Simulated User Studies for Explanation Evaluation

Valerie Chen, Gregory Plumb,[*] Nicholay Topin,[*] Ameet Talwalkar
Carnegie Mellon University
`valeriechen@cmu.edu`

Abstract

The goal of evaluating explanations is often to determine suitable explanations for a desired real-world use case. Traditionally, evaluation of explanations follows two steps: first, proxy metrics (an algorithmic evaluation based on desirable properties) and then, human user studies (an experiment with real users that puts explanations to the test in real use cases). The former is efficient to compute but is typically disconnected from the use case itself. Meanwhile, the latter is time-consuming to organize and often difficult to get right. We argue for the inclusion of a new step in the evaluation workflow that capitalizes on the strengths of both proxy metrics and user studies called *simulated user evaluations*, an algorithmic evaluation grounded in real use cases. We provide a two-phase framework to conduct simulated user evaluations and demonstrate that, by instantiating this framework for local explanations, we can use simulated user evaluations to recreate findings from existing user studies for two use cases (identifying data bugs and performing forward simulation). Additionally, we demonstrate the ability to use simulated user evaluations to provide insight into the design of new studies.

1 Introduction

There exists a gap between the desired real-world use cases of interpretability and the technical objectives for which interpretable machine learning (IML) methods are optimized [1]. As a result, it is difficult for (a) end-users to identify suitable IML methods for their use cases and (b) researchers to ground their new IML methods in real applications. Towards bridging this gap, one would ideally conduct *user studies* measure how accurately users can complete a real-world task when given explanations (Figure 1, Right). Indeed, a few recent works have conducted these types of studies (e.g., [2–6]). While user studies are the gold standard for IML evaluation [7], they are often infeasible for most IML researchers to run because: (1) they are resource intensive, requiring the recruitment of and compensation for real users; (2) they are time-consuming to conduct, requiring the coordination of many participants; and (3) they are difficult to set up correctly in a way that is carefully controlled to isolate the effect of explanations, and consequently often yield noisy results.

Due to the difficulties of running user studies, explanations are more commonly first evaluated using *proxy metrics* (Figure 1, Left), which measure the extent to which an explanation captures seemingly desirable properties (e.g., faithfulness to model [8, 9], stability over runs [10, 11]). The algorithmic nature of proxy metrics is advantageous for researchers because they make evaluations easy to run and reproduce, but such metrics often fail to reflect desired aspects of the use case that would have been captured in a user study [1, 7].

To contextualize this discussion, we provide a running example based on a use case studied in [3], where a researcher is interested in studying whether explanations can help ML engineers identify bugs in an ML pipeline. Specifically, the bug studied occurs when the data used to train the predictive ML model was corrupted (e.g., a feature was imputed incorrectly). To evaluate an explanation method's

[*]equal contribution

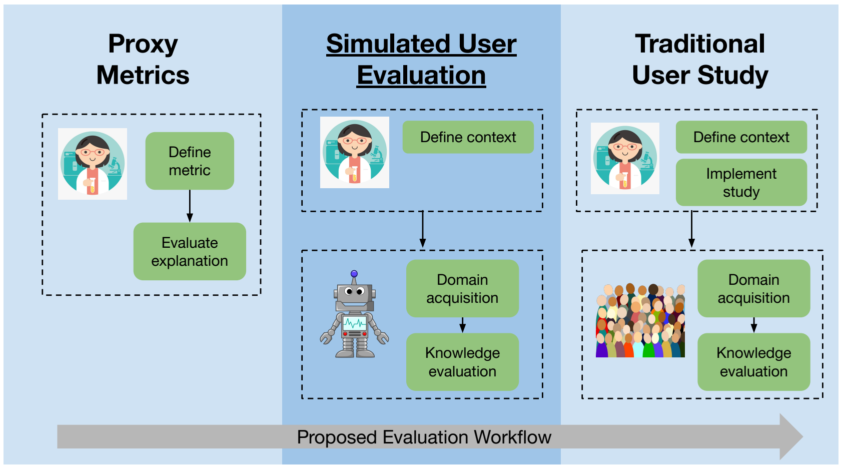

Figure 1: Our proposed IML evaluation workflow, including current, common evaluation steps (Proxy Metrics and Traditional User Study) as well as our new evaluation step *Simulated User Evaluations*. For the researcher, running simulated user evaluation is a natural preceding step to a traditional user study because defining a use-case grounded set-up is necessary for both. However, for a traditional user study, there is the additional step of implementing the study, which would entail recruiting participants and setting up a realistic use case environment for the participants to interact in. Note that the exact way that humans would acquire domain knowledge might differ from the agent.

usefulness for identifying the bug, the researcher could conduct an expensive user study where the researcher would have to recruit many ML engineers to participate. The researcher would also need to ensure that their study design controls for all possible confounding factors. On the other hand, the researcher could try to evaluate the explanations using proxy metrics, but would get stuck because there is no known proxy metric to measure the utility of an explanation for finding this particular bug.

In this work, we argue for the addition of a new step in the evaluation pipeline called *simulated user evaluations* (Figure 1, Middle). Simulated user evaluations leverage the strengths of both proxy metrics and traditional user studies by being both *algorithmic* (consequently, fast and reproducible) and *use-case specific* (i.e., grounded in a real world use case such as model debugging). To emulate the real user evaluation, simulated user evaluations are designed to encode as much of the same context of the use case as the traditional user study.

Our approach trains an algorithmic agent to measure whether an explanation method contains the *predictive information* necessary for a researcher's real world use-case of interest. Because an explanation must be predictive of the use-case in order for it to be useful to a human, researchers can use simulated user evaluations to efficiently screen candidate explanation methods for a user study. Importantly, we note that our algorithmic agent does not capture the many complex psychological or cognitive factors that influence the ability of humans to reason about explanations [12]. Incorporating these additional factors remains an open direction for future work.

While for these reasons simulated user evaluations cannot replace user evaluations, simulated user evaluations can provide critical information on whether or not an explanation method is predictive for a use case. This information is typically not captured by any existing proxy metrics. We argue that simulated user evaluations should be regarded as a foundational step toward building more efficient and reproducible evaluations of explanation methods.

Our contributions in this work is two-fold:

- Our first contribution in Section 3 is a general two-phase framework to conduct simulated user evaluations. Our framework uses an algorithmic agent to simulate an end user that uses explanations in a real world use case. This framework consists of step (1) *domain acquisition*, where the agent learns the relevant domain knowledge for the use case, and step (2) *knowledge evaluation*, where the agent is evaluated on unseen examples of the use case as a way to measure explanation utility.
- Our second contribution in Section 4 instantiates our general framework to evaluate the utility of *local explanations* (e.g. [8, 13]). In Section 5, we demonstrate that our framework

is able to corroborate existing findings from two user studies papers [3, 4] related to local explanations for two use cases. We also showcase how, once the set-up has been done for a given use case, our framework can be used to easily provide additional insights. For example, we find that certain explanation methods perform just as well as the ones considered in the original studies.

2 Related Work

Types of User Studies on Explanations: A large number of prior works have user studies to evaluate IML methods: [14] surveys the wide and diverse goals of IML user studies. Specifically, IML user studies can differ in the independent variable that is changed (e.g., the explanation method used [3, 4] or the visualization of the explanation [15]) or the dependent variable(s) being measured (e.g., the user's accuracy on a task [3, 4, 12, 16, 17], time required to complete the task [12], or trust in the ML model [18]). We focus on the set of user studies that measure how *explanation methods* (independent variable) affect how *accurately* users can complete a real-world task (dependent variable).

We group existing user studies on explanations that measure the accuracy of users on a real-world task into two categories. The first category studies the utility of explanations in use cases where the *predictiveness* of the explanation method is already known to researchers. For example, [12, 16, 17] ask users to answer simple comprehension questions about a decision set explanation, where these explanations contain the information necessary to complete the task with perfect accuracy. Thus, these studies measure humans' ability to comprehend and interpret the explanations without error. The second category includes studies where the predictiveness of the explanation method is *not* known to researchers. Our framework was developed to aid researchers in this second setting to quickly validate whether or not an explanation method is predictive. Researchers can then efficiently eliminate poor explanation methods or discover promising explanation methods for their use case.

Connecting Methods and Real-world Use Cases: [19] was one of the first to highlight the disconnect between the many goals of interpretability that are potentially discordant with existing methods, and as such [1, 20] stressed the need for more studies that explicitly study connections between methods and a specific downstream use case. There have been an increasing number of user studies that have started making these connections including [2–5] which were conducted to understand whether explanations are helpful for humans in a variety of real-world use cases ranging from fraud detection [5] to model debugging [3]. The findings from these user studies are mixed, with some finding that explanations are not always helpful, particularly when compared to more naive baselines like providing the score or model prediction to the user. As such, we also consider these naive baselines in our study. We reproduce findings from use cases studied in [3, 4] with an automated algorithmic approach that then allowed us to dive deeper and perform follow-up (simulated) experiments.

Algorithmic Evaluation Frameworks: Many existing algorithmic evaluation frameworks focus on measuring desirable properties of explanations relating to the faithfulness of the explanation to the model [21–23]. These evaluations are typically disconnected from the downstream use case of interest. In terms of algorithmic evaluations that are grounded in a use case, we were able to identify only a handful of examples (e.g., [6, 24, 25]). In this work, we present a generalizable, algorithmic framework that allows a researcher to specify their own use case to perform simulated user evaluation, and we demonstrate in Section 3 how these prior use cases can be cast in our general framework.

While we study whether explanations can be helpful for a downstream use case by simulating a user evaluation, another line of work studies the utility of explanations to improve the model training process [26–28]. The most salient difference between these two paradigms is the former assumes that explanations might help humans on a use case that they might not have been able to do as easily otherwise, while the latter does not include a human in the loop.

3 General Framework

We provide a general framework (as shown in Figure 1, Middle) for training an algorithmic agent that simulates a human using explanations for a downstream use case. Our two-part framework[2] is as follows:

[2]Note on terminology: in the following sections, we refer to the *framework* as the process to train the agent through these two phases and the *agent* as the simulated user.

First, our framework begins with the phase of *domain acquisition*. In the domain acquisition phase, the agent sees labeled training examples for the downstream use case as a way to learn information about the use case (e.g., what data-points and explanations look like with a bug or not) and also the dataset context. This step is motivated by our observation that user study participants are sometimes provided some examples to convey relevant knowledge for the use case (e.g., presented example explanations and expected conclusions). For the simulated algorithmic agent, this step is synonymous to training the agent model.

Once the agent has learned the relevant knowledge, as evidenced by a high accuracy on the training examples, we move onto the second phase of *knowledge evaluation*. This phase is analogous to the evaluation portion of typical user studies, where humans are put to the test on new examples to see if they are able to use a given explanation effectively. Similarly, the algorithmic agent is evaluated on unseen test examples. Its accuracy on these examples is a measure of the utility of the explanation for the use case. We note that existing one-off examples of algorithmic evaluations of explanations can also be cast under these two steps as described in Table 1.

Table 1: Unifying existing ad-hoc algorithmic evaluations under our framework. These algorithmic agents had considerably simpler domain acquisition processes, which were largely heuristic based as opposed to learning from data.

	Expo[6]	Influence Function[25], RPS[24]
Use Case	Use explanation to determine how to modify features of an instance to achieve target prediction	Use explanation to identify examples that were mislabeled
Domain Acquisition	The agent uses a randomized, greedy heuristic and picks the largest feature	The agent uses a heuristic, such as picking a point with large influence
Knowledge Evaluation	Evaluate the number of modifications that were made to reach the target prediction	Evaluate the number of correct points the agent selected for inspection

Compared to a human user, the researcher would be able to more carefully control the evaluation process with an algorithmic agent by eliminating the confounding factors that affect a human user's performance in a real user study. For example, while a human user study may have high per-subject variation dependent on each participant's prior knowledge or comfort with the task, a simulated agent makes deterministic predictions on each test example. This is important because it helps the researcher to isolate the effect of explanations (i.e., its predictiveness) on the agent's performance.

4 Simulated User Evaluations for Local Explanations

We instantiate our general framework to study the utility of local explanations, which encompasses the set of techniques that explain why a model made a prediction at a specific point. We apply our local explanation framework to two broad use cases: (1) performing forward simulation and (2) detecting data bugs, replicating use cases considered in previous user studies [2–4].

More specifically, our framework trains an agent for a researcher's given *context specification*, which includes the relevant information (e.g. data-points and their corresponding explanations) to predict the the ground truth answer for a use case, which we call the *task label* (e.g., What is the model's prediction for this X value? Does the model which produced these explanation contain a bug?).

Context Specification: Just as in a real user study, the researcher must first make several important decisions to specify the scope of their evaluation:

- First, the researcher must decide which explanation methods they would like to evaluate.
- Second, the researcher must decide the the *base model family*, which is the family of predictive models to be explained.
- Third, the researcher must choose a *base dataset* on which the explanation method's utility will be evaluated.

- Fourth, the researcher must specify the *use case* they would like to study, such as model debugging or forward simulation.

The simulated user evaluation then compares the utility of the researcher's selected explanation methods for the researcher's selected use case when explaining models from the base model family on the base dataset. For our experiments in Section 5, we replicate the context specification from previous user studies [2–4] to the best of our ability.

Training & Evaluation: After the researcher completes context specification, the next step is to train and evaluate the simulated agent. Our algorithmic framework generates observations that mimic the series of observations containing information (such as data-point and explanation pairs) that humans use to complete the task in real user studies. Our agent is then trained on these observations to predict the task label. Sections 4.1 and 4.2 describe how we construct the observations given as input to the algorithmic agent for each use case. Section 4.3 discusses how these observations are used to train (domain acquisition) and evaluate (knowledge evaluation) our algorithmic agent.

4.1 Forward Simulation

The forward simulation use case is motivated by [4], where the authors test whether providing users with explanations (the observation contains both X_i and $E(X_i, f)$) helps them better predict a model's behavior when compared to a control setting in which users are not given explanations (the observation would only contain X_i). As such, we train an agent $\mathcal{A}$ to predict a model f's output, so the task label is $f(X_i)$.

Following the procedure from [4], we split the base dataset $\mathcal{D}$ into train, test, and validation sets. In the domain acquisition phase, the agent is trained on a dataset of observations (here each observation $O_i = (X_i, E(X_i, f))$ is a tuple of one data-point and explanation) generated from the train set (X_i from the train set). Then, in the knowledge evaluation phase, the agent $\mathcal{A}$ is evaluated on observations generated the unseen validation set (X_i from the validation set). Clarification on the terminology for forward simulation is provided in Table 2. A more detailed algorithm is provided in Appendix A.

Table 2: Useful terminology to understand how our agent is trained for forward simulation.

Terminology	**Description**
Base Model Class $\mathcal{F}$	E.g., Random Forest, LightGBM
Base Dataset $\mathcal{D} = \{(X_i, y_i)\}$	Each $X_i \in R^d$
Model instance f	Trained on $\mathcal{D}_{train}$
Explanation Method $E : X_i \times f \rightarrow e$	Either E is an explanation method or baseline
Observation $O_i = (X_i, E(X_i, f))$	If E is a local explanation and $X_i \in R^d$, then $O_i \in R^{2d}$.
Agent $\mathcal{A} : O_i \rightarrow f(X_i)$	Our agent learns to map data-point and explanation pairs $O_i = (X_i, E(X_i, f))$ to $f(X_i)$

4.2 Data Bugs

The set of data bug use cases consider whether explanations help a user to predict whether the model from which the explanation was derived was trained on a dataset containing a systematic error. For this use case, the researcher additionally needs to provide a "bug definition" B, which corrupts the given dataset. We consider bug definitions that were studied in prior user studies [2, 3] and summarize their motivation for identifying each of these bugs:

- **Missing value [3]:** Impute missing values for a feature using the mean value. This can inject bias into the models towards the mean value.
- **Redundant feature [3]:** The dataset contains features that are redundant, or contain the same predictive information. The true importance of the quantity that the redundant features measure may be distributed between the redundant features. Thus, none of the redundant features' importance scores reflect the true underlying measure's importance.
- **Label Error [2]:** A portion of the data-points are incorrectly labeled. This can cause the model to learn wrong associations.

A distinguishing feature of identifying data bugs from performing forward simulation is that the humans in [3] were given *multiple* data-point and explanation pairs per observation because it might be easier to detect that a model bug if given more than one data-point. For example, the missing value bug is identified by looking at a distribution over the explanation scores across data-points for a given feature (see Appendix B.4) and would not have been possible if the observation contained only one data-point. As such, we also give our agent the ability to learn from multiple data-points, which means that each observation is a set of data-point and explanation pairs. Note the slight change in notation for model debugging in Table 3.

Table 3: Useful terminology to understand how our agent is trained for model debugging.

Terminology	**Description**
Base Model Class $\mathcal{F}$	E.g., Random Forest, LightGBM
Base Dataset $\mathcal{D} = \{(X_i, y_i)\}$	Each $X_i \in R^d$
Subset of base dataset $\mathcal{D}_j \subset \mathcal{D}$	
Model instance f_j	Trained on D_j
Explanation Method $E : X_i^j \times f_j \to e$	$X_i^j \in \mathcal{D}_j$, E is an explanation method or baseline
Bug definition $B : \mathcal{D}_j \to \mathcal{D}_j^{'}$	
Observation $O^j = \{(X_i^j, E(X_i^j, f_j))\}_{i=1}^S$	Set of S data-point and explanation tuples $(X_i^j, E(X_i^j, f_j))$ for $X_i^j \sim \mathcal{D}_j$
Agent $\mathcal{A} : O^j \to y^j$	Task labels $y^j = 1$ if $\mathcal{D}_j$ has a bug, 0 otherwise

To train our agent, we generate a dataset of size N of observations $\{O^1, ..., O^N\}$. This dataset of observations is then split into a train and test set that corresponds to the domain acquisition and knowledge evaluation respectively. Notice that each observation O^j given as input to the agent is a *set* of S data-point, explanation pairs. To create each observation O^j, we first need a model instance f_j to generate explanations. We train f_j on a subsample $\mathcal{D}_j$ of the the original base dataset $\mathcal{D}$. We subsample the base dataset so that the agent can learn how to use explanations that are defined on a consistent set of features, while also observing potentially different behaviors on the different subsampled datasets. To get ground truth task labels y^j, half of the time we apply bug B to subset $\mathcal{D}_j$ (set $y^j = 1$) so that half of the observations have explanations from bugged models and the other half from unbugged models. A more detailed algorithm is provided in Appendix A.

4.3 Instantiating Algorithmic Agent

For the agent model, we leverage DeepSets [29], a canonical model that is able to handle permutation invariance on its input. This property is desirable and necessary for our agent to have because in the data bugs use case, the agent is given as input a *set* of data-point and explanation tuples that do not have a pre-defined ordering. The exact model architecture is provided in Appendix B.2. In our experiments we use Adam with learning rate $1e-4$ for our optimizer and binary cross-entropy for the loss function.

We observe in the following section that the DeepSets agent model class accurately captures whether or not there is predictive information in an explanation across the wide range of datasets/tasks we used in our experiments. An open direction for future work is how to select the appropriate agent model class for other types of data, such as image or multi-modal data.

5 Experiments

We provide experimental details to *reproduce* the experimental set-up of the original studies for both the forward simulation and debugging use cases.[3] This means we replicate the context specifications studied in [3, 4].

Base Datasets: Both user studies that we replicate used the UCI Adult dataset[4]. For diversity of results, we additionally run experiments with three additional UCI datasets (Bank Marketing[5],

[3]Code repository can be found at `https://github.com/valeriechen/sim-eval-public`.

[4]https://archive.ics.uci.edu/ml/datasets/Adult

[5]https://archive.ics.uci.edu/ml/datasets/Bank+Marketing

Occupancy Detection[6], and Online Shopper Purchasing Intention[7]) for the label errors bug. In Appendix B.1, we discuss how each dataset was processed. All datasets considered had a binary target label and had at least $10,000$ points.

Base Models: For the forward simulation use case, we use a Random Forest as the base model, following [4]. For the data bugs use case, we use LightGBM [30] as the base model, following [3].

Explanations: For forward simulation, the agent receives no explanation in the baseline setting (so E is simply the empty set). For data bugs, we provide the agent with the model prediction as a baseline (so $E : X_i \times f \rightarrow f(X_i)$).

We focus on local feature attribution explanation methods (e.g. LIME [8] and SHAP [13]). However, our framework can also handle global feature attribution explanation methods, such as Anchors [31] and GAMs [32] which were considered in [4] and [3] respectively, if a local explanation can be derived from them. We discuss how we convert Anchors and GAMs to local explanations in Appendix B.3.

Framework Hyperparameters: For forward simulation, we generate an observation dataset of up to size 2000 and evaluate on 250 new observations. For data bugs, we generate a training set for domain acquisition of size 1000 and perform knowledge evaluation on a dataset of size 250.

5.1 Forward Simulation

To recreate the original setting from [4], we compare the prediction accuracy of our agent on the 'No explanation' setting where it receives either 16 or 32 randomly sampled validation observations without explanation with its prediction accuracy on 16 or 32 observations when given an explanation. With our algorithmic framework, we were also able to explore what happens if the participants is given a larger number of data-point and explanation pairs in the domain acquisition phase by increasing the number of observations from 16-32 up to 2000. We then tested the agent on 3 different samples of 32 observations from the 250 validation observations, where results are shown in Table 4.

Table 4: We vary the number of training observations the agent receives to perform forward simulation, recreating results from [4] in the 16 and 32 column and providing additional results for increased number of training observations. LIME and Anchors were explanations considered in the original paper while we additionally evaluated SHAP and GAM.

Explanation	16	32	100	1000	2000
LIME	89.3%	89.2%	91.9%	94.8%	95.6%
Anchors	81.9%	83.4%	96.7%	99.1%	99.4%
SHAP	91.2%	94.8%	97.1%	99.7%	99.5%
GAM	81.7%	86.2%	96.1%	98.0%	98.3%
No explanation	77.8%	78.6%	85.1%	91.2%	92.4%

Sim-User-Eval Findings: As shown in Table 4, LIME is a good choice for the forward simulation use case as an agent exhibits a significant increase in accuracy when using it over the baseline. At 16 or 32 training observations, the difference between Anchors and No explanation is small. This corroborates findings from [4], where they found that only LIME had a significant difference over the no explanation baseline when they provided 16 or 32 training observations. However, if they had performed simulated user evaluations, they would have noticed that SHAP could have also been a good option to try.

5.2 Data bugs

Missing Values: The missing values bug was originally implemented by replacing the 'Age' value with the mean value of 38 for 10% of adults with > 50k income. We note that while we attempted to recreate the same setting used in [3], the exact hyperparameters of their LightGBM model and dataset preprocessing were not provided. Thus, we had to artificially increase the bugged percentage to 30% to reproduce the middle plot in Figure 1 from [3] (for more details, see Appendix B.4).

[6]https://archive.ics.uci.edu/ml/datasets/Occupancy+Detection+

[7]https://archive.ics.uci.edu/ml/datasets/Online+Shoppers+Purchasing+Intention+Dataset

Table 5: Recreating [3]'s missing values setting, showing that both SHAP and GAM were successful in finding the bug with high accuracy. We additionally vary the size of the observation set.

	Observation Set Size				
Explanation	1	10	100	1000	2000
SHAP	47.9%	83.4%	99.3%	99.8%	99.5%
GAM	60.4%	86.6%	99.6%	99.5%	99.8%
LIME	52.5%	51.3%	62.5%	82.1%	82.8%
Model Prediction	52.9%	57.5%	55.4%	56.2%	52.5%

Table 6: Performing additional experiments on the Adult dataset by varying the strength of the missing values bug for a fixed observation set size of 1000, where we observe that similar patterns as Table 5 hold in terms of what explanations were successful.

Explanation	5%	10%	15%	20%	30%
SHAP	77.9%	92.5%	99.1%	99.8%	99.8%
GAM	61.8%	97.5%	99.5%	99.1%	99.5%
LIME	52.9%	55.8%	61.2%	59.2%	82.1%
Model Prediction	51.7%	52.9%	53.5%	55.1%	56.2%

Sim-User-Eval Findings: Similar to [3], we found that SHAP and GAMs are good choices for identifying the missing values bug. Furthermore, as shown in Table 5, we find that accuracy of the algorithmic agent starts to drop if there are too few explanations (i.e., observation set of size 10 or less). This means that too few explanations do not contain sufficient signal to detect the bug. As such, the researcher would need to figure out how to present a large number of explanations to a human user in a human-interpretable manner. This suggests the need for a user interface where the explanation feature values are presented in an aggregated fashion, which is indeed what the approach taken by [3] for the user interface. Further, as seen in Table 6, we find that a researcher could make the bug more subtle by decreasing the percentage of corrupted data-points and the explanations would still contain enough signal for an agent to detect the bug.

Redundant Features: The redundant features bug was implemented by including two features, 'Education' and 'EducationNum,' that represent the same information. To create a version where there is no bug, we randomize the value of one of the features so there is no correlation between the two and only one feature contains the original information. Note that, in the original paper, they did not have a *strength* of the bug, but we are easily able to study this variant of the bug with our algorithmic framework. We increase the strength of the bug by increasing the number of data-points in $\mathcal{D}_j$ that have the two correlated features.

Table 7: We vary the strength of the redundant features bug on the Adult dataset for a fixed observation set size of 1000 and corroborate results from [3].

Explanation	10%	30%	50%	70%	90%
SHAP	61.2%	74.1%	86.2%	99.5%	99.5%
GAM	59.1%	56.2%	81.3%	99.8%	99.7%
LIME	57.1%	52.0%	55.6%	54.6%	51.3%
Model Prediction	57.9%	52.1%	51.5%	52.0%	52.9%

Sim-User-Eval Findings: Using our algorithmic agent, we found that the simulated agent had significantly higher accuracy when given SHAP and GAM explanations as shown in Table 7. This matches findings from [3], where a fraction of users were able to use these explanations to detect the bug. Its accuracy was no better than random guessing when given LIME explanations or the model prediction baseline.

Label Error: The label error bug was implemented by randomly flipping the label of a fraction of the dataset as motivated by [2]. We experimented with the same set-up by flipping a fixed percentage of labels on multiple UCI datasets. We first used the Adult dataset to select a setting where the best explanations did reasonably well (but not perfect) to ensure a sufficiently difficult use case: a label error rate of 1% and set of 5 explanations.

Table 8: We study the label errors bug on the Adult dataset by varying the strength $\{1\%, 5\%, 10\%\}$ of the bug for observation sets of sizes $\{1, 5, 10\}$. Note that this bug required far smaller set sizes to detect than the previous two data bugs.

Explanation	1%			5%			10%		
	1	5	10	1	5	10	1	5	10
SHAP	63.5%	81.2%	87.5%	92.5%	98.5%	97.9%	96.7%	99.2%	99.5%
GAM	68.8%	85.8%	86.7%	93.3%	99.2%	99.1%	95.4%	99.5%	99.8%
LIME	55.0%	62.5%	53.3%	58.7%	56.8%	56.7%	51.5%	55.8%	55.4%
Model Prediction	56.8%	55.8%	60.0%	53.3%	50.8%	52.1%	49.5%	57.5%	52.9%

Table 9: We compare the same instantiation (1% label error and observation set of size 5) of the label error bug on four different UCI datasets and find that the same explanation method has varying utilities on different datasets.

Explanation	Adult	Bank	Shopping	Occupancy
SHAP	81.2%	71.6%	93.3%	99.5%
GAM	85.8%	66.7%	68.3%	99.6%
LIME	62.5%	53.3%	55.8%	58.3%
Model Prediction	55.8%	62.5%	54.5%	52.1%

Sim-User-Eval Findings: We are able to draw the same negative conclusions as [2] about the utility of saliency maps for label error detection on image datasets (discussed in Appendix B.5). However, as shown in Table 8, we find positive results for the same type of bug in the tabular setting. Specifically, explanations like SHAP or GAMs could be potentially useful for detecting label errors. Additionally, as shown in Table 9, the same explanation methods do not have the same utility for the same exact bug on different datasets, so there is no single best explanation for all use cases. This further suggests the need for a researcher to perform simulated user evaluations to guide the selection of candidate explanation methods for a real user study.

6 Conclusion

In this work, we argued for the inclusion of simulated user evaluations as an integral part of the IML evaluation workflow. To make this new type of evaluation more concrete, we proposed a two-phase framework for describing the general simulated user evaluation problem, that endows an algorithmic agent with necessary domain knowledge and then evaluates it on the downstream use case. We then instantiated the framework for local explanations to evaluate the utility of a set of local explanations on two use cases based on real user studies. We demonstrate that an algorithmic agent is able to corroborate findings about which explanation methods are best able to help a human.

There are several exciting directions for future work on simulated user evaluations including running a simple user study to verify some of the hypotheses that arose from the algorithmic evaluation process on these two use cases, expanding the instantiated framework to include other modalities like images, and adding other explanation methods to the evaluated set.

7 Acknowledgements

We would like to thank Nari Johnson for her extensive help as well as Alex Cabrera and Manuela Veloso for their valuable feedback. This work was supported in part by DARPA FA875017C0141, the National Science Foundation grants IIS1705121, IIS1838017, IIS2046613 and IIS-2112471, an Amazon Web Services Award, a Facebook Faculty Research Award, funding from Booz Allen Hamilton Inc., and a Block Center Grant. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of any of these funding agencies.

References

[1] Valerie Chen, Jeffrey Li, Joon Sik Kim, Gregory Plumb, and Ameet Talwalkar. Towards connecting use cases and methods in interpretable machine learning. *CoRR*, abs/2103.06254, 2021. URL `https://arxiv.org/abs/2103.06254`.

[2] Julius Adebayo, Michael Muelly, Ilaria Liccardi, and Been Kim. Debugging tests for model explanations. In *NeurIPS*, 2020. URL `https://proceedings.neurips.cc/paper/2020/hash/075b051ec3d22dac7b33f788da631fd4-Abstract.html`.

[3] Harmanpreet Kaur, Harsha Nori, Samuel Jenkins, Rich Caruana, Hanna Wallach, and Jennifer Wortman Vaughan. *Interpreting Interpretability: Understanding Data Scientists' Use of Interpretability Tools for Machine Learning*, page 1–14. Association for Computing Machinery, New York, NY, USA, 2020. ISBN 9781450367080. URL `https://doi.org/10.1145/3313831.3376219`.

[4] Peter Hase and Mohit Bansal. Evaluating explainable ai: Which algorithmic explanations help users predict model behavior? In *ACL*, 2020. URL `https://arxiv.org/abs/2005.01831`.

[5] Sérgio Jesus, Catarina Belém, Vladimir Balayan, João Bento, Pedro Saleiro, Pedro Bizarro, and João Gama. How can i choose an explainer? an application-grounded evaluation of post-hoc explanations. In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '21, page 805–815, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383097. doi: 10.1145/3442188.3445941. URL `https://doi.org/10.1145/3442188.3445941`.

[6] Gregory Plumb, Maruan Al-Shedivat, Ángel Alexander Cabrera, Adam Perer, Eric Xing, and Ameet Talwalkar. Regularizing black-box models for improved interpretability. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 10526–10536. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/770f8e448d07586afbf77bb59f698587-Paper.pdf`.

[7] Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.

[8] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

[9] Gregory Plumb, Denali Molitor, and Ameet Talwalkar. Model agnostic supervised local explanations. *arXiv preprint arXiv:1807.02910*, 2018.

[10] David Alvarez-Melis and Tommi S. Jaakkola. Towards robust interpretability with self-explaining neural networks. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 7786–7795, Red Hook, NY, USA, 2018. Curran Associates Inc.

[11] Amirata Ghorbani, Abubakar Abid, and James Zou. Interpretation of neural networks is fragile. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3681–3688, 2019.

[12] I Lage, E Chen, J He, M Narayanan, B Kim, S Gershman, and F Doshi-Velez. An evaluation of the human-interpretability of explanation. *arXiV preprint arXiv:1902.00006*, 2019.

[13] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Proceedings of the 31st international conference on neural information processing systems*, pages 4768–4777, 2017.

[14] A Abdul, J Vermulen, D Wang, B Lim, and M Kankanhalli. Trends and trajectories for explainable, accountable and intelligible systems: An hci research agenda. *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 2018.

[15] J Krause, A Perer, and E Bertini. A user study on the effect of aggregating explanations for interpreting machine learning models. *Proceedings of the 2018 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2018.

[16] H Lakkaraju, S Bach, and J Leskovec. Interpretable decision sets: A joint framework for description and prediction. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

[17] H Lakkaraju, E Kamar, R Caruana, and J Leskovec. Faithful and customizable explanations of black box models. *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, 2019.

[18] Z Bucinca, P Lin, K Gajos, and E Glassman. Proxy tasks and subjective measures can be misleading in evaluating explainable ai systems. *Proceedings of 25th International Conference on Intelligent User Interfaces*, 2020.

[19] Zachary C Lipton. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue*, 16(3):31–57, 2018.

[20] Maximilian Idahl, Lijun Lyu, Ujwal Gadiraju, and Avishek Anand. Towards benchmarking the utility of explanations for model debugging. *arXiv preprint arXiv:2105.04505*, 2021.

[21] Joon Sik Kim, Gregory Plumb, and Ameet Talwalkar. Sanity simulations for saliency methods. *arXiv preprint arXiv:2105.06506*, 2021.

[22] Julius Adebayo, Justin Gilmer, Michael Muelly, Ian Goodfellow, Moritz Hardt, and Been Kim. Sanity checks for saliency maps. *arXiv preprint arXiv:1810.03292*, 2018.

[23] Mengjiao Yang and Been Kim. BIM: towards quantitative evaluation of interpretability methods with ground truth. *CoRR*, abs/1907.09701, 2019. URL `http://arxiv.org/abs/1907.09701`.

[24] Chih-Kuan Yeh, Joon Kim, Ian En-Hsu Yen, and Pradeep K Ravikumar. Representer point selection for explaining deep neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL `https://proceedings.neurips.cc/paper/2018/file/8a7129b8f3edd95b7d969dfc2c8e9d9d-Paper.pdf`.

[25] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *International Conference on Machine Learning*, pages 1885–1894, 2017.

[26] Danish Pruthi, Bhuwan Dhingra, Livio Baldini Soares, Michael Collins, Zachary C Lipton, Graham Neubig, and William W Cohen. Evaluating explanations: How much do explanations from the teacher aid students? *arXiv preprint arXiv:2012.00893*, 2020.

[27] Sayna Ebrahimi, Suzanne Petryk, Akash Gokul, William Gan, Joseph E. Gonzalez, Marcus Rohrbach, and trevor darrell. Remembering for the right reasons: Explanations reduce catastrophic forgetting. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=tHgJoMfy6nI`.

[28] Joseph D Viviano, Becks Simpson, Francis Dutil, Yoshua Bengio, and Joseph Paul Cohen. Saliency is a possible red herring when diagnosing poor generalization. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=c9-WeM-ceB`.

[29] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper/2017/file/f22e4747da1aa27e363d86d40ff442fe-Paper.pdf`.

[30] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30:3146–3154, 2017.

[31] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[32] Trevor Hastie and Robert Tibshirani. Generalized additive models: some applications. *Journal of the American Statistical Association*, 82(398):371–386, 1987.

A Use Case Algorithms

A.1 Forward simulation

- Given base dataset $\mathcal{D}$, split into train/test/validation
- Train the base model f on the train set
- Let N_T be size of desired training dataset and N_V be size of the desired validation dataset
- Sample N_T points from the train set and for each point $X_i \in \mathcal{D}_{train}$ generate an explanation of f at that point, add the tuple $(X_i, E(X_i, f))$ to the training dataset.
- Repeat process for N_V points but from the validation set

Note the baseline condition would be generated in the same way, but excluding the explanation.

A.2 Data bugs

- Given base dataset $\mathcal{D}$, base model family $\mathcal{F}$, and bug specification B
- Let N be size of desired dataset, x be the set size
- For j in N
  - Subsample base dataset $\mathcal{D}_j$ and split into train/test/validation
  - With probability 0.5, apply bug B to corrupt $\mathcal{D}_j$
  - Train f_j on $\mathcal{D}_j$
  - Sample a set of S points from the $\mathcal{D}_j$
  - Generate an explanation $E(X_i^j, f_j)$ for each of the S points
  - Add this observation $O^j = \{(X_i^j, E(X_i^j, f_j))\}_{i=1}^S$ consisting of the set of data-points and their respective explanations as the jth observation in the final dataset

B Experiments

B.1 Dataset preprocessing

In the adult dataset, we dropped the `finalweight` feature. In the occupancy dataset, we dropped the `date` feature. In the bank dataset, we dropped the `poutcome` and `contact` features. For all datasets, we first encoded categorical variables and MinMax scaled all features.

B.2 Deepset Architecture

The Deepset architecture [29] is characterized by two parts, the feature extractor ϕ and then a standard network ρ, which takes in a summed representation from ϕ. Our ϕ network has a sequence of Dense layers of size ($2d$ where d is number of features, 200, 100), each followed by an ELU activation with the exception of the last layer. The ρ network is also a sequence of Dense layers of size (100, 30, 30, 10), each followed by an ELU activation with the exception of the last layer which is followed by a Sigmoid activation.

B.3 Converting Global Explanations to Local

For both GAM and Anchors, we used official Github implementations published by the authors. To convert GAM to a local explanation, there was already a method in the implementation to `explain_local`, so we used that out of the box.

The Anchors implementation works by returning a set of anchors for a given local instance. For categorical variables, the anchor is one of the values that the feature can take on. For continuous variables, the anchor could take the form of a lower and upper bound. As such, we needed to modify the input from $2d$ number of features to include one additional feature for each continuous variable. For example, if there are 5 continuous variables as in the Adult dataset, the input is now size $2d + 5$.

B.4 Missing Values Adjustment

Figure 2 and Figure 3 show our attempt to recreate the original bug described in [3] as closely as possible. We observe that the 30% bug for both explanation methods much more closely resembles the figure that was presented in the original paper.

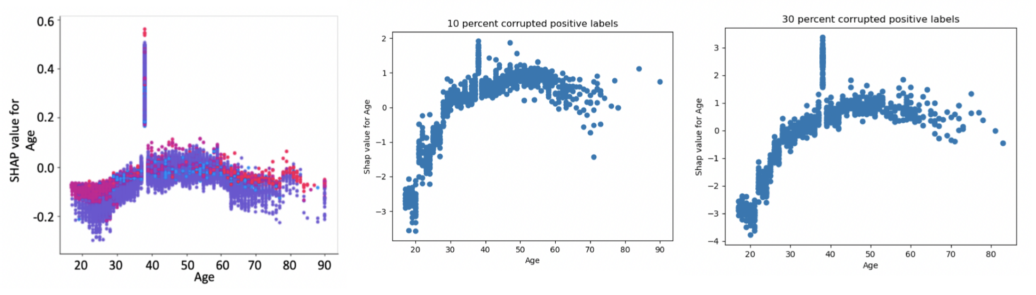

Figure 2: (Left) Image taken from the user interface from [3] that plots the distribution of SHAP values for the age feature (Middle/Right) Distribution of SHAP values for 1000 points for 10% and 30% corruption.

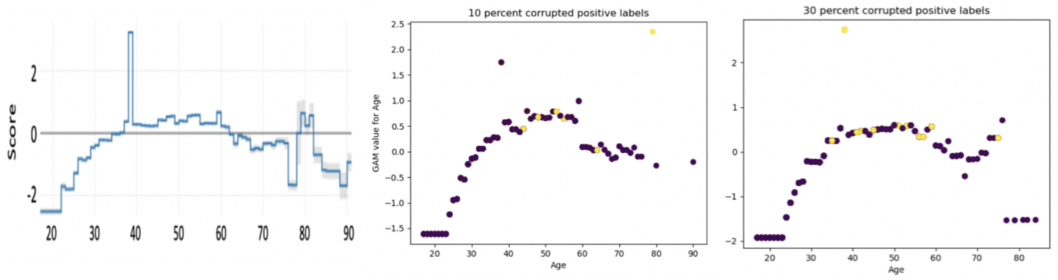

Figure 3: (Left) Image taken from the user interface from [3] that plots the distribution of GAM values for the age feature (Middle/Right) Distribution of GAM values for 1000 points for 10% and 30% corruption.

B.5 Image Results for Label Error

The experiments for [2] were performed on a dog breed dataset, where the authors considered a few different types of bugs. One in particular was a label error bug, where the dog breed label was randomized in the dataset before the base model was trained. For all explanation methods considered for this bug, participants said they were able to detect the bug overwhelmingly because the participants observed the incorrect label as opposed to using the explanation. Since the dataset modality in [2] is images as opposed to tabular, we modified the DeepSet architecture so that the ϕ network is a convolutional architecture. Following the experimental details otherwise from Section 5, we found that with 100% accuracy, the model prediction alone was sufficient for the agent to detect the bug. We plan to explore variations of the label error bug in the image setting to try to find a version where explanations might actually be useful.