Data versioning in machine-learning architecture

Peter Bartoš, Stanislav Krištof

Faculty of Informatics and Information Technologies
Slovak University of Technology in Bratislava

November 27, 2024

Abstract

Data versioning plays a crucial role in modern machine learning architecture, ensuring that the complex and ever-evolving datasets that provide the basis for models can be tracked, compared, and managed efficiently. At its core, data versioning refers to the practice of creating unique references for different states of a dataset over time, allowing us to trace changes, restore previous versions, and debug issues. This is vital in machine learning workflows, where even small changes in data can significantly impact model performance.

In this domain, data versioning supports reproducibility by maintaining a consistent link between datasets and the models trained on them. Without version control, it becomes challenging to recreate experiments, leading to inconsistencies in predictions and hindering model audits. Versioning also simplifies collaboration across teams, enabling multiple stakeholders to work on the same data without overwriting each other's progress.

Basic approaches to data versioning systems (DVS) include full duplication of datasets, where copies are saved with each change, and metadata-based versioning, where timestamps indicate the validity of each record. Advanced solutions (like lakeFS and DVC) deal with versioning as a core component of machine learning architecture. They enable storage-efficient data commits, branching, and comparison, similar to how Git handles version control in software development.

Overall, data versioning enhances productivity, reduces errors, and fosters an engineering-driven approach to handling data in machine learning pipelines, ultimately enabling smoother transitions between development stages and more robust model deployment. The objective of the project is to go over these systems and provide detailed overviews of how data versioning has such a crucial role in machine learning architecture.

1 Introduction

Reproducibility, as defined by ACM [6], is the ability to obtain precise measurements by different teams under the same conditions. In AI advancements, reproducibility is crucial, with Peng [20] considering it the ultimate measure of scientific validity. However, Pawlik et al. [19] found that only 7.64% of analyzed papers were reproducible. Improved data version control can enhance reproducibility and enable more complex studies, increasing understanding of neural networks. Pawlik et al. [19] suggest that datasets should include not only input data but also raw data and preparation instructions to improve context and reproducibility.

To understand how data versioning integrates into machine learning architecture, it's essential to examine key components and workflows. Machine learning pipelines rely on iterative experimentation, where data is critical at every stage, from preprocessing to model training, evaluation, and deployment. Maintaining consistent, traceable versions of datasets and models is vital for ensuring reproducibility. [5]

2 Version control in Machine Learning

In the process of developing a machine learning model, various artifacts beyond source code are produced. An artifact refers to any file that serves as an input or output of a given process. In traditional software development, inputs such as source code and libraries qualify as artifacts, while outputs like object files and executables are also considered artifacts. For instance, in a build process, these outputs represent the results of transforming inputs through the compilation pipeline. [5, 22]

In the context of machine learning, the most critical artifacts are datasets and models. Datasets act as the inputs to the training process, while models, which encapsulate learned parameters and structures, serve as the outputs of this process. [22]

Figure 1: Artifacts in Machine Learning [5]

2.1 Use of Data Versioning in Machine Learning

In machine learning workflows, managing datasets is crucial for reproducibility and collaboration. A systematic data versioning approach allows teams

to track, store, retrieve, and switch between dataset versions, ensuring consistency across development and experimentation phases. [5, 22]

The process begins by versioning the dataset, similar to tracking changes in source code. Metadata files store information about each dataset version, keeping the repository efficient while pointing to specific dataset versions used in different pipeline stages. [22, 2]

The actual dataset is stored separately, either locally or in the cloud, to avoid bloating the version control system. This ensures every dataset version is accounted for without overwhelming the repository. [2]

Retrieving a specific dataset version is possible through metadata, ensuring reproducibility by matching data with the state of the code and experiments. [2, 1]

The ability to switch between dataset versions allows flexibility during development, supporting iterative testing, analysis, and debugging, ensuring synchronization between code and data. [2, 1]

Implementing data versioning practices ensures machine learning workflows are robust, efficient, and collaborative, facilitating seamless transitions through different stages of development and research. [5, 2, 1]

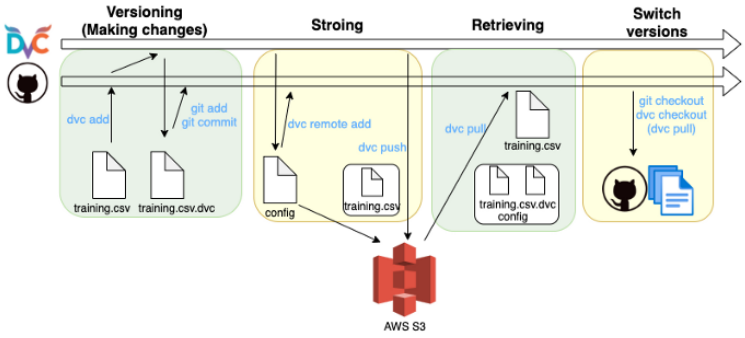

Figure 2: Example of how can be data versioning used in machine learning (DVC and GIT as examples) [2]

2.2 Advantages of Data Versioning in Machine Learning

Data versioning in machine learning workflows goes beyond tracking changes; it promotes collaboration across teams and ensures consistency in model development. By structuring versioning, practitioners can manage datasets and models at different stages, enabling seamless integration, testing, and pipeline improvement. [5, 2]

It allows branching and merging of datasets, akin to code branching in software development. Teams can modify foundational datasets for specific needs, organizing changes into separate versions while maintaining links to the original data. [5, 2]

Versioning supports "harvesting," where improvements in one branch can integrate back into the main dataset or other streams, benefiting broader workflows without disrupting progress. It also ensures consistency by propagating changes across projects. [2, 4]

Hierarchical versioning tracks streams from enterprise-level to individual workflows, enabling scalability and evolution alongside machine learning models. [2, 4]

A robust versioning framework ensures data lineage, collaboration, and reproducibility, enhancing confidence in building, testing, and deploying models. [2, 4]

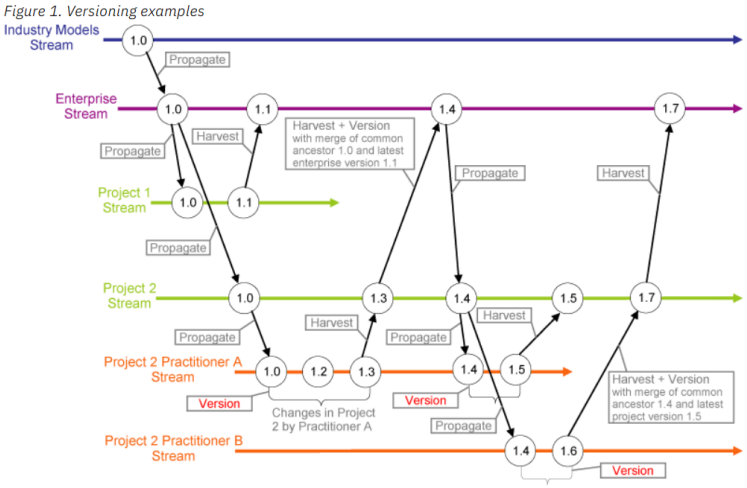

Figure 3: An example of hierarchical data versioning streams showcasing branching, propagation, and merging across industry, enterprise, project, and individual practitioner levels, enabling collaborative and scalable data management in machine learning workflows [4]

3 Insight into...

Types of Data Version Control Systems (DVCS) According to Zolkifli et al. [24] in centralised DVCS, a singular repository is located on the server. These DVCS are suitable for teams with smaller number of members, with the team being located in a single place. Only the last version of a file is retrieved and there is a single point of failure. Examples include Apache Subversion or Perforce Revision Control System. In distributed DVCS, on the other hand, each user has one local repository. Examples include Git, Mercurial, Bazaar or Bitkeeper. This type is suitable regardless of team size.

It also enables users being located in different parts of the world. Clients can create their own branches and sync them with the server.

Git As stated by Komsiyski [16], Git works on basis of patch files. As noted by Bryan [7], large and often changing files are not suitable for Git, since they can slow down pushes and pulls. According to Perez et al. [21], binary files in git are stored as a single large entity. Therefore, even small changes lead to new copies in the repository. This also makes it difficult to compare changes in files using diff and may lead to frequent merge conflicts.

Git LFS Git also offers an extension for large files called Git LFS (Large File Storage), which enables more efficient handling of large files [21]. Content of the file is stored in cloud, with the repository containing only pointers to the files, as noted in the documentation [13] (Figure 4). Since Git LFS is a GitHub extension, its advantages are compatibility with Git [14] and file agnosticism (compatibility with all file formats) [17]. Its disadvantages are inefficient storage management - similar to Git, edited files are stored as new files. Therefore, it is not suitable for large frequently-changing files, especially if they are compressed (such as file formats frequently used in computer vision, e.g. jpg, png) [15]. In addition, data in Git LFS do not stay in place. It also does not scale as well and its data retrieval is slow. As such, this approach is suitable mostly for game developers and not for for ML and data science purposes.

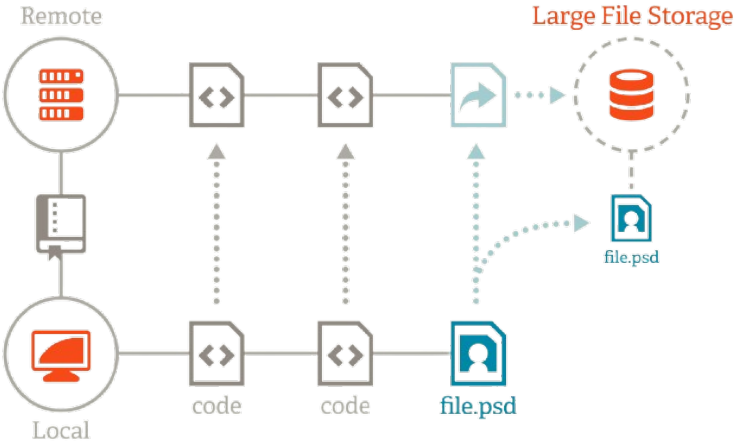

Figure 4: Software architecture of Git LFS [18]

Dolt Dolt is a version-controlled SQL database and as such it may serve as an example of a centralised DVS. Analogous to Git, one can track schemas and changes in the database, create and merge branches [8]. Under the hood,

Dolt uses Prolly trees - a data structure related to both B-trees [23], commonly used in RDBMS and Merkle trees (used by distributed version control systems such as Git or Mercurial)[17]. This data structure provides both fast performance (including diffs and merges) and efficient storage management (each portion of data shared between trees is shared only once). However, like other RDBMS, while Dolt may be the ideal solution for structured data, it is not suitable for unstructed data.

DVC by Iterative Another solution, suitable for ML learning may be DVC [3]. DVC achieves faster data retrieval by its data staying in place. In addition, DVC also uses a caching layer (Figure **??**), which allows faster data retrieval for multiple members of team [17]. DVC supports both structured and unstructed data. It however does not support RDBMS. Since it caters to data scientists, it offers several features which greatly lead to higher reproducibility, such as defining data pipelines [9], visualisation of pipelines [12], experiment tracking [11] and hydra compatibility [10]. It also does not scale as well, mostly due to the same reasons as git.

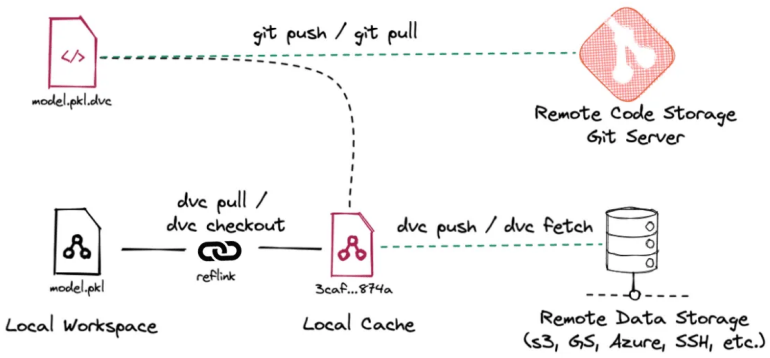

Figure 5: Software architecture of DVC [17]

4 Conclusions and Further Work

Data versioning is essential in modern machine learning workflows, addressing challenges like reproducibility, collaboration, and scalability. Implementing data versioning systems (DVS) helps teams manage datasets efficiently, ensuring consistency and traceability. Tools like Git LFS, Dolt, and DVC showcase different approaches to version control, highlighting trade-offs in data management, performance, and scalability.

Challenges persist despite advancements. Centralized systems face scalability issues and single points of failure, while distributed systems must handle large, changing datasets efficiently. Tools like Git LFS struggle with large

binary files, Dolt is limited to structured data, and DVC, while beneficial for data scientists, lacks support for relational databases.

Future research in data versioning will focus on handling large, frequently updated datasets, particularly in fields like computer vision, genomics, and IoT. Additionally, hybrid systems are needed to manage both relational and non-relational data for diverse machine learning workflows.

Addressing these challenges will help data versioning systems evolve, solidifying their role in machine learning architecture. This will enhance the efficiency, reliability, and reproducibility of pipelines, while fostering greater collaboration in the AI and data science community.

References

[1] Mlops: Data versioning with dvc — part i. 2021.

[2] How data versioning can be used in machine learning. 2022.

[3] Dvc documentation. 2024.

[4] In-depth guide to data versioning: Benefits and formats. 2024.

[5] Intro to mlops: Data and model versioning. 2024.

[6] ACM. Artifact review and badging. August 24, 2020.

[7] J. Bryan. Excuse me, do you have a moment to talk about version control? *The American Statistician*, 72(1):20–27, 2018.

[8] Dolt. *Data and Model Quality Control.* Accessed on 2024-11-6.

[9] DVC. *Defining Pipelines.* Accessed on 2024-10-12.

[10] DVC. *Hydra Composition.* Accessed on 2024-10-12.

[11] DVC. *Reviewing and Comparing Experiments.* Accessed on 2024-10-12.

[12] DVC. *Running pipelines.* Accessed on 2024-10-12.

[13] GitHub. *About Git Large File Storage.* Accessed on 2024-11-05.

[14] GitHub. *Collaboration with Git Large File Storage.* Accessed on 2024-11-05.

[15] GitHub. When to use git lfs. Accessed on 2024-10-13.

[16] V. Komsiyski. Binary differencing for media files. 2013.

[17] E. Orr. Data versioning as your 'get out of jail' card - dvc vs. git-lfs vs. dolt vs. lakefs. 2024. Accessed on 2024-10-12.

[18] A. Pasha. Understanding git large file storage (lfs): Efficiently managing large files in git repositories. 2023. Accessed on 2024-11-08.

[19] M. Pawlik, T. Hütter, D. Kocher, W. Mann, and N. Augsten. A link is not enough–reproducibility of data. *Datenbank-Spektrum*, 19:107–115, 2019.

[20] R. D. Peng. Reproducible research in computational science. *Science*, 334(6060):1226–1227, 2011.

[21] Y. Perez-Riverol, L. Gatto, R. Wang, T. Sachsenberg, J. Uszkoreit, F. d. V. Leprevost, C. Fufezan, T. Ternent, S. J. Eglen, D. S. Katz, et al. Ten simple rules for taking advantage of git and github, 2016.

[22] M. R. Pulicharla. Data versioning and its impact on machine learning models. *Journal of Science & Technology*, 5(1):22–37, 2024.

[23] T. Sehn. Prolly trees. 2024. Accessed on 2024-11-08.

[24] N. N. Zolkifli, A. Ngah, and A. Deraman. Version control system: A review. *Procedia Computer Science*, 135:408–415, 2018.