SliceTeller: A Data Slice-Driven Approach for Machine Learning Model Validation

Xiaoyu Zhang*, Jorge Piazentin Ono*, Huan Song, Liang Gou, Kwan-Liu Ma, Liu Ren

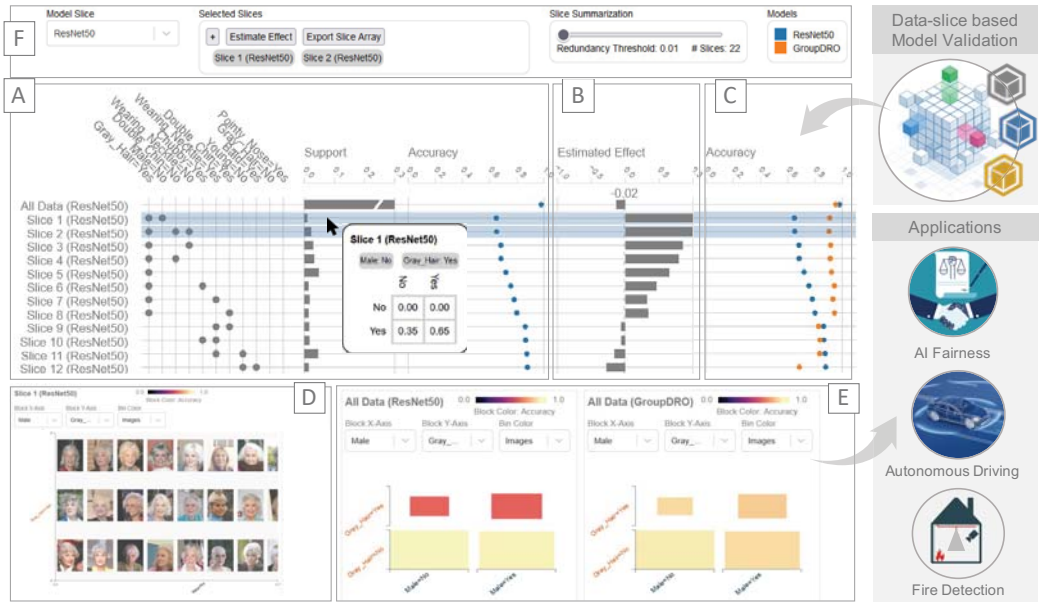

Fig. 1. *SliceTeller* applied to the comparison of two machine learning models (*ResNet50* and *GroupDRO*) for hair color classification (gray hair, not gray hair), trained on the CelebFaces Attributes Dataset (CelebA). (A) Slice Matrix: The data slices (represented as rows), slice descriptions (encoded as columns), and slice metrics (Support and Accuracy). Slices are sorted by model accuracy. (A - *Tooltip*) Confusion matrix for Slice 1. (B) Estimated effects of optimizing the model for two data slices (Slices 1 and 2, highlighted in blue). (C) Accuracy comparison between the two models, *ResNet50* and *GroupDRO*. (D) Slice Detail View containing image samples from a data slice. (E) Slice Detail View containing the comparison of two data slices using the *MatrixScape* visualization. (F) System menu, containing options for model selection, effect estimation of focusing on a slice during model training, and data slice summarization.

Abstract— Real-world machine learning applications need to be thoroughly evaluated to meet critical product requirements for model release, to ensure fairness for different groups or individuals, and to achieve a consistent performance in various scenarios. For example, in autonomous driving, an object classification model should achieve high detection rates under different conditions of weather, distance, etc. Similarly, in the financial setting, credit-scoring models must not discriminate against minority groups. These conditions or groups are called as "*Data Slices*". In product MLOps cycles, product developers must identify such critical data slices and adapt models to mitigate data slice problems. *Discovering* where models fail, *understanding* why they fail, and *mitigating* these problems, are therefore essential tasks in the MLOps life-cycle. In this paper, we present ***SliceTeller***, a novel tool that allows users to debug, compare and improve machine learning models driven by *critical* data slices. *SliceTeller* automatically discovers problematic slices in the data, helps the user understand why models fail. More importantly, we present an efficient algorithm, ***SliceBoosting***, to estimate trade-offs when prioritizing the optimization over certain slices. Furthermore, our system empowers model developers to compare and analyze different model versions during model iterations, allowing them to choose the model version best suitable for their applications. We evaluate our system with three use cases, including two real-world use cases of *product development*, to demonstrate the power of *SliceTeller* in the debugging and improvement of product-quality ML models.

Index Terms—Model Validation, Data Slicing, Data Validation, Model Evaluation, Data-Centric AI, Human-in-the-loop

1 Introduction

Recently, Machine Learning (ML) has been used in a variety of critical applications, including autonomous driving, medical imaging, industrial fire detection, and credit scoring [18, 27, 35, 42]. Such applications need to be thoroughly evaluated before deployment in order to assess model capabilities and limitations. Unforeseen model mistakes may cause serious consequences in the real world: for example, a false sense of security in ML models may cause safety issues in driver-assistance [18] and industrial systems [27], misdiagnoses in medical analysis [42], and biases against minorities [35].

During our collaborations with MLOps (Machine Learning Oper-

- * *These authors contributed equally.*
- *Xiaoyu Zhang and Kwan-Liu Ma are with UC Davis. Xiaoyu Zhang was an intern at Bosch Research. E-mails: {xybzhang, klma}@ucdavis.edu.*
- *Jorge P. Ono, Huan Song, Liang Gou and Liu Ren are with Robert Bosch Research and Technology Center, USA - Bosch Center for Artificial Intelligence. E-mails: {jorge.piazentinono, huan.song, liang.gou, liu.ren}@us.bosch.com.*

Manuscript received 31 March 2022; revised 1 July 2022; accepted 8 August 2022. Date of publication 30 September 2022; date of current version 2 December 2022. This article has supplementary downloadable material available at https://doi.org/10.1109/TVCG.2022.3209465, provided by the authors.
Digital Object Identifier no. 10.1109/TVCG.2022.3209465

ations) engineers for product-quality model development, we have identified that the evaluation of critical ML models is usually conducted beyond the aggregated level (e.g., a single performance metric). Instead, they need to thoroughly evaluate model performance on carefully specified usage scenarios or conditions in order to meet important ML product requirements. Based on this analysis, experts can then take actions to 1) attempt to make the model more robust to various conditions and 2) make customers aware of model limitations in certain conditions, aiding in the development of mitigating measures. During the evaluation of ML models, model developers often have to slice their data based on the specified product usage conditions, to ensure satisfactory performance under such critical conditions. For example, in the autonomous driving setting, experts need to ensure high detection rates for multiple environmental conditions, such as sunny weather and rain, and specific object types, such as cars and pedestrians. Figure 2 shows a common workflow for critical model analysis and iteration.

Challenges on Model Evaluation and Iteration. While the slice-based analysis is essential for the critical applications, this approach has several limitations. 1) Manually slicing the data is a very time-consuming task. In our interviews, experts mentioned that this task involved manually creating rules to slice the data, running evaluation scripts on the data subsets, and comparing the results on various data subsets. 2) ML experts cannot explore all possible data subsets to identify relevant failures cases for their application. Data slices can be created by any number of interpretable meta-data (e.g., weather and temperature for autonomous driving), resulting in an exponentially large search space. Therefore, they must rely on domain-specific priors to select what meta-data they will slice the data based on. 3) Once the critical failures are identified, experts have the options to either collect more data to cover the weakness scenarios or retrain their models by prioritizing the critical slices. While the former requires additional investment on data collection, the latter is usually time-consuming, particularly for training neural network architectures. Moreover, it is unclear how the new model will trade-off performance on other slices and whether the result can still meet the product requirements.

Our Approach. We develop *SliceTeller*, a novel data slice-driven model validation tool that automates slice finding, enables slice-based model validation and comparison, and allows what-if analysis for slice prioritization. Our tool takes as input the data (non-interpretable features), metadata (interpretable properties that can be used to slice the data), dataset labels and model predictions. A state-of-the-art slice-finding algorithm is adapted to find slices on the data for which a performance metric (for example, accuracy) is significantly different from the overall model metric. Once these slices are found, we use a binary matrix graphical encoding to show the data slices compactly, as well as metrics for these data slices (Figure 1). We also provide various visualization to help users understand and interpret these data slices. After a data slice of interest is found, users can estimate the performance impact of optimizing the model for this data slice using our effect estimation algorithm, called *SliceBoosting*. Finally, users can quickly find problematic data slices on their model and derive actionable insights that can be used to improve the results according to their product requirements in a next training iteration.

Our main contributions include the following:

1. A novel Visual Analytics (VA) tool, *SliceTeller*, for the evaluation and comparison of ML models using a data slicing approach. *SliceTeller* combines data slice finding, together with effective visual representations for data slices and model metrics, to facilitate the iteration and comparison of ML models.

2. An efficient algorithm, *SliceBoosting*, for the quick estimation of data slice trade-offs during model training to improve model iteration efficiency. This algorithm estimates the performance effect of focusing on one or more data slices during model training, highlighting potential trade-offs between data slice optimization and overall model performance.

3. Three use cases, including two real-world use cases of product development, to demonstrate the effectiveness of our approach in assessing, comparing, and iterating over ML model results. We believe our method and design process together with product R&D partner and MLOps Engineers can benefit the practice and research of using VA for model validation at large.

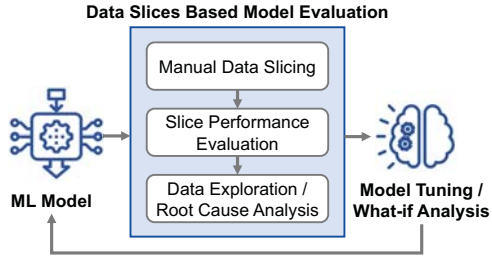

Fig. 2. Product MLOps engineer's workflow for model validation and iteration over critical data slices. Experts sliced their data based on product and domain requirements, computed model performances per slice, and explored the data to identify the root causes for potential model mistakes. Based on these observations, they would iterate over the model, by retraining while re-prioritizing certain data slices over others.

In summary, *SliceTeller*, is a novel VA tool for ML model validation with a data-slice driven approach. This tool is model-agnostic and can be easily plugged in MLOps life-cycles. This work also resonates with recent data-centric AI trends focusing on data instead of models. We hope this work can inspire more research questions innovating VA approaches to address data-driven model validation challenges.

2 Related Work

2.1 Slice-Oriented Model Validation

Finding the subgroups of specific quality is one of the classical combinatorial optimization problems, named Constraint Satisfaction Problem (CSP) [51]. This problem is defined as slice finding in the context of ML model evaluation, the aim of which is to identify the data subgroups over which the ML subgroups underperforms [2].

Most of the existing solutions in commercial tools use greedy heuristics to balance the tradeoff between searching speed and accuracy. For instance, the FreaAI in IBM IGNITE [1], Slice Finder in Tensorflow [7,8], Amazon Sagemaker [32], and RobustnessGym [15] are built upon heuristic techniques such as clustering, self-defined metrics (e.g., highest posterior density), model-based, and rule-based data slicing. Efficient as they are, heuristic solutions could miss critical data slices if they fall into the blind area of the heuristic rules (e.g., the slice size is too small). Thanks to the development of parallel computing devices, researchers could solve this problem by using exhaustive searching with a reasonable time cost. For example, SliceLine includes size and score pruning to boost the performance of exhaustive search, which can be easily deployed onto the parallel devices [47]. DivExplorer [43–45] enumerate the data lattice to look for all candidate itemsets with the highest divergence and support the customization of itemset size. Since such flexibility would allow us to let the users decide the slices they are interested in, we choose DivExplorer as the slice finding tool for our system. The research work in this direction is still in the exploratory stage, and there is an increasing trend of introducing ML for solving the classical combinatorial optimization problems [21, 34].

However, these approaches mainly focus on search efficiency and scalability, but largely ignore how to understand and interpret the impact of these data slices. Also, it is a non-trivial task to customize and select data slices of interest based on domain-specific requirements. This is where our VA-based approach can help.

2.2 Model Robustness over Data Groups

ML robustness research focuses on achieving consistent model performance under various conditions. Due to data collection, sampling or annotation biases, ML models trained on real-world data tend to identify *spurious correlations*, connections that appear extensively in the training data, but do not hold in novel scenarios [28, 39]. Under distribution shift, models that rely on these spurious correlations often degrade significantly in performance [13, 36].

One powerful approach to alleviate this issue is group distributionally robust optimization (DRO) [20, 41]. Group DRO utilizes prior

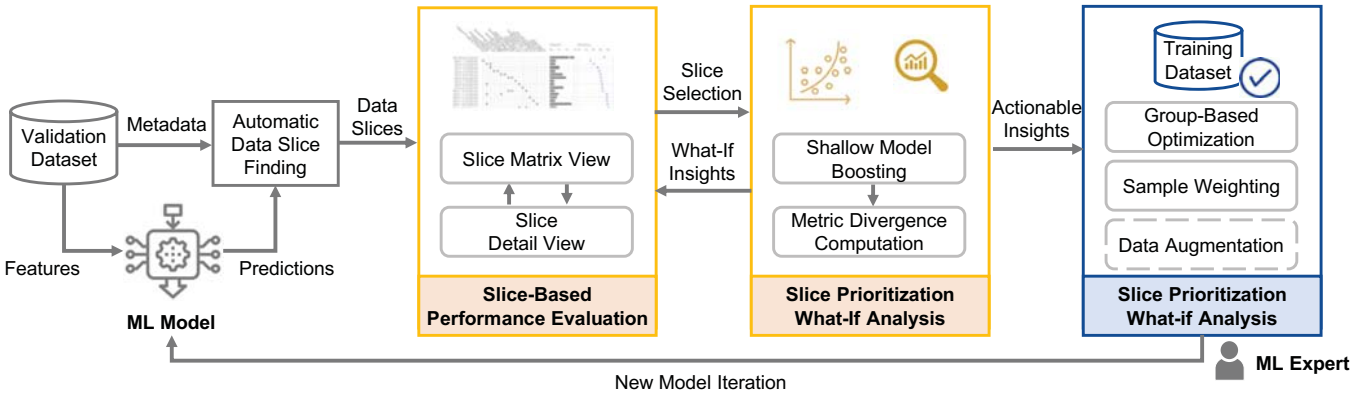

Fig. 3. Workflow of the model analysis and improvement using *SliceTeller*. The validation data, together with the model predictions, are used for the automatic slice identification. The produced data slices can be explored using our VA solution (Slice Matrix and Slice Detail View). Users can prioritize groups of data slices and quickly evaluate the effect of this action on the rest of the model slices. Finally, experts can use the insights gained from the system to fine tune the model and continue the analysis with *SliceTeller*.

knowledge of spurious correlation to define groups over training data, and optimizes the worst-case loss among the groups, instead of the expected loss of the entire distribution. As a result, the trained model is capable of significantly improving minority group performance while maintaining similar performance over the majority groups. In [48], the authors investigate group DRO in the context of over-parameterized deep neural networks (DNNs). They discovered that as opposed to shallow ML models that can directly benefit from group DRO, DNNs require strong regularizations during training to achieve minority group improvements. Recent works along the direction of group DRO research incorporates instance-level weighting to tackle imperfect group partition [57], leverages human annotation to discover and optimize over unmeasured variables [50], and scales up the optimization method for large-scale problems [29].

In this paper, we utilize distributionally robust deep neural network described in [48] as the slice optimization model. Our VA framework addresses the following core limitations of group DRO for practical usage: a) Group DRO requires that the group information of training data is known beforehand. Although empirical results have provided evidence of spurious correlations in popular public datasets [28, 41], such insights are not readily available for most real-world ML problems. b) Group DRO training is time-consuming. It is therefore infeasible to exhaustively apply group DRO on all combinations of data slices in order to determine the ideal slices for optimization. By integrating slice discovery, effect estimation and model optimization under the same VA framework (Figure 3), we help users identify problematic slices, understand the optimization trade-off among slices, and eventually optimize the model, thus significantly speeding up model iteration.

2.3 Visualization for Slice-Based Model Optimization

There are various VA techniques supporting the creation and analysis of data slices [12, 25, 54]. In the context of model exploration, CoFact [22] is a VA system that helps users explore counterfactual subsets and thus understand the confounding facts in large and complex datasets. CoFact is close to the first part of our work, but its focus is on spurious feature correlations, not providing any model or data optimization recommendations. FairVIS [3] and Manifold [56] also allow data subgroup analysis based on human interaction, but do not provide automated methods for data slicing. Finally, Errudite [52] provides a domain-specific language for data grouping with unstructured text data analysis, while we mainly focus on structured data.

Most of these works focus on the data exploration stage and rarely provide optimization solutions to close the entire loop. Therefore, we provide a VA solution for the entire loop of slice-based model optimization that supports slice finding, model optimization, and model comparison with a Matrix visualization inspired by UpSet [30] and PipelineProfiler [40], along with carefully-designed interactions driven by the domain requirements described in Sec. 3.1. To the best of our knowledge, our system is the first of its kind to provide an end-to-end solution for users to identify, understand and optimize model performance on problematic data slices.

3 *SliceTeller*

In this section, we describe *SliceTeller*, a system for data slice-driven validation of ML models. We first present a desiderata for our system, distilled from interviews with four industry partners from the product R&D team. Next, we describe the visual components of our system. Finally, we present *SliceBoosting*, an algorithm that can estimate performance divergences after a slice-based model optimization.

3.1 Data and Domain Requirements

SliceTeller was developed based on a collaborative product project with three industry MLOps engineers over the period of six months. We received continuous feedback from our partners, who worked on tabular and image-based classification problems. These experts work on products in critical applications, including autonomous driving and fire detection for building security. Therefore, they often conduct extensive validation of their models before deployment to production.

In the context of autonomous driving, experts were interested in modeling the ultrasonic sensors to understand the car surroundings. It is a critical modality in the sensor-fusion pipeline to enhance the overall system robustness. The raw ultrasonic sensor data are not directly interpretable by human. However, every sample also contained metadata describing the experiment setup, for example, the object type, distance, sensor location, time of day, etc. Experts had trained a tree boosting model to classify nearby objects' heights (as "high" or "low") using the sensor-derived tabular features. While evaluating their models, they wanted to make sure that certain critical objects had a low error rate. In some cases, they would trade-off between the performance of non-critical objects for the performance of the critical objects. For instance, children, curbstones, and nearby cars should have the highest priority. Therefore, in every evaluation iteration, they had to slice the data, evaluate the model on the data subsets, and retrain the model with different parameters to mitigate critical mistakes. Experts mentioned that these tasks were tedious and time-consuming.

For the fire detection application, experts trained a deep neural network to detect smoke and fire on video frames. In this setup, every video segment was associated with interpretable metadata that described the video collection process in detail, for example, the recording location, time of day, the smoke density, and whether there were blinking lights in the scene. While the overall performance of this model was high, experts were interested in identifying situations where it failed. Therefore, they spent a large amount of time inspecting the model and using the video metadata to identify these situations. Transparency with customers was a high priority for model release: they wanted to clearly communicate the model capabilities, where it was effective and where it failed. Furthermore, they wanted to understand why the model

was failing, and what were the possible confounding features on their dataset.

While these two applications used different techniques and data types, the experts developing them shared a common goal. Figure 2 shows the model evaluation workflow derived from our interviews. In both cases, experts desired to slice the data into various scenarios, thoroughly evaluate their models, understand the failure cases, and develop strategies to tune the models to improve performance. They noted that this workflow usually took several iterations, requiring significant effort and trial-and-error. Based on these observations, we have compiled the following desiderata for a data slice-driven model validation system:

[R1] Data Slice Finding and Overview: Users often spend a significant amount of time slicing the data based on certain heuristics to learn the boundaries of the model. Our system should automatically identify these data slices in the validation dataset and present an overview to the user.

[R2] Slice-Based Data Understanding and Valuation: Users should be able to explore the data slices in order to understand them and value how critical they are. The system should allow the user to explore the data, model metrics, and distributions to explain these scenarios.

[R3] Slice-Based Model Optimization: In critical applications, experts need to trade-off the performance of certain scenarios in order to focus on critical use cases. To do so, they need to train models from scratch, which can be very time-consuming. The system should enable the quick experimentation with the slice-based model optimization, highlighting possible trade-offs in the data.

[R4] Slice-Based Model Comparison: Users need to train and evaluate multiple models in order to tune parameters and mitigate problems. However, this comparison is usually done at the aggregated level (e.g., a single metric value). The system should allow the comparison of model performances at the slice level, facilitating the identification of trade-offs between data slices.

3.2 System Workflow

In order to fulfill the requirements identified in the previous section, we developed *SliceTeller*, a system that tells a story about the evolution of ML models from the perspective of data slices, allowing their evaluation, exploration and comparison. Figure 3 shows the general workflow for model analysis and improvement with *SliceTeller*. The input to *SliceTeller* is the validation dataset consisting of validation data (e.g., raw images or tabular features extracted from the sensor signals), metadata (interpretable features that can be used to slice the data), and ground truth labels (e.g., object classes or obstacle height). Note that we use a validation dataset for model analysis instead of training data since it is unseen by the model. In the case of model overfitting, we observe all slices in the training data having high accuracies. Given the input, the system works as follows:

1) First, the system uses an automatic slice finding algorithm to identify data slices where the performance measures (e.g., accuracy) are the most different from the overall model performance. We use the DivExplorer algorithm [44], a Frequent Pattern Mining-based approach for this task (Section 3.3).

2) Next, a VA system allows the users to quickly visualize and summarize the produced data slices using the Slice Matrix View (where rows correspond to slices, and columns, to slice descriptions and associated metrics). The user can select slices to view its details using the Slice Distribution View, which can present metadata distributions and correlations to the user. These two views allow the user to identify critical slices in the data, i.e. slices where the model performance has issues (Section 3.4).

3) When a critical slice is found, the user can test mitigating measures using the 'Slice Prioritization - What-If Analysis' tool. This tool uses our *SliceBoosting* algorithm to evaluate the effect of optimizing the model for particular data slices. *SliceBoosting* fits a shallow boosting model on top of the original model to estimate the effect of prioritized optimization (Section 3.5).

4) Finally, when the user found a group of slices to optimize, they can export the selected slices back to their programming environment, make changes on data, hyperparameter or model, and insert the new model back into *SliceTeller* to compare models (Section 3.6).

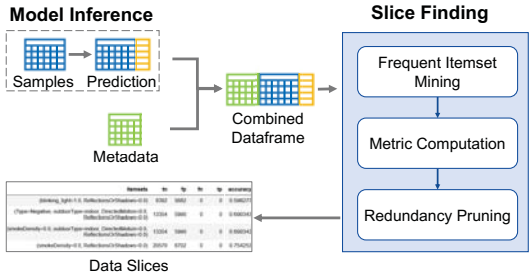

Fig. 4. Data slicing workflow. It takes as input model predictions combined with interpretable metadata and utilizes frequent itemset mining to automatically identify the most critical slices. It then performs slice merging and redundancy removal to generate concise data slice results.

3.3 Data Slicing

We begin our analysis by automatically finding problematic data slices using the DivExplorer algorithm [44] (illustrated in Figure 4). The algorithm takes the model predictions and the meta-data (interpretable features of the dataset) as input, and executes an exhaustive slice search by frequent pattern mining [17]. The *minimum support* (i.e., minimum slice size) is defined as a parameter by the user. Then, a model metric such as accuracy is computed for every data slice found.

To reduce the searching time for datasets with a larger number of metadata features, we conduct a two-iteration slice finding procedure using DivExplorer. First, we run DivExplorer with a large minimum support to identify the relevant metadata features which are most correlated with poor performance. Then, we run DivExplorer again using the relevant metadata features, this time with a lower minimum support to perform a more fine-grained search. The level of granularity (minimum support) can be fine-tuned by the user in order to find the relevant slices for their model. For example, users can fine-tune the parameter to find slices with sufficiently high support and low performance (such attributes are problem-specific and user-defined).

The DivExplorer algorithm can find an exponential number of data slices (exponential in the number of unique feature-value pairs). Therefore, we use a summarization approach to reduce the number of slices to be explored by the user **[R1]**. We allow users to summarize data slices with a redundancy pruning approach [44] (slider in Figure 1(F)). If the introduction of an item α (e.g., $Weather = Sunny$) in a slice S will cause an absolute performance change below a redundancy threshold ε, only the slice without α (denoted $S \setminus \{\alpha\}$) will be presented to the user. This guarantees that the more general slices can be investigated first. More specifically, let p be the function that computes the performance score on a data slice. A data slice S is pruned if: $|p(S) - p(S \setminus \{\alpha\})| < \varepsilon$.

3.4 Visualization Design

Figure 1 demonstrates the visualization design of *SliceTeller*. The main visualization components of *SliceTeller* are the Slice Matrix (A) and the Slice Detail View (D-E). The Slice Matrix shows a summary of all the data slices with a performance metric that diverges from the overall model. The user can drill down on the slices in order to explore one or more data slices simultaneously using the Slice Detail View. The System Menu (F) allows users to switch between data slices from the multiple ML models, summarize model slices and perform What-if analyses to estimate the effect of optimizing the model for a particular data slice. These operations are described later in this section.

Slice Matrix

The Slice Matrix (Figure 1(A)) provides an overview of the problematic data slices to the user **[R1]**. First, the data slices are identified using DivExplorer [44], described in Section 3.3. After the data slices are found, they are graphically represented using *Slice Matrix*, an adaptation of the UpSet [30] matrix encoding, where sets are represented as rows, and set members, as columns. In the context of data slices, we

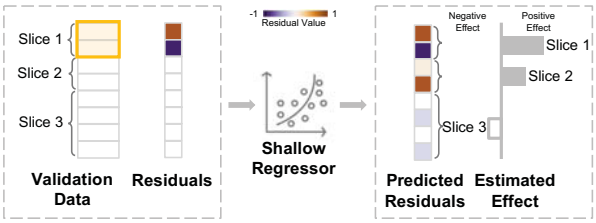

Fig. 5. Illustration of *SliceBoosting* algorithm to estimate model optimization effect. Given the selected slice 1, we train a shallow regressor to estimate that, under the ideal scenario where the optimization model correctly fits to slice 1, how will the performance on slice 2 and 3 be affected. We design the prediction target of the regressor as the residuals of the original model predictions to the ground truth validation labels. To focus on the effect estimation of slice 1, we set the residual values only for slice 1, while keeping the residuals of all other slices as 0. The predicted residuals from the regression are in the range of $[-1,1]$. We then aggregate the sample-level residual predictions to obtain slice-level estimation results: how will the accuracy on these slices increase or decrease?

use a similar encoding where each slice is represented as a row (set), and slice descriptions (items), as columns. Our adaptation also includes data slice metrics on the UpSet visualization encoding.

Model and data metrics are computed and displayed together with their respective slices ([**R2]** and [**R4]**). For model-agnostic metrics, a bar chart is displayed and, for model-specific metrics, a color-coded 1-D scatter plot is shown. In Figure 1(A), the metrics "Support" and "Accuracy" are displayed. We show a truncated scale for "Support", since the support of the entire dataset (slice "All Data") is always equal to 1. Additional metrics can be defined, including "Precision", "Recall", and "F1 Score". Previous VA systems have enabled users to interactively compare ML models [4, 11, 14, 40, 53, 56]. However, to the best of our knowledge, *SliceTeller* is the first of its kind to allow a detailed model comparison using automatically computed data slices to guide the analysis process.

Slice Detail View

During our interviews, experts expressed their interest in understanding the content of the data slices and valuing how much impact they have in their application [**R2]**. We design two types of slice detail view to cater two major data types in our use cases: a *Slice Distribution View* (Figure 7(D)) for tabular data, and a *MatrixScape View* (Figure 1(D-E) [18]) for image data.

Slice Distribution View. In this view, users can select the metadata to understand the distribution shifts across slices. We present the distribution of each metadata feature as a sorted histogram and align those for the same feature of different slices to facilitate a more convenient comparison. For example in Figure 7(D), two data slices are selected ("All Data" and "Slice 1") and "Object" metadata distribution is shown.

MatrixScape View. While looking at distributions was useful, the experts working with image data wanted to be able to explore the images themselves. In particular, they wanted to check whether they could identify potential sources for model mistakes in these samples. To allow this task, we used the MatrixScape visualization [18], which can contextualize images with metadata information. In MatrixScape, images can be laid out in a canvas according to different metrics and aggregated at multiple levels of detail. At the coarsest aggregation level, MatrixScape shows a heatmap of a particular data metric (for example, accuracy), grouped by metafeatures chosen by the user (Figure 1(E). Upon zooming in, users can see individual data samples as well (Figure 1(D)).

3.5 *SliceBoosting*: Estimating the Effect of Data Slice Optimization

During our interviews with the domain experts, they expressed the need to create multiple models and evaluate trade-offs between them [**R3]** from the perspective of manually created data slices. In their current approach to train multiple models from scratch for comparison, there are mainly three pain points: 1) Since the model training process is time-consuming, the experimentation cycle is easily interrupted, and the model iteration is slow. 2) It requires significant efforts from the experts to keeping track of multiple models trained across different data slices. 3) To draw experimentation conclusions and identify the slice trade-offs, they have to switch between development tools multiple times.

Denote the original input model to *SliceTeller* as f parameterized by θ. Let the training data be $X^{train} \in \mathbb{R}^{N^{train} \times D}$, where N^{train} is the number of samples in training set and D is the feature dimension. Similarly, let the validation data be $X^{val} \in \mathbb{R}^{N^{val} \times D}$. We use S^{val} to denote the slices selected by user as in Section 3.4, and S^{train} to denote the training data slices that correspond to the **same description** as S^{val} (e.g., $Weather = Sunny, Object = Wall$). We can utilize the optimization approaches (details in Section 3.6) to retrain f on X^{train} to prioritize on S^{train}, in order to obtain the optimized model f'. However, due to the scale of X^{train} and the high complexity of f, the optimization is time-consuming. It is therefore infeasible to try out different slice combinations to obtain the optimal f' that could satisfy the product requirements.

To facilitate fast slice-based experimentation, our objective is to estimate the performance difference between f' and f without explicitly training for f'. We develop a novel *SliceBoosting* algorithm to perform the estimation. The main idea is that instead of training the full model to evaluate slice trade-offs, we can train a shallow model to approximate the residuals (errors) of the slices, in an approach similar to boosting [23]. We denote the shallow model as h. Due to the shallowness, the training process is significantly faster than training the full model from scratch.

We have two assumptions: 1) The validation set X^{val} has a similar distribution to the training data X^{train} while being significantly smaller. This allows us to train the shallow model on the validation set to approximate the full model behavior on the training set. This assumption is valid in most cross-validation experiment settings. 2) The optimization approach (details in Section 3.6) is sufficiently powerful to steer the model to make correct predictions on the selected validation slices. Under these assumptions, we train the shallow model to fit to the selected validation slices S^{val} together with the associated labels. After it is trained, its predictions on other slices will contain the approximation of the full model's behavior with further optimization. Similar approaches have used weak learners to reduce model biases and improve performance, including Multicalibrated Predictor [19] and MultiAccuracy Boosting [26]. However, while these approaches train and evaluate multiple boosted models to improve model accuracy, *SliceBoosting* uses a simplified approach with a single boosted model to estimate the effect of subgroup optimization and allow quick experimentation.

Since the shallow model is a "weak learner", it is challenging to encode all validation data and labels. Inspired by gradient boosting [23] and surrogate model explanation approaches [9, 10], we design the shallow model to instead fit to the residuals (errors) of the original model on the selected slices. Since the original model is powerful (e.g., ResNet-50 Deep Neural Networks), its prediction is close to ground truth labels. Therefore, predicting the residual is a significantly easier task for the shallow model. As shown in Figure 5, the residual is calculated as the difference between the ground truth validation labels and the predicted labels, in one-hot-encoded format:

$$\text{residual}_i = y_i^{val} - \hat{y}_i^{val} \tag{1}$$

where y_i^{val} denotes the one-hot-encoded ground truth validation label for sample x_i^{val}, and $\hat{y}_i^{val}$ denotes the one-hot-encoded predicted label from the original model f. We illustrate the residual for a certain class in Figure 5. There are three possible values in the residuals calculated from Eq. (1): 0 denoting the model prediction is correct, 1 denoting the model missed the detection of the class, and -1 denoting the model wrongly predicted the class. Note that since we focus on the selected slices, samples from all other slices have residual of 0.

We then train the shallow regressor h using XGBoost [5] to learn the residuals from S^{val}. To achieve fast response for visual interaction, we use only 3 training iterations and maximum tree height 5 in XGBoost

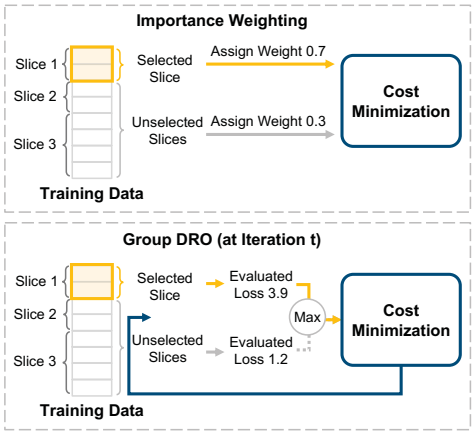

Fig. 6. Illustration of the model optimization methods considered in our work. During re-training, they prioritize slices in the training data according to user's decision.

(these parameters can be fine-tuned depending on the problem). To emphasize on the small set of misclassified samples, we increase their weights in the loss function. After h is trained, we infer the residual and prediction label for the full optimized model ($\tilde{y}_j^{val}$) as follows:

$$pred\_residual_j = h(x_j^{val})$$
$$\tilde{y}_j^{val} = pred\_residual_j + \hat{y}_j^{val}$$

Here, x_j^{val} contains data features of the validation set belonging to Slice j. A good estimation is achieved if $\tilde{y}_j^{val}$ is close to the true label y_j. After obtaining all estimated predictions, we measure the new accuracy in each slice and compare it with the original model accuracy to determine final estimated effect. More specifically, let A be the vector containing the accuracy of the original model on all data slices, and A' be the vector containing the accuracy of the boosted model f' on all data slices. The estimated effect E' is given by $E' = A' - A$. As illustrated in Figure 5, in the estimation effect, a positive number indicates that the performance on the slice might improve with model optimization. On the other hand, a negative number suggests that the performance on the slice could be reduced.

In order to evaluate *SliceBoosting*, we can check whether its estimated effects agree with the real optimized model performance (outlined in Section 3.6) . We measure this *Agreement Score* using Pearson correlation coefficient [49] for the first two use cases in Section 4. More specifically, let A be the original model accuracy on all data slices and A'' be the retrained model accuracy on all data slices. The real performance effect E is given by $E = A'' - A$. We compute the Pearson correlation coefficient between the estimated effect E' and the real effect E as: $Agreement\ Score = corr(E, E')$. We note that in both use cases, the Agreement Score was greater than 0.8, showing high correlation between the estimated slice optimization effects and the real effects. We further validate the *SliceBoosting* algorithm by evaluating the Agreement Score of ten estimated effects, computed for the top five worst data slices of the two aforementioned use cases (a new estimate and model are computed for each data slice). In Case 1, the estimates for five models optimized on the five worst slices had an Agreement Score of 0.860 ± 0.050. In Case 2, the estimates for five models optimized on the five worst slices had an Agreement Score of 0.776 ± 0.054. A detailed description the *SliceBoosting* evaluation is available in the supplementary materials.

3.6 Model Optimization

In this section, we utilize state-of-the-art model optimization methods to improve the performance on the selected slices, while minimizing the trade-off for the averaged model performance on the entire dataset [**R3**]. These methods adapt the loss function based on identified slice prioritization and subsequently perform additional training to steer the model towards the user requirement. Note that our framework is compatible with data-centric model improvement strategies as well (e.g., additional data collection and data augmentation/synthesis) and we leave the discussion to Section 6. Here, we focus on optimization-based model improvements without any change of the dataset.

Figure 6 illustrates the model optimization methods in *SliceTeller*, i.e. importance weighting and group DRO. Note that we merge all unselected slices into a single slice for optimization. In general, importance weighting method changes the loss function by assigning heavier weights to the training samples in the worst-performing slices. On the other hand, group DRO prioritizes the worst-performing slices during the training process. Here we briefly describe the two approaches. Detailed descriptions can be found in [48].

Importance Weighting. Importance weighting modifies the expected loss by emphasizing training samples belonging to the slices S^{train}. Denote the number of samples in S^{train} as n^{train}, the number of samples in the training set as N^{train}, and the total number of slices as M. The weight for slice S is calculated as:

$$w_{S^{train}} = \frac{N^{train}}{M \times n^{train}}$$

Intuitively, the selected slices with lower performance correspond to the minority groups in training set. We can therefore specify the weights of the slices as inverse proportional to the respective slice size. Then, the modified expected loss can be defined as follows [48]:

$$\mathbb{E}_{(x^{train}, y^{train}, S^{train}) \sim P}[w_{S^{train}} l(\theta; (x^{train}, y^{train}))] \quad (2)$$

where P is the distribution of training data X^{train} and l is the loss.

Group DRO. Compared to importance weighting that upweights the selected slices by heuristic rule, group DRO adopts a different optimization scheme. Instead of optimizing for the averaged loss over entire training data, it optimizes for the worst-case loss over the groups in the training data. Specifically, the expected loss is defined as [48]:

$$\max_{S^{train}} \mathbb{E}_{(x^{train}, y^{train}) \sim P_{S^{train}}}[l(\theta; (x^{train}, y^{train}))] \quad (3)$$

During training, the optimization can be conducted by either recording the historical losses of all groups [41], or utilizing gradient ascent [48].

4 Use Cases

In this section, we present three use cases to demonstrate the power of *SliceTeller* on the analysis, validation and improvement of ML models. Case 1 shows how a fictional user would validate an image classification problem for data biases. The next two case studies are taken from our interviews with three MLOps engineers who work on real industry products. Case 2 shows how one engineer working on an ultrasonic object height classification model can use our system to improve their models on critical data slices. Finally, Case 3 shows how two MLOps engineers used our system to explore an image-based fire detection model and identify potential data issues. Table 1 shows a summary of the three use cases.

Table 1. Use Case Summary

	1) Bias Detection	2) Height Classification	3) Fire Detection
Data Type	Image	Tabular	Image
Validation Size	40,520	47,322	126,912
Input Metadata	40 binary	8 numeric / categ.	7 numeric / binary
Target (Binary)	Yes / No	High / Low	Yes / No

4.1 Case 1: Bias Detection for AI Fairness in Image Classification Models

To showcase how *SliceTeller* helps ML experts and practitioners detect bias caused by the imbalanced distribution of image dataset, we describe how Hedy, a PhD student interested in computer vision, profiles the performance of ResNet50 on the CelebFaces Attributes Dataset (CelebA) [33] and identify a gender-related bias inherited in the model.

The CelebA dataset contains 202,599 face images of 10,177 celebrities, along with 40 binary attribute annotations including gender, skin color, smiling, etc. for each image. It is widely used in the computer vision community for tasks including image classification [24].

Table 2. Accuracy of Image Classification Models (Val / Test)

Slice	Description	ResNet-50	Group DRO
0	All Data	**0.98 / 0.98**	0.95 / 0.95
1	Gray_Hair=Yes, Male=No	0.65 / 0.5	**0.91 / 0.81**
2	Gray_Hair=Yes, Double_Chin=No, Wearing_Necktie=No	0.65 / 0.69	**0.90 / 0.93**

Hedy first divides the data into three splits: training (70%), validation (20%) and testing (10%). She fine-tunes a ResNet50 model on the CelebA dataset to classify *gray hair* and obtains an overall validation accuracy of 0.98. She wanted to investigate how different models performed over attributes of *Male* and *Gray_Hair*. The system initially identified 22 slices for Hedy to explore (using DivExplorer support=0.02). Upon inspecting the model with the matrix view of *SliceTeller* (Figure 1(A)), she notices that the model can only achieve a low accuracy of 0.65 on Slice 1, which is defined as **females** ($Male = No$) with **gray hair** ($Gray\_Hair = Yes$) (Fig. 1). She checks the distribution of the dataset over the dimension of *Male* and *Gray_Hair*, and finds that such a significant performance drop is caused by the highly skewed data distribution in this slice (Figure 1(E) left). She verifies this observation by browsing the sample images in Slice 1 (Figure 1(D)). Interestingly, Hedy observes that Slice 2, defined by ($Wearing\_Necktie = No$, $Double\_Chin = No$, and $Gray\_Hair = Yes$), also only achieves an accuracy of 0.65. Although the attribute *Wearing_Necktie* seems related to gender bias as well, it is unclear if this is the true cause of the low performance. Therefore, she decides to take a closer look of these two slices by using the effect estimation functionality in *SliceTeller*.

Hedy starts by conducting a what-if analysis with *SliceTeller* to estimate the effect of optimizing upon Slice 1 and 2 jointly. The estimation result from *SliceTeller* indicates that the model performance on the worst eight slices will all improve significantly if optimizing Slice 1 and 2 together (Figure 1(B)). The improvement is higher as compared to optimizing over Slice 1 alone. Hedy checked the support size of Slice 2 (Figure 1(A)) and its data distribution, then realized that this slice accounts for more minority attributes than females with gray hair. This explains why optimizing Slices 1 and 2 together can improve the performance of the worst eight slices. Such optimization effect appears promising to Hedy, so she retrains a *GroupDRO* model [48] that dynamically optimizes upon the two slices. Table 2 shows the validation and test accuracy of the two models.

After the optimization, Hedy adds the performance result of *GroupDRO* into *SliceTeller* to compare it with the previous ResNet50 model. She immediately observes that the new model's performance on the gray-haired female images improve significantly (Figure 1(C)). As predicted in the effect estimation stage, the optimized model's performance on the worst eight slices are improved, with a small trade-off on overall model performance. To confirm that she has alleviated the model bias towards females on classifying *gray hair*, she places the "All Data" slice of both ResNet50 and *GroupDRO* side by side in the MatrixScape view and compares them from the 2D dimension *Male* and *gray hair* 1 (Figure 1(E)). The result confirms that the *GroupDRO*'s performance classifying gray-hair females has significantly improved, with slight performance change on the other blocks.

4.2 Case 2: Ultrasonic Object Height Classification for Autonomous Driving

We now consider a real-world application: object height detection for autonomous driving. One of our domain experts worked on a model for object height prediction based on vehicle ultrasonic sensors. The sensors produced tabular data has 157,743 records that consists of 71 numerical features, and the expert's goal was to predict object height as a binary label: 'high' or 'low'. Such predictions would help improve the overall robustness of sensor fusion in the model pipeline, preventing collisions and aiding in the decision-making process of the car.

The car's on-board processor has limited compute power and cannot handle very complex models, such as neural networks. Therefore, the expert chose to train an XGBoost [5] model for this problem, which we will call *Model_1*. The expert split the data into three splits: training (60%), validation (30%) and testing (10%). They obtained an overall validation accuracy of 0.88, and were interested in evaluating the model using *SliceTeller* in order to find failure cases. Every sample in the dataset contained associated metadata about the environmental and sensor conditions. The metadata included 'Object Type', 'Distance', 'Sensor Approach', 'Scene Clutter', 'Direction', 'Speed', 'Temperature' and 'Weather'. In this section, we show an example of such analysis. Because this is a private dataset, the results are anonymized.

SliceTeller identified 145 data slices (using the DivExplorer algorithm with support=0.05). As a first step, the expert used the Slice Summarization tool to reduce the number of data subsets to analyze. They set the Redundancy Threshold to 0.10 and obtained 11 distinct data slices for analysis. Figure 7(A) shows the analysis of the data slices from *Model_1*. The expert noticed that Slices 1 (*sunny weather*, *high clutter* and *high temperature* (in the range (20.6, 29])), 2 (*high clutter* and *low speed*), 7 (*curbstones*) and 10 (*container sides*) performed significantly worse than the overall model.

Before investing into additional data acquisition for the characterized scenario, the expert wanted to check whether optimizing the model for these slices would be possible. Therefore, they used the *SliceBoosting* algorithm to estimate the effect of training the model with higher weights on these slices. The result of this run is shown in Figure 7(B). Note that the performance of these slices is expected to improve, as well as the performance of slices 11 and 21.

They trained a new XGBoost model using the importance weighting method described in Section 3.6, this time adding higher sample weights to the samples belonging to Slices 1, 2, 7 and 10. *Model_2* (Figure 7(C)) contains the result of this optimized model. The performance of *Model_2* is higher on the optimized slices, with an accuracy increase of more than 0.2 for every optimized slice (the *Agreement Score* of the real model with the estimate model is 0.83412.). This improvement was appreciated by the expert. However, they noticed that there was a trade-off with slices 139, 142 and 143. In particular, the expert noted that slices 139 (Weather=Rain) and 142 (Object=Wall) are critical for the autonomous driving application, and therefore should not have a significant drop in performance.

To fix this issue, they trained a new model, this time weighting the samples by the previously optimized slices, as well as slices 139 and 142. Figure 8(A) shows the results of this optimization. *Model_3* has better performance than *Model_1* on the data slices where the model performs the worst, and comparable results on the slices Rain and Wall. To conclude the analysis, the expert wanted to visualize the worst data slices from all 3 trained models (Figure 8(B)). We see that overall, *Model_3* performs better than the other models on the their worst data slices. We highlighted the slices where *Model_3* performs worse, noting that this difference was not considered significant by the expert. Table 3 shows the performance of the three models on the data slices of interest for validation and test data.

Table 3. Accuracy of Object Height Classification Models (Val / Test)

Slice	Description	Model_1	Model_2	Model_3
0	All Data	0.88 / 0.74	0.92 / **0.84**	**0.95** / 0.84
1	Clutter=High, Weather=Sunny, Temp=(20.6, 29.0]	0.53 / 0.60	0.82 / **0.87**	**0.96** / 0.85
2	Clutter=High, Speed=(1.5, 3.4]	0.56 / 0.67	0.82 / **0.86**	**0.89** / 0.84
7	Object=Charging Curbstone	0.67 / 0.36	0.94 / 0.85	**0.91 / 0.87**
10	Object=Containerside	0.70 / 0.66	0.97 / 0.99	**1.00 / 1.00**
139	Weather=Rain	**0.98** / 0.83	0.88 / 0.88	0.95 / **0.90**
142	Object=Wall	**0.99 / 0.88**	0.81 / 0.80	0.93 / 0.83

4.3 Case 3: Image-Based Fire Detection

Image-based fire detection is an important problem in the industrial setting, having the potential to identify fires in its early stages, preventing accidents and losses. In order to address this problem, our partner MLOps product team trained a Convolutional Neural Network on image frames to predict the label 'Fire' / 'No Fire', obtaining a validation accuracy score of 0.94. Transparency with customers is paramount in

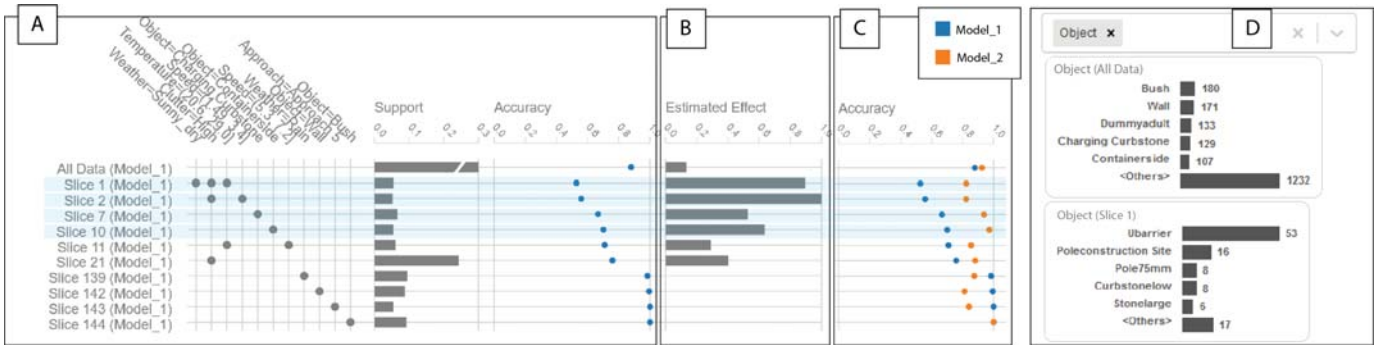

Fig. 7. Analysis of the original model for object height classification (*Model_1*). (A) The summarized data slices for this model. The highlighted slices were selected by the expert to be improved in further model iterations. (B) *SliceBoosting* results showing the estimated effect of optimizing the model for the selected slices. (C) The comparison between the original model (*Model_1*) and the model optimized for the selected slices (*Model_2*). (D) The Slice Distribution View, containing the distribution of metadata 'Object' for Slices 'All Data' and 'Slice 1'.

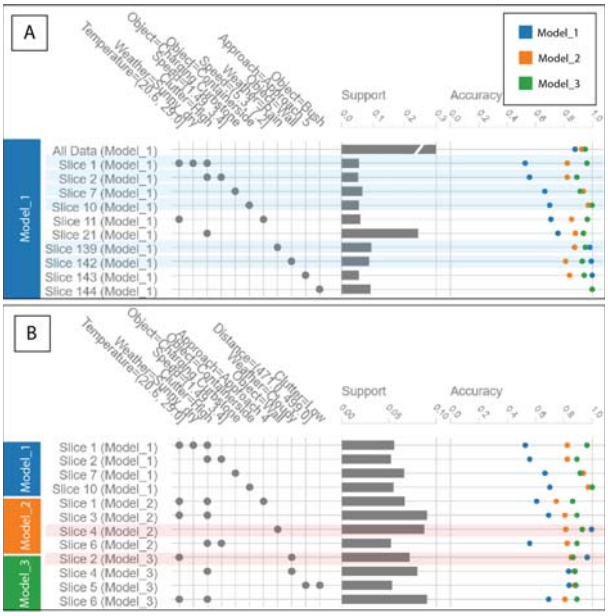

Fig. 8. A third iteration of the model is added for comparison. (A) Performance of the three models on the worst data slices from *Model_1*. Samples from highlighted slices were more heavily weighted on the training of *Model_3*. *Model_3* has better accuracy than *Model_1* on slices 1, 2, 7, 10, and comparable accuracy on slices 139 and 142. (B) Visualization of the worst data slices from the three models. Slices highlighted in red have *Model_3* performance worse than *Model_1*.

this business application. Therefore, the engineering team wanted to identify and convey the model limitations to their customers.

Each video had six interpretable metadata that could be used for exploration with *SliceTeller*. These metadata were: 'Location' (Outdoor, Indoor), 'Reflections or Shadows' (Yes, No), 'Motion' (Yes, No), 'Approaching Object' (Yes, No), 'Blinking Light' (Yes, No), and 'Smoke Density' (integer between 0 and 4). This metadata was assigned to every video frame, together with a new value: 'Normalized Frame' (real number in the range [0, 1]), describing the frame position in the video. After inserting this data into *SliceTeller*, the system initially discovered 115 data slices (using the DivExplorer algorithm with support=0.2). They used the slice summarization slider at multiple thresholds in order to inspect the results and select data slices for exploration. Figure 9 shows the most interesting data slices found by them.

The experts first inspected the worst data slice in the model, which was defined by scenes with 'Blinking Lights' and 'no reflections or shadows' (Figure 9(A)). Using the confusion matrix, they saw that this data slice contained solely videos without fire, but it had many false alarms (Accuracy of 0.61). According to them, it was known that blinking lights negatively influenced the classifier prediction. However, they did not expect this large effect, with an accuracy decrease of 0.33.

They also identified data slices with problems in the beginning and ending of the videos. The accuracy of the beginning frames was lower (Figure 9(B)), with the worst slice having an accuracy of 0.86. The problems in the start of the videos were expected, as cameras were moved around at this time and, as a result, frames would get blurry. The problems at the end of the videos (Figure 9(C)), however, were surprising. They investigated the video frames using the Slice Distribution View and formulated the hypothesis that at the end of the videos, the smoke was already dissipated and hard to see. This was a useful insight for the experts, who decided to ignore those frames because their goal was to stop fires when they started.

Another surprising data slice contained samples in the middle of the recording (Figure 9(D)). This slice contained many false alarms (error rate of 0.14). However, the expert mentioned that "it looks like we should be able to get this case right". Using the Slice Detail View, they attributed these mistakes to mislabeled data. Based on this insight, they decided to double check the sample labels. Furthermore, they were very interested in using *SliceTeller* to find more cases like this, noting that too many false alarms can annoy customers.

In order to mitigate the problems found in the data slices, their strategy consisted of increasing the training data size, using data collection and data augmentation. To improve particular data slices, they said they would collect more samples in the same conditions of the slices of interest. They would thoroughly inspect the new samples in order to ensure data quality. Another mitigation strategy mentioned is data augmentation. Currently, the MLOps team is in the process of testing different augmentation strategies, such as including frames with added noise and blur to their training data. They were very interested in comparing multiple model versions using our system.

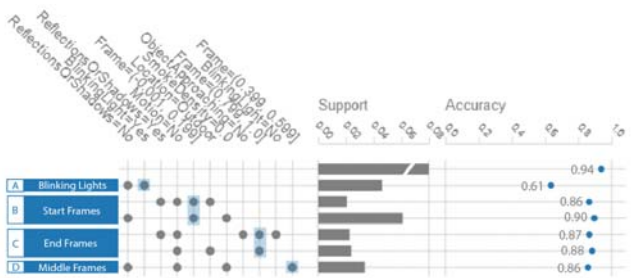

Fig. 9. Image-Based Fire Detection Use Case: interesting data slices found by the MLOps engineers. (A) Blinking light. (B) Start of video. (C) End of video. (D) Middle of video.

5 Expert Interviews

SliceTeller was developed with continuous feedback from our product partners (Section 3.1) over the course of six months. In this section, we discuss their analyses, insights and feedback for our system during a final round interview. First, we discuss the feedback from one expert working on the autonomous driving problem, who have used *SliceTeller*

to evaluate and iterate over their object height classification models. Next, we discuss the feedback from two experts working on the image-based fire detection problem, who derived actionable insights and identify data slices that require more training data using *SliceTeller*.

Expert User Demographics. We interviewed three ML experts (MLOps engineers) to evaluate *SliceTeller*. All experts have more than four years of experience in machine learning and have a graduate degree in STEM. They did not report any visual disabilities and were not color blind. The experts are not authors of this paper.

5.1 Ultrasonic Object Height Classification Experts

In Section 4.2, we have shown an example of the analysis conducted together with one MLOps engineer from the product R&D team working on the ultrasonic object height classification problem. The expert used our system to evaluate their current model, as well as experiment on new models. The expert was interested in finding potential problems in their models. In particular, they wanted to identify misclassified samples close to the cars, as these mistakes can be critical. In their current workflow, they frequently evaluate their models on hand-curated data slices and fine tune them to achieve near-zero errors within a distance threshold. This workflow relies on many handcrafted scripts that have to be executed sequentially to evaluate and iterate over their models, making the process difficult to use. Therefore, they needed a simpler and more efficient alternative to this workflow.

During their analyses, they found a set of problematic slices that they wanted to optimize their model for. They did two model iterations, and were able to produce a solution that maintained a good trade-off across the multiple data slices of interest. They mentioned that the automatic data slicing and the slice matrix component were "very useful", because they could reduce the model testing time and effort significantly. However, they noted that they still needed to guarantee a customer-defined quality on the hand-crafted slices. Therefore, they were interested in combining *SliceTeller* in their current evaluation strategy, which would allow them to identify problems in their models that they did not think to look at before. As a feature request, they would like to propose problem-specific metrics for slice evaluation. We are currently working with the ultrasonic team to learn about their metrics and include them in the future system iteration.

5.2 Image-Based Fire Detection Experts

We interviewed two MLOps engineers working on the image-based fire detection problem. During their analyses, the experts were able to discover potential model issues, as well as formulate strategies to mitigate these issues. They used the Slice Matrix View and the Slice Summarization Slider the most frequently and expressed that these two views could facilitate the model exploration and help reduce the amount of time they needed to spend looking at data slices. They also liked to use the Slice Detail View: after identifying critical data slices, the experts used the Detail View to explore video frames and formulate hypothesis about the root cause.

Regarding model iteration, these experts did not show too much interest in fine-tuning models. Instead, they wanted to collect quality data to retrain and improve the existing models. They described their own experiments with data augmentation, such as noise and blur, with positive results. They mentioned the model comparison feature would be a powerful tool during model iteration.

They also had feature requests. They were interested in investigating the explanations for the model predictions. In particular, having importance maps displayed together with the images, so they could see where the model was "looking" when making a prediction. Furthermore, they wanted to be able to manually add data slices to *SliceTeller*, in order to keep track of critical slices for which 100% accuracy was required. These features will be implemented in the future.

SliceTeller provided our experts with new insights about their model, as well as validated hypothesis previously held by them. The experts mentioned that the system could be very useful, especially because they wanted to identify "interpretable, quantifiable boundaries for our system". These quality boundaries could be shared with customers, who can make an informed decision about their models.

6 Discussion

Other Mitigation Solutions for Model Improvements. As mentioned in Section 3.6, we focused on optimization-based model improvement approaches for *SliceTeller* without changing the training data. In practice, data-centric model improvement [38] is a promising direction to tackle real-world challenges. For example, after *SliceTeller* identifies particular weakness scenarios (e.g., snowy condition for autonomous driving) as indicated by the most critical slices, additional data can be collected corresponding to the scenarios. Recent work in data-centric AI focus on weakly-supervised learning [6, 46] and self-supervised learning [16, 37, 55] to reduce the annotation cost and facilitate fast model iteration. Another approach is to use image processing and deep learning-based image synthesis techniques to perform data augmentation. For example, adversarial objects can be placed on top of the images in order to improve the robustness of a model [18].

Limitations of *SliceBoosting*. The *SliceBoosting* algorithm in Section 3.5 was developed based on the assumption that, with a powerful optimization method, the model is able to make correct predictions on the selected slices in the validation data. The hardness of this task is determined by the generalization gap between training and validation slices. As shown in [48], strong regularization strategies in DRO training help to significantly close the gap and we utilized them in the implementation of *GroupDRO*. Robust optimization [31, 57] and domain generalization [58] are active research topics in the ML community. Although our empirical results in Section 4 demonstrated strong correlation between *SliceBoosting* estimation of the model improvement and the real improvement from importance weighting and *GroupDRO*, we plan to evaluate the effect with additional model optimization techniques as well as more public datasets.

Application Domains. In this paper, we have shown how *SliceTeller* can be used for the analysis of classification models for image and tabular data. However, our system design is not specific to these domains, and can be applied to other data types, e.g., text data. The Slice Matrix abstracts the data features by using interpretable metadata to describe the data slices. Therefore, it does not need to be adapted to other data types. The Slice Detail View, however, is dependent on the domain, and needs to be adapted to display a summary of the selected slices. For example, in the case of text domains, we foresee the use of text visualizations to convey the data slice's content (e.g., word clouds). Other data types would require custom visualization implementations for the inspection of data samples. An extension of *SliceTeller* is also possible for regression models: in this case, the slice finding algorithm needs to be adapted to have a measure of how correct a prediction is, and when it should be considered a defect.

Requirement on Metadata. One limitation of our system is the need for interpretable metadata to slice the models. This metadata usually requires intensive manual annotation, and is therefore expensive to create. Therefore, as future work, we would like to investigate automatic methods to generate such information. For example, for images, one possible research direction is to use self-supervised learning [16,55] to automatically identify interpretable visual concepts.

7 Conclusions

In this paper, we have presented *SliceTeller*, a novel VA system for data slice-driven validation of ML models. Our tool allows users to quickly identify problematic data slices, investigate the failure cases, understand the potential optimization trade-offs, and eventually iterate on new model solutions.

We demonstrated the power of *SliceTeller* with three use cases that show how *SliceTeller* can be used to analyze, validate, and improve ML models in diverse application areas. *SliceTeller* was developed and improved in close collaboration with industry ML Ops engineers and domain experts working on product R&D. Based on the positive feedback, we are currently working on incorporating *SliceTeller* into their ML development workflows to facilitate fast product iteration and model release.

REFERENCES

[1] S. Ackerman, O. Raz, and M. Zalmanovici. FreaAI: Automated extraction of data slices to test machine learning models. In *International Workshop on Engineering Dependable and Secure Machine Learning Systems*, pp. 67–83. Springer, 2020.

[2] G. Barash, E. Farchi, I. Jayaraman, O. Raz, R. Tzoref-Brill, and M. Zalmanovici. Bridging the gap between ML solutions and their business requirements using feature interactions. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 1048–1058, 2019.

[3] Á. A. Cabrera, W. Epperson, F. Hohman, M. Kahng, J. Morgenstern, and D. H. Chau. FairVis: Visual analytics for discovering intersectional bias in machine learning. In *Proceedings of 2019 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pp. 46–56. IEEE, 2019.

[4] A. Chatzimparmpas, R. M. Martins, K. Kucher, and A. Kerren. Stackgenvis: Alignment of data, algorithms, and models for stacking ensemble learning using performance metrics. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):1547–1557, 2020.

[5] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 785–794, 2016.

[6] V. Chen, S. Wu, A. J. Ratner, J. Weng, and C. Ré. Slice-based learning: A programming model for residual learning in critical data slices. *Advances in neural information processing systems*, 32, 2019.

[7] Y. Chung, T. Kraska, N. Polyzotis, K. H. Tae, and S. E. Whang. Automated data slicing for model validation: A big data-ai integration approach. *IEEE Transactions on Knowledge and Data Engineering*, 32(12):2284–2296, 2019.

[8] Y. Chung, T. Kraska, N. Polyzotis, K. H. Tae, and S. E. Whang. Slice finder: Automated data slicing for model validation. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pp. 1550–1553. IEEE, 2019.

[9] D. Collaris and J. Van Wijk. Strategyatlas: Strategy analysis for machine learning interpretability. *IEEE Transactions on Visualization and Computer Graphics*, 2022.

[10] D. Collaris and J. J. van Wijk. Explainexplore: Visual exploration of machine learning explanations. In *2020 IEEE Pacific Visualization Symposium (PacificVis)*, pp. 26–35. IEEE, 2020.

[11] S. Das, D. Cashman, R. Chang, and A. Endert. Beames: Interactive multimodel steering, selection, and inspection for regression tasks. *IEEE computer graphics and applications*, 39(5):20–32, 2019.

[12] M. F. De Oliveira and H. Levkowitz. From visual data exploration to visual data mining: A survey. *IEEE transactions on visualization and computer graphics*, 9(3):378–394, 2003.

[13] Y. Ganin and V. Lempitsky. Unsupervised domain adaptation by backpropagation. In *International conference on machine learning*, pp. 1180–1189. PMLR, 2015.

[14] M. Gleicher, A. Barve, X. Yu, and F. Heimerl. Boxer: Interactive comparison of classifier results. In *Computer Graphics Forum*, vol. 39, pp. 181–193. Wiley Online Library, 2020.

[15] K. Goel, N. Rajani, J. Vig, S. Tan, J. Wu, S. Zheng, C. Xiong, M. Bansal, and C. Ré. Robustness gym: Unifying the nlp evaluation landscape. *arXiv preprint arXiv:2101.04840*, 2021.

[16] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. Richemond, E. Buchatskaya, C. Doersch, B. Avila Pires, Z. Guo, M. Gheshlaghi Azar, et al. Bootstrap your own latent-a new approach to self-supervised learning. *Advances in Neural Information Processing Systems*, 33:21271–21284, 2020.

[17] J. Han, H. Cheng, D. Xin, and X. Yan. Frequent pattern mining: current status and future directions. *Data mining and knowledge discovery*, 15(1):55–86, 2007.

[18] W. He, L. Zou, A. K. Shekar, L. Gou, and L. Ren. Where can we help? A visual analytics approach to diagnosing and improving semantic segmentation of movable objects. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):1040–1050, 2021.

[19] U. Hébert-Johnson, M. Kim, O. Reingold, and G. Rothblum. Multicalibration: Calibration for the (computationally-identifiable) masses. In *International Conference on Machine Learning*, pp. 1939–1948. PMLR, 2018.

[20] W. Hu, G. Niu, I. Sato, and M. Sugiyama. Does distributionally robust supervised learning give robust classifiers? In *International Conference on Machine Learning*, pp. 2029–2037. PMLR, 2018.

[21] IPAM. Workshop: Deep Learning and Combinatorial Optimization. publisher: IPAM 2021.

[22] S. Kaul, D. Borland, N. Cao, and D. Gotz. Improving visualization interpretation using counterfactuals. *IEEE Transactions on Visualization and Computer Graphics*, 28(1):998–1008, 2021.

[23] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30, 2017.

[24] T. Kehrenberg, M. Bartlett, O. Thomas, and N. Quadrianto. Null-sampling for interpretable and fair representations. In *European Conference on Computer Vision*, pp. 565–580. Springer, 2020.

[25] D. A. Keim. Information visualization and visual data mining. *IEEE transactions on Visualization and Computer Graphics*, 8(1):1–8, 2002.

[26] M. P. Kim, A. Ghorbani, and J. Zou. Multiaccuracy: Black-box postprocessing for fairness in classification. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, pp. 247–254, 2019.

[27] A. Kotriwala, B. Klöpper, M. Dix, G. Gopalakrishnan, D. Ziobro, and A. Potschka. Xai for operations in the process industry-applications, theses, and research directions. In *AAAI Spring Symposium: Combining Machine Learning with Knowledge Engineering*, 2021.

[28] S. Lapuschkin, S. Wäldchen, A. Binder, G. Montavon, W. Samek, and K.-R. Müller. Unmasking clever hans predictors and assessing what machines really learn. *Nature communications*, 10(1):1–8, 2019.

[29] D. Levy, Y. Carmon, J. C. Duchi, and A. Sidford. Large-scale methods for distributionally robust optimization. *Advances in Neural Information Processing Systems*, 33:8847–8860, 2020.

[30] A. Lex, N. Gehlenborg, H. Strobelt, R. Vuillemot, and H. Pfister. Upset: visualization of intersecting sets. *IEEE transactions on visualization and computer graphics*, 20(12):1983–1992, 2014.

[31] D. Li, Y. Yang, Y.-Z. Song, and T. M. Hospedales. Learning to generalize: Meta-learning for domain generalization. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[32] E. Liberty, Z. Karnin, B. Xiang, L. Rouesnel, B. Coskun, R. Nallapati, J. Delgado, A. Sadoughi, Y. Astashonok, P. Das, et al. Elastic machine learning algorithms in amazon sagemaker. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pp. 731–737, 2020.

[33] Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[34] V. Marin, S. Jialin, F. Aaron, A. Brandon, M. Georg, D. Bistra, and Y. Yisong. Workshop: Learning Meets Combinatorial Algorithms. publisher: NeurIPS 2020.

[35] N. Mehrabi, F. Morstatter, N. Saxena, K. Lerman, and A. Galstyan. A survey on bias and fairness in machine learning. *ACM Computing Surveys (CSUR)*, 54(6):1–35, 2021.

[36] J. P. Miller, R. Taori, A. Raghunathan, S. Sagawa, P. W. Koh, V. Shankar, P. Liang, Y. Carmon, and L. Schmidt. Accuracy on the line: on the strong correlation between out-of-distribution and in-distribution generalization. In *International Conference on Machine Learning*, pp. 7721–7735. PMLR, 2021.

[37] I. Misra and L. v. d. Maaten. Self-supervised learning of pretext-invariant representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6707–6717, 2020.

[38] M. Motamedi, N. Sakharnykh, and T. Kaldewey. A data-centric approach for training deep neural networks with less data. *arXiv preprint arXiv:2110.03613*, 2021.

[39] V. Nagarajan, A. Andreassen, and B. Neyshabur. Understanding the failure modes of out-of-distribution generalization. *arXiv preprint arXiv:2010.15775*, 2020.

[40] J. P. Ono, S. Castelo, R. Lopez, E. Bertini, J. Freire, and C. Silva. Pipelineprofiler: A visual analytics tool for the exploration of automl pipelines. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):390–400, 2020.

[41] Y. Oren, S. Sagawa, T. B. Hashimoto, and P. Liang. Distributionally robust language modeling. *arXiv preprint arXiv:1909.02060*, 2019.

[42] C. Panigutti, A. Perotti, and D. Pedreschi. Doctor xai: an ontology-based approach to black-box sequential data classification explanations. In *Proceedings of the 2020 conference on fairness, accountability, and transparency*, pp. 629–639, 2020.

[43] E. Pastor, L. de Alfaro, and E. Baralis. Identifying biased subgroups in ranking and classification. *arXiv preprint arXiv:2108.07450*, 2021.

[44] E. Pastor, L. de Alfaro, and E. Baralis. Looking for trouble: Analyzing

classifier behavior via pattern divergence. In *Proceedings of the 2021 International Conference on Management of Data*, pp. 1400–1412, 2021.

[45] E. Pastor, A. Gavgavian, E. Baralis, and L. de Alfaro. How divergent is your data? *Proceedings of the VLDB Endowment*, 14(12):2835–2838, 2021.

[46] A. J. Ratner, C. M. De Sa, S. Wu, D. Selsam, and C. Ré. Data programming: Creating large training sets, quickly. *Advances in neural information processing systems*, 29, 2016.

[47] S. Sagadeeva and M. Boehm. Sliceline: Fast, linear-algebra-based slice finding for ml model debugging. In *Proceedings of the 2021 International Conference on Management of Data*, pp. 2290–2299, 2021.

[48] S. Sagawa, P. W. Koh, T. B. Hashimoto, and P. Liang. Distributionally robust neural networks. In *International Conference on Learning Representations*, 2019.

[49] P. Schober, C. Boer, and L. A. Schwarte. Correlation coefficients: appropriate use and interpretation. *Anesthesia & Analgesia*, 126(5):1763–1768, 2018.

[50] M. Srivastava, T. Hashimoto, and P. Liang. Robustness to spurious correlations via human annotations. In *International Conference on Machine Learning*, pp. 9109–9119. PMLR, 2020.

[51] E. Tsang. *Foundations of constraint satisfaction: the classic text*. BoD–Books on Demand, 2014.

[52] T. Wu, M. T. Ribeiro, J. Heer, and D. Weld. Errudite: Scalable, reproducible, and testable error analysis. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

[53] K. Xu, M. Xia, X. Mu, Y. Wang, and N. Cao. Ensemblelens: Ensemble-based visual exploration of anomaly detection algorithms with multidimensional data. *IEEE transactions on visualization and computer graphics*, 25(1):109–119, 2018.

[54] J. S. Yi, Y. ah Kang, J. Stasko, and J. A. Jacko. Toward a deeper understanding of the role of interaction in information visualization. *IEEE transactions on visualization and computer graphics*, 13(6):1224–1231, 2007.

[55] J. Zbontar, L. Jing, I. Misra, Y. LeCun, and S. Deny. Barlow twins: Self-supervised learning via redundancy reduction. In *International Conference on Machine Learning*, pp. 12310–12320. PMLR, 2021.

[56] J. Zhang, Y. Wang, P. Molino, L. Li, and D. S. Ebert. Manifold: A model-agnostic framework for interpretation and diagnosis of machine learning models. *IEEE transactions on visualization and computer graphics*, 25(1):364–373, 2018.

[57] C. Zhou, X. Ma, P. Michel, and G. Neubig. Examining and combating spurious features under distribution shift. In *International Conference on Machine Learning*, pp. 12857–12867. PMLR, 2021.

[58] K. Zhou, Z. Liu, Y. Qiao, T. Xiang, and C. Change Loy. Domain generalization: A survey. *arXiv e-prints*, pp. arXiv–2103, 2021.