DiffML: End-to-end Differentiable ML Pipelines

Benjamin Hilprecht*
hilprechtbenjamin@gmail.com
Technical University of Darmstadt

Christian Hammacher*
chammacher@outlook.de
Software AG

Eduardo Reis
eduardo.reis@cs.tu-darmstadt.de
Technical University of Darmstadt

Mohamed Abdelaal
mohamed.abdelaal@softwareag.com
Software AG

Carsten Binnig
carsten.binnig@cs.tu-darmstadt.de
Technical University of Darmstadt & DFKI

ABSTRACT
In this paper, we present our vision of differentiable ML pipelines called DiffML that truly allows to automate the construction of ML pipelines in an end-to-end fashion. DiffML allows to jointly train not just the ML model itself but also the entire pipeline including data engineering steps, e.g., data cleaning, data augmentation, etc. Our core idea is to formulate all steps in a differentiable way such that the entire pipeline can be trained using backpropagation. However, this is a non-trivial problem and opens up many new research questions. To show the feasibility of this direction, we demonstrate initial ideas and a general principle of how typical data engineering steps can be formulated as differentiable programs and jointly learned with the ML model. Moreover, we discuss a research roadmap and core challenges that have to be systematically tackled to enable fully differentiable ML pipelines.

CCS CONCEPTS
• Information systems → Data cleaning.

KEYWORDS
data engineering, differentiable ML pipelines, data cleaning

ACM Reference Format:
Benjamin Hilprecht*, Christian Hammacher*, Eduardo Reis, Mohamed Abdelaal, and Carsten Binnig. 2023. DiffML: End-to-end Differentiable ML Pipelines. In Data Management for End-to-End Machine Learning (DEEM '23), June 18, 2023, Seattle, WA, USA. ACM, New York, NY, USA, 7 pages.
https://doi.org/10.1145/3595360.3595857

1 INTRODUCTION
In recent years, there have been many breakthroughs in machine learning (ML). However, developing ML pipelines to solve particular tasks is still far from trivial and requires expertise in many areas. In particular, besides training a model, to develop an ML pipeline, we not only have to tackle core tasks regarding model selection and hyperparameter tuning, but we also have to solve many tasks related to data engineering such as data cleaning, data augmentation, data transformation, etc. These tasks typically require a high-level of expertise and are thus a barrier for the broad adoption of ML. Furthermore, even with expert knowledge, these steps are highly time-consuming since they require many manual decisions and experimentation.

Hence, recent efforts concentrate on automating the steps of typical ML pipelines and in particular also data engineering. For instance, it was proposed to automate data cleaning using ML techniques in particular to detect [5, 6] or correct data errors [3, 7], impute missing values [7], or even compensate a potential bias in the data [2].

However, while these automation techniques typically help with the individual tasks, it is not guaranteed that they improve the end-to-end performance of the ML model. For instance, different papers have shown that while automated data cleaning can improve the performance, an unsuitable cleaning method can also significantly deteriorate the accuracy of a (downstream) model [4]. The reason behind this deterioration is that the automated individual techniques often do not target the final accuracy of the ML model but metrics of the particular task (e.g., the accuracy of the imputation strategy).

Vision and Contributions. In this paper, we thus propose our vision of DiffML that is a first approach, which allows to automatically construct the entire ML pipeline including the necessary data engineering steps automatically, while optimizing the end-to-end
In particular, to enable trainable ML pipelines, we propose to express those pipelines as a differentiable program. That way, we can train all steps of an ML pipeline end-to-end; e.g., we can train the data cleaning steps jointly with the ML model by using backpropagation along the full pipeline. However, it is not straightforward to formulate entire ML pipelines as a differentiable program. As a concrete contribution in this paper, we discuss a concrete direction of using so-called mixtures of pipeline alternatives (cf. Section 2) to enable differentiable ML pipelines.

To show the generality of using the idea of mixtures for enabling differentiable ML pipelines, we demonstrate how various pipeline steps can be trained jointly with an ML model by making them differentiable using this idea. As concrete use cases to show the feasibility of our approach, we implemented three concrete scenarios, which differ in the nature of the data engineering steps that are combined with the model training. Furthermore, we think that this paper is only a starting point and more additional research is required to automate other pipeline steps or even explore other directions to enable differentiable ML pipelines beyond the idea of using mixtures.

Note that there have been previous attempts to learn ML models with some ML pre-processing steps such as feature normalization [10]. However, in these approaches data engineering steps are not integrated in an end-to-end fashion since these steps are more complex than simple ML pre-processing steps. Moreover, another direction is AutoML [11] that typically only automates model selection and hyperparameter tuning, whereas steps such as data cleaning are not considered. Moreover, compared to AutoML a major advantage of our approach is that just a single pipeline has to be trained whereas AutoML requires searching the space of potential ML pipelines by training multiple pipelines (e.g., with different data preprocessing steps, different model architectures, and different hyperparameters), which increases the computational overhead significantly.

Outline. In Section 2, we present our vision of expressing the various steps in ML pipelines as a single differentiable program and discuss the advantages. Afterwards, in Section 3, we present more details on how the general idea of DiffML can be applied to the problems of data cleaning, dataset and feature selection and show promising initial results that suggest that differentiable ML pipelines can achieve competitive ML performance with only a single training phase. In Section 4, we present a research roadmap to enable differentiable ML pipelines for a broad set of tasks and finally provide an outlook in Section 5.

2 DIFFERENTIABLE ML PIPELINES

In the following, we give an overview of DiffML and discuss both the advantages of such a design and the key challenges to realize this vision.

2.1 Overview of the Approach

Figure 2 shows the high-level idea of DiffML: we parameterize the search space of data engineering steps in a pipeline and train it jointly with the ML model itself. This way, the data engineering steps that are best suited for the model and the particular task are instantiated automatically.

A major challenge to enable the vision of DiffML is that many steps of ML pipelines can not be trivially expressed in a differentiable way. The reasons behind this challenge are twofold: First, for each pipeline step, users can typically choose from very different methods. For example, various methods exist to impute missing values, e.g., mean imputation, KNN-based imputation, etc. Overall, selecting the best method from a set of methods is a non-differentiable problem by nature. Second, as mentioned before, for some individual pipeline steps, there already exist learned methods which themselves are differentiable (e.g., learned data cleaning). However, integrating these learned approaches into a differentiable ML pipeline and training them end-to-end with an ML model is not trivial, since the optimization objectives typically differ.

Hence, the main idea of DiffML to make ML pipelines differentiable is to express different alternatives as a mixture of pipeline alternatives, where during the training process the weights for the different alternatives are learned (cf. Figure 2 (lower part)). Moreover, we believe that mixtures are just one way and many more options exist, as we discuss in Section 4. In the following, to illustrate the idea, we first sketch how ML pipelines are trained today before we introduce our approach based on the idea of mixtures to achieve differentiable ML pipelines.

ML Pipelines today. Traditionally, the data engineering steps $S$ in an ML pipeline are typically considered fixed and only the ML model $M_0$ itself is trained. Hence, to obtain a prediction for a particular example $x$ in our dataset, we first apply the data engineering steps and feed the result into our model: $M_0(S(x))$. During training, we find the parameters $\theta$ that minimize our loss. For instance, the data engineering step could just be a missing value imputation, where missing values are replaced by some constant dummy value. If we

![Figure 2: Differentiable pipelines can be realized as learning mixtures of alternatives for pipeline steps. This way, we can learn which preprocessing steps are suitable to improve the end-to-end model performance. For example, we could learn alternatives such as $\lambda_1 = 1$, $\lambda_2 = 0$ which means that only the KNN-imputer is used whereas $\lambda_1 = 0$, $\lambda_2 = 1$ uses only the mean imputer. However, we can also learn to combine both imputers (e.g., for $\lambda_1 = \lambda_2 = 0.5$).](image-url)
now want to evaluate an imputation method such as K-nearest neighbors (KNN) for replacing missing values, we would train a different model $M_{\theta'}$ with different parameters $\theta'$ and compare the performance of both pipelines on the validation set. Such retraining can be costly, since the search space of different pipelines grows exponentially as more steps are considered.

**ML Pipelines with DiffML.** Instead in our approach, we not only parameterize the model $M_{\theta}$ but also the data engineering step $S_\lambda(x)$ and optimize both sets of parameters to also train the data engineering step itself. As mentioned before, we require $S_\lambda(x)$ to be differentiable w.r.t. $\lambda$ s.t. a gradient-descent-based optimization can be applied. Unfortunately, expressing pipelines steps as differentiable programs is often non-trivial. However, we believe that often different options can be combined using mixtures, where during training the weights of different alternatives can be learned.

For instance in Figure 2, instead of using a fixed procedure to impute missing values, we use a mixture of KNN-based and mean imputation and learn the weights $\lambda$ of each approach, i.e., we feed the weighted sum of both imputations into the model. In the extreme case of using $\lambda_0 = 0$ and $\lambda_1 = 1$, this reduces to just using a fixed mean imputation in our pipeline. As such, the case of using just a single imputation strategy is simply a special case of this formulation and can still be expressed. In addition, this approach also enables combinations of different imputation strategies (e.g., by using $\lambda_0 = \lambda_1 = 0.5$) which can be beneficial, as we will show in our initial case studies.

**Multi-step Pipelines.** Note that the idea of mixtures can also be applied to multi-step ML pipelines, e.g., if the pipeline consists of more than a single data engineering step and combines data cleaning with feature data augmentation or it chains several data cleaning steps (i.e., value imputation and outlier removal). A general approach is to simply treat each combination of pipeline steps as an alternative in the mixture, e.g., if there are two data cleaning methods $C_1$ and $C_2$ and two feature selection methods $F_1$ and $F_2$, they could be combined by considering $C_1 + F_1, C_2 + F_1, C_1 + F_2$ and $C_2 + F_2$ in the mixture. While this results in an exponential growth of pipeline alternatives, we believe that in the future this search space can be pruned significantly or this technique might not be required at all as we discuss in Section 4.2. However, to demonstrate that the idea of mixtures is feasible, we will apply this technique later in the data cleaning use case for the sub-steps error detection and repair (cf. Section 3.1).

### 2.2 Discussion

As discussed before, our goal is to train the entire ML pipeline using gradient-based optimization, which basically requires that the ML model itself is also trainable using methods such as SGD or similar techniques. While this is true for many popular classes of models including DNNs, linear and logistic regression or support vector machines (SVMs), there are other model types which do not directly adhere to this training regime such as decision trees.

However, recently it was shown that a much broader class of models can be made differentiable by translating the models (e.g., decision trees) into NN layers [10]. This technique can also be applied directly in our setup to enable a more diverse set of models.

Furthermore, motivated by the breakthroughs achieved using DNNs, we believe that it is already very attractive to only focus on DNNs in an initial realization of our vision, which we will be our focus in the remainder of this paper.

Besides having a differentiable model, a key challenge to enable our vision is to express the different steps of ML pipelines in a differentiable way. Hence, to demonstrate the feasibility we will next showcase how three pipeline steps can in fact be formulated in a differentiable manner, namely data cleaning, dataset selection and feature selection. We will discuss the challenges of additional operators as well as other potential future directions in Section 4.

### 3 CASE STUDIES

In the following, we present how three important preprocessing steps (data cleaning, dataset selection and feature selection) can be trained jointly with the ML model by expressing them as differentiable programs.

#### 3.1 Learned Data Cleaning

In this section, we aim to learn which data cleaning procedure to use jointly with the actual model training. Before we explain how our approach works, we provide necessary background.

**Background.** Data cleaning is usually organized in two subtasks: error detection [5, 6] and repair [3, 7]. In the first task, we aim to identify which values in a tuple are erroneous. Having identified the errors, a repair method then aims to predict the correct values before replacing such values in the dataset. Hence, for a combination of a detection method $d$ and a repair method $r$, we obtain a different repaired tuple $\tilde{x}_{dr}$ for every original tuple $x$ in our dataset.

Figure 3: Prediction errors on dirty datasets for data cleaning methods and DiffML (black line, top) and learned weights for data cleaning methods (bottom) in DiffML. DiffML assigns higher weights to more effective data cleaning methods and thus always achieves a competitive performance. Moreover, for finding these weights, DiffML only requires to train a single pipeline.
Traditionally, to find an effective detection and repair method, different ML pipelines with different data cleaning methods had to be trained.

Method. We now show how to express data cleaning as a differentiable program based on the general idea of mixtures introduced in Section 2.1. More precisely, we formulate the search space of different error detection and repair methods as a mixture where every detection and repair method is assigned a weight $\lambda_d$ and $\lambda_r$, respectively. A particular tuple $x_d$, should then be weighted by $\lambda_d + \lambda_r$. During training, we will assign higher weights to more effective pairs. The weighted sum of all repaired tuples $\sum_{d \in D, r \in R} (\lambda_d + \lambda_r)x_{dr}$ could now simply be fed into the model to make the entire pipeline differentiable.

To approximate a probability distribution for the learned weights, we do not directly use the weights $\lambda_d$ and $\lambda_r$ but the softmax, i.e., $e^{\lambda_d + \lambda_r}$ / $\sum_{d \in D, r \in R} e^{\lambda_d + \lambda_r}$. For learning and inference, we use a tuple $x_{dr}$, with renormalized weights as an input to the model $M$: $M_0(\sum_{d \in D, r \in R} \tilde{\lambda}_{dr} x_{dr})$. During training, we can then jointly learn the model parameters $\theta$ and weights $\lambda$ by computing a loss function on the output and then computing gradients for both during backpropagation. We also found that it is also slightly beneficial to not learn both the weights $\lambda$ and the model parameters $\theta$ in a single pass, but in each step of mini-batch learning sample two different batches and update the parameters sequentially.

Initial Results. We study the effectiveness of our approach in an initial experiment where the task is to solve a regression problem on tabular data. In particular, we consider three datasets where each of them has a specific error type which is common in practice: Bike with missing values, Nasa with outliers and Airbnb with typos on 10% of the values each.

To evaluate how state-of-the-art data cleaning can improve the performance of the downstream model, we repaired the datasets using existing state-of-the-art detection methods, including DBoost [6], ED2 [5] and a simple missing value detector. To repair the errors, we used an ML-based imputation strategy (KNN) as well as using the mean. We then trained a Multilayer Perceptron (MLP) on the dirty data as well as the repaired versions and compare the root mean square error (RMSE). In addition, we evaluate DiffML which can choose from all of these detection and repair methods. As an additional baseline, we also train the model on the dirty version of the dataset, i.e., without any cleaning.

As we can see in Figure 3 (top chart), DiffML achieves RMSE values (black solid line) which are competitive compared to the best method for each of the datasets. Importantly, DiffML trains only a single pipeline whereas traditionally multiple pipelines would have to be trained to identify the most effective data cleaning strategy. Moreover, when comparing the individual methods to the baseline model trained on the dirty data without cleaning (black dashed line), we can see that there exist certain combinations which result in inferior performance compared to training on the dirty data, which confirms prior studies. However, this is not the case for DiffML which always improves the model performance.

An interesting observation in Figure 3 is also that the best data cleaning method also depends strongly on the dataset and type of errors, and thus selecting the best strategy is nontrivial. For instance, the ED2 detector works reasonably well for outliers but results in a deterioration in RMSE for typos (Airbnb). In contrast, DiffML finds a competitive combination of detectors and repairs for each data set; i.e., it chooses different detectors and repairs for each of the datasets, as we can see by studying the weights of the individual methods in Figure 3 (bottom). There is a clear trend that DiffML assigns higher weights to data cleaning methods that perform well in isolation (e.g., DBoost and the missing value detectors and ML imputation strategy for missing values). Interestingly, sometimes the combination (i.e., a mixture) of different data cleaning methods results in a superior overall RMSE than each of the individual methods, as we can see for instance in the case of outliers on the (Nasa) dataset.

An interesting future extension of our initial approach could be to also make the actual detection and repair methods also differentiable to train them end-to-end with our approach. A simple example could be to not replace missing values with the mean, but to use a learned value. We believe that there are many more sophisticated approaches to formulate differentiable data cleaning pipelines yet to be explored which can further improve the performance.

3.2 Learned Dataset Selection

Often, ML engineers need to incorporate training data from multiple sources into model training. Especially if only limited data is available, it often has to be complemented with freely available open data [1]. However, for open datasets the label quality is often not clear and thus additional data might not always improve the downstream ML performance when being included in training. As such, selecting which datasets should be included is another problem that data scientists often need to deal with.

Traditionally, to decide which datasets should be included in training, we would train the ML pipelines on different combinations of available datasets. However, this is costly since many pipelines need to be trained to identify which subsets of training datasets should be used. Hence, in this use case we aim to learn which training datasets should be incorporated, where the challenge is to formulate the inherently discrete dataset selection step in a differentiable way. Our idea is to learn a mixture of models, where each of the models is conceptually trained on a different dataset. In the following, we now show how dataset selection can be formulated as a differentiable program.

Method. We now introduce our approach to derive differentiable dataset selection. For ease of exposition, we assume that there are only two datasets $D_1$ and $D_2$ and we want to decide which subset of datasets should be used, i.e., either both the datasets or just a single dataset. However, our method generalizes to $n$ datasets.

Our idea is based on the following observation: if the subset $D_2$ with erroneous labels is ignored, this is equivalent to not considering the gradients $\nabla_{\w} L(x_i)$ coming from examples $x_i \in D_2$ in the model update phase (i.e., during backpropagation). Hence, by weighting the gradients depending on the dataset they belong to,
we can obtain a mixture of models trained on different subsets. For instance, if all examples of $D_2$ obtain weight zero, this is equivalent to just training on $D_1$. More formally, we assign a weight $\lambda_i$ to each dataset and use the following modified update rule instead of standard SGD

$$\theta \leftarrow \theta - \frac{1}{n} \sum_{i=1}^{n} \lambda_{d(x_i)} \nabla_{\theta} L(x_i)$$

(1)

where the function $d(x_i)$ returns the index of the dataset the example was sampled from, i.e., 1 or 2 for datasets $D_1$ and $D_2$, respectively. This can be seen as training a mixture of models using the weights $\lambda_{d(x_i)}$ where we weight the gradients based on the source which the data stems from. For instance, if DiffML uses the weights $\lambda_1 = 0, \lambda_2 = 1$, we obtain a model that is similar as if it would only be trained on the second dataset whereas for $\lambda_1 = 0.5, \lambda_2 = 0.5$ we obtain a model incorporating both datasets equally. Note that a similar weighting of gradients is also compatible with alternative optimization algorithms beyond simple SGD such as Adagrad or Adam. The question now becomes how the weights $\lambda$ can be learned.

Our idea is to learn the weights $\lambda$ based on how the gradients $\nabla_{\theta} L(x_i)$ influence the loss of the model. Intuitively, if the model update caused by the gradient of the example $x_i$ increases the loss of the model, the weight of the corresponding dataset should be lowered. To this end, we first compute the model gradients $\nabla_{\theta} L(x_i)$ of a data batch as usual, and afterwards draw a second batch for the updated model (according to Equation 1). However, this time we compute the gradient w.r.t. the weights $\lambda$ depending on how the previous $\theta$ gradient updates influence the loss. Accordingly, we update the weights $\lambda$ s.t. gradients from high-quality datasets improving the model performance are preferred. This method is in particular beneficial if the second batch to update $\lambda$ is drawn from a clean validation set, which could in practice be a small high-quality dataset where the user is sure that it only contains little errors. As such, we can choose larger weights for datasets, which overall reduces the loss of the model when they are included in the training. Moreover, similar to the first use case, we do not use the weights directly but the softmax of the weights. Overall, the idea of differentiable dataset selection is thus related to methods aiming to assign weights to individual records in the dataset [8]. However, our approach is different since we aim to solve dataset selection instead of deciding on a per-example basis whether it should be included.

*Initial Results.* For the setup in our initial evaluation, we consider two datasets $D_1$ and $D_2$ where the second dataset has low label quality, and it may deteriorate the model performance if it is included in the training. We obtained the datasets by splitting the datasets used in the first experiment in two subsets and perturbing a certain percentage of labels of the second dataset by randomly swapping labels. We then compare the RMSE of DiffML, which can learn to assign lower weights to the dirty dataset, to a baseline method which uses both datasets equally; i.e., the baseline is an MLP trained over the union of both datasets.

As we can see in Figure 4, the proposed method of using DiffML always provides an improvement over the baseline model (referred to as Default in Figure 4) trained on the full dataset. In general, the improvement is dependent on the dataset. For instance, the Bike dataset is very sensitive to noise, therefore our method drastically reduces the RMSE. On the other hand, on datasets such as Airbnb that are robust to noise (i.e., where the introduced errors do not deteriorate the model performance significantly) the method only yields slightly more accurate results as expected.

### 3.3 Learned Feature Selection

In ML pipelines, it can sometimes be beneficial to exclude some features from the training process, for instance in cases where features are uninformative or correlated (and thus do not provide additional information). However, selecting the best suited set of features is non-trivial but can have a significant impact on the end-to-end performance of the ML task.

State-of-the-art methods for feature selection include filter-based methods, which compute metrics such as correlation with the target variable to select the $k$ best features according to the metric (where $k$ is a hyperparameter), wrapper-based methods which incrementally train models on subsets of features to find a suitable set of features, and embedded methods which exploit characteristics of a model class (e.g., Lasso for linear models, feature importance for decision trees etc.). While both filter-based models and wrapper-based techniques require multiple ML pipelines to be trained (which can quickly become costly for a larger number of features), embedded methods are limited to specific model classes and can often not be trained in a differentiable way. An alternative to feature selection are dimensionality reduction techniques such as PCA, which again require multiple pipelines to be trained to find a suitable number of target dimensions $k$.

*Method.* For this paper, we propose differentiable feature selection, where we again rely on the intuition of mixtures of different ML pipelines. In particular, a feature subset can be seen as setting some features to zero in the feature vectors, which are thus ignored. Hence, instead of using all features in our ML pipeline $(x_0, x_1, \ldots, x_n)$, we again assign a learnable weight to each feature which is normalized between zero and one using the sigmoid function $\sigma(\cdot)$. Hence, we use the feature vector $(\sigma(\lambda_0)x_0, \ldots, \sigma(\lambda_n)x_n)$ in our ML pipeline and again learn the values of the weights $\lambda$. 

![Figure 4: Performance of DiffML for learned dataset selection. In all cases, DiffML reduces the error (RMSE) by assigning lower weights to the latter one compared to a vanilla model that is trained on a union of the clean and dirty datasets.](image)
FAMD Reduction

**4 THE ROAD AHEAD**

There is still research needed to generalize our findings in the case studies in different directions. The main research challenges are (i) coverage, i.e., expressing other pipeline steps besides the three of our case studies in a differentiable way and (ii) support for end-to-end pipelines.

### 4.1 Other Pipeline Steps

A first interesting research direction is how to express additional operations beyond the ones presented in our case study such as data augmentation or data transformation in a differentiable way since this broadens the applicability of differentiable ML pipelines. An initial approach could be to apply our notion of mixtures also to the remaining operators, e.g., by applying our methods for the dataset selection use case on different augmented versions of a dataset to find a suitable set of augmentations.

While this approach can already extend the set of differentiable pipeline steps, there is a fundamental limitation to a mixtures-based formulation: in a nutshell, we simply learn from a (combination of) alternative pipeline steps which ones to select (e.g., mean imputation vs. KNN imputer for missing values) whereas the alternatives themselves remain static (i.e., either mean or the result of the KNN model or a mix is imputed). An alternative could be to also express not just alternatives of pipeline operators but the operators themselves in a differentiable way (e.g., to use a differentiable model for the imputation) which significantly increases the space of solutions. While this is straightforward to apply for data imputation, it is unclear how to express for instance data augmentation strategies using differentiable models.

### 4.2 Multi-step Pipelines

As mentioned before, our approach using mixtures also generalizes to multi-step pipelines, e.g., with feature selection and data cleaning by considering all combinations of feature selection and data cleaning as alternatives in our mixtures. However, this approach does not scale to longer pipelines since the number of combinations of pipeline steps, which would all have to be considered, grows exponentially in the length of the pipeline. However, we believe that there are many research opportunities to overcome this problem by either (i) applying pruning or (ii) designing inherently differentiable pipeline steps without mixtures.

First, we believe that pruning can already drastically reduce the number of pipeline combinations, which have to be considered. For instance, if two missing value imputation and outlier detection methods operate on different columns, not all combinations have to be considered since they operate independently. Hence, the number of pipeline alternatives in the mixture can be pruned significantly. We believe that there are more involved techniques, which could further reduce the search space.
Second, the pipeline steps themselves could be implemented as differentiable programs as previously mentioned. In this case, the exponential explosion of mixture alternatives is entirely avoided and the differentiable pipeline steps can simply be chained and learned end-to-end. For instance, a simple missing value imputation strategy could be a DNN that predicts the missing value given the other column values. This strategy can be used before and after other differentiable pipeline steps because it is fully differentiable w.r.t. to its input. For instance, two such imputers for different columns could simply be chained together and no cartesian product of alternatives would have to be considered. While it is not straightforward to express all pipeline steps in a differentiable way without mixtures, it scales to arbitrary pipeline lengths and therefore poses an interesting avenue for future research.

5 CONCLUSION

In this paper, we have presented our vision of differentiable ML pipelines based on the idea of mixtures. This vision has the advantage that the entire ML pipeline is optimized for downstream performance, which prevents that pre-processing steps can harm the model accuracy. In our initial case studies, we have shown how to express different data engineering steps as differentiable programs, which shows the generality of our approach.

We believe that there are many research challenges to enable our vision, both in formulating more pipeline operators in a differentiable way but also supporting complex multistep pipelines. We hope that this research can converge into a new class of systems for differentiable ML pipelines which enable efficient and fully automated construction of ML pipelines with high accuracy and could thus be an important contribution to further democratize ML.

ACKNOWLEDGMENTS

This research is funded by the BMBF projects AICoM and KomPAKI (grant numbers 02P20A064 and 02L19C150), the IML4E project (grant number 01IS21021A, ITEA project number 20219), the Merck Sustainability Hub (project LC IT Age), the state of Hesse as part of the NHR Program, as well as the HMWK cluster project 3AI (The Third Wave of AI). Finally, we want to thank hessian.AI, TU Darmstadt as well as DFKI Darmstadt for their support.

REFERENCES

[1] Sonia Castelo, Rémi Rampin, Aécio Santos, Aline Bessa, Fernando Chirigati, and Juliana Freire. 2021. Auctus: A Dataset Search Engine for Data Augmentation. arXiv preprint arXiv:2102.05716 (2021).
[2] Benjamin Hilprecht and Carsten Binnig. 2021. ReStore - Neural Data Completion for Relational Databases. In SIGMOD ’21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021, Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava (Eds.). ACM, 710–722. https://doi.org/10.1145/3448016.3457264
[3] Mohammad Mahdavi and Ziawasch Abedjan. 2020. Baran: Effective Error Correction via a Unified Context Representation and Transfer Learning. Proc. VLDB Endow. 13, 12 (Jul 2020), 1948–1961. https://doi.org/10.14778/3407790.3407801
[4] Felix Neutatz, Binger Chen, Ziawasch Abedjan, and Eugene Wu. 2023. From Cleaning before ML to Cleaning for ML: Data Engineering (2021), 24.
[5] Felix Neutatz, Mohammad Mahdavi, and Ziawasch Abedjan. 2019. ED2: A Case for Active Learning in Error Detection. In Proceedings of the 28th ACM International Conference on Information and Knowledge Management, CIKM 2019, Beijing, China, November 3–7, 2019, Wenwu Zhu, Dacheng Tao, Xueqi Cheng, Peng Cui, Elke A. Rundensteiner, David Carmel, Qi He, and Jeffrey Xu Yu (Eds.). ACM, 2249–2252. https://doi.org/10.1145/3357384.3358129
[6] Clément Pit-Claudel, Zélida Mariet, Rachael Harding, and Sam Madden. 2016. Outlier Detection in Heterogeneous Datasets using Automatic Tuple Expansion. (02 2016).
[7] Theodoros Rekatsinas, Xu Chu, Bab F. Ilyas, and Christopher Ré. 2017. HoloClean: Holistic Data Repairs with Probabilistic Inference. Proc. VLDB Endow. 10, 11 (Aug 2017), 1190–1201. https://doi.org/10.14778/3137628.3137631
[8] Mengye Ren, Wenyuan Zeng, Bin Yang, and Raquel Urtasun. 2018. Learning to Reweight Examples for Robust Deep Learning. In Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research), Jennifer Dy and Andreas Krause (Eds.), Vol. 80. PMLR, 4334–4343. https://proceedings.mlr.press/v80/ren18a.html
[9] Gilbert Saporta. 1990. Simultaneous analysis of qualitative and quantitative data. In Societa Italiana di Statistica. XXXV riunione scientifica, Vol. 1. CEDAM, 62–72.
[10] Gyeong-In Yu, Saeed Amizadeh, Sehoon Kim, Artidoro Pagnoni, Ce Zhang, Byung-Gon Chun, Markus Weimer, and Matteo Interlandi. 2021. WindTunnel: Towards Differentiable ML Pipelines beyond a Single Model. Proc. VLDB Endow. 15, 1 (Sep 2021), 11–20. https://doi.org/10.14778/3485450.3485452
[11] Marc-André Zöller and Marco F. Huber. 2021. Benchmark and Survey of Automated Machine Learning Frameworks. J. Artif. Intell. Res. 70 (2021), 409–472. https://doi.org/10.1613/jair.1.11854

Received 16 March 2023; accepted 20 April 2023