time of this study, the largest context window available for GPT-4 is 8000 tokens. This means that the context is limited to approximately 32000 characters.

3 TEXT SUMMARIZATION

We generate several high-quality summaries using GPT-4 [37] as a proof of concept for our user study. Fig. 2 shows the sequence of data transformations involved in producing text summaries of computational experiments. We first run an experiment while recording provenance ①. We then preprocess the provenance data ② ③ and then use the GPT-4 model from OpenAI [37] to generate the summaries for our user study ④. The LLM-generated summaries should contain 1) enough information that a user can understand the experiment well enough to reproduce it and 2) no unnecessary or false information. We outline further goals and expectations in §3.3.

① **Provenance capture** We developed a system level provenance collection tool to capture provenance during experiments [7]. System level provenance describes data at the granularity of system calls, files and processes. We wrote our own tool because most existing system provenance collection tools have a large installation overhead [32, 39]. Our tool uses eBPF [2], a Linux framework that allows users to monitor operating system events without modifications to the kernel. Previous work that uses eBPF for provenance capture mainly focuses on security [27, 45], whereas our tool only captures the information necessary for computational experiment reproducibility. We call the provenance data captured by our system-level tool a *provenance log*.

3.1 Data Preprocessing

For most LLMs, including GPT-4 [37], the context window is limited. Since many of our provenance logs are larger than the context window, we need a more concise representation. Additionally, the provenance log we get from the data capture stage is a machine-readable JSON file. The JSON provenance format is long and verbose, which increases the context size. Previous work shows that LLM response quality degrades and loses information around the middle of a document when the context is too long [28].

We reduce log size by removing duplicate edges and converting the JSON log to natural language. We perform both the edge reduction and the natural language formatting automatically using Python scripts. Both of these methods also provide the benefit of reducing noise in the input to the LLM. Duplicate edge reduction helps prevent edges from being erroneously categorized as more

```
{"type":"Entity","id":"4554","annotations":{"inode_inum":"4554","uid":"0","path":"/
    root/usr/bin/python3.11"}}
{"type":"Activity","id":"227899","annotations":{"pid":"227899"}}
{"type":"Used","to":"4554","from":"227899","annotations":{"operation":"execute","
    datetime":"2023-10-21 20:01:55:427"}}
```

(a) Example machine-readable provenance log.

```
2023-10-21 20:01:55:427 process with pid:227899 executed file:/root/usr/bin/python3.11
```

(b) Provenance log converted to natural language format.

Figure 3: A provenance log in machine-readable JSON format (Fig. 3a) is converted to natural language format (Fig. 3b). The machine-readable JSON format size is 88 tokens and the natural language log size is 30 tokens.

important than they are, and the natural language format aligns more with an LLM's training corpus than the JSON output of our system-level provenance collection tool.

② **Edge Reduction** The operating system sometimes produces many system events for a single user action. For example, if a user is modifying a file using a text editor, the operating system might execute multiple writes in a row. Our provenance collection system will record each write event as an edge. Conceptually, there is no difference between a single large write event and many consecutive small write events. Therefore, we use a simplified version of edge aggregation described by Xu et al. [57] to remove repeated edges from the graph. This reduced log sizes by 43-53% for the logs in our study.

③ **Natural Language Formatting** Through empirical experiments, we found that converting the JSON logs into natural language sentences improved both log size and summary quality. The new log format follows a simple natural language structure where a short sentence describes each relationship in the graph. For example, when a process writes to a file, this is recorded in the log as a JSON object for each of: a process node, a file node, and an edge that connects the two nodes. We simplify this relationship as "Process p writes to file f", where p and f are the identifiers for the process and the file. Fig. 3 shows an example of the natural language log format conversion. Since we can enumerate all the possible relationship types in our provenance graph, we can generate a mapping of sentences to relationships in the provenance graph. We can then automatically generate a log in natural language format, filling in the blanks with values from the provenance data. This format produces higher quality summaries than the machine-readable log, using the evaluation criteria in §3.3. The natural language format reduces the study provenance log size by an additional 58-63%. In combination these two techniques reduce the logs to between 17 and 24% of their original size. The code for generating the natural language format is publicly available (details in Appendix A).

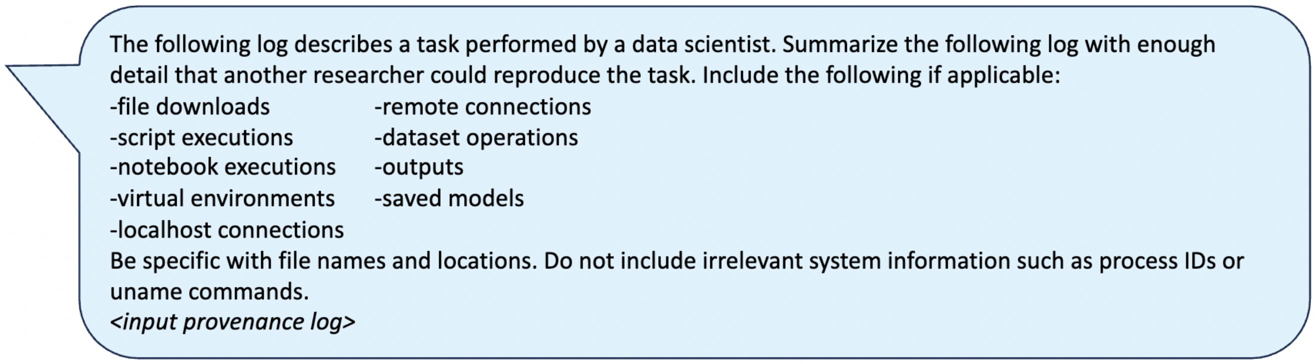
3.2 Prompting

After preprocessing, we use LLM prompting to generate text summaries from the preprocessed provenance logs. Prompt engineering is the process of designing LLM prompts to achieve a desired response. Prompt engineering does not require model training or fine-tuning. Existing work has shown that prompt engineering effectively generates well-written summaries of long-form text [17].

④ **Prompt Engineering** We use GPT-4-0613 [37], the latest openly available model from OpenAI at the time of our study. OpenAI provides guidelines and strategies for developing prompts [1]. We followed these guidelines and adjusted our prompts until we achieved a desirable output. Using clear and specific instructions achieved the best results. In Fig. 4, we show the final prompt we used to generate the summaries for our user study. We discuss how we evaluated output quality and how we arrived at our final prompt in §3.3.

Temperature Parameter Additionally, we set the GPT temperature parameter to 0 to ensure consistent responses. The temperature is a randomness control parameter for the GPT model. A lower temperature means less randomness and a higher temperature means the outputs will have more variability. Higher temperatures sometimes introduces interesting prose and more high-level descriptions, but the responses were inconsistent and more likely to contain false information. Setting the temperature to 0 creates almost deterministic responses, as the model chooses the most likely token from the generative distribution. Unfortunately, it is not perfectly deterministic due to two factors. First, GPU computations have non-deterministic threading which can cause a different reduction order, and as floating point addition is not associative this can produce slightly different results. Second, it is widely believed that GPT-4 uses a Mixture-of-Experts (MoE) architecture [**?**]. In a MoE each expert has a maximum capacity which is shared across all inputs in the batch. Therefore, an input can be routed to different experts depending on the other inputs in the batch, leading to non-determinism.

Summarizing Large Provenance Logs Even after preprocessing, some of our provenance logs still exceeded the model context window. The GPT-4 context window is 8,000 tokens at the time of our study. In comparison, our processed provenance logs ranged from 3945 to 12815 tokens. In cases where the log was too large, we used prompt chaining, a technique that has been used for large, complex tasks [52, 55]. If a log exceeded the size of the context window, we divided it into two or more logs. We define break points as edges in the graph that correspond to a user executing a command. These break points represent a natural break in the log information such as a user executing a python script from the command line. We



The following log describes a task performed by a data scientist. Summarize the following log with enough detail that another researcher could reproduce the task. Include the following if applicable:

-file downloads -remote connections
-script executions -dataset operations
-notebook executions -outputs
-virtual environments -saved models
-localhost connections

Be specific with file names and locations. Do not include irrelevant system information such as process IDs or uname commands.
<input provenance log>

Figure 4: The prompt we used to generate our user study text summaries. *Input provenance log* denotes the log specific to each task.

maximize the size of the first chunk and put the remainder in the second chunk, ensuring that the first section of the next chunk starts at a break point. The model then summarizes the first section of the log, and we give the response, the next section of the log, and a second prompt back to the model, We repeat this process until the model has summarized the entire log. This method generated high quality summaries using the evaluation method described in §3.3.

3.3 Summary Evaluation

There is currently no standard for evaluating LLM-generated summaries. Existing methods for evaluating LLM responses use both qualitative and quantitative methods depending on the application [12]. Quantitative methods involve statistically measuring responses compared to reference text written by a human. Recent research shows no strong correlation between statistical metrics and summary quality [50]. We do not have a strict expected output structure for the text summaries; therefore, the statistical difference between the generated and reference summaries is not meaningful. Therefore, we use qualitative methods to evaluate LLM-generated summaries. We define a rubric with four categories:

Completeness	Is all the necessary information included?
Conciseness	Is any unnecessary information included?
Truthfulness	Does it include any false information?
Readability	Is it easy to read and well formatted?

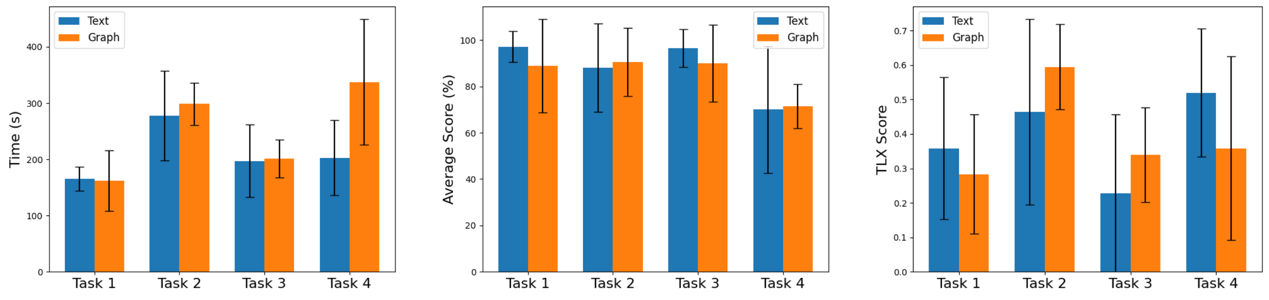
For each of the four categories, we manually assign a score out of 4, giving a total score out of 16. We developed a prompt that produced summaries that scored 16/16 for each of the provenance logs used in the study. We used the prompt in Fig. 4 to summarize logs smaller than the context window. We also use this prompt as the first prompt in the chaining approach. The prompt (excluding the input provenance) uses only 101 tokens, leaving the rest of the 8K context window for input logs and the output summary.

It took approximately one month of iteration to create our final solution. We had many discussions with our team members to develop the rubric, refine the prompt and come to a consensus on the best responses. To develop our prompt, we started with a basic prompt, "Summarize the following log." and provided a small log describing a user executing a python script. We adjusted the prompt, using different wording and adding further instructions and context. As we increased the size of the input log, the model required more specific prompting to steer it in the right direction. Once we engineered a prompt that consistently produced well written summaries, we used this prompt to generate the summaries in our user study.

We also experimented with providing examples, another technique discussed in the OpenAI guidelines. The example technique involves manually writing a "conversation history". That is, one writes a prompt and then manually generates a desired response to that prompt. The prompt/response pair is provided to the model as an example before giving the model a prompt for which you want the model to produce a response. The model is likely to follow the response format from the conversation history when using this technique. While the responses generated from prompts that included examples were of high quality, the examples counted against the context window limit, leaving fewer tokens available for the real provenance log. We did not use this example technique for the summaries in our study. Rather, we opted to use a detailed instruction that uses less of the context window limit as shown in Fig. 4.

4 USER STUDY

We conduct a user study to evaluate whether users were better able to understand workflows given text-based provenance summaries than they were when given node-link diagrams. The study uses a mixed methods approach. Participants are quantitatively evaluated on their ability to answer questions about several computational experiments using only the provenance summarization. We then analyze qualitative feedback through long answer text responses and audio recordings. Appendix D contains our study materials. Our study was approved by the University of British Columbia's Research Ethics Board [certificate #H23-00382].



(a) Average time to complete each task. **(b) Average score for each task normalized out of 100.** **(c) Normalized NASA Task Load Index (TLX).**

Figure 5: Quantitative metrics showing performance using both graph (orange) and text (blue) provenance summaries.

ID	**Occupation**	**Data Science**	**Computer Science**	**Forestry**	**Physics**	**Bioinformatics**	**Visualization**	**Information Technology**	**Human Robot Interaction**	**Environmental Science**
1	Lecturer			✓						
2	Data Scientist			✓						
3	Graduate Student	✓					✓			
4	Data Scientist	✓				✓				✓
5	Graduate Student									✓
6	Scientist								✓	
7	Professor	✓	✓		✓					
8	Database Admin							✓		
9	Graduate Student	✓		✓						✓
10	Graduate Student	✓								
11	Graduate Student			✓						✓
12	Graduate Student					✓				

Table 1: Participant Fields

4.1 Study Methods

The study session consisted of a brief introduction and overview of the study purpose, demographic questionnaire, activity, and post-activity questionnaire.

Study Activity Each participant completed four tasks. For each task, the participant was given either a node-link diagram provenance summary or an automatically generated text summary representing a computational experiment that they had not seen before. Participants used the provenance summary to answer questions about the computational experiment. The questions concerned information one would need to reproduce said experiment, such as, "*Which scripts write to this data file?*" and "*How many output files are created during this experiment?*". We describe the study's computational experiments (workflows) in Table 2. We used the one shot approach described in §3.2 to develop the summaries for the first and second tasks and the chaining approach to create the third and fourth task summaries. We manually generated the node-link diagrams to make them as readable as possible. Details on how we manually generated the summaries are in Appendix B. Each task's node-link diagrams and text summaries are available in Appendix D.

Task ID	Description
0	User executes a Python script and the script creates a plot of input data.
1	User executes an R preprocessing script followed by a Python model training script.
2	User executes a model training script three times, each with a different learning rate passed as a command line argument.
3	User executes a script that reads in data but does not produce any output files. The user then edits the script using the vim text editor. The user finally executes the edited script, and this time, the script execution produces a model checkpoint file.

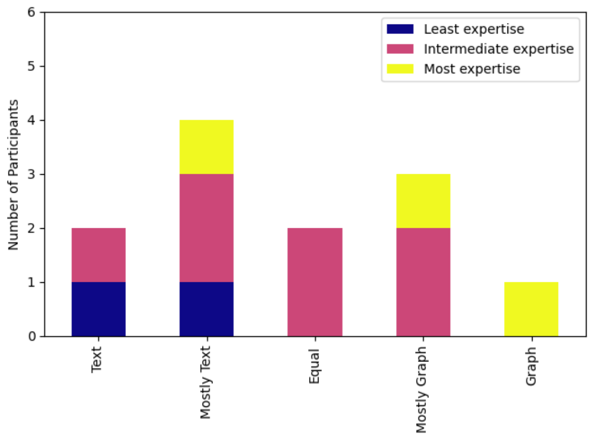
Table 2: Workflow description for each task in the user study.

4.2 Participants

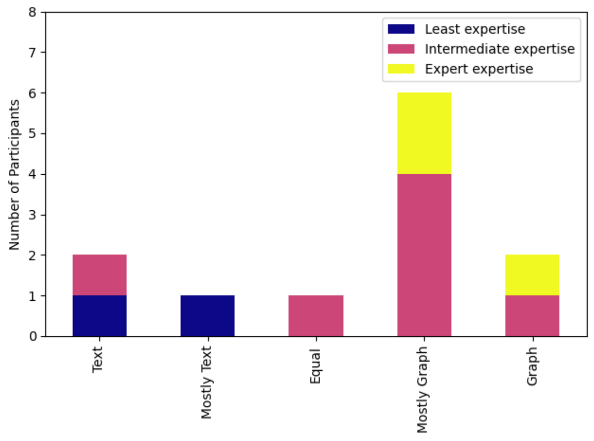
Twelve people participated in the user study. The participants were six graduate students, two data scientists, one research scientist, one professor, one lecturer, and one database administrator. Their fields spanned data science, bioinformatics, environmental science, and forestry. Table 1 shows the complete list of fields. Using the information provided in the demographic survey, we categorized the participants into three expertise categories based on their data science and programming experience.

4.3 Quantitative Results

We evaluate three performance metrics for each task: score, time to complete, and perceived cognitive load (Fig. 5). Overall, participants were able to answer most questions correctly regardless of which



(a) Which provenance summarization was more useful?



(b) Which provenance summarization was more enjoyable?

Figure 6: Participants rank their preference for either the text summarization or node link diagram. Participants are categorized by their computational expertise.

representation they used. Time to complete and perceived cognitive load varied across tasks.

Time to Complete We measured the time to complete each task by recording how long the participant spent on that page in the survey (Fig. 5a). The average time to complete each task was similar for tasks 1-3 independent of whether the participants used the text summary or the node-link diagram summary. In task 3, we see that participants using the node-link diagram took slightly longer, on average, to complete this task. We suspect this difference occurs because task 4 contained the only long answer question that asks "why?" rather than "what?". This question required participants to consider the larger picture of the whole provenance graph and its implications.

Question Score Each participant successfully answered most questions correctly regardless of representation (Fig. 5b). For tasks 1-3, at least 8 out of 12 participants scored 100% and 11 out of 12 scored over 70% using either the text or node-link diagram summary. The scores were lowest for task 4, where only one participant scored perfectly, although 10 out of 12 participants scored over 75%. The participants who scored the highest on this task were able to answer the multistep reasoning more easily with the text-based provenance summary than they were with the graph-based one. We assigned a score for this question manually, giving two points if they answered correctly with plausible reasoning and one point for partially correct responses (i.e. correct reasoning but incorrect answer or vice versa). All other questions in the study had only one answer and were marked as either correct or incorrect.

Perceived Cognitive Load We measure perceived cognitive load using the NASA Task Load Index (TLX) standard scoring system [19]. After each task, participants recorded their response to the TLX questions in Appendix C. As with the other quantitative metrics, the cognitive load scores are similar when comparing the two summarization methods (Fig. 5c).

The quantitative results show no obvious difference in overall performance when using the text or the node-link summary. We begin to see larger differences when we look at user preference and qualitative feedback.

4.4 Qualitative Results

In the post-activity survey, we asked questions regarding the entire study experience. At this point, the participants have completed two tasks using a text provenance summary and two tasks with a node-link provenance summary. We asked them to rate their preference for either summary technique using a 5-level Likert scale [26]. The participants recorded whether either was more useful during the activities and whether either was more enjoyable than the other. We show the results of these questions in Fig. 6b.

At first glance, there is no trend in either direction. Some participants strongly prefer the node-link diagram, and others strongly prefer the text, with a few in the middle. But, when we include participants' overall experience with research programming, a stronger trend emerges. Users with little experience (blue) find the text both more useful and enjoyable. Users with intermediate (pink) to advanced (yellow) experience varied in whether they found the text or node-link summary more useful, but tended more towards the graph in terms of enjoyment. We uncover some explanations for these trends using the long answer survey responses and audio transcriptions. We outline the prominent themes below. In the following sections, we refer to participants by number (e.g. P0) to preserve anonymity.

Text summaries are accessible for all expertise levels. As observed in Fig. 6a, users with less computational expertise preferred the text summaries. Less experienced participants were more comfortable and less overwhelmed with the text summaries. For instance, P6 felt the graphs required some background knowledge they did not have.

Text summaries tell a story. P6 described the text summaries as *"Text reads more like a storyline, which is more intuitive for me"*. Multiple participants found the text summaries followed a logical order.

P12, who studies bioinformatics, remarked that they are required to closely follow written protocols in their work. This experience translated well to understanding the text summaries, which have a similar format to a written experiment protocol. However, the graph differed from any data format they were familiar with and required more effort to understand. Advanced users, many of whom found the graph more enjoyable, still appreciated aspects of the text summary. P8 notes *"The text format felt more useful in identifying the workflow steps in order."*

Text summary requires attention to detail. Several participants who preferred the node-link diagram summarization found the text too long to read. P10 found it less enjoyable to *"read through each sentence and remember what is being done at each step"*. While the length of prose was a barrier for some participants, other participants found it helpful. Particularly, users who are confident and often read written protocols were comfortable extracting information from the text summaries.

The text summaries lacked some structure compared to the graphs. As in natural prose, the subjects and verbs do not appear in the same place in each sentence in the text summaries. For P9, *"The text summary tended to jump around more and was difficult to follow."* We discuss alternative text summary formats in §5.2.

Advanced Users Identify Patterns in Graph. Users with high computational expertise often preferred the graph format. Many users in this category enjoyed the extra details and workflow visualization for identifying relationships and patterns. P8 found the graph *"made it easier to identify relationships between different components."* Similarly, P9 found that when using the graph *"it was easier to see repeated steps and patterns."*

Users noted that the text summary was harder to skim and quickly extract information. As such, several participants identified areas where the text could be improved, potentially affording similar benefits to the graph. Several participants noted that keyword highlighting in the text might allow pattern matching similar to the graph. We discuss this further in §5.1.

Text summary can help users to get up to speed on node-link diagrams. Several users noted they would like to see both provenance representations in a real application. For less experienced users, some noted they could use the text summary to help understand the node-link diagram. P6 would prefer to have *"both text and [node-link diagram] side by side [...] so that I could eventually learn how to read [the node-link diagrams] with some practice."* Even users who preferred the graph noted that *"a plain or natural language commentary is always useful [alongside the graph]"* (P9).

4.5 Remote Study

We released a second version of our user study as an online survey and allowed participants to complete the survey on their own. The remote version of the study had minor changes from the in-person version including small changes to question wording, two additional demographic questions and an additional long-form answer question for task 2. 10 participants completed the remote study. We did not see any significant trends across the quantitative metrics. All the participants that completed the remote study were categorized as intermediate or expert in their computational and data science expertise. Participants' overall preference for the text summarization versus the node link diagram was similar to the initial study. The qualitative feedback matched the themes we identified in the first study. Several participants remarked that they enjoyed the visual cues in the node link diagram but also found the text useful for answering questions about what happened during an experiment.

4.6 Threats to Validity

Our study evaluation has several limitations. To limit the complexity of the tasks in our study, we evaluate the provenance summaries by asking questions about reproducing an experiment rather than having the users actually reproduce an experiment. Additionally, we provide the provenance summaries without any corresponding code, data or computational environment. Future studies should consider evaluating more realistic scenarios. For example, participants could use the provenance to reproduce past experiments or try provenance collection tools in their own workflows.

We do not observe any strong trends in our quantitative analysis, possibly due to our small study size and specific population. With our sample size (24 total participants), it is challenging to draw statistically significant conclusions. Additionally, the majority of our participants are academic researchers, so we cannot generalize our findings to other populations. Nonetheless, our results set the groundwork for future studies and highlight the need for larger studies comparing provenance representations for comprehension.

5 DISCUSSION & FUTURE WORK

Our qualitative analysis yields several areas of improvement for text-based provenance summaries as well as reproducibility tools. The participants' enthusiasm while sharing feedback on reproducibility tools sparks optimism for future research in provenance and reproducibility.

5.1 Design Recommendations

In the post-activity questionnaire, we asked participants if there are additional features they would like for a reproducibility tool and if there is anything that would prevent them from using a reproducibility tool. We give several recommendations based on our takeaways from the qualitative analysis. These guidelines can also be applied more broadly to any tools that assist with user comprehension of experimental workflows.

Visual Features For both visualization-based and text-based summarizations, several users noted they would like highlighting, zooming, panning, and search. As P7 describes, *"adding colors to file names, and scripts/outputs/paths [...] would make it more readable."* P3 also mentions *"highlighting of linked routes (when you hover over an item it shows all the related items)"*. In graph summaries, users complained of difficulty tracing the edges between nodes. In text summaries, several participants noted that the text required users to read the entire entry, sometimes multiple times, to ensure they did not miss any details. We imagine that simply color coding and bolding keywords such as verbs (e.g., read, write, execute) and file paths would help users to extract important details more quickly and easily.

Hide Low-Level Details With either provenance summarization, users still felt overloaded with information on first impressions. P3 suggested *"the ability to have hierarchical drop down tree to help organize larger amounts of data"*. Similarly, P9 wanted the tool to *"allow the user to 'zoom in' on different parts of the experiment"*. The option to view a high-level summary first and expand on the details later might reduce the initial cognitive overload and make the summaries more approachable.

Integration with Existing Tools Four participants expressed interest in integration with tools they already use, such as Git [10] or RMarkdown [3]. Several would have liked a provenance summarization directly linked to their code repository. Others mentioned it would help them to understand previous experiments if they integrated a provenance summary into their computational notebook.

Installation and Use Overheads Many participants mentioned that they would be unlikely to use any tool if the overhead for use was too high. This overhead includes installation and workflow modifications. P6 noted *"if set up would slow me down a lot, I might be less likely to use it."* Specifically, barriers include having to rewrite any of their existing code or switching programming environments.

5.2 Text Summary Limitations and Improvements

Although we cannot guarantee perfect summaries using current models, our positive results using a generic large language model leave us hopeful. We expect that using a domain specific LLM, trained on experiment provenance data, would be better still. For our user study, we generated text summaries using the out-of-the-box GPT-4 model from OpenAI [37] with no fine-tuning. GPT-4 is closed-source, and we assume OpenAI trained it with general-purpose data. We expect that training or fine-tuning a model using provenance graphs and summary data would reduce false information and give more consistent structure. Fine-tuning involves training a pre-trained model on a smaller task-specific dataset. We envision using our evaluation criteria from §3.3 in this task-specific dataset. Additionally, LLM providers such as OpenAI and Meta are frequently releasing improved models with larger context windows [51]. At the time of writing, OpenAI has already released a new version of GPT-4 with a 128K context window [20] As such, we are optimistic for a path forward.

With LLM limitations in mind, we also note that there are several ways one could automatically generate similar, or even better, summaries. The simplest method is to use rule-base methods to populate text fields in a structured report [49]. A significant benefit of rule-based text generation is that it does not require model training. Additionally, rule-based methods address some concerns mentioned by study participants who wanted more structure in the text. Conversely, the rigid structure of rule-based methods could decrease the output quality, making the summaries less compelling to read.

Another way to extend LLM provenance summarization is to incorporate multiple provenance formats such as language level provenance [25, 41] and application provenance [5]. Since different provenance tools use different abstractions, it is challenging to combine the information. One could take advantage of the unstructured input and output format and use LLMs to summarize provenance logs from various sources. Further, a tool could potentially combine summaries to link provenance from various tools together.

Lastly, extending the summary generation using LLMs, several study participants expressed interest in using a tool similar to ChatGPT for assistance with reproducing experiments. ChatGPT is OpenAI's interactive tool for model prompting [36]. In this interactive mode, users would be able to "chat" with the model to ask questions about the experiment and how to reproduce it. Querying the provenance through this interface might reduce some confusion for users were overwhelmed with too much data in the summaries. Assuming we could fine-tune a model for provenance summarization, one could imagine such a tool for querying provenance graphs. The benefits would be that users could express queries in natural language and get nuanced, simple responses.

6 RELATED WORK

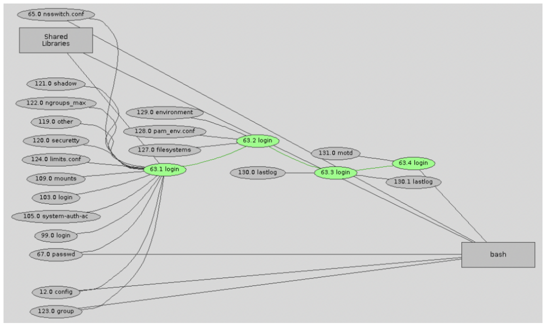
Our work focuses on determining how receptive users are to a tool conveying information from provenance. Given that we propose a technique using LLMs to do so, we examine prior work on visualizing and summarizing provenance, as well as LLM summarization techniques and limitations.

6.1 Provenance Graph Visualization

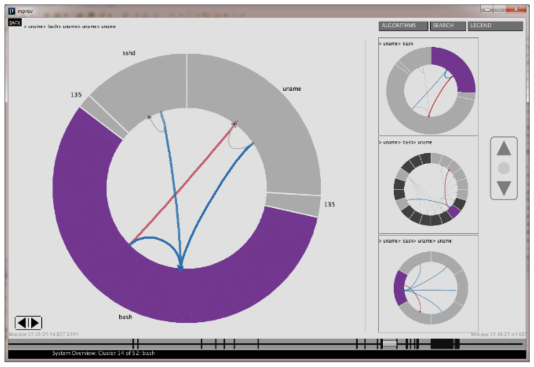
Provenance data are historically displayed using node-link diagrams [15, 21, 30]. Some applications such as Probe-It [15] include additional views, graph-style visualizations have practically become standard practice. Many tools store provenance data in graph databases, e.g., Neo4j [35], and then use the tools that accompany those systems or other graph-centric tools, e.g., GraphViz [16], to explicitly represent provenance data. However, generic graph tools often produce illustrations that are cluttered and difficult to read. Provenance tools for experimental workflow tracking also, unsurprisingly, use node-link diagram illustrations. Vistrails [5] captures provenance for workflows in their applications and displays the provenance data using node-link diagrams. Users must execute their entire workflow in the Vistrails application to capture provenance. Language-level provenance tools common in research programming, such as RDataTracker [25] and noWorkflow [41], also use node-link diagrams. For our study, we chose to manually generate our node-link diagrams rather that use existing tools to generate the graphs. We made this decision because we use a different provenance abstraction than the language-level tools and some application specific provenance visualization tools such as VisTrails [5]. Additionally, the graphs we generated using GraphViz [16] and Neo4J database viewer [35] were not well organized and did not display all the information necessary for reproducibility comprehension. Therefore, we did not think it would be a fair comparison to use these in the study. We manually created the graphs in our study to highlight workflow-level detail necessary for reproducibility.

6.2 Alternative Provenance Summarization

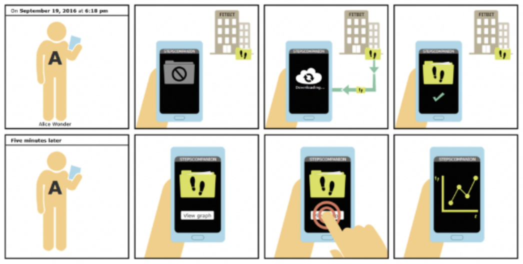
Some prior work eschewed the node-link diagram and provided alternative visualizations of provenance data. Schreiber and Struminski use comics to describe user sensor data, such as metrics from a smartwatch [44]. The comic provenance visualization in Fig. 7c



(a) Macko and Seltzer use node-link diagrams to visualize provenance data.



(b) Borkin et al. use a radial diagram to show file system provenance.



(c) Schreiber and Struminski use comics to present smartwatch provenance data.

Figure 7: We compare the most common node-link visualization (Fig. 7a) with two alternative approaches (Fig. 7c, Fig. 7b).

provides an easy-to-read, high-level summary of self-tracking sensor data [44]. Borkin et al. used a radial representation to represent file system provenance data (Fig. 7b [6]). They compare this radial representation to the node-link representation of Orbiter [30] and find that system administrators are better able to interpret system level behavior using the radial diagrams. Similar to what we observed with text-based summaries, these tools lowered the cognitive load for some users while performing tasks. VinciDecoder [48] uses machine learning to summarize provenance graphs into forensic reports. These forensic reports are also similar to our work, as the reports are text-based and use machine learning techniques to produce them. However, since our summarizations are LLM-generated, our reports have less regular structure than VinciDecoder's. One could envision adopting similar generation techniques to create more structured experimental workflow provenance summarizations. While these visualization techniques perform well in some domains, none are designed or evaluated specifically for computational experiment comprehension.

6.3 LLM Summarization

Summarization is a popular application of large language models. Zhang et al. and Goyal et al. use LLMs to summarize new articles [17, 62]. The authors find that users prefer LLM-generated summaries to other summarization models [17] and like them much as human-written summaries [62]. Similarly, Laskar et al. evaluated how well various LLMs summarize meeting notes [24] and ultimately deployed such summaries in a real-world setting. Other work introduces new techniques for prompting LLMs to summarize even longer text. Wu et al. propose their "Extract-then-Evaluate" method [56], while Chang et al. and Zhang et al. demonstrate techniques based on iterative and incremental updates [11, 61].

Despite the successes found in LLM summarization, they are not without their drawbacks. Yang et al. demonstrated that the predictions ChatGPT provides in a mental health care setting are unstable and can change based on tweaking the severity of adjectives used in the prompt [58]. They also found that while LLMs are capable of providing explanations for their answer, those explanations are not always correct and do not mean that the models are interpretable. Meanwhile, Shen et al. tested LLMs for automatic evaluation of summaries, and found them inconsistent and not yet at the level of human evaluators [47]. Liu et al. observe that LLMs frequently overlook information in the middle of a document [28]. Tang et al. evaluates medical journal summaries produced by GPT 3.5 [38] and ChatGPT [36] and demonstrate that the GPT-generated summaries are often untruthful or used indecisive language that could lead to misinformation. Unlike provenance logs, these summarizations' inputs (news articles, paper abstracts) are pure natural language sentences and paragraphs, which is what led us to create the natural language format from our provenance logs (§3). To our knowledge, there is no published work on using LLMs for summarizing provenance logs or log-structured data.

7 CONCLUSION

Our study demonstrates the viability of automatically generating text-based provenance summaries for enhancing user understanding of data science workflows. We evaluate the text summarization approach with a user study, comparing text summaries to node-link diagrams for presenting provenance data to users. We found that users with less programming expertise often prefer the text summaries as they adopt a familiar format that is more approachable. In contrast, users with advanced programming experience enjoyed the extra details and visual cues provided in the graphs. We provide direction to improve both the text and node-link diagram summarizations in our qualitative analysis of our survey results. Not only do our results demonstrate the effectiveness of the textual summaries, but the results also provide insight into how to present provenance data to enhance user comprehension.

ACKNOWLEDGMENTS

We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC). Nous remercions le Conseil de recherches en sciences naturelles et en génie du Canada (CRSNG) de son soutien.

The author contributions are as follows: Nichole Boufford: conceptualization, software, formal analysis, investigation, writing – original draft preparation, methodology, validation, visualization, resources, writing– review and editing, data curation Joseph Wonsil: conceptualization, investigation, resources, writing – review and editing Adam Pocock: conceptualization, writing – review and editing Jack Sullivan: conceptualization, writing –review and editing Margo Seltzer: conceptualization, supervision, writing – review and editing Thomas Pasquier: conceptualization, supervision, writing – review and editing, project administration, funding acquisition.

REFERENCES

[1] [n. d.]. Open AI Prompt Engineering Guide. https://platform.openai.com/docs/guides/prompt-engineering. Accessed: 2023-11-22.

[2] (accessed January 23, 2024). eBPF. online. https://ebpf.io/..

[3] JJ Allaire, Yihui Xie, Christophe Dervieux, Jonathan McPherson, Javier Luraschi, Kevin Ushey, Aron Atkins, Hadley Wickham, Joe Cheng, Winston Chang, and Richard Iannone. 2023. *rmarkdown: Dynamic Documents for R.* https://github.com/rstudio/rmarkdown R package version 2.25.

[4] Monya Baker. 2016. 1,500 scientists lift the lid on reproducibility. *Nature* 533, 7604 (2016), 452–454.

[5] L. Bavoil, S.P. Callahan, P.J. Crossno, J. Freire, C.E. Scheidegger, C.T. Silva, and H.T. Vo. 2005. VisTrails: enabling interactive multiple-view visualizations. In *VIS 05. IEEE Visualization, 2005.* 135–142. https://doi.org/10.1109/VISUAL.2005.1532788

[6] Michelle A. Borkin, Chelsea S. Yeh, Madelaine Boyd, Peter Macko, Krzysztof Z. Gajos, Margo Seltzer, and Hanspeter Pfister. 2013. Evaluation of Filesystem Provenance Visualization Tools. *IEEE Transactions on Visualization and Computer Graphics* 19, 12 (2013), 2476–2485. https://doi.org/10.1109/TVCG.2013.155

[7] Nichole Boufford. 2023. Thoth. https://github.com/ubc-systopia/thoth.

[8] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. *CoRR* abs/2005.14165 (2020). arXiv:2005.14165 https://arxiv.org/abs/2005.14165

[9] Lucian Carata, Sherif Akoush, Nikilesh Balakrishnan, Thomas Bytheway, Ripduman Sohan, Margo Seltzer, and Andy Hopper. 2014. A Primer on Provenance: Better Understanding of Data Requires Tracking Its History and Context. *Queue* 12, 3 (2014), 10–23. https://doi.org/10.1145/2602649.2602651

[10] Scott Chacon and Ben Straub. 2014. *Pro git.* Apress.

[11] Yapei Chang, Kyle Lo, Tanya Goyal, and Mohit Iyyer. 2023. BooookScore: A systematic exploration of book-length summarization in the era of LLMs. *arXiv preprint arXiv:2310.00785* (2023).

[12] Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, Wei Ye, Yue Zhang, Yi Chang, Philip S. Yu, Qiang Yang, and Xing Xie. 2023. A Survey on Evaluation of Large Language Models. arXiv:2307.03109 [cs.CL]

[13] Fernando Chirigati, Rémi Rampin, Dennis Shasha, and Juliana Freire. 2016. ReproZip: Computational Reproducibility With Ease. In *Proceedings of the 2016 International Conference on Management of Data* (San Francisco, California, USA) *(SIGMOD '16)*. ACM, 2085–2088. https://doi.org/10.1145/2882903.2899401

[14] April Clyburne-Sherin, Xu Fei, and Seth Ariel Green. 2019. Computational reproducibility via containers in social psychology. *Meta-Psychology* 3 (2019).

[15] Nicholas Del Rio and Paulo Pinheiro Da Silva. 2007. Probe-it! visualization support for provenance. In *International Symposium on Visual Computing*. Springer, 732–741.

[16] John Ellson, Emden Gansner, Lefteris Koutsofios, Stephen C North, and Gordon Woodhull. 2002. Graphviz—open source graph drawing tools. In *Graph Drawing: 9th International Symposium, GD 2001 Vienna, Austria, September 23–26, 2001 Revised Papers 9*. Springer, 483–484.

[] William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research* 23, 120 (2022), 1–39.

[17] Tanya Goyal, Junyi Jessy Li, and Greg Durrett. 2023. News Summarization and Evaluation in the Era of GPT-3. arXiv:2209.12356 [cs.CL]

[18] Philip J Guo and Margo I Seltzer. 2012. Burrito: Wrapping your lab notebook in computational infrastructure. (2012).

[19] Sandra G. Hart and Lowell E. Staveland. 1988. Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research. In *Human Mental Workload*, Peter A. Hancock and Najmedin Meshkati (Eds.). Advances in Psychology, Vol. 52. North-Holland, 139–183. https://doi.org/10.1016/S0166-4115(08)62386-9

[20] https://openai.com/blog/new-models-and-developer-products-announced-at devday. 2023. New models and developer products announced at DevDay. blog post.

[21] Jane Hunter and Kwok Cheung. 2007. Provenance Explorer-a graphical interface for constructing scientific publication packages from provenance trails. *International Journal on Digital Libraries* 7 (2007), 99–107.

[22] Shaoxiong Ji, Tianlin Zhang, Kailai Yang, Sophia Ananiadou, and Erik Cambria. 2023. Rethinking Large Language Models in Mental Health Applications. *arXiv preprint arXiv:2311.11267* (2023).

[23] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, Carol Willing, and Jupyter development team. 2016. Jupyter Notebooks - a publishing format for reproducible computational workflows. In *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, Fernando Loizides and Birgit Scmidt (Eds.). IOS Press, Netherlands, 87–90. https://eprints.soton.ac.uk/403913/

[24] Md Tahmid Rahman Laskar, Xue-Yong Fu, Cheng Chen, and Shashi Bhushan Tn. 2023. Building Real-World Meeting Summarization Systems using Large Language Models: A Practical Perspective. *arXiv preprint arXiv:2310.19233* (2023).

[25] B.S. Lerner and E.R. Boose. 2014. RDataTracker: Collecting Provenance in an Interactive Scripting Environment. In *USENIX Workshop on the Theory and Practice of Provenance (TaPP).* https://doi.org/10.1007/978-3-319-16462-5_36

[26] Rensis Likert. 1932. A technique for the measurement of attitudes. *Archives of psychology* (1932).

[27] Soo Yee Lim, Bogdan Stelea, Xueyuan Han, and Thomas Pasquier. 2021. Secure namespaced kernel audit for containers. In *Proceedings of the ACM Symposium on Cloud Computing*. 518–532.

[28] Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2023. Lost in the middle: How language models use long contexts. *arXiv preprint arXiv:2307.03172* (2023).

[29] Yucheng Low, Ajit Banerjee, and Rajat Arya. 2021. XetHub. https://about.xethub.com/

[30] Peter Macko and Margo Seltzer. 2011. Provenance Map Orbiter: Interactive Exploration of Large Provenance Graphs. In *3rd USENIX Workshop on the Theory and Practice of Provenance (TaPP 11).* USENIX Association, Heraklion, Crete Greece. https://www.usenix.org/conference/tapp11/provenance-map-orbiter-interactive-exploration-large-provenance-graphs

[31] Andrew Mleczko, Sebastian Schuberth, Lars Schneider, and Brian M. Carlson. 2021. git-lfs. https://github.com/git-lfs/git-lfs

[32] Kiran-Kumar Muniswamy-Reddy, David Holland, Uri Braun, and Margo Seltzer. 2006. Provenance-Aware Storage Systems. *USENIX* (01 2006), 43–56.

[33] T. Munzner. 2015. *Visualization Analysis and Design.* CRC Press. 210 pages. https://books.google.de/books?id=NfkYCwAAQBAJ

[34] National Academies of Sciences, Engineering, and Medicine. 2019. *Reproducibility and Replicability in Science.* The National Academies Press, Washington, DC. https://doi.org/10.17226/25303

[35] Neo4j. 2012. Neo4j. http://neo4j.org/

[36] OpenAI. 2023. ChatGPT. https://chat.openai.com/chat

[37] OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL]

[38] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. arXiv:2203.02155 [cs.CL]

[39] Thomas Pasquier, Xueyuan Han, Mark Goldstein, Thomas Moyer, David Eyers, Margo Seltzer, and Jean Bacon. 2017. Practical Whole-System Provenance Capture. In *Symposium on Cloud Computing (SoCC'17).* ACM, ACM.

[40] Thomas Pasquier, Matthew K. Lau, Ana Trisovic, Emery R. Boose, Ben Couturier, Mercè Crosas, Aaron M. Ellison, Valerie Gibson, Chris R. Jones, and Margo Seltzer. 2017. If these data could talk. *Nature Scientific Data* 4 (2017). https://www.nature.com/articles/sdata2017114

[41] João Felipe Pimentel, Leonardo Murta, Vanessa Braganholo, and Juliana Freire. 2017. NoWorkflow: A Tool for Collecting, Analyzing, and Managing Provenance from Python Scripts. *Proc. VLDB Endow.* 10, 12 (aug 2017), 1841–1844. https://doi.org/10.14778/3137765.3137789

[42] João Felipe Pimentel, Leonardo Murta, Vanessa Braganholo, and Juliana Freire. 2019. A large-scale study about quality and reproducibility of jupyter notebooks. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR).* IEEE, 507–517.

[43] Chandrasekhar Ramakrishnan, Michele Volpi, Fernando Perez-Cruz, Lilian Gasser, Firat Ozdemir, Patrick Paitz, Mohammad Alisafaee, Philipp Fischer, Ralf Grubenmann, Eliza Jean Harris, et al. 2023. Renku: a platform for sustainable data science. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

[44] Andreas Schreiber and Regina Struminski. 2017. Visualizing Provenance using Comics. In *9th USENIX Workshop on the Theory and Practice of Provenance (TaPP 2017)*. USENIX Association, Seattle, WA. https://www.usenix.org/conference/tapp17/workshop-program/presentation/schreiber

[45] R Sekar, Hanke Kimm, and Rohit Aich. 2023. eAudit: A Fast, Scalable and Deployable Audit Data Collection System. In *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 87–87.

[46] Omid Setayeshfar, Christian Adkins, Matthew Jones, Kyu Hyung Lee, and Prashant Doshi. 2021. Graalf: Supporting graphical analysis of audit logs for forensics. *Software Impacts* 8 (2021), 100068.

[47] Chenhui Shen, Liying Cheng, Xuan-Phi Nguyen, Yang You, and Lidong Bing. 2023. Large Language Models are Not Yet Human-Level Evaluators for Abstractive Summarization. In *Findings of the Association for Computational Linguistics: EMNLP 2023*. 4215–4233.

[48] Azadeh Tabiban, Heyang Zhao, Yosr Jarraya, Makan Pourzandi, and Lingyu Wang. 2023. VinciDecoder: Automatically Interpreting Provenance Graphs Into Textual Forensic Reports With Application To OpenStack. In *Secure IT Systems: 27th Nordic Conference, NordSec 2022, Reykjavic, Iceland, November 30–December 2, 2022, Proceedings* (Reykjavic, Iceland). Springer-Verlag, Berlin, Heidelberg, 346–367. https://doi.org/10.1007/978-3-031-22295-5_19

[49] Azadeh Tabiban, Heyang Zhao, Yosr Jarraya, Makan Pourzandi, Mengyuan Zhang, and Lingyu Wang. 2022. ProvTalk: towards interpretable multi-level provenance analysis in networking functions virtualization (NFV). In *The Network and Distributed System Security Symposium 2022 (NDSS '22)*.

[50] Liyan Tang, Zhaoyi Sun, Betina Idnay, Jordan G Nestor, Ali Soroush, Pierre A Elias, Ziyang Xu, Ying Ding, Greg Durrett, Justin F Rousseau, et al. 2023. Evaluating large language models on medical evidence summarization. *npj Digital Medicine* 6, 1 (2023), 158.

[51] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).

[52] Dietrich Trautmann. 2023. Large Language Model Prompt Chaining for Long Legal Document Classification. *arXiv preprint arXiv:2308.04138* (2023).

[53] Ana Trisovic, Matthew Lau, Thomas Pasquier, and Merce Crosas. 2022. A large-scale study on research code quality and execution. *Scientific Data* 9 (02 2022), 60. https://doi.org/10.1038/s41597-022-01143-6

[54] Joseph Wonsil, Nichole Boufford, Prakhar Agrawal, Christopher Chen, Tianhang Cui, Akash Sivaram, and Margo Seltzer. 2023. Reproducibility as a service. *Software: Practice and Experience* 53, 7 (2023), 1543–1571. https://doi.org/10.1002/spe.3202 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.3202

[55] Tongshuang Wu, Ellen Jiang, Aaron Donsbach, Jeff Gray, Alejandra Molina, Michael Terry, and Carrie J Cai. 2022. Promptchainer: Chaining large language model prompts through visual programming. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts*. 1–10.

[56] Yunshu Wu, Hayate Iso, Pouya Pezeshkpour, Nikita Bhutani, and Estevam Hruschka. 2023. Less is More for Long Document Summary Evaluation by LLMs. *arXiv preprint arXiv:2309.07382* (2023).

[57] Zhang Xu, Zhenyu Wu, Zhichun Li, Kangkook Jee, Junghwan Rhee, Xusheng Xiao, Fengyuan Xu, Haining Wang, and Guofei Jiang. 2016. High Fidelity Data Reduction for Big Data Security Dependency Analyses. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (Vienna, Austria) *(CCS '16)*. Association for Computing Machinery, New York, NY, USA, 504–516. https://doi.org/10.1145/2976749.2978378

[58] Kailai Yang, Shaoxiong Ji, Tianlin Zhang, Qianqian Xie, Ziyan Kuang, and Sophia Ananiadou. 2023. Towards interpretable mental health analysis with large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. 6056–6077.

[59] Vahan Yoghourdjian, Yalong Yang, Tim Dwyer, Lee Lawrence, Michael Wybrow, and Kim Marriott. 2020. Scalability of network visualisation from a cognitive load perspective. *IEEE transactions on visualization and computer graphics* 27, 2 (2020), 1677–1687.

[60] J.D. Zamfirescu-Pereira, Richmond Y. Wong, Bjoern Hartmann, and Qian Yang. 2023. Why Johnny Can't Prompt: How Non-AI Experts Try (and Fail) to Design LLM Prompts. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI '23)*. Association for Computing Machinery, New York, NY, USA, Article 437, 21 pages. https://doi.org/10.1145/3544548.3581388

[61] Haopeng Zhang, Xiao Liu, and Jiawei Zhang. 2023. SummIt: Iterative Text Summarization via ChatGPT. *arXiv preprint arXiv:2305.14835* (2023).

[62] Tianyi Zhang, Faisal Ladhak, Esin Durmus, Percy Liang, Kathleen McKeown, and Tatsunori B. Hashimoto. 2023. Benchmarking Large Language Models for News Summarization. arXiv:2301.13848 [cs.CL]

[63] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. 2023. A Survey of Large Language Models. arXiv:2303.18223 [cs.CL]

A AVAILABILITY

The work presented in this paper is open-source. Detailed installation instructions are available online.

- System Provenance Collection Tool. Available for download under GPL-2.0 license at https://github.com/ubc-systopia/thoth.
- Provenance Summarization. Available for download under Apache 2.0 license at https://doi.org/10.5281/zenodo.10672536.
- The user study documents are available at https://doi.org/10.5281/zenodo.106725369.

B DIAGRAM CREATION

It is common for graph users to visualize their data as node-link graphs with tools like Neo4j [35] and GraphViz [16]. However, these general-purpose tools are not a sufficient fit for this study. They do not readily show in a static way all the necessary attributes for nodes and edges a participant needs to see to answer the questions from our tasks. Additionally, since we created automatically generated text summaries tailored towards reproducibility and workflow executions, it would not be a fair comparison to use a general-purpose visualization. Therefore, we chose to make our own diagrams.

We devised a new type of node-link diagram tailored towards displaying information from provenance logs about a workflow. We manually created our node-link diagram visualizations using a set of pre-defined rules rather than write a program to do so automatically. Making them manually allowed us a finer grain of control over the various aspects of the diagram to ensure legibility; however, we believe the process could be automated with some effort.

Our node-link diagrams highlight the processes that comprise the computational workflow. Overall, our diagrams display events in order they executed from top to bottom; however, the operations are not displayed proportional to the time they occurred, only the order. We represent each process with a large arrow that points downward to indicate the order of execution. Each process' arrow receives a unique color, except for instances where that process has spawned additional processes. The child processes are large arrows of the same color placed directly to the right of the original process.

The top of the arrow has a block containing the command that initiated the process. Smaller arrows attached to the right side of a process show the various system operations the process performed over the course of its existence. These arrows represent edges to other nodes, such as files it reads or writes, libraries it loads, or programs it executes.

Nodes representing the same file are not duplicated, so it is clear in the graph when a process reads a file that another process created. In this situation when multiple processes have edges to a node, the order the arrows point to the node are in execution order. The arrow attached at the top executed first, moving downwards to the last arrow at the bottom which is the operation executed last.

C TASK LOAD INDEX QUESTIONS

(1) How mentally demanding was this task? (1-Very Low, 5-Very High)
(2) How hurried or rushed were you during this task? (1-Very Low, 5-Very High)
(3) How successful would you rate yourself in accomplishing this task? (1-Perfect, 5-Failure)
(4) How hard did you have to work to accomplish your level of performance? (1-Very Low, 5-Very High)
(5) How insecure, discouraged, irritated, stressed, and annoyed were you? (1-Very Low, 5-Very High)
(6) How useful was the provenance summary in answering the questions? (1-Very Useful, 5-Not Useful)

D STUDY ACTIVITIES

D.1 Questions

D.1.1 Task 1.

(1) What is the name of the dataset the student is using?
(2) Which directory is the dataset saved in?
(3) What is the name of the file containing the experiment code?
(4) Which directory is the experiment code located in?
(5) How many output files are produced? (Include intermediate outputs)
(6) Which programming languages are used to conduct the analysis in this experiment?

D.1.2 Task 2.

(1) How many times is the script train_model.py executed?
(2) How many times is the script preprocess.R executed?
(3) Which scripts write to the file data.csv?
(4) Which scripts read from the file data.csv?
(5) Which scripts write to the file temp_data.csv?
(6) Which scripts read from the file "temp_data.csv"?
(7) Which of the following are dependencies of train_model.py?

D.1.3 Task 3.

(1) Where is the dataset located?
(2) How many output files were created during this experiment (including intermediate files)?
(3) Please explain the difference between the first and second executions of the train_model.py script in no more than two sentences.

D.1.4 Task 4.

(1) What is the name of the dataset the student is using?
(2) Which directory is the dataset saved in?
(3) What is the name of the file containing the experiment code?
(4) Which directory is the experiment code located in?
(5) How many output files are produced? (Include intermediate outputs)
(6) Which programming languages are used to conduct the analysis in this experiment?

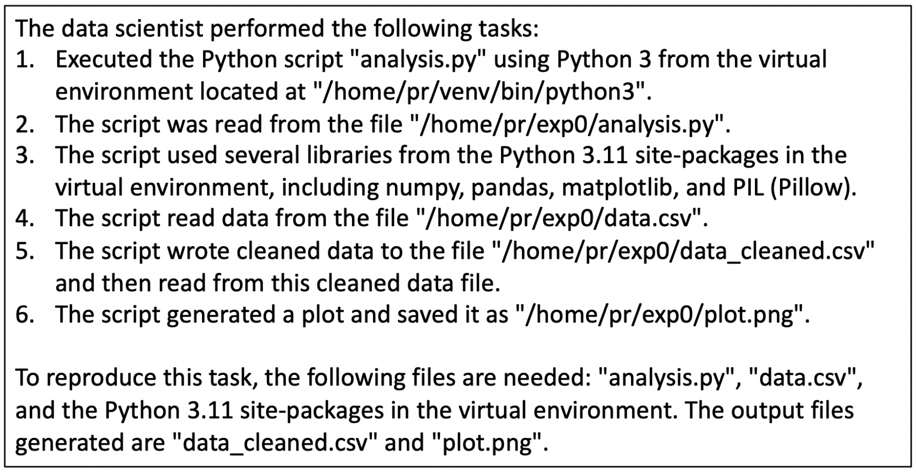
D.2 Text Summaries

Fig. 8, Fig. 9, Fig. 10, Fig. 11 show the text summaries used in our study. We used the single prompt method for task 1 and 2 and the chaining method for task 2 and 3.

D.3 Node Link Diagrams

Fig. 12, Fig. 13, Fig. 14, Fig. 15 show the node link diagrams we created and used in our study.

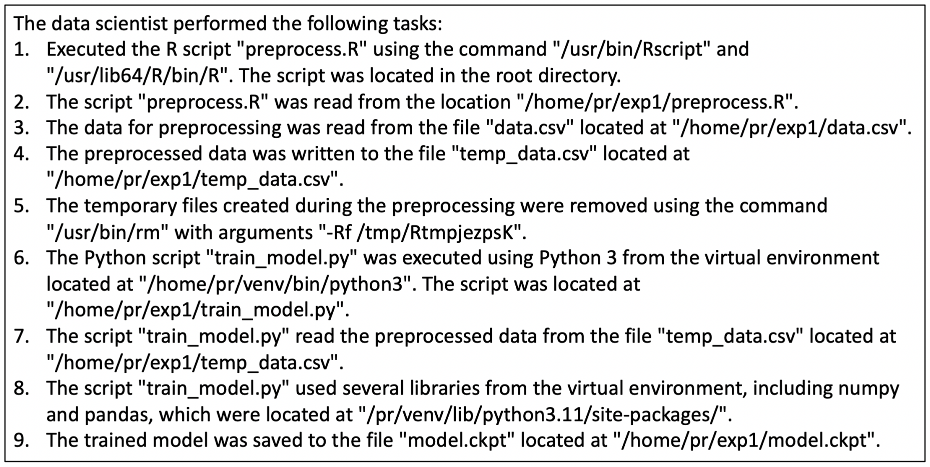
Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009


The data scientist performed the following tasks:

1. Executed the Python script "analysis.py" using Python 3 from the virtual environment located at "/home/pr/venv/bin/python3".
2. The script was read from the file "/home/pr/exp0/analysis.py".
3. The script used several libraries from the Python 3.11 site-packages in the virtual environment, including numpy, pandas, matplotlib, and PIL (Pillow).
4. The script read data from the file "/home/pr/exp0/data.csv".
5. The script wrote cleaned data to the file "/home/pr/exp0/data_cleaned.csv" and then read from this cleaned data file.
6. The script generated a plot and saved it as "/home/pr/exp0/plot.png".

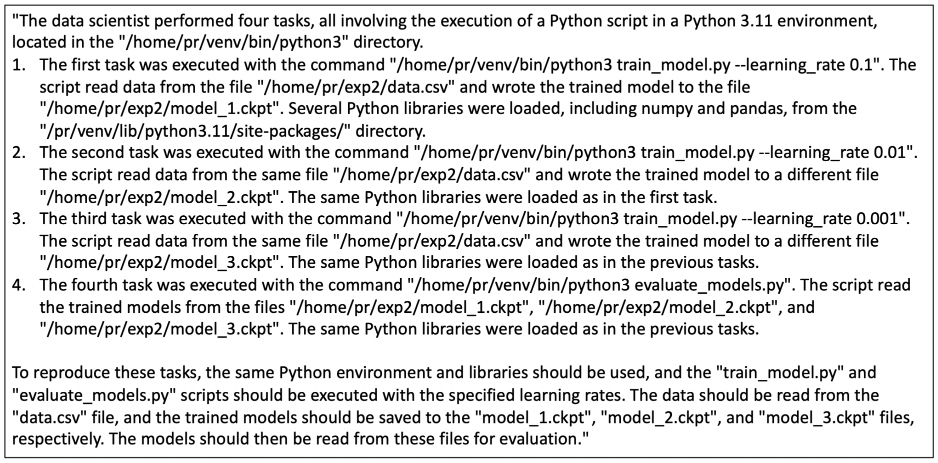
To reproduce this task, the following files are needed: "analysis.py", "data.csv", and the Python 3.11 site-packages in the virtual environment. The output files generated are "data_cleaned.csv" and "plot.png".

Figure 8: Workflow 1 Text Summary


The data scientist performed the following tasks:

1. Executed the R script "preprocess.R" using the command "/usr/bin/Rscript" and "/usr/lib64/R/bin/R". The script was located in the root directory.
2. The script "preprocess.R" was read from the location "/home/pr/exp1/preprocess.R".
3. The data for preprocessing was read from the file "data.csv" located at "/home/pr/exp1/data.csv".
4. The preprocessed data was written to the file "temp_data.csv" located at "/home/pr/exp1/temp_data.csv".
5. The temporary files created during the preprocessing were removed using the command "/usr/bin/rm" with arguments "-Rf /tmp/RtmpjezpsK".
6. The Python script "train_model.py" was executed using Python 3 from the virtual environment located at "/home/pr/venv/bin/python3". The script was located at "/home/pr/exp1/train_model.py".
7. The script "train_model.py" read the preprocessed data from the file "temp_data.csv" located at "/home/pr/exp1/temp_data.csv".
8. The script "train_model.py" used several libraries from the virtual environment, including numpy and pandas, which were located at "/pr/venv/lib/python3.11/site-packages/".
9. The trained model was saved to the file "model.ckpt" located at "/home/pr/exp1/model.ckpt".

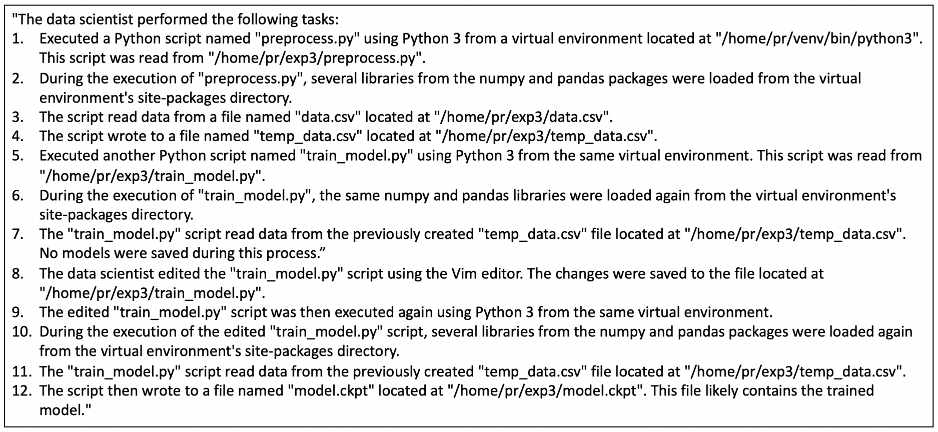
Figure 9: Workflow 2 Text Summary


"The data scientist performed four tasks, all involving the execution of a Python script in a Python 3.11 environment, located in the "/home/pr/venv/bin/python3" directory.

1. The first task was executed with the command "/home/pr/venv/bin/python3 train_model.py --learning_rate 0.1". The script read data from the file "/home/pr/exp2/data.csv" and wrote the trained model to the file "/home/pr/exp2/model_1.ckpt". Several Python libraries were loaded, including numpy and pandas, from the "/pr/venv/lib/python3.11/site-packages/" directory.
2. The second task was executed with the command "/home/pr/venv/bin/python3 train_model.py --learning_rate 0.01". The script read data from the same file "/home/pr/exp2/data.csv" and wrote the trained model to a different file "/home/pr/exp2/model_2.ckpt". The same Python libraries were loaded as in the first task.
3. The third task was executed with the command "/home/pr/venv/bin/python3 train_model.py --learning_rate 0.001". The script read data from the same file "/home/pr/exp2/data.csv" and wrote the trained model to a different file "/home/pr/exp2/model_3.ckpt". The same Python libraries were loaded as in the previous tasks.
4. The fourth task was executed with the command "/home/pr/venv/bin/python3 evaluate_models.py". The script read the trained models from the files "/home/pr/exp2/model_1.ckpt", "/home/pr/exp2/model_2.ckpt", and "/home/pr/exp2/model_3.ckpt". The same Python libraries were loaded as in the previous tasks.

To reproduce these tasks, the same Python environment and libraries should be used, and the "train_model.py" and "evaluate_models.py" scripts should be executed with the specified learning rates. The data should be read from the "data.csv" file, and the trained models should be saved to the "model_1.ckpt", "model_2.ckpt", and "model_3.ckpt" files, respectively. The models should then be read from these files for evaluation."

Figure 10: Workflow 3 Text Summary



"The data scientist performed the following tasks:

1. Executed a Python script named "preprocess.py" using Python 3 from a virtual environment located at "/home/pr/venv/bin/python3". This script was read from "/home/pr/exp3/preprocess.py".
2. During the execution of "preprocess.py", several libraries from the numpy and pandas packages were loaded from the virtual environment's site-packages directory.
3. The script read data from a file named "data.csv" located at "/home/pr/exp3/data.csv".
4. The script wrote to a file named "temp_data.csv" located at "/home/pr/exp3/temp_data.csv".
5. Executed another Python script named "train_model.py" using Python 3 from the same virtual environment. This script was read from "/home/pr/exp3/train_model.py".
6. During the execution of "train_model.py", the same numpy and pandas libraries were loaded again from the virtual environment's site-packages directory.
7. The "train_model.py" script read data from the previously created "temp_data.csv" file located at "/home/pr/exp3/temp_data.csv". No models were saved during this process.”
8. The data scientist edited the "train_model.py" script using the Vim editor. The changes were saved to the file located at "/home/pr/exp3/train_model.py".
9. The edited "train_model.py" script was then executed again using Python 3 from the same virtual environment.
10. During the execution of the edited "train_model.py" script, several libraries from the numpy and pandas packages were loaded again from the virtual environment's site-packages directory.
11. The "train_model.py" script read data from the previously created "temp_data.csv" file located at "/home/pr/exp3/temp_data.csv".
12. The script then wrote to a file named "model.ckpt" located at "/home/pr/exp3/model.ckpt". This file likely contains the trained model."

Figure 11: Workflow 4 Text Summary

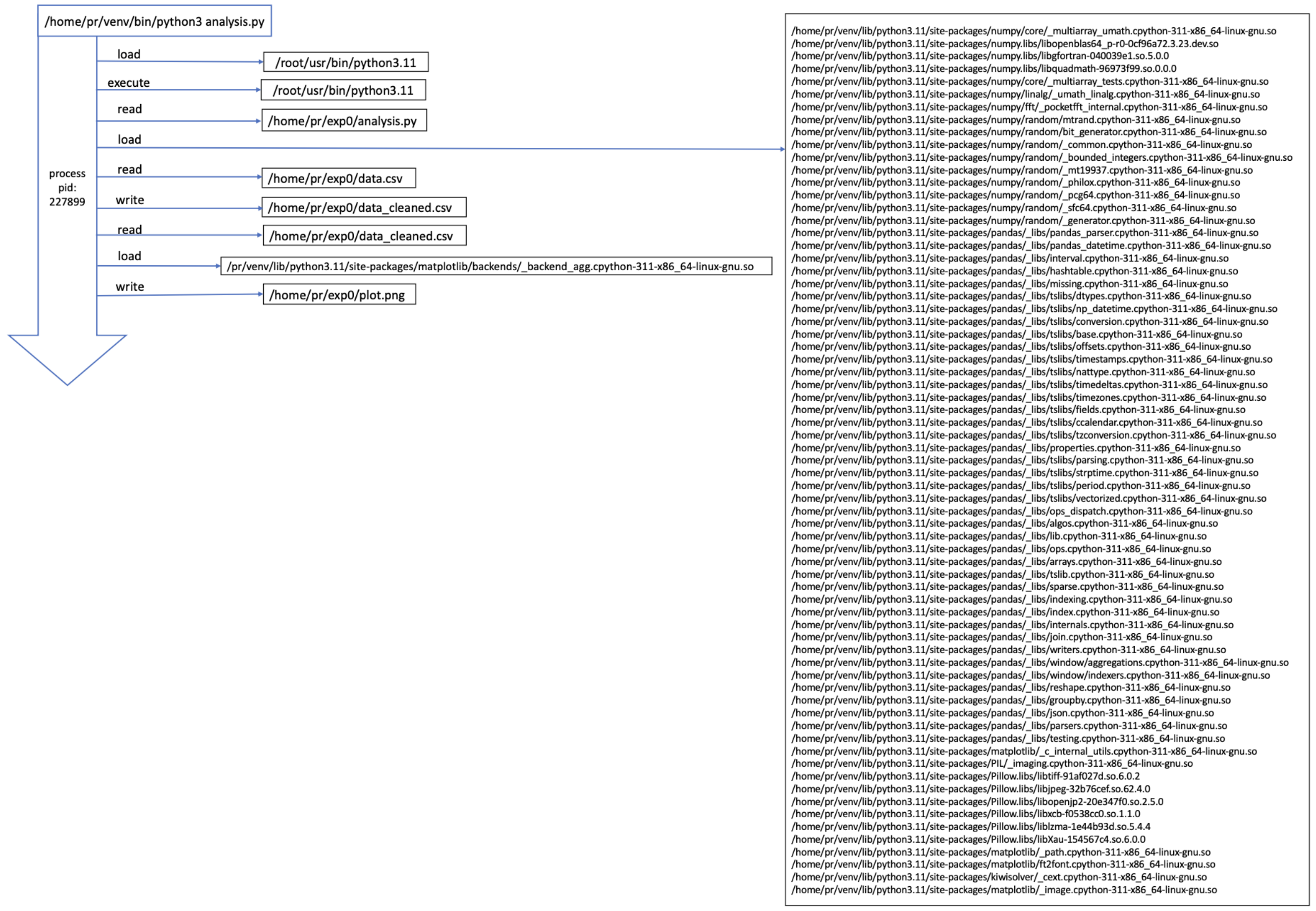

/home/pr/venv/bin/python3 analysis.py
process
pid:
227899
load
/root/usr/bin/python3.11
execute
/root/usr/bin/python3.11
read
/home/pr/exp0/analysis.py
load
read
/home/pr/exp0/data.csv
write
/home/pr/exp0/data_cleaned.csv
read
/home/pr/exp0/data_cleaned.csv
load
/pr/venv/lib/python3.11/site-packages/matplotlib/backends/_backend_agg.cpython-311-x86_64-linux-gnu.so
write
/home/pr/exp0/plot.png
/home/pr/venv/lib/python3.11/site-packages/numpy/core/_multiarray_umath.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy.libs/libopenblas64_p-r0-0cf96a72.3.23.dev.so
/home/pr/venv/lib/python3.11/site-packages/numpy.libs/libgfortran-040039e1.so.5.0.0
/home/pr/venv/lib/python3.11/site-packages/numpy.libs/libquadmath-96973f99.so.0.0.0
/home/pr/venv/lib/python3.11/site-packages/numpy/core/_multiarray_tests.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/linalg/_umath_linalg.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/fft/_pocketfft_internal.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/random/mtrand.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/random/bit_generator.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/random/_common.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/random/_bounded_integers.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/random/_mt19937.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/random/_philox.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/random/_pcg64.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/random/_sfc64.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/random/_generator.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/pandas_parser.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/pandas_datetime.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/interval.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/hashtable.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/missing.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/dtypes.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/np_datetime.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/conversion.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/base.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/offsets.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/timestamps.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/nattype.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/timedeltas.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/timezones.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/fields.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/ccalendar.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/tzconversion.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/properties.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/parsing.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/strptime.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/period.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/vectorized.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/ops_dispatch.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/algos.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/lib.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/ops.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/arrays.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslib.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/sparse.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/indexing.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/index.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/internals.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/join.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/writers.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/window/aggregations.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/window/indexers.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/reshape.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/groupby.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/json.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/parsers.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/testing.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/matplotlib/_c_internal_utils.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/PIL/_imaging.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/Pillow.libs/libtiff-91af027d.so.6.0.2
/home/pr/venv/lib/python3.11/site-packages/Pillow.libs/libjpeg-32b76cef.so.62.4.0
/home/pr/venv/lib/python3.11/site-packages/Pillow.libs/libopenjp2-20e347f0.so.2.5.0
/home/pr/venv/lib/python3.11/site-packages/Pillow.libs/libxcb-f0538cc0.so.1.1.0
/home/pr/venv/lib/python3.11/site-packages/Pillow.libs/liblzma-1e44b93d.so.5.4.4
/home/pr/venv/lib/python3.11/site-packages/Pillow.libs/libXau-154567c4.so.6.0.0
/home/pr/venv/lib/python3.11/site-packages/matplotlib/_path.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/matplotlib/ft2font.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/kiwisolver/_cext.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/matplotlib/_image.cpython-311-x86_64-linux-gnu.so

Figure 12: Workflow 1 Node Link Diagram

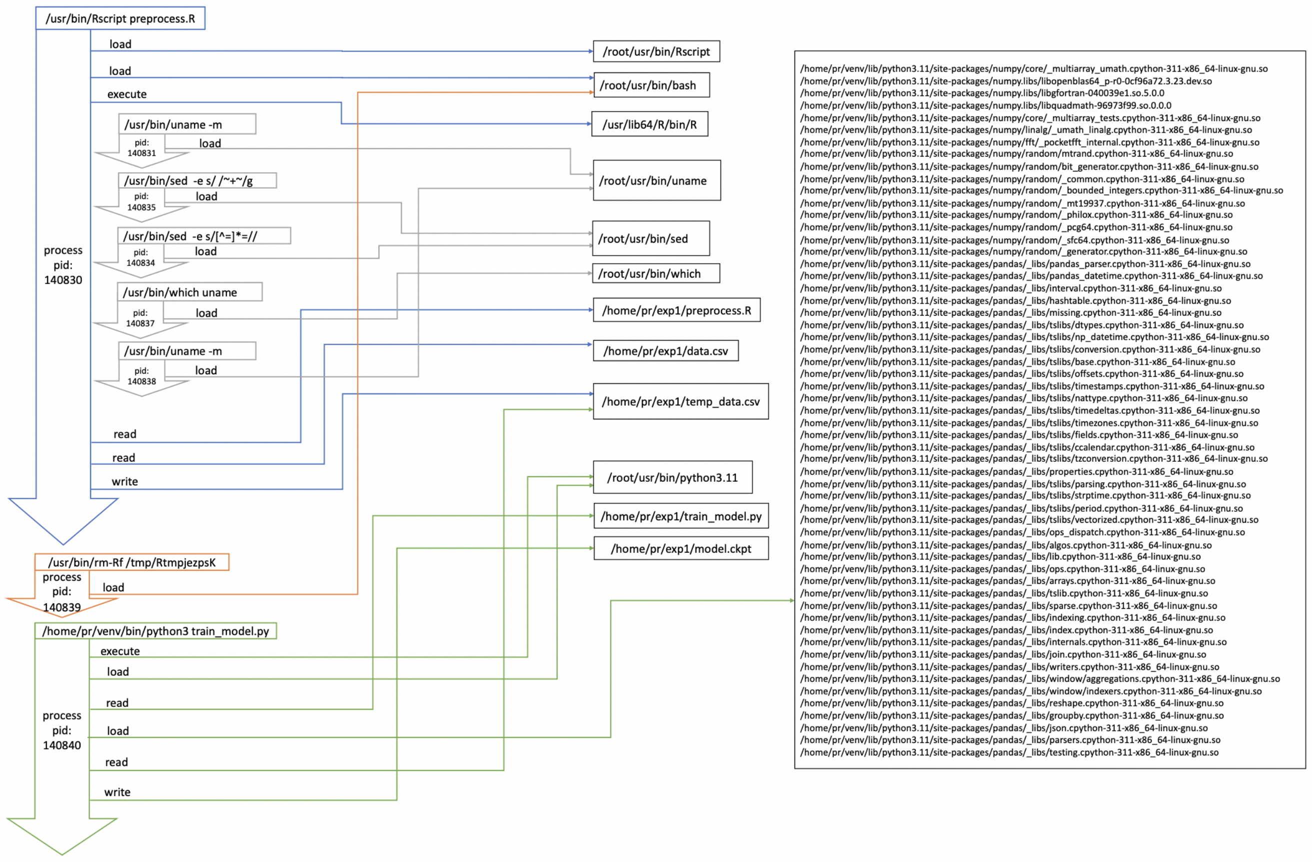

/usr/bin/Rscript preprocess.R
process
pid:
140830
load
load
execute
/usr/bin/uname -m
pid:
140831
load
/usr/bin/sed -e s/ /~+~/g
pid:
140835
load
/usr/bin/sed -e s/[^=]*=//
pid:
140834
load
/usr/bin/which uname
pid:
140837
load
/usr/bin/uname -m
pid:
140838
load
read
read
write
/usr/bin/rm-Rf /tmp/RtmpjezpsK
process
pid:
140839
load
/home/pr/venv/bin/python3 train_model.py
process
pid:
140840
execute
load
read
load
read
write
/root/usr/bin/Rscript
/root/usr/bin/bash
/usr/lib64/R/bin/R
/root/usr/bin/uname
/root/usr/bin/sed
/root/usr/bin/which
/home/pr/exp1/preprocess.R
/home/pr/exp1/data.csv
/home/pr/exp1/temp_data.csv
/root/usr/bin/python3.11
/home/pr/exp1/train_model.py
/home/pr/exp1/model.ckpt
/home/pr/venv/lib/python3.11/site-packages/numpy/core/_multiarray_umath.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy.libs/libopenblas64_p-r0-0cf96a72.3.23.dev.so
/home/pr/venv/lib/python3.11/site-packages/numpy.libs/libgfortran-040039e1.so.5.0.0
/home/pr/venv/lib/python3.11/site-packages/numpy.libs/libquadmath-96973f99.so.0.0.0
/home/pr/venv/lib/python3.11/site-packages/numpy/core/_multiarray_tests.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/linalg/_umath_linalg.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/fft/_pocketfft_internal.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/random/mtrand.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/random/bit_generator.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/random/_common.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/random/_bounded_integers.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/random/_mt19937.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/random/_philox.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/random/_pcg64.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/random/_sfc64.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/random/_generator.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/pandas_parser.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/pandas_datetime.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/interval.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/hashtable.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/missing.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/dtypes.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/np_datetime.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/conversion.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/base.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/offsets.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/timestamps.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/nattype.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/timedeltas.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/timezones.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/fields.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/ccalendar.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/tzconversion.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/properties.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/parsing.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/strptime.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/period.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/vectorized.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/ops_dispatch.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/algos.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/lib.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/ops.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/arrays.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslib.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/sparse.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/indexing.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/index.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/internals.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/join.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/writers.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/window/aggregations.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/window/indexers.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/reshape.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/groupby.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/json.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/parsers.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/testing.cpython-311-x86_64-linux-gnu.so

Figure 13: Workflow 2 Node Link Diagram

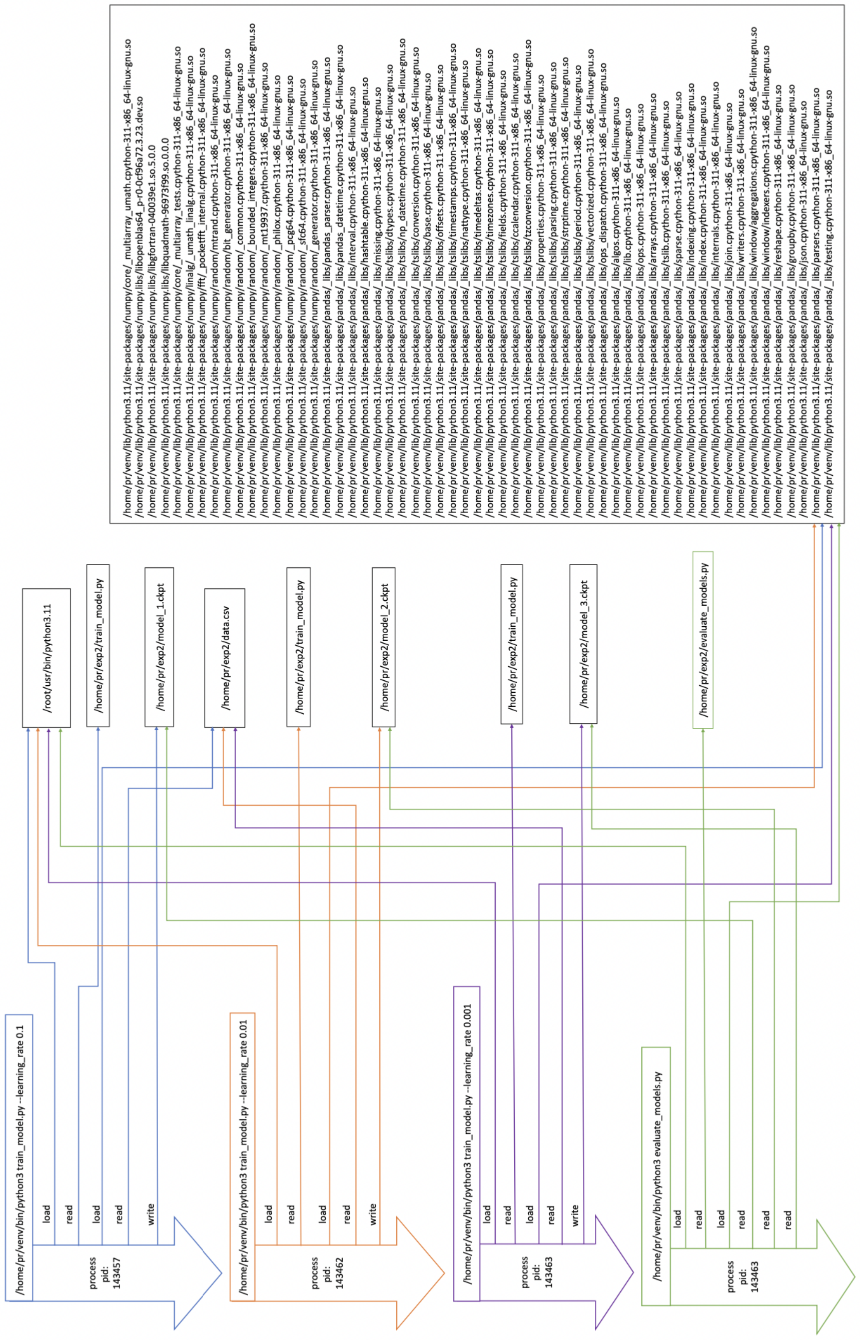

/home/pr/venv/bin/python3 train_model.py --learning_rate 0.1
process
pid:
143457
load
read
load
read
write
/home/pr/venv/bin/python3 train_model.py --learning_rate 0.01
process
pid:
143462
load
read
load
read
write
/home/pr/venv/bin/python3 train_model.py --learning_rate 0.001
process
pid:
143463
load
read
load
read
write
/home/pr/venv/bin/python3 evaluate_models.py
process
pid:
143463
load
read
load
read
read
read
/root/usr/bin/python3.11
/home/pr/exp2/train_model.py
/home/pr/exp2/model_1.ckpt
/home/pr/exp2/data.csv
/home/pr/exp2/train_model.py
/home/pr/exp2/model_2.ckpt
/home/pr/exp2/train_model.py
/home/pr/exp2/model_3.ckpt
/home/pr/exp2/evaluate_models.py
/home/pr/venv/lib/python3.11/site-packages/numpy/core/_multiarray_umath.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy.libs/libopenblas64_p-r0-0cf96a72.3.23.dev.so
/home/pr/venv/lib/python3.11/site-packages/numpy.libs/libgfortran-040039e1.so.5.0.0
/home/pr/venv/lib/python3.11/site-packages/numpy.libs/libquadmath-96973f99.so.0.0.0
/home/pr/venv/lib/python3.11/site-packages/numpy/core/_multiarray_tests.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/linalg/_umath_linalg.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/fft/_pocketfft_internal.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/random/mtrand.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/random/bit_generator.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/random/_common.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/random/_bounded_integers.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/random/_mt19937.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/random/_philox.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/random/_pcg64.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/random/_sfc64.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/random/_generator.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/pandas_parser.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/pandas_datetime.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/interval.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/hashtable.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/missing.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/dtypes.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/np_datetime.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/conversion.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/base.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/offsets.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/timestamps.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/nattype.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/timedeltas.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/timezones.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/fields.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/ccalendar.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/tzconversion.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/properties.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/parsing.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/strptime.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/period.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/vectorized.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/ops_dispatch.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/algos.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/lib.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/ops.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/arrays.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslib.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/sparse.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/indexing.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/index.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/internals.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/join.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/writers.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/window/aggregations.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/window/indexers.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/reshape.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/groupby.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/json.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/parsers.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/testing.cpython-311-x86_64-linux-gnu.so

Figure 14: Workflow 3 Node Link Diagram

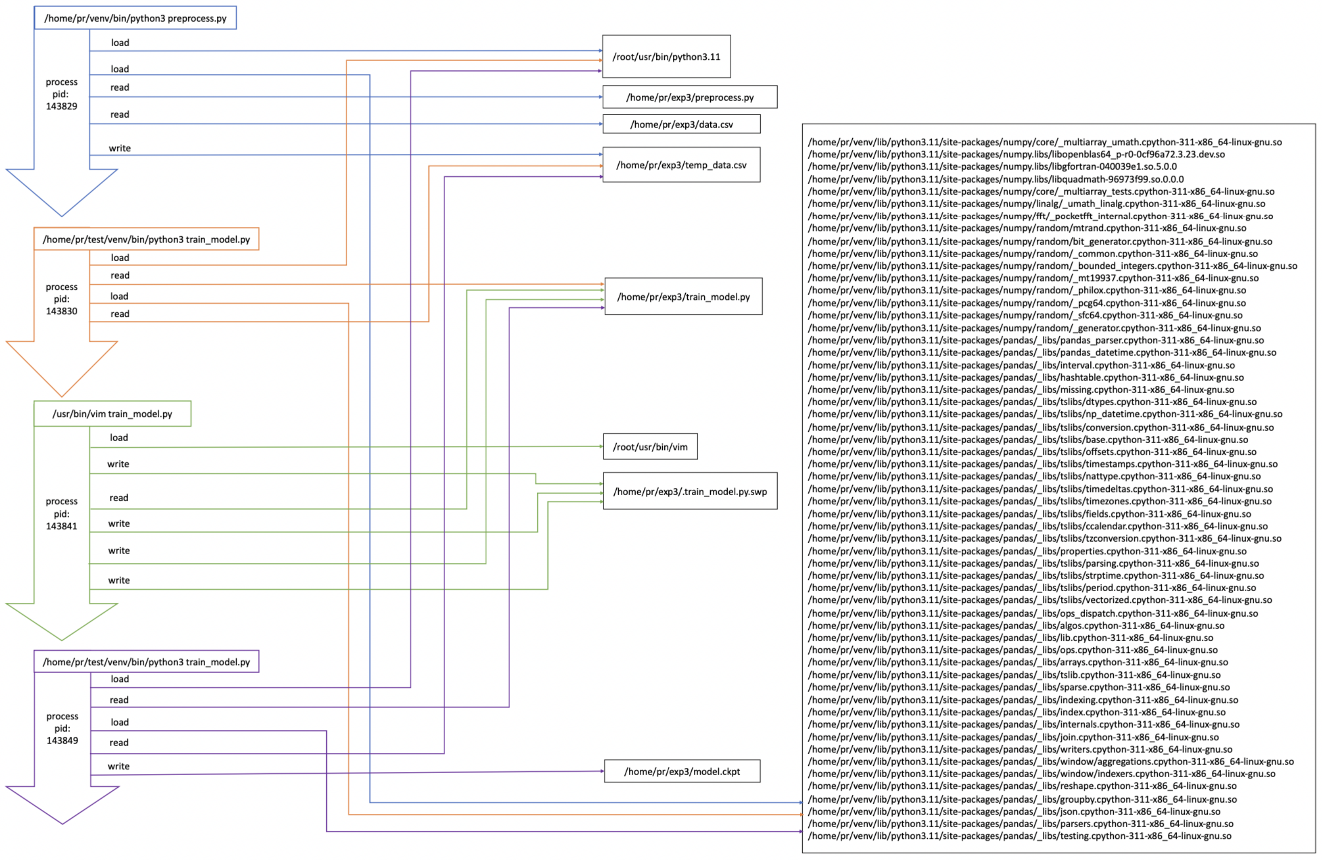

/home/pr/venv/bin/python3 preprocess.py
process pid: 143829
load
load
read
read
write
/home/pr/test/venv/bin/python3 train_model.py
process pid: 143830
load
read
load
read
/usr/bin/vim train_model.py
process pid: 143841
load
write
read
write
write
write
/home/pr/test/venv/bin/python3 train_model.py
process pid: 143849
load
read
load
read
write
/root/usr/bin/python3.11
/home/pr/exp3/preprocess.py
/home/pr/exp3/data.csv
/home/pr/exp3/temp_data.csv
/home/pr/exp3/train_model.py
/root/usr/bin/vim
/home/pr/exp3/.train_model.py.swp
/home/pr/exp3/model.ckpt
/home/pr/venv/lib/python3.11/site-packages/numpy/core/_multiarray_umath.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy.libs/libopenblas64_p-r0-0cf96a72.3.23.dev.so
/home/pr/venv/lib/python3.11/site-packages/numpy.libs/libgfortran-040039e1.so.5.0.0
/home/pr/venv/lib/python3.11/site-packages/numpy.libs/libquadmath-96973f99.so.0.0.0
/home/pr/venv/lib/python3.11/site-packages/numpy/core/_multiarray_tests.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/linalg/_umath_linalg.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/fft/_pocketfft_internal.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/random/mtrand.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/random/bit_generator.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/random/_common.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/random/_bounded_integers.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/random/_mt19937.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/random/_philox.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/random/_pcg64.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/random/_sfc64.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/numpy/random/_generator.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/pandas_parser.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/pandas_datetime.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/interval.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/hashtable.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/missing.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/dtypes.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/np_datetime.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/conversion.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/base.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/offsets.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/timestamps.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/nattype.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/timedeltas.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/timezones.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/fields.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/ccalendar.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/tzconversion.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/properties.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/parsing.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/strptime.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/period.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslibs/vectorized.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/ops_dispatch.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/algos.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/lib.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/ops.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/arrays.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/tslib.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/sparse.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/indexing.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/index.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/internals.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/join.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/writers.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/window/aggregations.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/window/indexers.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/reshape.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/groupby.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/json.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/parsers.cpython-311-x86_64-linux-gnu.so
/home/pr/venv/lib/python3.11/site-packages/pandas/_libs/testing.cpython-311-x86_64-linux-gnu.so

Figure 15: Workflow 4 Node Link Diagram