Computational Experiment Comprehension using Provenance Summarization

by

Nichole Boufford

B.Sc., University of British Columbia, 2021

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

(Computer Science)

The University of British Columbia

(Vancouver)

March 2024

© Nichole Boufford, 2024
The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

**Computational Experiment Comprehension using Provenance Summarization**

submitted by Nichole Boufford in partial fulfillment of the requirements for the degree of **Master of Science** in **Computer Science**.

**Examinining Committee:**

Thomas Pasquier, Assistant Professor, Computer Science, UBC  
*Supervisor*

Margo Seltzer, Professor, Computer Science, UBC  
*Supervisory Committee Member*
Abstract

Scientists often use complex multistep workflows to computationally analyze data. These workflows might include downloading datasets, installing packages, data processing, model training and evaluating the results. It is difficult to effectively manage and track these computational workflows. The fast-paced and iterative nature of research programming leads these workflows to include unused code, multiple versions of the same script, and untracked dependencies. These issues cause difficulties when researchers try to reproduce code that someone else has written, or even code that they have written themselves. Research programmers can address these problems by collecting data provenance: a record of what happened during an experiment, including files touched, execution order, and software dependencies. Provenance provides a record of experiment execution, but provenance graphs are often large and complicated, and quickly become incomprehensible. We propose a new method for summarizing provenance graphs using recent advances in prompting large language models. We use large language model prompting to develop textual summaries of provenance graphs. We perform a user study to compare textual summaries to traditional node-link diagrams for experiment reproduction tasks. Our results show that textual summaries are a promising approach to summarizing provenance for experiment reproduction. We use qualitative results from the user study to motivate future designs for reproducibility tools.
Lay Summary

Historically, scientists across disciplines use a lab notebook to document their experimental procedures. In doing so, they make their experiments reproducible. Scientists now program computers to do experiments, but the concept of a lab notebook translates poorly to the computational setting. To solve part of this problem, researchers use software tools to collect data that describes their experiment’s execution. This is a step in the right direction, but capturing the record is only half of the problem. Conventionally, researchers present this data as a graphical visualization, but this format can be cognitively overwhelming. We present a natural language format for experiment reproducibility. In our user study, we evaluate automatically generated natural language summaries with the conventional graphical diagrams. We demonstrate that users that do not have a strong background in computer science prefer the natural language format. Our results motivate future reproducibility tool design.
Preface

This thesis is an original work by Nichole Boufford, under the supervision of Dr. Thomas Pasquier. All the work presented henceforth was conducted collaboratively in the Systopia Lab at the University of British Columbia, Point Grey campus. A version of Chapter 4 and Chapter 5 will be published in the 2024 ACM Conference on Reproducibility and Replicability. I was the lead investigator where I was responsible for all major areas of concept formation, data collection and analysis, and manuscript composition. The user study in Chapter 5 and associated methods were approved by the University of British Columbia’s Research Ethics Board [certificate #H23-00382]. Joseph Wonsil and I prepared the materials for the user study in Chapter 5. Joseph Wonsil, Adam Pocock and Jack Sullivan were involved in concept formation and contributed to paper edits. Thomas Pasquier and Margo Seltzer were the supervisory authors on this project and were involved throughout the project in concept formation and manuscript edits.
# Table of Contents

Abstract ................................................................. iii  
Lay Summary .............................................................. iv  
Preface ................................................................. v  
Table of Contents ......................................................... vi  
List of Tables .......................................................... ix  
List of Figures .......................................................... x  
List of Abbreviations .................................................. xiii  
Acknowledgments ......................................................... xiv  

1 Introduction ........................................................... 1  

2 Background .............................................................. 4  
  2.1 Provenance ......................................................... 4  
  2.2 Defining Reproducibility ......................................... 5  

3 System Provenance ..................................................... 6  
  3.1 Background ......................................................... 8  
    3.1.1 System Provenance Capture ............................... 8  
    3.1.2 Extended Berkeley Packet Filter (EBPF) ................. 9  
  3.2 Design ............................................................ 11
A Supporting Materials ........................................ 57
  A.1 Availability ............................................. 57
  A.2 Diagram Creation ........................................ 57
  A.3 Task Load Index Questions ............................. 58
  A.4 Study Activities ........................................ 59
    A.4.1 Questions ........................................... 59
    A.4.2 Text Summaries .................................... 60
    A.4.3 Node Link Diagrams ................................. 60
List of Tables

Table 3.1  Thoth provenance format in accordance with the World Wide Web Consortium (W3C) PROV Data Model [7]. . . . . . . . . . 12
Table 3.2  EBPF hooks and their corresponding provenance graphs. . . . . 13
Table 3.3  Thoth command line interface description. . . . . . . . . . . . 14
Table 5.1  Participant Fields . . . . . . . . . . . . . . . . . . . . . . . . 31
Table 5.2  Workflow description for each task in the user study. . . . . . 32
List of Figures

| Figure 3.1 | EBPF programs are compiled and verified before being loaded into the kernel. EBPF hooks trigger the programs. EBPF maps are shared data structure for communication between userspace and kernel programs. |
| Figure 3.2 | Thoth is split into a userspace program, an eBPF kernel program, and a command line interface (CLI). The userspace program communicates with the EBPF programs in the kernel using EBPF maps. The user communicates with Thoth via the CLI, and the userspace program writes the provenance log to a local file. |
| Figure 3.3 | Simple provenance graph using our data format. Process 1 executes the example.py script using Python 3.11. This script reads data.csv and writes to output.data. |
| Figure 3.4 | Example usage of Thoth provenance collection. This workflow will produce the graph in Figure 3.3. |
| Figure 4.1 | 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. |
| Figure 4.2 | Steps to create a text summary of a computational experiment. |
Figure 4.3 A provenance log in machine-readable JSON format is converted to natural language format. The machine-readable JSON format size is 88 tokens and the natural language log size is 30 tokens.

Figure 4.4 The prompt we used to generate our user study text summaries. *Input provenance log* denotes the log specific to each task.

Figure 4.5 A basic prompt and the GPT generated summary. In this example we use GPT 3.5, without preprocessing on the provenance log. The summary lacks focus, clarity and understanding.

Figure 4.6 A prompt using the conversation history to provide examples. The first two instruction/response pairs are manually generated and the final summary is generated by the GPT-3.5 model. *Input provenance log 1, 2 and 3* denote provenance logs for three different tasks.

Figure 5.1 Workflow 1 Text Summary

Figure 5.2 Workflow 1 Node Link Diagram (cropped image, full image in Section A.4)

Figure 5.3 Quantitative metrics showing performance using both graph (orange) and text (blue) provenance summaries.

Figure 5.4 Participants rank their preference for either the text summarization or node link diagram. Participants are categorized by their computational expertise.

Figure 5.5 Participants rank their preference for either the text summarization or node link diagram in the remote study. Participants are categorized by their computational expertise.

Figure 6.1 We compare the most common node-link visualization (6.1a) with two alternative approaches (6.1c, 6.1b).

Figure A.1 Workflow 1 Text Summary

Figure A.2 Workflow 2 Text Summary

Figure A.3 Workflow 3 Text Summary

Figure A.4 Workflow 4 Text Summary
Figure A.5  Workflow 1 Node Link Diagram . . . . . . . . . . . . . . . 64
Figure A.6  Workflow 2 Node Link Diagram . . . . . . . . . . . . . . . 65
Figure A.7  Workflow 3 Node Link Diagram . . . . . . . . . . . . . . . 66
Figure A.8  Workflow 4 Node Link Diagram . . . . . . . . . . . . . . . 67
# List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLI</td>
<td>command line interface</td>
</tr>
<tr>
<td>EBPF</td>
<td>Extended Berkeley Packet Filter</td>
</tr>
<tr>
<td>LLM</td>
<td>large language model</td>
</tr>
<tr>
<td>LSM</td>
<td>Linux Security Module</td>
</tr>
<tr>
<td>W3C</td>
<td>the World Wide Web Consortium, the standards body for web technologies</td>
</tr>
</tbody>
</table>
Acknowledgments

Thank you to everyone who has supported me during this chapter of my life.

Firstly, I am thankful for my supervisor, Thomas Pasquier, for guiding me during this degree. Your advice and support, even before officially becoming my supervisor, made graduate school a rewarding and positive experience. I would also like to thank Margo Seltzer for her support throughout my entire research experience, from undergrad to now. Thomas and Margo, your commitment to your students goes beyond research and I cannot thank you enough.

Thank you to Joe Wonsil for a few years of awesome collaboration. You are not only a talented researcher but an incredible friend.

Thank you to Adam Pocock and Jack Sullivan for your collaboration and support. I am not lying when I say I will truly miss the provenance meetings!

Thank you to everyone in Systopia and students of the CS department. I could not have asked for better labmates to work, ski, run, drink coffee and celebrate with.

Lastly, I am filled with gratitude for all of my friends and family. Thank you to Brianne for your support and for always helping me to procrastinate. There is no one I would rather [ski, run, hike, crochet, thrift] all day with. Thank you to my parents for your unconditional support and love. Thank you to Ryan for many things, including always being my first reader and source of encouragement.

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 g’énie du Canada (CRSNG) de son soutien.
Chapter 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. 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 [47, 61, 62]. 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. Manually recording everything that happened during a computational experiment is extremely tedious, if not impossible. As such, researchers use many tools that help with mechanical experiment reproducibility such as code repositories [12], dataset repositories [31, 34] and computational environment trackers [15, 50]. 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 [44].
Provenance is metadata describing the history of data objects [11]. The data objects are often scripts and datasets, whose attributes might contain version information, and their relationships describe dependence or causality. 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.

As provenance is graph-structured data, researchers typically visualize provenance data as a directed acyclic graph. The node-link diagram is by far the most common way to present this visualization [6, 24, 27, 33, 46, 54]. Although visual diagrams are meant to help users to understand their data, this is often not the case. Provenance graphs are large and complicated, making them difficult for users to comprehend. They often contain more information than a human can store in their working memory and thus impose too high a cognitive load [37].

Since users have difficulty understanding provenance data, provenance on its own does not inherently help users to understand experiments. Some prior work investigates alternative provenance representations [9, 52], but they still rely on graphical representations of provenance data. We hypothesize that explaining what a provenance graph represents is a better approach to facilitate a user’s understanding of the experiment.

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.

Overall, we address several challenges in experiment comprehension and re-
producibility. We give an overview of data provenance in the context of reproducibility in Chapter 2. In Chapter 3, we describe our system provenance capture tool. We show that by designing system provenance collection with experiment tracking in mind, we can accurately describe a computational workflow. Additionally, we explore text-based provenance summarization to help users understand experiments. We discuss our provenance summarization technique using large language models in Chapter 4. In Chapter 5 we conduct a user study to compare our text-based provenance representation with the traditional node-link diagram. We discuss related works in Chapter 6 and conclude in Chapter 7.

The work in this thesis will appear in the 2024 ACM Conference on Reproducibility and Replicability. Our code, datasets, and study materials are publicly available (further details in Section A.1).
Chapter 2

Background

We begin with a brief background on provenance and provide definitions that will be used throughout the remainder of this thesis. In the following chapters, we provide more specific background sections as appropriate.

2.1 Provenance

Provenance is a record that describes what has happened to a piece of data [11]. This includes when the data object was created, who has created/modified it, and how it has been modified. As such, provenance collection is an effective mechanism to record steps in a computational experiment.

Previous work used provenance to capture experiment protocols at various levels of abstraction. Muniswamy-Reddy et al. describes application and system level provenance systems - the prior as systems that record provenance at the semantic level of the target application and the latter as provenance collection of operating system semantics. In this work, we use application provenance to refer to provenance describing user applications. We use system provenance to refer to provenance describing operating system level activities.

Researchers most frequently use application provenance to record experiments. Systems such as Vistrails [6] and Genepattern [51] are some of the first systems to integrate provenance into computational research pipelines. These systems provide integrated reproducibility but require users to perform their entire analysis
within the tool’s ecosystem. In reality, researchers use many tools and programs. More recently, machine learning applications are including provenance in training pipelines to record data versions, tuning parameters, and random seeds. For example, Merit [63] uses provenance in a Java machine learning framework to reproduce machine learning pipelines. Again, these tools provide part of the solution by tracking code execution within an application. Burrito integrates system provenance and application provenance to provide experiment tracking across a system [20]. Burrito shares similar goals to our own, acknowledging the diverse experiment ecosystem, although, it still requires developer effort to add provenance collection for each application. We further describe prior work on provenance systems and experiment tracking in Chapter 3.

### 2.2 Defining Reproducibility

We use the definition of reproducibility from the National Academies of Science, Engineering and Medicine [39]. “Reproducibility is obtaining consistent results using the same input data, computational steps, methods, code, and conditions of analysis.” When we described tracking a workflow in the context of experiment reproducibility, we mean recording and understanding each step that a user performed to achieve a result.
Chapter 3

System Provenance

Scientists use a variety of tools for research programming, many of which are designed for software development rather than research. While software engineering tools, such as code repositories [12], dataset repositories [31, 34] and computational environment trackers [15, 50], play a crucial role in reproducibility, they do not capture some critical information. To reproduce an experiment, we need the dataset, the code, the computational environment and some form of instructions. Software engineering tools provide code and data and sometimes environment parameters that help to recreate the computational environment. Although these tools help with reproducibility, they fail to capture an execution record: the steps the researcher took to perform their experiment. These missing steps include the order the researcher ran their experiment code, intermediate data files, and intermediate code versions that were executed but did not get committed to version control.

Consider a chemist performing an experiment. To prepare their experiment, they have a list of reagents, equipment, and a written experimental procedure. We can compare these materials to the code and the data in a computational experiment. During their experiment, the chemist writes notes that describe exactly how they performed the experiment in practice. These notes include the exact measurements of the reagents, the room temperature, any variation from the expected procedure, and visual observations. Without these experiment notes, another scientist would not be able to exactly reproduce the experiment. They might be able to follow the planned procedure, but a small adjustment to reagent amount or the tem-
perature can drastically change the results in practice. We liken these experiment notes to the execution record in computation.

In computational sciences, it would be extremely tedious for a researcher to track this information manually due to the fast-paced nature of research programming. Further, the responsibility falls onto the researcher to ensure they frequently commit their code and follow reproducibility best practices [39].

Provenance addresses software engineering tools’ limitations by providing a record of the steps taken during a computational experiment. It can be captured automatically, mitigating the pressure on the researcher to manually record their steps. But, existing provenance tools have limitations in the context of experiment reproducibility and are not well adopted.

Application and language level provenance lack awareness of activities across different applications. Since application provenance is limited to the application that collects the provenance, any activity outside the application is not captured. For example, a researcher might be performing a Python analysis and using noWorkflow [46] to record provenance, but they notice that one of the headers in their dataset has a spelling error. So, they open the data in a spreadsheet application and edit the column name before continuing their analysis. noWorkflow does not capture this edit outside the Python environment, and this could lead to mistakes in reproducing the experiment later on.

Another challenge that most provenance applications face is that they are not easily integrated into existing workflows. Setting up a new project environment or learning to use a new tool can take away valuable time from a researcher’s domain work. We found in our user study (Chapter 5) that users do not want to use a tool that requires large changes to their existing workflow, or is time-consuming to set up.

The aforementioned limitations highlight the need for specialized provenance capture to benefit research programmers. For our study on provenance summarizations in the following chapters, we required a system provenance tool to capture multifaceted workflows. There are many system provenance collection tools, but none provide everything we require for experiment tracking. For reproducible experiments, we not only require an execution record, but the tool that provides the capture must be easy to install and generate data relevant to the experiment at
hand, rather than the whole system. The key challenges with using existing system provenance tools for scientific experiment tracking are that most tools are difficult to integrate, and they collect data at a different granularity than that needed for reproducibility.

To address these limitations, we developed Thoth, a system provenance capture tool for experiment reproducibility. Thoth addresses the multi-application issue by capturing operating system interactions, including file operations and network connections. Unlike prior work, these operating system interactions are limited to only those objects used in the experiment. We use the Extended Berkeley Packet Filter (EBPF) so that Thoth requires minimal developer effort to install and use. The design and implementation are based on previous work in provenance collection using eBPF [29] that we tailored to our reproducibility use case. We use Thoth as the provenance collection tool for our study on provenance summarizations for computational reproducibility in Chapter 4 and Chapter 5. 1

3.1 Background

Rather than use existing provenance collection tools, we designed Thoth specifically for experimental workflow tracking. We discuss limitations of existing provenance capture tools that motivate our design.

3.1.1 System Provenance Capture

System provenance tools are often used for security and auditing applications because they provide a view of an entire machine [5, 21, 32]. These specialized tools capture large amounts of data, on the order of thousands of nodes and edges, to cover all system events. Whole-system provenance capture tools such as Camflow [43] and the PASS system [35] can be used for a variety of applications but still capture extensive data across a machine. While whole-system provenance is useful for applications that require detailed visibility, experiment reproducibility does not require such extensive tracking, especially if the experiment code touches only a few code and data files.

Many provenance tools require significant effort to install and use. There are

1The Thoth code is available at https://github.com/ubc-systopia/thoth.
EBPF programs are compiled and verified before being loaded into the kernel. EBPF hooks trigger the programs. EBPF maps are shared data structure for communication between userspace and kernel programs.

**Figure 3.1:** EBPF programs are compiled and verified before being loaded into the kernel. EBPF hooks trigger the programs. EBPF maps are shared data structure for communication between userspace and kernel programs.

Various mechanisms for system provenance capture such as using Linux Security Modules [43, 49] and Linux tracepoints [32]. These systems require users to use custom operating systems [32, 35, 43, 49], which are difficult to maintain. Further, non-computer scientists that want to use provenance tools may not have the time or expertise to debug such programs. As such, we developed a more lightweight provenance collection tool that is simpler to install and use.

### 3.1.2 EBPF

EBPF is a Linux framework that allows users to run sandboxed programs in the kernel [2]. Compared to kernel modules or custom kernels, EBPF programs are lightweight and safe. EBPF is available on generic Linux kernels and therefore users need not modify their kernel to run EBPF programs. Consequently, EBPF is
Figure 3.2: Thoth is split into a userspace program, an eBPF kernel program, and a command line interface (CLI). The userspace program communicates with the eBPF programs in the kernel using eBPF maps. The user communicates with Thoth via the CLI, and the userspace program writes the provenance log to a local file.

EBPF is commonly used for auditing, logging and network filtering [2]. Figure 3.1 shows the EBPF program ecosystem. EBPF achieves safety through static verification. Each EBPF program is verified for safety before the final compilation to ensure that the program is safe to run in the kernel. After verification, EBPF programs are JIT compiled into machine code to enhance performance. EBPF programs run when certain hooks are triggered. This event-driven nature makes EBPF programs well-suited for kernel observability. EBPF programs communicate with userspace programs through shared data structures called EBPF maps.

ProvBPF [29] and eAudit [53] use EBPF [2] as a lightweight solution to collect whole-system provenance. Similarly, we chose to use EBPF to implement our provenance collection mechanism because it is a lightweight alternative to using a custom kernel. In contrast to ProvBPF and eAudit, our system captures only a subset of provenance limited to a user-specified project directory. We describe our implementation using EBPF in further detail in Section 3.3.
3.2 Design

The main goals for our system are to reduce barriers to install and use and to only generate data relevant to the project at hand. These goals are based on the challenges outlined in Section 3.1.

Our first goal is to enable users to use generic operating systems without modifications to decrease the barrier for use. More so, we want to avoid the set-up cost of adding kernel modules or compiling a custom kernel. Users can run EBPF programs using generic Linux distributions such as Ubuntu and Fedora. We also want minimal disruption to the user’s workflow after installation. Since provenance capture can occur in the background, a user should specify only their directory of interest and leave the rest up to the tool to capture their workflow. We model our collection after ProvBPF [29] and eAudit [53], but rather than targeting security we target experiment reproducibility. Therefore, our goals and the data we collect are different.

Our second goal focuses on developer effort and data storage. We designed our provenance capture system to track only information that is useful in the context of experiment reproducibility. It is common in research programming to have experiment artifacts in a project directory. This directory contains experiment code and data, and it is the directory that would likely be tracked using version control. We eliminate the extraneous data that we would get from whole-system provenance capture by capturing only those system events that occur inside the project directory.

Rather than capturing all system events and filtering the data, we capture only the data we need. The minimum data we need to describe simple experiments are file operations, user commands, and shared libraries. For example, a user executes a Python script from the command line to perform data processing. Thoth tracks the Python file, the dataset, the command they used to run the script, libraries versions and outputs. We also include network activity to gain information about file downloads and network connections. Figure 3.2 shows our overall system design. Users are presented with a simple command line interface (CLI). Thoth uses EBPF programs to log kernel events and log provenance data to a local file. We outline extensions to our system in Section 3.4.
Table 3.1: Thoth provenance format in accordance with the World Wide Web Consortium (W3C) PROV Data Model [7].

<table>
<thead>
<tr>
<th>Vertices</th>
<th>PROV Data Model Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Object</td>
<td></td>
</tr>
<tr>
<td>Process</td>
<td>Activity</td>
</tr>
<tr>
<td>File</td>
<td>Entity</td>
</tr>
<tr>
<td>Socket</td>
<td>Entity</td>
</tr>
<tr>
<td>Command</td>
<td>Entity</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Edges</th>
<th>PROV Data Model Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Event</td>
<td></td>
</tr>
<tr>
<td>lsm/file_permission (read)</td>
<td>Used</td>
</tr>
<tr>
<td>lsm/file_permission (write)</td>
<td>WasGeneratedBy</td>
</tr>
<tr>
<td>lsm/bprm_creds_for_exec</td>
<td>Used</td>
</tr>
<tr>
<td>tracepoint/syscalls/sys_enter_execve</td>
<td>WasGeneratedBy</td>
</tr>
<tr>
<td>lsm/socket_connect</td>
<td>WasGeneratedBy</td>
</tr>
<tr>
<td>lsm/mmap_file</td>
<td>Used</td>
</tr>
</tbody>
</table>

3.2.1 Provenance Data Model

We represent system events as a directed acyclic graph. Our provenance data model is based on the W3C PROV data model [7]. In the PROV data model, entities represent data objects and activities represent actions. In our model, activities represent processes/tasks. The entities we capture are files, sockets, and commands. Edges correspond to system events and are named according to the EBPF hook that causes them to be captured. Table 3.1 describes our data model as it corresponds to the W3C PROV data model.

We use SPADE JSON format for our output [18]. This format consists of an array of JSON objects, where each JSON object represents either a node or an edge. We use this format to simplify integration with provenance querying and viewing applications such as SPADE [18]. Developers can use SPADE to import Thoth provenance directly into a Neo4J database to query or visualize the provenance.
Table 3.2: EBPF hooks and their corresponding provenance graphs.

<table>
<thead>
<tr>
<th>Hook</th>
<th>Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>lsm/file_permission (read)</td>
<td><img src="process-read-file" alt="Graph" /></td>
</tr>
<tr>
<td>lsm/file_permission (write)</td>
<td><img src="process-write-file" alt="Graph" /></td>
</tr>
<tr>
<td>lsm/bprmcreds_for_exec</td>
<td><img src="process-execute-file" alt="Graph" /></td>
</tr>
<tr>
<td>tracepoint/syscalls/sys_enter_execve</td>
<td><img src="process-took-command-command" alt="Graph" /></td>
</tr>
<tr>
<td>lsm/socket_connect</td>
<td><img src="process-connect-socket" alt="Graph" /></td>
</tr>
<tr>
<td>lsm/mmap_file</td>
<td><img src="process-map-memory-file" alt="Graph" /></td>
</tr>
</tbody>
</table>

In Table 3.2, we show the graphs for each kernel hook. Figure 3.3 shows a small example graph generated in this format.

3.3 Implementation

We implemented and tested our system using Fedora 37 on a virtual machine with 8GB RAM. We used the EBPF framework to implement our provenance capture tool.
Figure 3.3: Simple provenance graph using our data format. Process 1 executes the example.py script using Python 3.11. This script reads data.csv and writes to output.data.

Table 3.3: Thoth command line interface description.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>Starts Thoth daemon.</td>
</tr>
<tr>
<td>stop</td>
<td>Stops Thoth daemon.</td>
</tr>
<tr>
<td>track-dir</td>
<td>Specifies directory to be tracked. Thoth daemon must be started before running this command.</td>
</tr>
<tr>
<td>remove-dir</td>
<td>Stop tracking given directory.</td>
</tr>
</tbody>
</table>

3.3.1 User Interface

We designed our user interface to require minimal disruption to a user’s workflow. The user interacts with a command line interface to start and stop provenance collection, and to specify the project directory. Table 3.3 specifies the command line interface. After installing the tool, the user starts the collection daemon and specifies which directory to track through the command line interface. After this, provenance collection occurs in the background as a system daemon process. Provenance is saved in a local file that the user can view or import into a provenance
Figure 3.4: Example usage of Thoth provenance collection. This workflow will produce the graph in Figure 3.3.

viewing application. Figure 3.4 demonstrates how a user interacts with Thoth from the command line to produce the provenance graph in Figure 3.3. More commonly, a user might start the Thoth daemon when they start working on their project and stop the daemon after they have finished working for the day.

3.3.2 EBPF Programs

We wrote an eBPF program for each kernel hook described in Table 3.2. When the system call is invoked, the kernel observes that there is an eBPF program at the hook point and then invokes the program with the current parameters. The eBPF program then records the provenance for this system call in an eBPF map. The userspace program reads the provenance data from the eBPF map, formats the log and writes it to a local file. The userspace also use eBPF maps to communicate information to the kernel program such as which directories to track.

File Operations

We use eBPF Linux Security Module (LSM) hooks to track file operations. The eBPF program records the process ID, file path, and operation type (read, write, execute, mmap) to the provenance log. We use inode numbers to identify files and process IDs to identify processes.

Commands

We use the “execute program” system call hook to record commands that users execute on the command line. We record the process ID, pathname of the executable, and arguments. The command provenance subgraphs help to decipher the purpose of the file operations.
Network Activity

We also use Linux Security Module (LSM) hooks to track network activity. Table 3.2 shows the possible relationships. Sockets are identified by the IP address, port, and protocol. The EBPF program records the process ID, socket ID, and operation type to the provenance log. We found that we did not require extensive network tracking for our experiment execution and thus record only socket connection operations. This gives us the IP address of network connections that are initiated from within the experiment directory. This is useful for experiment tracking if a user downloads a dataset or runs a notebook on a local server.

3.4 Discussion

We were successful in using Thoth to collect provenance for our evaluation and user study in the following chapters. There are also several ways that Thoth provenance can be extended that would further benefit reproducible science.

We envision adding an annotation framework to the Thoth provenance system. Often there is context that cannot be captured through provenance that would be useful for experiment tracking, such as the reason for editing a file (i.e., to fix a bug, add to the analysis). We imagine adding a note object that allows a user to write a note and insert it into the provenance graph. The note can specify the purpose of the experiment or assign a contextual label to a subgraph. An example is labeling the subgraphs that represent each of: downloading data, processing the data and then running the analysis. Using the PROV data model [7], one could assign a “start” event at the beginning of a subgraph and an “end” event at the end of the subgraph. These notes would provide contextual information to a researcher who is reproducing the experiment. We also imagine that with enough manual annotations, one could train a model to automatically assign contextual notes to specific patterns in the provenance.

In reality, multiple layers of provenance are needed to capture an entire experiment. We also envision layering Thoth provenance with other provenance applications to create a full picture of the system. Muniswamy-Reddy et al. proposed layering provenance [36]. While we focus on system level provenance, we keep in mind the ability to extend our system to include application provenance.
In the following chapters we discuss how we applied Thoth to experiment reproducibility tasks and evaluated various representations of Thoth provenance.
Chapter 4

Provenance Summarization

System provenance capture provides researchers with a record of their experiments, addressing the issue of mechanic reproducibility, which is a necessary, but not sufficient, part of automatic reproducibility. Tools that use data provenance typically present it visually as a directed acyclic graph that represents the relationships between data objects. Unfortunately, since provenance graphs are large and complicated, even experienced software engineers find interpreting these graphs challenging [9].

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 (Figure 4.1) [6, 24, 27, 33, 46, 54]. Large graphs are difficult for humans to understand [37]. They often contain more information than a human can store in their working memory and thus, impose too high a cognitive load.

Imagine a researcher who is returning to an analysis they had written a few months prior for a paper. They need to rerun their experiment to produce the graph for the final version of their paper. They used version control, a data repository, an environment manager, and still, they have trouble reproducing their analysis. Even though their scripts run without errors, the researcher has lost track of the order in which they executed the scripts. To make matters worse, they did not save a copy of their dataset. As a result, they must redo the preprocessing, which involved some manual editing in a spreadsheet editor.
Figure 4.1: 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.

Luckily the researcher collected provenance during their last experiment execution. But the provenance log is machine-readable only and not so helpful. They use a graph database to view the provenance, but there are hundreds of nodes! It is going to take them just as long to sift through the provenance as it would to solve the problem through trial and error.

This scenario illustrates 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 [9, 52], 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 demonstrate how we automatically generate such summaries using large language models. The work in this chapter will appear in the 2024 ACM Conference on Reproducibility and Replicability. We evaluate our summarization technique in Chapter 5, where we discuss our user study.
4.1 Background

We examine provenance visualization and summarization across domains to give context to our work and discuss the current state of the art in LLM text summarization.

4.1.1 Provenance Visualization

Most provenance applications, including those used for experiment and code management, present provenance as a node-link diagram [6, 24, 27, 33, 46, 54]. 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 [68] as they cannot fit in a human’s working memory [37]. Alternative provenance visualizations [9, 52] have failed to see meaningful adoption (see Chapter 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.

4.1.2 Summarization using Generative AI

Recent work in generative artificial intelligence shows that LLMs are able to effectively summarize large quantities of text [19, 71]. 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 [72]. 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 [25, 30, 55, 58]. LLMs require careful prompting to generate useful responses [69]. LLM responses
are sometimes verbose, redundant, and unclearly organized. Additionally, with current generative AI models, we cannot guarantee response correctness [10]. 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 [41], 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 [41]. This means that the context is limited to approximately 32,000 characters.

4.2 Summarization Using Large Language Models

We generate several high-quality summaries using GPT-4 [41] as a proof of concept for our user study. Figure 4.2 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 [41] to generate the summaries for our user study ④. The LLM-generated summaries should contain 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 Section 4.2.3.

① Provenance capture We use Thoth, our system described in Chapter 3, to capture provenance during experiment. We call the provenance data captured by Thoth a provenance log.
4.2.1 Data Preprocessing

For most LLMs, including GPT-4 [41], 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 [30].

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 important than they are, and the natural language format aligns more closely with an LLM's training corpus than does the JSON output.

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 consecutive writes. 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. [66] 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. Figure 4.3 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 Section 4.2.3. 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 Section A.1).
4.2.2 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 [19].

Prompt Engineering

We use GPT-4-0613 [41], 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 Figure 4.4, 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 Section 4.2.3.

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 introduce interesting prose and more high-level descriptions, but the responses were inconsistent and more likely to contain false information. A temperature of 0 means that the model will choose the next word with the highest probability each time, although due to non-determinism in GPU calculations there may be slight discrepancies. In our experience, setting the temperature to 0, we get responses that are nearly the same each time, differing by only few words, if any.

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 [60, 64]. 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 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 Section 4.2.3.

4.2.3 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 [14]. 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 [58]. 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 Figure 4.4 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
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
Figure 4.6: A prompt using the conversation history to provide examples.

The first two instruction/response pairs are manually generated and the final summary is generated by the GPT-3.5 model. *Input provenance log 1, 2 and 3* denote provenance logs for three different tasks.
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. Figure 4.6 shows an
example summary that we generated 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 Figure 4.4.

4.3 Discussion

Although we cannot guarantee perfect summaries using current models, our posi-
tive 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 bet-
ter still. For our user study, we generated text summaries using the out-of-the-box
GPT-4 model from OpenAI [41] 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 train-
ing a pre-trained model on a smaller task-specific dataset. We envision using our
evaluation criteria from Section 4.2.3 in this task-specific dataset. Additionally,
LLM providers such as OpenAI and Meta are frequently releasing improved mod-
els with larger context windows [59]. At the time of writing, OpenAI has already
released a new version of GPT-4 with a 128K context window [23]. As such, we
are optimistic for a path forward.

We also note that there are several alternative techniques one could use to auto-
matically generate provenance summaries. The simplest method is to use rule-base
methods to populate text fields in a structured report [56]. 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
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 [40]. 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 who 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.
Chapter 5

User Study

We conduct a user study to evaluate whether users are 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 summaries. We then analyze qualitative feedback through long answer text responses and audio recordings. Section A.4 contains our study materials.

5.1 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 5.2. We used the single prompt approach described in Section 4.2.2 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. Figure A.1 shows the text summary and Figure A.5 shows the node-link diagram for task 1. Details on how we manually generated the summaries are in Section A.2. The node-link diagrams and text summaries for all four tasks are available in Section A.4.

### 5.2 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 5.1 shows the complete list of fields. Using

Table 5.1: Participant Fields

<table>
<thead>
<tr>
<th>ID</th>
<th>Occupation</th>
<th>Data Science</th>
<th>Computer Science</th>
<th>Forestry</th>
<th>Physics</th>
<th>Bioinformatics</th>
<th>Visualization</th>
<th>Information Technology</th>
<th>Human Robot Interaction</th>
<th>Environmental Science</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Lecturer</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Data Scientist</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Graduate Student</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Data Scientist</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Graduate Student</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Scientist</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Professor</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Database Admin</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Graduate Student</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Graduate Student</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Graduate Student</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Graduate Student</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 5.2: Workflow description for each task in the user study.

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

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 5.1: Workflow 1 Text Summary
the information provided in the demographic survey, we categorized the participants into three expertise categories based on their data science and programming experience.

5.3 Quantitative Results

We evaluate three performance metrics for each task: score, time to complete, and perceived cognitive load (Figure 5.3). Overall, participants were able to answer most questions correctly regardless of which 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 (5.3a). 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?”.
Figure 5.3: Quantitative metrics showing performance using both graph (orange) and text (blue) provenance summaries.
Figure 5.4: Participants rank their preference for either the text summarization or node link diagram. Participants are categorized by their computational expertise.
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 (5.3b). 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 [22]. After each task, participants recorded their response to the TLX questions in Section A.3. As with the other quantitative metrics, the cognitive load scores are similar when comparing the two summarization methods (5.3c).

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.

### 5.4 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 [28]. 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 5.4b.

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 (yellow) to advanced (pink) 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 5.4a, 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 Section 4.3.

**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 Section 5.6.1.

**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).

### 5.5 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 an-
swer question for task 2. 10 participants completed the remote study. We did not see any significant trends across the quantitative metrics. Participants’ overall preference for the text summarization versus the node link diagram was similar to that in the initial study. Figure 5.5 shows the charts indicating user preference for the text vs the graph. All the participants that completed the remote study were categorized as intermediate or expert in their computational and data science expertise. These participants’ preferences range from strongly preferring the node-link diagram to equally preferring both text and node link summaries. This is consistent with the intermediate and expert participants in the in-person 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.

5.6 Discussion

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.

5.6.1 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
Figure 5.5: Participants rank their preference for either the text summarization or node link diagram in the remote study. Participants are categorized by their computational expertise.
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 [12] 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.
Chapter 6

Related Work

Our work focuses on developing provenance tools for reproducibility and evaluating user comprehension of tools conveying information from provenance. We discuss provenance capture in the context of experiment reproducibility and how our tool, Thoth, fits into that landscape. Given that we propose a technique using LLMs to summarize provenance, we examine prior work on visualizing and summarizing provenance, as well as LLM summarization techniques and limitations.

6.1 Provenance Capture

Researchers have built provenance tools for experiment tracking, often at the application level. Application specific provenance collection tools record provenance for a specific application and are often integrated within the application itself. For instance, Tribuo [48] and LIMA [45] are systems that collect provenance during machine learning experiments for reproducing experiments and deduplicating computation. Tribuo is a Java machine learning framework while LIMA is implemented for Apache SystemDS [8], so users can only collect provenance specific to those systems. VisTrails [6] uses provenance to track data analysis projects. Users build their workflows within the VisTrails application and can then execute their analyses and visualize the data. Similarly, GenePattern provides a similar service for genomics research [51]. VisTrails and GenePattern require users to execute their workflows in their framework, so users would have to modify their workflows
to incorporate the tool. These systems are useful for researchers that work with these specific tools to reproduce their models. In practice, many researchers use a variety of tools and applications. We focus on system-level provenance collection, rather than collecting provenance for a specific application, so that we can still track the experiment when a user switches applications or uses multiple tools.

Language-level provenance tools such as RDataTracker [27] for the R language and NoWorkflow [46] for Python capture provenance at the programming language level. These provenance tools collect provenance of data objects within scripts and are useful for analyzing and reproducing scripts. Language-level provenance, though beneficial, suffers from the same limitations as application provenance. Often researchers will use more than one programming language or application to perform their analyses and language level provenance is unique to each language.

Burrito [20] is an experiment tracking and note-taking system. It uses data provenance to observe researcher’s workflow across their workspace. This system requires application specific implementation in order to track objects across applications. Our goal for our system is to be more generalizable, requiring no additional implementation to track the effect of different applications.

6.2 Provenance Graph Visualization

Provenance data are historically displayed using node-link diagrams [16, 24, 33]. Although some applications such as Probe-It [16] include additional views, graph-style visualizations have practically become standard practice. Many tools store provenance data in graph databases, e.g., Neo4j [38], and then use the tools that accompany those systems or other graph-centric tools, e.g., GraphViz [17], to explicitly display 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 [6] 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 [27] and noWorkflow [46], also use node-link diagrams. For our study, we chose to manually gen-
erate our node-link diagrams rather than 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 [6]. Additionally, the graphs we generated using GraphViz [17] and Neo4J database viewer [38] 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.

### 6.3 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 [52]. The comic provenance visualization in 6.1c provides an easy-to-read, high-level summary of self-tracking sensor data [52]. Borkin et al. used a radial representation to represent file system provenance data (6.1b [9]). They compare this radial representation to the node-link representation of Orbiter [33] 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 [57] uses machine learning to summarize provenance graphs into forensic reports. The authors use rule-based techniques and neural-machine translation to generate text from provenance graphs. 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.
Macko and Seltzer use node-link diagrams to visualize provenance data.

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

Schreiber and Struminski use comics to present smartwatch provenance data.

Figure 6.1: We compare the most common node-link visualization (6.1a) with two alternative approaches (6.1c, 6.1b).
6.4 LLM Summarization

Summarization is a popular application of large language models. Zhang et al. and Goyal et al. use LLMs to summarize new articles [19, 71]. The authors find that users prefer LLM-generated summaries to other summarization models [19] and like them as much as human-written summaries [71]. Similarly, Laskar et al. evaluated how well various LLMs summarize meeting notes [26] 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 [65], while Chang et al. and Zhang et al. demonstrate techniques based on iterative and incremental updates [13, 70].

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 [67]. 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 [55]. Liu et al. observe that LLMs frequently overlook information in the middle of a document [30]. Tang et al. evaluates medical journal summaries produced by GPT 3.5 [42] and ChatGPT [40] 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 (Chapter 4). To our knowledge, there is no published work on using LLMs for summarizing provenance logs or log-structured data.
Chapter 7

Conclusion

We investigate computational reproducibility through provenance capture and representation. Our study demonstrates the viability of automatically generating text-based provenance summaries using large language models in a landscape where network graphs dominate user-provenance interfaces. We developed Thoth, a provenance collection tool for reproducibility, to generate provenance graphs for our study. We evaluate the text summarization approach with a user study, comparing text summaries to the state-of-the-art node link diagrams for reproducing computational experiments. Users with less programming expertise often prefer the text summaries as they are presented in a familiar format whereas users with advanced programming experience enjoyed the extra details and visual cues provided in the graphs. Our results demonstrate the effectiveness of the textual summaries and provide insights to guide future reproducibility tool design.

We envision several directions for future work. First, continuing development of our system-level provenance collection tool, we imagine connecting Thoth provenance graphs to other reproducibility tools such as version control or application level provenance. Collecting information at multiple levels of granularity would provide more context for reproducing experiments. With regard to text summarization, one could enhance the summarization technique by using model-fine tuning on provenance data or by using newer models. We expect that fine-tuning would produce even better summarizations consistently. Lastly, we expect that performing a larger user study on provenance representations in the context of experi-
ment reproducibility would give further insight to those developing reproducibility tools.
Bibliography


[27] B. Lerner and E. Boose. RDataTracker: Collecting provenance in an interactive scripting environment. In USENIX Workshop on the Theory and Practice of Provenance (TaPP), 2014. doi:10.1007/978-3-319-16462-5_36. → pages 2, 18, 20, 43


53


[61] A. Trisovic, M. Lau, T. Pasquier, and M. Crosas. A large-scale study on research code quality and execution. Scientific Data, 9:60, 02 2022. doi:10.1038/s41597-022-01143-6. → page 1


Appendix A

Supporting Materials

A.1 Availability
The work presented in this paper is open-source. Detailed installation instructions are available online.


- The user study documents are available at https://doi.org/10.5281/zenodo.106725369.

A.2 Diagram Creation
It is common for graph users to visualize their data as node-link graphs with tools like Neo4j [38] and GraphViz [17]. 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.

A.3 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)

A.4 Study Activities

A.4.1 Questions

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?

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?

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.

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?

A.4.2 Text Summaries

Figure A.1, Figure A.2, Figure A.3, Figure A.4 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.

A.4.3 Node Link Diagrams

Figure A.5, Figure A.6, Figure A.7, Figure A.8 show the node link diagrams we created and used in our study.
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 A.1: 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 "--force -r /tmp/Rtmpjjeszpsk".
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 A.2: 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 A.3: Workflow 3 Text Summary**

62
"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 A.4: Workflow 4 Text Summary**
Figure A.6: Workflow 2 Node Link Diagram
Figure A.7: Workflow 3 Node Link Diagram
Figure A.8: Workflow 4 Node Link Diagram