ProvBuild: Improving Data Scientist Efficiency with Provenance (An Extended Abstract)

Jingmei Hu
Harvard University
Cambridge, MA, USA

Jiwon Joung
University of Michigan
Ann Arbor, MI, USA

Maia Jacobs
Harvard University
Cambridge, MA, USA

Krzysztof Z. Gajos
Harvard University
Cambridge, MA, USA

Margo I. Seltzer
University of British Columbia
Vancouver, BC, Canada

ACM Reference Format:

1 THE PROBLEM
Data scientists frequently analyze data by writing scripts. We conducted a contextual inquiry with interdisciplinary researchers, which revealed that parameter tuning is a highly iterative process and that debugging is time-consuming. As analysis scripts evolve and become more complex, analysts have difficulty conceptualizing their workflow. In particular, after editing a script, it becomes difficult to determine precisely which code blocks depend on the edit. Consequently, scientists frequently re-run entire scripts instead of re-running only the necessary parts. We present ProvBuild, a data analysis environment that uses change impact analysis [1] to improve the iterative debugging process in script-based workflow pipelines. ProvBuild is a tool that leverages language-level provenance [2] to streamline the debugging process by reducing programmer cognitive load and decreasing subsequent runtimes, leading to an overall reduction in elapsed debugging time. ProvBuild uses provenance to track dependencies in a script. When an analyst debugs a script, ProvBuild generates a simplified script that contains only the information necessary to debug a particular problem. We demonstrate that debugging the simplified script lowers a programmer’s cognitive load and permits faster re-execution when testing changes. The combination of reduced cognitive load and shorter runtime reduces the time necessary to debug a script. We quantitatively and qualitatively show that even though ProvBuild introduces overhead during a script’s first execution, it is a more efficient way for users to debug and tune complex workflows. ProvBuild demonstrates a novel use of language-level provenance, in which it is used to proactively improve programmer productivity rather than merely providing a way to retroactively gain insight into a body of code. To the best of our knowledge, ProvBuild is a novel application of change impact analysis and it is the first debugging tool to leverage language-level provenance to reduce cognitive load and execution time.

2 PROVBUILD: PIPELINE DEBUGGING USING PROVENANCE
ProvBuild utilizes noWorkflow [5] to collect language-level provenance [3] to record the actions a script takes and the dependencies between these actions, variables, values and functions. ProvBuild uses these dependencies to identify precisely the parts of a script affected by a user’s debugging. It then produces a shortened script, called the ProvScript, that contains only those parts of the script necessary to debug the original script. This shortened script provides two benefits: 1) it makes it easier for the user to reason about the script and the effect of a user’s modification to it, and 2) it reduces the re-execution time.

ProvBuild consists of a backend engine (Figure 1) and a user interface (Figure 2). The interface allows users to debug functions or variables on the simplified ProvScript and seamlessly merge those modifications back into the original script. To facilitate evaluation, the ProvBuild prototype interface supports both conventional editing (i.e., editing on the entire script) and the ProvBuild provenance-driven editing of a ProvScript. In either case, the user begins by selecting the mode of interaction (conventional or ProvBuild) and identifying the script with which they are working. In ProvBuild mode, the interface activates the provenance tracking backend.

Users interact with their scripts through the three main modules shown in Figure 2.

- **Search:** The user inputs the name of the function or variable to edit (see (1) in Figure 2). ProvBuild extracts the object’s dependencies based on the stored provenance information and generates a ProvScript containing only code pertaining to the chosen object.

![Figure 1: ProvBuild Architecture.](image-url)
tasks with and without ProvBuild. After each task, we examined their digital memorization behavior to evaluate cognitive load and gave them a questionnaire with NASA-TLX standard questions, which evaluate perceived workload [4], and questions about their perceived self-efficacy, subjective assessment of ease of use, and effectiveness. We found that after programming with ProvBuild, participants significantly shorter average completion time ($F(1, 20) = 66.64, \text{raw } p < 0.0001, \text{adjusted } p < 0.0003$) and greater number recall accuracy ($F(1, 20) = 16.00, \text{raw } p = 0.0007, \text{adjusted } p = 0.0014$), and also reported more satisfied overall ($F(1, 20) = 7.42, \text{raw } p = 0.0131, \text{adjusted } p = 0.0131$). Those main effect were statistically significant.

**Study 2: Performance Evaluation:** We collected Python scripts from published works, compared script length and running times with and without ProvBuild to evaluate performance. noWorkflow introduces overhead during dynamic provenance tracking, which caused execution time to increase dramatically, in the best case, by only 56%, but in the worst case by around a factor of 30. However, ProvBuild still saves time in later debugging phases. We measured ProvBuild’s performance after making three types of changes: 1) directly altering script output, 2) altering an input file or input variable, and 3) modifying the parameter of a function in the script. The speedup inherently depends on the length of the code path following the edit and ranges from a factor of 1.78 to 39.31. The resulting ProvScripts retained, on average, 77% and 58% of the lines of the original script, respectively, while the speed-ups averaged 1.23X and 2.46X, respectively.

**Study 3: Deployment in the wild:** We conducted a real-world deployment to evaluate ProvBuild’s usefulness and efficacy for data scientists from different domains. We gave participants access to ProvBuild for one week, which allowed them to explore and use the tool for Python debugging in their daily work. We used surveys to obtain feedback from 12 participants. All participants chose to use ProvBuild at least once, while four participants used it more than once in a one-week period. Participants mentioned different benefits and several concerns after their use with ProvBuild. They expressed a preference for using ProvBuild and mentioned that ProvBuild improves the debugging process mainly by reducing programming time, allowing users to find dependencies and understand their workflow more easily, reducing the need for memorization. They also addressed issues we knew about (e.g., initial run time) or that could be easily addressed (e.g., integration with Jupyter). Participants did not report any significant barriers to independent use of ProvBuild.

**REFERENCES**


