B. König-Ries et al. (Hrsg.): Datenbanksysteme für Business, Technologie und Web (BTW 2023),
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 1

MLProvCodeGen: A Tool for Provenance Data Input and Capture of Customizable Machine Learning Scripts

Tarek Al Mustafa[1,3], Birgitta König-Ries[1,2,3], Sheeba Samuel[1,2,3]

Abstract: Over the last decade Machine learning (ML) has dramatically changed the application of and research in computer science. With growing complexity, it becomes increasingly complicated to assure the transparency and reproducibility of advanced ML systems from raw data to deployment. In this paper, we describe an approach to supply users with an interface to specify a variety of parameters that together provide complete provenance information and automatically generate executable ML code from this information. We introduce *MLProvCodeGen* (Machine Learning Provenance Code Generator), a *JupyterLab* extension to generate custom code for ML experiments from user-defined metadata. ML workflows can be generated with different data settings, model parameters, methods, and training parameters and reproduce results in *Jupyter Notebooks*. We evaluated our approach with two ML applications, image and multiclass classification, and conducted a user evaluation.

Keywords: Provenance Management; Code Generation; Machine Learning; JupyterLab; Jupyter Notebooks; Reproducibility

1 Introduction

Machine Learning (ML) is the dominating data science approach today. ML solves various problems in many sectors. It also benefits the scientific community by supporting scientific workflows [De19] and database systems [Ma20; Va17]. ML workflows include steps to obtain results for given problems from raw data. These steps range from data preprocessing to deployment. Though they are common for every ML workflow, the specifics of the implementation, metadata of the entire experiment, and history of data points and sources used, differ for each ML model. Reproducibility of ML experiments, an increasingly important issue [Ba16; Hu18; SK21], can be enhanced by capturing this information as *provenance data*. We propose a method that allows users to generate code for ML pipelines by filling in templates with pre-defined parameters and variables. These templates incorporate all information needed for provenance tracking. We argue that this reduces the complexity of creating ML models while enhancing reproducibility. The main contributions of this work are: (1) define the minimum requirements to reproduce chosen ML workflows. (2) use these minimum requirements as a data model to build a template based system to automatically generate ML code in Jupyter notebooks[4] with multiple, user-chosen

[1] Heinz Nixdorf Chair for Distributed Information Systems, Jena, Germany
[2] Michael Stifel Center Jena, Jena, Germany
[3] Friedrich Schiller University Jena **{tarek.almustafa, birgitta.koenig-ries, sheeba.samuel}@uni-jena.de**
[4] `https://jupyter-notebook.readthedocs.io/en/stable/`

parameters. (3) automatically capture and display provenance data from the generated notebooks to allow one-to-one reproductions by (4) inputting captured data into the system.

2 Related Work

Provenance Data and Reproducibility. Provenance plays a key role in reproducibility [Mi16]. Prospective provenance describes the specifications and steps that must be followed to generate a data product [Fr08]. Retrospective provenance captures what happened during the execution of a computational task. It is important that both provenance data types are captured and documented [De15; HDB17]. In our previous work, we investigated more factors that influence the reproducibility of ML experiments [SLK20].
Provenance Data Models and Ontologies. Provenance data models specify the format of metadata and which data points are represented. The *W3C PROV family of specifications* [MBC13] includes *The PROV Data Model (PROV-DM)* [Be13] and *The Provenance Ontology (PROV-O)* [Le13], an encoding of *PROV-DM* into *OWL2 Web Ontology Language*. Our previous work, the *REPRODUCE-ME Ontology* [SK17; SK18a], extends *PROV-O* and includes the *provenance-plan (P-PLAN)*[5] vocabulary to describe all computational and non-computational steps and data of scientific experiments in a machine-readable way.
Provenance Capture Systems. There have been a number of applications of these specifications and ontologies that may adopt or adjust existing data models. Other significant works include *PROV-ML* defined in [So19], which uses *W3C PROV* and *ML-Schema* to specify a provenance data model for complex tasks in the computational science and engineering domains and multiple systems that aim to capture provenance data automatically from either ML scripts [Na20; Sc17], model outputs [Ma17], computational notebooks [SK18b; SK20], specific workflow steps like data cleaning [PML20], or whole systems [Sc18].
MLOps. Systems applying DevOps practices to ML [Ta20] include AutoML[6], MLflow [Za18], and ModelDB [Hi04]. They support ML development and deployment, including workflow management, data engineering, provenance management, and reproducibility. These systems target complex, custom-made ML products requiring contributions by several experts including data scientists and developers. In contrast, our work focuses on customizing predefined ML pipelines by lay users without the need for ML expertise.
Code Generation and Templates. Automatic code generation can increase productivity and consistency in ML scripts. Code generation tools can assist the development of automatic programming tools to improve programming productivity [LCB20]. However, supporting automatic code generation with multiple parameters raises complexity exponentially. *Train-Generator* provides and generates custom template code for ML[7]. It offers multiple options for preprocessing, model setup, training, and visualization. We build upon this system by developing a framework that can generate code for multiple ML tasks, generating executable notebooks, and integrating provenance data capture and visualization.

[5] `http://vocab.linkeddata.es/p-plan/version/13032014/`
[6] `https://cloud.google.com/automl/docs`
[7] `https://traingenerator.streamlit.app/`

3 MLProvCodeGen

In this section, we introduce *MLProvCodeGen*, a *JupyterLab* extension that explores how to support the reproducibility of ML experiments by combining template based code generation and provenance data capture, input, and visualization into one system. We implemented two example use cases: Image Classification and Multiclass Classification on tabular data, each with its set of customizable parameters. *MLProvCodeGen* was designed such that it can be extended to others. *MLProvCodeGen* is available online.[8]
Fig. 1 shows the system architecture consisting of a frontend plugin to capture information, and a backend plugin to process that information and generate notebooks from it.

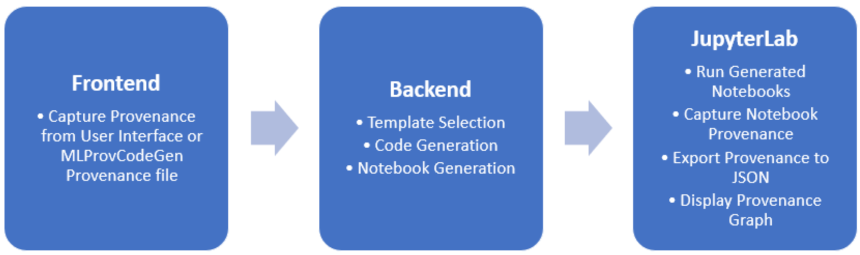

Fig. 1: System Architecture of MLProvCodeGen

Frontend. The frontend plugin provides a user interface as shown in Fig. 2. Users can open

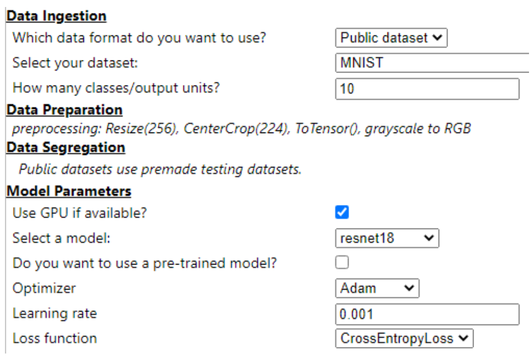

Fig. 2: Excerpt from Image Classification Input Elements in the User Interface

the extension by clicking the *MLProvCodeGen* button in the *other* section of *JupyterLab*'s home interface. At the bottom, users can submit their selected parameters to the system's backend. The user interface also allows users to input a provenance file from an experiment generated by *MLProvCodeGen* in the past in order to reproduce it.
Backend. The backend's main goal is to generate a notebook for either Image Classification or Multiclass Classification from user inputs. Each use case has a set of templates associated

[8] https://mybinder.org/v2/gh/fusion-jena/MLProvCodeGen/main?urlpath=lab

with it from which code can be generated. Therefore, the backend first selects a set of templates based on the specified use case, and then links variables from the user inputs to the templates. Since Jupyter Notebooks consist of cells, each cell is generated from a distinct template. Templates contain placeholder variables that are filled by the backend. For example, the template contains a placeholder called $dataset$ and the backend extracts a $dataset$ variable from the user inputs using $dataset = user\_inputs['entity']['ex : DataIngestionData']['ex : dataset\_id']$ that is called $dataset$ and has a value $[ex : dataset\_id]$. When the templates are rendered, the value $ex : dataset\_id$ is written into the $dataset$ placeholder and the output is appended to a notebook file. This way, a notebook file that was empty at the start is filled with rendered outputs from templates for all markdown and code cells. We use *Jinja*[9] as our templating language.

Notebooks. The notebooks are structured as follows: At the top is a markdown cell containing information about the ML task itself. The code cells below contain the installation command for the requirements and packages needed to run the notebook. Import statements are added directly after and provenance data capture is initialized. The remaining cells follow the structure of an ML pipeline. Each notebook has a cell for data ingestion, data preparation, data segregation, the model, training, and evaluation. At the bottom of each notebook are cells to generate a provenance graph, generate a provenance data file in JSON format, and cells to view the provenance data file and graph.

experiment_info	creation_date, file_size, modification_date, task_type, title
hardware_info	CPU, GPUs, Operating_System, RAM
packages	All Python packages used in the notebook + the package version used
notebook	prov:type, creation_date, file_format, name, kernel, programming_language, programming_language_version
data_ingestion	start_time, end_time, execution_time, data_format, dataset_id, dataset_classes, feature_dimensions, dataset_description, root_location, training_samples, testing_samples, validation_samples
data_preparation	start_time, end_time, execution_time, number_of_operations, operations
data_segregation	start_time, end_time, execution_time, training_split, testing_split, validation_split
model_parameters	start_time, end_time, execution_time, gpu_enable, pretrained, save_checkpoint, model_name, model_description, activation_function, output_neurons, loss_function, optimizer, optimizer_learning_rate
model_training	start_time, end_time, execution_time, random_seed, resulting_model_seed, batch_size, epochs, print_progress
model_evaluation	start_time, end_time, execution_time, evaluation_metrics(accuracy, loss, AUC, Confusion Matrix, F1, MAE, MSE)

Tab. 1: Provenance Data Model of MLProvCodeGen

Provenance Data Capture. All provenance information captured for notebooks generated by *MLProvCodeGen* is listed in Tab. 1. We capture provenance data using the *prov*[10] Python package. This allows us to specify entities, agents, and activities according to *PROV-DM*

[9] https://jinja.palletsprojects.com/en/3.1.x/

[10] https://pypi.org/project/prov/

specifications and build p-plans and collections adjacent to *PROV-O*. If a specific function was used to capture that information, *MLProvCodeGen* generates an activity describing it. Each code cell is an entity, has an activity that describes the execution of that cell, and a second entity that describes the data generated by the execution of that cell. Cell entities are ordered by specifying how a given cell was influenced by the ones executed before it. At the end of the notebook, the captured provenance data is saved to a JSON file and used to generate a provenance graph as seen in Fig. 3 and Fig. 4. A major downside of using the *prov* package is that the provenance capture has to be hard coded into the notebook at the time of notebook generation. This means that changes made by users after that point are only saved if users write them into the provenance data package themselves.

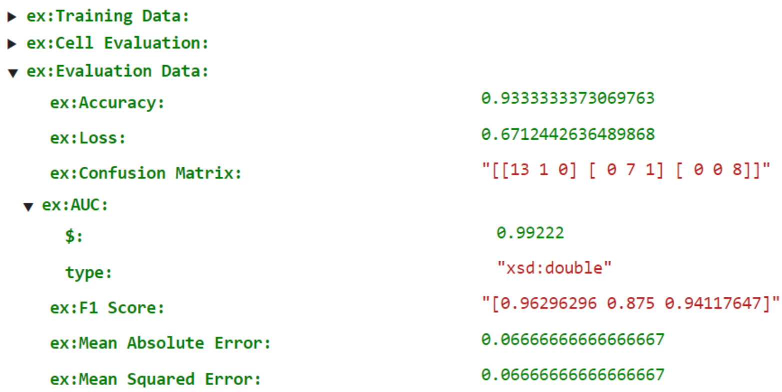

Fig. 3: Excerpt from a generated provenance JSON file in MLProvCodeGen

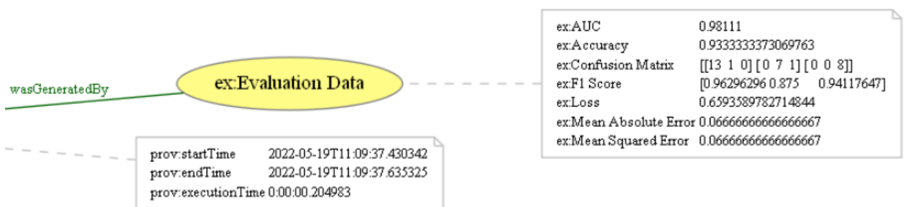

Fig. 4: Captured Evaluation Data in the Provenance Graph

Provenance Data Input. Any provenance data file generated by *MLProvCodeGen* can be uploaded to the system in the user interface to generate an identical reproduction of the code described by the provenance data. Uploaded files are processed by the backend in the exact same way as data input by users through the input elements in the user interface.
Extensibility. Due to the modular nature of *MLProvCodeGen*, users should have the ability to add new ML experiments to it. We have published step-by-step instructions in the online documentation. The different steps include: From an existing notebook (1) Write code generation templates according to the notebook's cells, (2) add provenance capture code to

the templates following the data model and prior examples, (3) add new input elements to the user interface in line with the variables used in the templates, and (4) connect frontend and backend through a server call for the new ML experiment. Further evaluation would be necessary to assess the difficulty of extending *MLProvCodeGen*.

4 Preliminary Evaluation

We conducted a user evaluation to measure *MLProvCodeGen*'s user experience by combining an online survey via *LimeSurvey*[11] and a virtual installation via *Binder*[12]. All questions and completed answers are available online[13]. We also include a comprehensive summary of the user evaluation[14]. Our goal was to test the appropriateness and general usability of MLProvCodeGen for users from the computer science domain who may or may not be familiar with ML experiments, data provenance, and reproducibility. We asked users to self assess their level of proficiency with these terms, to complete hands-on user tasks, and consequently rate their experience using a variety of metrics. Eight of the twelve participants answered the question regarding their professional background. All had a background in computer science or a related field. Prior knowledge about both machine learning and reproducibility was mixed with all values from "poor" to "excellent" selected.
The key conclusions of the online survey are: (1) The explanations and instructions given are adequate to use *MLProvCodeGen* without outside help. (2) The user interface is intuitive and easy to use. (3) The generated notebooks have comprehensible structure and, depending on the users expertise, the code is coherent and understandable. (4) The provenance graph displays the provenance data as intended. However, for users without domain expertise, the graph is difficult to interpret. Due to its size, it is also strenuous to find specific data points.

5 Conclusions and Future Work

In this paper, we presented *MLProvCodeGen*, a tool to support the reproducibility of machine learning experiments by combining template based code generation and provenance data capture, input, and visualization into one system. We evaluated our system by implementing two use case ML tasks, image classification and multiclass classification, and conducted a user evaluation. Future work on *MLProvCodeGen* includes improvements to the provenance graph, provenance data export, and adding more examples such as clustering. All source code, further information, explanations, a tutorial, the documented user evaluation, and an installation of *MLProvCodeGen* on a virtual machine are available online.[15]

[11] https://www.limesurvey.org/
[12] https://mybinder.org/, available at https://mybinder.org/v2/gh/fusion-jena/MLProvCodeGen/main?urlpath=lab
[13] https://github.com/fusion-jena/MLProvCodeGen/tree/main/EvaluationResults
[14] https://github.com/fusion-jena/MLProvCodeGen/blob/main/EvaluationResults/MLProvCodeGen_LongPaper.pdf
[15] https://github.com/fusion-jena/MLProvCodeGen

References

[Ba16] Baker, M.: 1,500 scientists lift the lid on reproducibility. Nature 533/7604, 2016.

[Be13] Belhajjame, K.; B'Far, R.; Cheney, J.; Coppens, S.; Cresswell, S.; Gil, Y.; Groth, P.; Klyne, G.; Lebo, T.; McCusker, J., et al.: Prov-dm: The prov data model. W3C Recommendation 14/, pp. 15–16, 2013.

[De15] Dey, S.; Belhajjame, K.; Koop, D.; Raul, M.; Ludäscher, B.: Linking prospective and retrospective provenance in scripts. In: 7th USENIX Workshop on the Theory and Practice of Provenance (TaPP 15). 2015.

[De19] Deelman, E.; Mandal, A.; Jiang, M.; Sakellariou, R.: The role of machine learning in scientific workflows. The International Journal of High Performance Computing Applications 33/6, pp. 1128–1139, 2019.

[Fr08] Freire, J.; Koop, D.; Santos, E.; Silva, C. T.: Provenance for computational tasks: A survey. Computing in science & engineering 10/3, pp. 11–21, 2008.

[HDB17] Herschel, M.; Diestelkämper, R.; Ben Lahmar, H.: A survey on provenance: What for? What form? What from? The VLDB Journal 26/6, pp. 881–906, 2017.

[Hi04] Hines, M. L.; Morse, T.; Migliore, M.; Carnevale, N. T.; Shepherd, G. M.: ModelDB: a database to support computational neuroscience. Journal of computational neuroscience 17/, pp. 7–11, 2004.

[Hu18] Hutson, M.: Artificial intelligence faces reproducibility crisis. Science 359/6377, pp. 725–726, 2018.

[LCB20] Le, T. H.; Chen, H.; Babar, M. A.: Deep learning for source code modeling and generation: Models, applications, and challenges. ACM Computing Surveys (CSUR) 53/3, pp. 1–38, 2020.

[Le13] Lebo, T.; Sahoo, S.; McGuinness, D.; Belhajjame, K.; Cheney, J.; Corsar, D.; Garijo, D.; Soiland-Reyes, S.; Zednik, S.; Zhao, J.: Prov-o: The prov ontology./, 2013.

[Ma17] Ma, S.; Aafer, Y.; Xu, Z.; Lee, W.-C.; Zhai, J.; Liu, Y.; Zhang, X.: LAMP: data provenance for graph based machine learning algorithms through derivative computation. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. Pp. 786–797, 2017.

[Ma20] Ma, L.; Ding, B.; Das, S.; Swaminathan, A.: Active learning for ML enhanced database systems. In: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. Pp. 175–191, 2020.

[MBC13] Missier, P.; Belhajjame, K.; Cheney, J.: The W3C PROV family of specifications for modelling provenance metadata. In: Proceedings of the 16th International Conference on Extending Database Technology. Pp. 773–776, 2013.

[Mi16] Missier, P.: The lifecycle of provenance metadata and its associated challenges and opportunities. Building Trust in Information/, pp. 127–137, 2016.

[Na20] Namaki, M. H.; Floratou, A.; Psallidas, F.; Krishnan, S.; Agrawal, A.; Wu, Y.; Zhu, Y.; Weimer, M.: Vamsa: Automated provenance tracking in data science scripts. In: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. Pp. 1542–1551, 2020.

[PML20] Parulian, N. N.; McPhillips, T. M.; Ludäscher, B.: A model and system for querying provenance from data cleaning workflows. In: Provenance and Annotation of Data and Processes. Springer, pp. 183–197, 2020.

[Sc17] Schelter, S.; Boese, J.-H.; Kirschnick, J.; Klein, T.; Seufert, S.: Automatically tracking metadata and provenance of machine learning experiments. In: Machine Learning Systems Workshop at NIPS. Pp. 27–29, 2017.

[Sc18] Schelter, S.; Böse, J.-H.; Kirschnick, J.; Klein, T.; Seufert, S.: Declarative metadata management: A missing piece in end-to-end machine learning. Proceedings of SYSML 18/, 2018.

[SK17] Samuel, S.; König-Ries, B.: REPRODUCE-ME: ontology-based data access for reproducibility of microscopy experiments. In: European Semantic Web Conference. Springer, pp. 17–20, 2017.

[SK18a] Samuel, S.; König-Ries, B.: Combining P-Plan and the REPRODUCE-ME ontology to achieve semantic enrichment of scientific experiments using interactive notebooks. In: European semantic web conference. Springer, pp. 126–130, 2018.

[SK18b] Samuel, S.; König-Ries, B.: ProvBook: Provenance-based Semantic Enrichment of Interactive Notebooks for Reproducibility. In: ISWC (P&D/Industry/BlueSky). 2018.

[SK20] Samuel, S.; König-Ries, B.: Reproducemegit: a visualization tool for analyzing reproducibility of jupyter notebooks. In: Provenance and Annotation of Data and Processes. Springer, pp. 201–206, 2020.

[SK21] Samuel, S.; König-Ries, B.: Understanding experiments and research practices for reproducibility: an exploratory study. PeerJ 9/, e11140, 2021.

[SLK20] Samuel, S.; Löffler, F.; König-Ries, B.: Machine learning pipelines: provenance, reproducibility and FAIR data principles. In: Provenance and Annotation of Data and Processes. Springer, pp. 226–230, 2020.

[So19] Souza, R.; Azevedo, L.; Lourenço, V.; Soares, E.; Thiago, R.; Brandão, R.; Civitarese, D.; Brazil, E.; Moreno, M.; Valduriez, P., et al.: Provenance data in the machine learning lifecycle in computational science and engineering. In: 2019 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS). IEEE, pp. 1–10, 2019.

[Ta20] Tamburri, D. A.: Sustainable mlops: Trends and challenges. In: 2020 22nd international symposium on symbolic and numeric algorithms for scientific computing (SYNASC). IEEE, pp. 17–23, 2020.

[Va17] Van Aken, D.; Pavlo, A.; Gordon, G. J.; Zhang, B.: Automatic database management system tuning through large-scale machine learning. In: Proceedings of the 2017 ACM international conference on management of data. Pp. 1009–1024, 2017.

[Za18] Zaharia, M.; Chen, A.; Davidson, A.; Ghodsi, A.; Hong, S. A.; Konwinski, A.; Murching, S.; Nykodym, T.; Ogilvie, P.; Parkhe, M., et al.: Accelerating the machine learning lifecycle with MLflow. IEEE Data Eng. Bull. 41/4, pp. 39–45, 2018.