



mlidea: Interactively Improving ML Data Preparation Code via “Shadow Pipelines”

Stefan Grafberger
BIFOLD & TU Berlin
grafberger@tu-berlin.de

Paul Groth
University of Amsterdam
p.groth@uva.nl

Sebastian Schelter
BIFOLD & TU Berlin
schelter@tu-berlin.de

ABSTRACT

Data scientists develop ML pipelines in an iterative manner: they repeatedly screen a pipeline for potential issues, debug it, and then revise and improve its code according to their findings. However, this manual process is tedious and error-prone. To address this challenge, we propose to assist data scientists with automatically derived *interactive suggestions for pipeline improvements* during this development cycle. We demonstrate `mlidea`, a library to generate interactive suggestions with so-called *shadow pipelines*, hidden variants of the original pipeline that modify it to auto-detect potential issues, try out modifications for improvements, and suggest and explain these modifications to the user. Our system uses incremental view maintenance to enable data scientists to quickly iterate on their code and to ensure low-latency maintenance of the shadow pipelines. We demonstrate how our system improves code for various domains with three interactive shadow pipelines: fixing mislabeled rows, enhancing robustness against data quality problems, and improving pipeline performance on data slices with subpar predictions.

PVLDB Reference Format:

Stefan Grafberger, Paul Groth, and Sebastian Schelter. `mlidea`: Interactively Improving ML Data Preparation Code via “Shadow Pipelines”. PVLDB, 18(12): 5359 - 5362, 2025.
doi:10.14778/3750601.3750671

PVLDB Artifact Availability:

The source code, data, and/or other artifacts have been made available at <https://github.com/stefan-grafberger/mlidea-demo>.

1 INTRODUCTION

Software systems that learn from data with machine learning (ML) have become ubiquitous, influencing critical decisions. Unfortunately, ML pipelines are often brittle with respect to their input data [1, 16, 19, 22]. Consequently, data-centric techniques are being developed to detect, quantify, and improve the reliability, fairness, and prediction quality of ML applications. Applying these techniques to ML pipelines currently requires a high level of expertise, as existing approaches [5, 7, 13, 17, 21] assume that data scientists know in advance what kind of errors they are looking for.

Interactively improving ML pipelines with `mlidea`. In practice, data scientists typically do not know in advance what pipeline issues

to look for, and often “discover serious issues only after deploying their systems in the real world” [10]. At development time, data scientists currently have to iteratively screen their pipeline for potential issues, debug these issues, and then revise and improve their code according to their findings. This process is tedious, as it requires repeated manual code re-organisation and re-execution in an environment like a Jupyter notebook.

We argue that ML pipeline development should be accompanied by *interactive suggestions* to improve the pipeline code, similar to code inspections in modern IDEs like IntelliJ [11] or text corrections in writing assistants like Grammarly [9]. Compared to previous work like `mlinspect` [7] and `mlwhatif` [5], we address three new challenges: (i) enabling low-latency auto-detection of pipeline improvement opportunities for seamless integration into the development workflow; (ii) identifying issues spanning multiple operators, rather than being constrained to isolated operator checks [7]; and (iii) providing provenance-enabled explanations [2] for detected problems and suggested improvements. Note that AI coding assistants like Copilot [4] are orthogonal to our approach. While they suggest code changes, they focus on static, code-centric improvements without assessing the impact on ML pipeline performance. In contrast, our approach generates and evaluates suggestions by dynamically executing the pipeline, analyzing the data flowing through the pipeline, experimenting with changes, and quantifying the performance impact of each suggestion. Thus, our approach complements existing AI tools by offering a data-centric platform for efficiently evaluating code suggestions, applying data-centric ML techniques, and encoding best practices.

We recently proposed `mlidea` [6], a library that interactively assists data scientists with suggestions for improving ML data preparation code. To enable low-latency suggestions, `mlidea` introduces “shadow pipelines”, which are hidden variants of the original pipeline that modify it to auto-detect potential issues, try out modifications for improvements, and suggest and explain these modifications to the user. Shadow pipelines generate potential modifications for improvement using both expert-designed rules and LLMs. When possible, the suggestions take the form of source code changes, which `mlidea` can automatically try to merge into the user code with the help of LLMs. Our system works directly with declaratively written code using popular data science libraries and applies incremental view maintenance for low-latency updates of the original pipeline and the maintenance of the shadow pipelines.

Demonstration details. We showcase three different shadow pipelines that `mlidea` can automatically generate from existing pipeline code to interactively assist data scientists. Attendees will be able to experiment with `mlidea` on pipelines from different domains, implemented using popular data science libraries like `scikit-learn`, `pandas`, `keras`, and `langchain`, using both retrieval-augmented

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit <https://creativecommons.org/licenses/by-nc-nd/4.0/> to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 18, No. 12 ISSN 2150-8097.
doi:10.14778/3750601.3750671

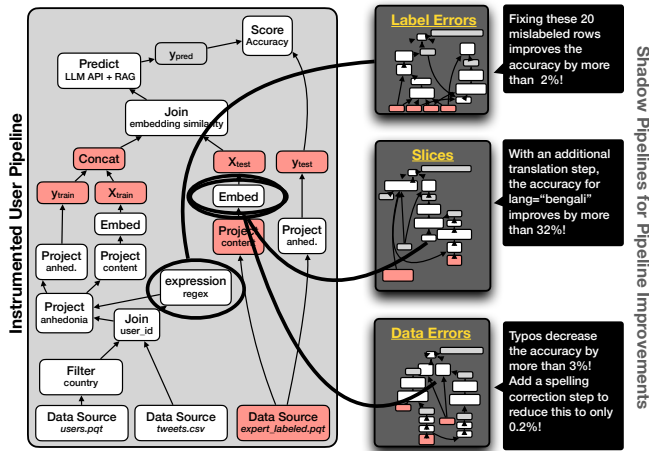


Figure 1: Overview of mliidea – several automatically maintained “shadow pipelines” identify data-related issues in a user’s ML pipeline code at development time and give actionable suggestions for improvements.

LLMs and traditional ML models. We provide a web-based user interface for the attendees to experience mliidea from the perspective of a data scientist to interactively improve their pipeline code by iteratively fixing mislabeled rows [12, 13], increasing the robustness against data quality issues [20], and improving prediction performance on underperforming data slices [3, 18]. Attendees can also explore how mliidea extracts a logical query plan from the original pipeline, generates shadow pipelines based on the original query plan, and uses incremental view maintenance to provide low-latency suggestions. We provide the web-based user interface for our demonstration, along with all example pipelines and datasets, at <https://github.com/stefan-grafberger/mlidea-demo>.

2 SYSTEM OVERVIEW

In the following, we provide a brief overview of mliidea. For further details, we refer to [6] and the open source code of our library at <https://github.com/stefan-grafberger/mlidea>.

Core ideas. Figure 1 gives a high-level overview of our approach: mliidea instruments a data scientist’s ML pipeline code and creates and maintains so-called “shadow pipelines” with low-latency to generate suggestions for improvements. Such a shadow pipeline is a hidden variant of the original pipeline, which modifies it to auto-detect potential issues and tries out different pipeline changes for improvement opportunities. Subsequently, each shadow pipeline provides the user with code suggestions to improve the pipeline, accompanied by a provenance-based explanation and a quantification of the expected impact on the pipeline outputs.

Warnings and suggestions via shadow pipelines. To generate shadow pipelines, mliidea starts by extracting the logical query plan of the original pipeline [5, 7]. Our system then uses this query plan to generate its shadow pipelines, each of which executes the following steps: (i) the first step is issue detection, for which the shadow pipeline introduces operators that take intermediates from

the original pipeline as input to screen for potential problems. (ii) For detected issues, the shadow pipeline integrates and evaluates potential fixes, generates the corresponding provenance-based change explanations, and provides a quantification of the expected impact on the pipeline outputs. To reduce overhead, the shadow pipelines only compute small changes compared to the original pipeline by efficiently using intermediates from it as input (sometimes re-using intermediates on a tuple-level [15]) and avoid costly operations like model re-training. In cases where such re-training is unavoidable, they use cheap proxy models [12] to estimate the impact of a change on expensive models like neural networks. For provenance-based explanations, mliidea tracks the fine-grained provenance of individual records throughout the user pipeline and its shadow pipelines. This provenance information is additionally used to construct the inputs for data-centric AI techniques like SliceLine [18] that require information from various parts of the ML pipeline, such as the feature matrix, predicted and ground-truth labels, and the corresponding rows from the initial unfeaturized input data [21].

Interactively improving ML pipelines via IVM. As users iteratively improve their pipelines with the help of our system, they continuously rewrite and rerun their pipelines. In light of such rewrites, we update the original pipeline and its shadow pipelines with incremental view maintenance (IVM) techniques [15], based on detected changes in operators or input tuples. mliidea applies IVM whenever possible, but falls back to full recomputation when necessary, particularly if there are multiple local subsequent operator changes in the query plan. Between different executions of the user’s code, mliidea compares the extracted logical query plan with the query plan from previous executions to detect changes. A query plan node is considered unchanged if the previous pipeline contained an operator with the same function call applied to the same parent nodes with the same non-data arguments. mliidea caches intermediate results for every node, as long as memory allows. During pipeline execution, mliidea incrementally updates intermediates based on detected query plan changes, and falls back to fully executing operations only when necessary.

3 DEMONSTRATION DETAILS

We demonstrate mliidea with a web-based interface (illustrated in Figure 2). This interface allows attendees to interactively improve ML pipelines based on suggestions from three shadow pipelines [3, 12, 13, 18, 20], and to automatically apply the suggested code improvements to existing pipeline code from different domains (social media posts, healthcare, census data) and different ML models (both traditional models and retrieval-augmented LLMs). Attendees will iteratively improve the source code of the ML pipelines, while interactively receiving detailed warnings and suggestions from mliidea. Two of the shadow pipelines can even directly suggest source code changes and apply them using LLM-based code rewriting. Furthermore, attendees can observe in real-time how mliidea drastically reduces execution times during iterative ML pipeline development by using IVM, allowing for quick iterations without fully re-running the pipelines after every small change. The interface also visualizes how mliidea extracts logical query plans from user code, generates shadow pipelines based on it, and incrementally maintains the original pipeline and its shadow pipelines.

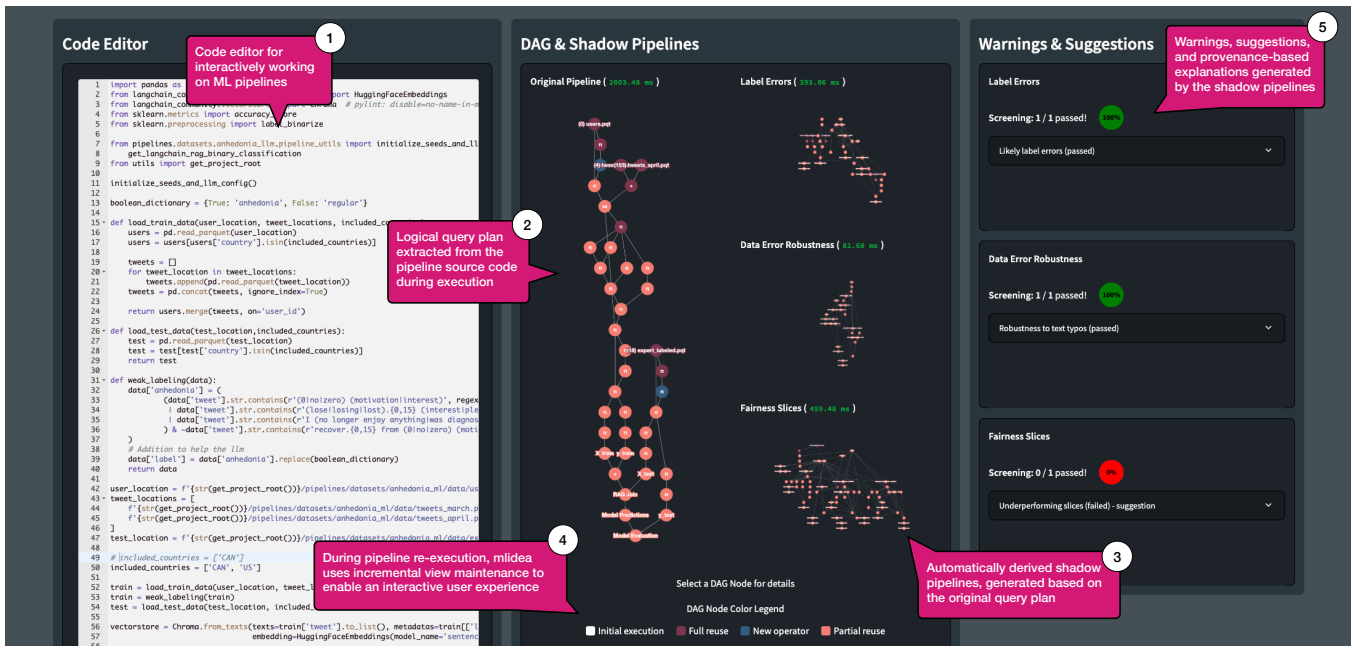


Figure 2: Web-based user interface: On the left, attendees interactively work on the pipeline code ①. In the middle, attendees can inspect how mIdea internally extracts a logical query plan from the code ②, generates shadow pipelines to detect issues and automatically try potential fixes ③, and how it uses incremental view maintenance to enable an interactive user experience ④. On the right, attendees observe issues detected by mIdea, along with suggestions and provenance-based explanations ⑤.

We demonstrate mIdea and its three exemplary shadow pipeline as follows:

- (i) Attendees choose one of our provided ML pipeline scenarios, and we briefly introduce them to the ML pipeline code and the shadow pipelines.
- (ii) Attendees run the ML pipeline with mIdea. Then, they inspect and discuss warnings and suggestions from the shadow pipelines with us and other attendees. These reports include provenance-based explanations for recommended changes and sometimes even automatically applicable code changes.
- (iii) Attendees then interactively address detected problems with the help of mIdea. They experience how mIdea’s IVM helps with quickly iterating on code, and how the shadow pipeline outputs change based on changes to the original pipeline.
- (iv) Finally, we provide attendees with a deeper understanding of how the shadow pipelines and mIdea’s IVM work. Our interface visualizes the query plans extracted from their pipeline code and the generated shadow pipelines, allowing attendees to explore them freely as they iterate on the pipeline code.

In detail, we include the following three shadow pipelines:

Shadow pipeline 1: Interactively refining label quality. This shadow pipeline enables attendees to iteratively refine label quality, a crucial factor in ML pipeline performance. With provenance-based explanations for the results from existing techniques for detecting label errors [12, 13] and IVM for efficient execution of ML pipelines on changed input data, mIdea allows quick iterations on ML pipeline code and data. Unlike previous approaches [14], which

do not consider end-to-end ML pipelines with data preparation steps like data integration, mIdea provides system support for interactively improving data preparation pipelines, a process that is tedious and error-prone without assistance [8].

Warnings and suggestions. mIdea automatically generates a shadow pipeline to detect records that degrade model performance [13]. If such records are detected, mIdea flips their labels in the training data and re-executes the pipeline with IVM. Attendees receive a report, highlighting rows with the lowest Shapley values, along with a provenance-based explanation built by joining problematic featurized rows, assigned labels, predicted labels, and the initial human-readable input data. Similarly, attendees will receive reports for the estimated impact of flipping these labels and provenance-based explanations on how that changes predictions. These reports enable quick iterations on source code and labels.

Optimizations. In case of long-running ML model training, mIdea uses cheap proxy models to estimate the impact of flipping uncertain labels by training them on both the original and updated training data. The shadow pipeline also uses additional optimizations like selectively rerunning operations, such as retrieval-augmented LLM inference, only for predictions that depend on flipped labels.

Interactivity. With mIdea’s warnings and suggestions, attendees then improve the pipeline iteratively. For example, one of our pipelines uses expert-designed regular expressions for weak labeling. Attendees will use mIdea’s provenance-based explanations to interactively refine the regular expressions for weak labeling, and observe in real-time how IVM accelerates this iterative process.

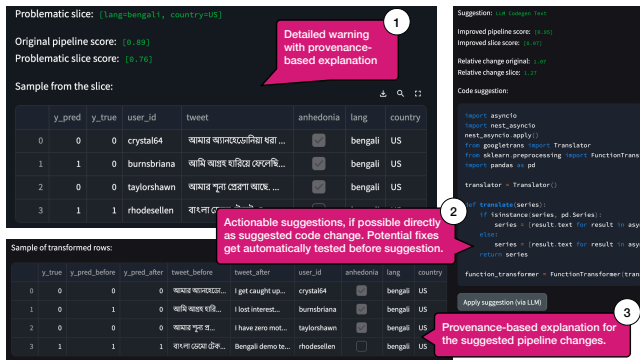


Figure 3: Attendees receive detailed warnings, actionable suggestions, and provenance-based explanations generated by the shadow pipelines.

Shadow pipeline 2: Improving robustness against data quality problems. This shadow pipeline allows attendees to test and improve the robustness of ML pipelines against data quality problems [20] before deployment. For example, in healthcare, a model using patient data and doctor’s notes may be impacted by typos made during stressful periods. Robustness can be improved by applying preprocessing steps like spell-checking to the data.

Warnings and suggestions. mlidea automatically generates shadow pipelines that corrupt columns of different data types in pre-defined ways (missing values for categorical columns, random scaling for numerical columns, and typos for text columns). If robustness issues are detected, mlidea automatically tests potential fixes, such as missing value imputation, outlier detection, and spell-checking. Attendees receive a report detailing the impact of corruptions, potential fixes, provenance-based explanations with samples of the corrupted and fixed rows, the performance impact, and suggested source code changes.

Optimizations. To test a large number of different pipeline variants with different corruptions and fixes, this shadow pipeline selectively reruns operations like model inference only for rows affected by corruptions or data cleaning. While data cleaning operations like spell-checking can be slow, mlidea reuses intermediate results across shadow pipeline executions for these expensive operations and only tests them on samples.

Interactivity. Attendees will investigate reports generated by mlidea with warnings, suggestions, detailed explanations, and executable code snippets. Our demo interface also offers LLM-based functionality for automatically integrating suggested code changes into the original pipeline. Attendees will iteratively refine the pipeline, re-executing it to address issues while leveraging the IVM.

Shadow pipeline 3: Improving prediction performance on problematic data slices. This shadow pipeline uses SliceLine [18] to identify data slices where the ML pipeline underperforms. When detected, mlidea automatically explores ways to improve performance on that slice. For example, in a social media scenario, a subset of user posts might be in a different language. mlidea will detect the problematic slice, apply a translation step, and quantify its impact.

Warnings and suggestions. Attendees receive a detailed report about the problematic slice (if detected) and the strategies mlidea tried to address it. One strategy involves prompting external LLMs with samples from the problematic slice to generate tailored data cleaning operations, such as the translation step on foreign-language posts. To avoid latency issues from heavy reliance on LLMs, mlidea also uses heuristics to come up with additional potential fixes.

Optimizations. Expensive operations like model inference are only run on rows from the problematic slice that were modified by potential fixes. Costly operations like LLM calls and translation steps reuse intermediate results between shadow pipeline executions.

Interactivity. Attendees will investigate warnings with provenance-based explanations about the problematic slice, along with suggestions and reports on how various improvement strategies affect its prediction performance. They will also receive provenance-based explanations of how the strategies transformed the problematic data. Additionally, attendees get code snippets for promising improvements, which they can integrate manually or with the help of LLM-based functionality, or develop their own fixes based on mlidea’s reports. As with all shadow pipelines, attendees will explore extracted and generated query plans and iteratively improve the pipeline while leveraging IVM.

REFERENCES

- [1] Eric Breck, Neoklis Polyzotis, Sudip Roy, Steven Whang, and Martin Zinkevich. 2019. Data Validation for Machine Learning. *MLSys* (2019).
- [2] Zaheer Chothia, John Liagouris, Frank McSherry, and Timothy Roscoe. 2016. *Explaining outputs in modern data analytics*. Technical Report. ETH Zurich.
- [3] Yeounoh Chung et al. 2019. Slice finder: Automated data slicing for model validation. *ICDE* (2019).
- [4] GitHub. 2021. GitHub Copilot - Your AI pair programmer. <https://copilot.github.com/>.
- [5] Stefan Grafberger et al. 2023. Automating and Optimizing Data-Centric What-If Analyses on Native Machine Learning Pipelines. *SIGMOD* (2023).
- [6] Stefan Grafberger et al. 2024. Towards Interactively Improving ML Data Preparation Code via "Shadow Pipelines". *DEEM workshop @ SIGMOD* (2024).
- [7] Stefan Grafberger, Paul Groth, Julia Stoyanovich, and Sebastian Schelter. 2022. Data distribution debugging in machine learning pipelines. *VLDBJ* (2022).
- [8] Stefan Grafberger, Bojan Karlaš, Paul Groth, and Sebastian Schelter. 2023. Towards Declarative Systems for Data-Centric ML. *DMLR workshop @ ICML* (2023).
- [9] Grammarly. [n.d.]. Demo. <https://demo.grammarly.com/>.
- [10] Kenneth Holstein et al. 2019. Improving fairness in machine learning systems: What do industry practitioners need? *CHI* (2019).
- [11] JetBrains. [n.d.]. Code inspections. <https://www.jetbrains.com/help/idea/code-inspection.html#access-inspections-and-settings>.
- [12] Ruoxi Jia et al. 2019. Efficient task-specific data valuation for nearest neighbor algorithms. *VLDB* (2019).
- [13] Bojan Karlaš et al. 2023. Data Debugging with Shapley Importance over Machine Learning Pipelines. *ICLR* (2023).
- [14] Sanjay Krishnan et al. 2016. ActiveClean: interactive data cleaning for statistical modeling. *VLDB* (2016).
- [15] Frank McSherry, Derek Gordon Murray, Rebecca Isaacs, and Michael Isard. 2013. Differential Dataflow. *CIDR* (2013).
- [16] Neoklis Polyzotis et al. 2018. Data lifecycle challenges in production machine learning: a survey. *SIGMOD Record* (2018).
- [17] Romila Pradhan, Jiongli Zhu, Boris Glavic, and Babak Salimi. 2022. Interpretable data-based explanations for fairness debugging. *SIGMOD* (2021).
- [18] Svetlana Sagadeeva and Matthias Boehm. 2021. Sliceline: Fast, linear-algebra-based slice finding for ml model debugging. *SIGMOD* (2021).
- [19] Sebastian Schelter et al. 2018. On challenges in machine learning model management. *IEEE Data Engineering Bulletin* (2018).
- [20] Sebastian Schelter et al. 2021. JENGA - A Framework to Study the Impact of Data Errors on the Predictions of Machine Learning Models. *EDBT* (2021).
- [21] Sebastian Schelter, Stefan Grafberger, Shubha Guha, Bojan Karlaš, and Ce Zhang. 2023. Proactively Screening ML Pipelines with ArgusEyes. *SIGMOD* (2023).
- [22] Julia Stoyanovich, Bill Howe, Serge Abiteboul, H.V. Jagadish, and Sebastian Schelter. 2022. Responsible Data Management. *Commun. ACM* (2022).