Provenance-Enabled Explainable AI

JIACHI ZHANG, Alibaba Cloud, China
WENCHAO ZHOU, Alibaba Cloud, China
BENJAMIN E. UJCICH, Georgetown University, USA

Machine learning (ML) algorithms have advanced significantly in recent years, progressively evolving into artificial intelligence (AI) agents capable of solving complex, human-like intellectual challenges. Despite the advancements, the interpretability of these sophisticated models lags behind, with many ML architectures remaining "black boxes" that are too intricate and expansive for human interpretation. Recognizing this issue, there has been a revived interest in the field of explainable AI (XAI) aimed at explaining these opaque ML models. However, XAI tools often suffer from being tightly coupled with the underlying ML models and are inefficient due to redundant computations.

We introduce **p**rovenance-enabled e**x**plainable **AI** (PXAI). PXAI decouples XAI computation from ML models through a provenance graph that tracks the creation and transformation of all data within the model. PXAI improves XAI computational efficiency by excluding irrelevant and insignificant variables and computation in the provenance graph. Through various case studies, we demonstrate how PXAI enhances computational efficiency when interpreting complex ML models, confirming its potential as a valuable tool in the field of XAI.

CCS Concepts: • **Computing methodologies** → **Artificial intelligence**; • **Information systems** → **Data mining**; • **Mathematics of computing** → **Approximation algorithms**.

Additional Key Words and Phrases: Explainable AI, Data Provenance, Probabilistic Graphical Model, Multi-Layer Perceptron, *k*-means Clustering

ACM Reference Format:
Jiachi Zhang, Wenchao Zhou, and Benjamin E. Ujcich. 2024. Provenance-Enabled Explainable AI. *Proc. ACM Manag. Data* 2, 6 (SIGMOD), Article 250 (December 2024), 27 pages. https://doi.org/10.1145/3698826

1 Introduction

In recent years, we have witnessed the great success of machine learning (ML) and artificial intelligence (AI) in all facets of daily life. From healthcare diagnostics to financial forecasting, the widespread deployment of AI/ML systems is transforming industries. Unfortunately, most AI/ML models remain black boxes, and the growing size and complexity of AI/ML models make those models difficult for humans to understand or explain. This lack of explainability undermines transparency and robustness, leading to significant challenges in user acceptance [8, 19] and regulatory compliance [56, 57].

The need for explainability is particularly pressing in complex AI pipelines, where understanding how specific inputs influence outputs can ensure accountability and foster trust. Traditional explainable AI (XAI) tools [28, 50] have attempted to bridge this gap. Depending on the objective to explain,

Authors' Contact Information: Jiachi Zhang, Alibaba Cloud, Hangzhou, China, zhangjiachi.zjc@alibaba-inc.com; Wenchao Zhou, Alibaba Cloud, Hangzhou, China, zwc231487@alibaba-inc.com; Benjamin E. Ujcich, Georgetown University, Washington, DC, USA, bu31@georgetown.edu.

ACM 2836-6573/2024/12-ART250
https://doi.org/10.1145/3698826

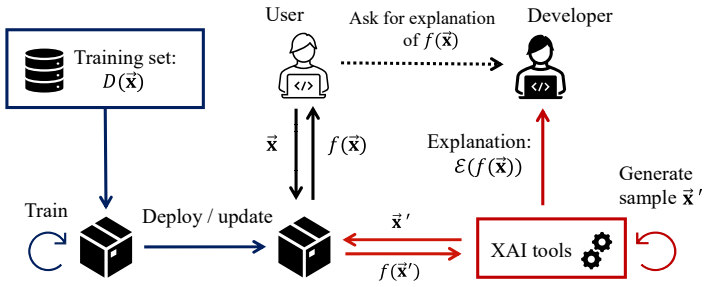

Fig. 1. A typical explainable AI (XAI) workflow.

there are *global* XAIs that characterize AI/ML models (e.g., GA^{2}Ms [10] and MMD-critic [31]), and *local* XAIs that explain a single model inference output (e.g., LIME [47] and counterfactual explanation [13, 40, 57]). Depending on the applicability, there are *model-specific* XAIs that are designed for specific models (e.g., LRP [5] and XGNN [61] for neural networks), and *model-agnostic* XAIs that apply to all models, (e.g., ICE [23], PDA [62] and SHAP [36]). In this paper, we focus on local and model-agnostic XAI tools that have the widest applicability.

Figure 1 illustrates a typical scenario in which AI developers can use XAI tools [25, 45]. First, black-box AI/ML models are trained from a training data set, denoted by $D(\vec{x})$. After the deployment or update, the model is ready for service to users. A user gives an input instance $\vec{x}$ and receives a model inference output $f(\vec{x})$. Sometimes, especially when $f(\vec{x})$ is undesirable, the user may ask the AI developer for explanations of $f(\vec{x})$.

To explain this output, developers typically take advantage of XAI tools following a *sample-then-inference* procedure: the XAI tools repeatedly generate a sample $\vec{x}'$ in the neighborhood of the original input instance $\vec{x}$, run a model inference for a corresponding output $f(\vec{x}')$, and finally compute an explanation $\mathcal{E}(f(\vec{x}))$ based on $\vec{x}'$ and $f(\vec{x}')$. The sample-then-inference procedure is common when using local and model-agnostic XAIs tools.

However, previous XAI tools fall short of practical deployability in two regards. First, *existing XAI tools are often coupled with underlying AI/ML models.* The explanations can be misleading when the output cannot be reproduced [44] (e.g., the model is updated, involves stochastic computations or the hyperparameters of model inference are modified) as the data creation and transformation within the model is not recorded. Second, *existing XAI tools are inefficient due to redundant computation.* For example, the differences between sampled instances and the original input instance might be small (e.g., ICE and counterfactual explanation). Consequently, only a small subset of the variables and computations will change during the process of model inference. The unchanged variables and computations are irrelevant to the XAI computation but are redundantly computed. However, it is difficult to trace the irrelevant variables and computations without knowing the computation dependencies.

To mitigate these challenges, we leverage the insight that we can collect data inference derivations, which allows us to use *data provenance* [9, 24] for AI/ML models. Although data provenance has been proposed in different stages of machine learning, including the model training phase [7, 59], the model inference phase [37, 58] and both phases [51, 54], no prior work addresses the limitations of local and model-agnostic XAI tools. The "approximate provenance" approach [46, 58] accelerates provenance-related computation by excluding "insignificant" derivations. However, to the best of our knowledge, no provenance work approximates the explanations derived from the local and model-agnostic XAI tools.

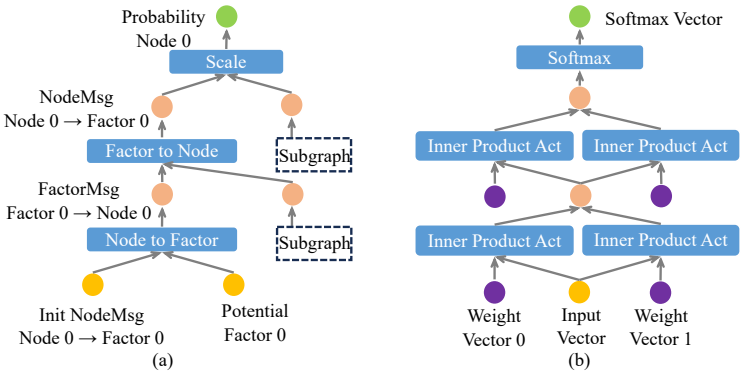

Fig. 2. Examples of PXAI provenance.

In this paper, we develop a local and model-agnostic XAI tool that we call **p**rovenance-enabled e**x**plainable **AI** (PXAI). To decouple XAI tools from the original AI/ML models, we develop a provenance model that acts as a comprehensive log of the model inference process, tracking the creation and transformation of all data within the model. For example, Figure 2 presents simplified provenance graphs that track the loopy belief propagation [41] of a probabilistic graphical model [32] (a), and the forward pass of a multi-layer perceptron (b). In these provenance graphs, round vertices represent various variables, including inputs (yellow), model parameters (purple), intermediate results (orange), and outputs (green). computation dependencies are recorded by edges, while operator vertices (blue) specify the computational operations performed. By constructing provenance alongside model inference, PXAI ensures a comprehensive and transparent documentation of the inference pipeline.

Provenance enables dependency analysis, which accelerates XAI computation by excluding redundant computations. A user provides a model to PXAI, which builds and maintains the relevant provenance graph. When a user requests the explanations of a model inference output, PXAI queries the provenance graph by backward tracing to remove the computation dependencies of other outputs and forward tracing to show how an input feature contributes to other variables. PXAI optimizes for cases in which such traces are large by creating an approximate subgraph that approximates the outputs and explanations derived from the original graph (i.e., the XAI-approximate property). Based on the aforementioned data structures and algorithms, PXAI enables explainable and efficient model inferences.

To demonstrate the efficacy of PXAI, our experiments across several case studies show that PXAI significantly lowers the performance overhead compared to the original XAI (i.e., ICE) computation *by up to 5 orders of magnitude*. As a result, PXAI enables efficient explanations across models and provides a promising direction for local and model-agnostic XAI tools.

Our main contributions include the following:

(1) We define and develop a **provenance model** that tracks the creation and transformation of all data within AI/ML models.
(2) We develop **provenance-enabled model inference**, which reproduces model inference results and excludes irrelevant variables and computations. Additionally, we propose **XAI-approximate** and develop **provenance-enabled approximate subgraph searching** methods, which exclude insignificant variables and computations without violating the XAI-approximate property.

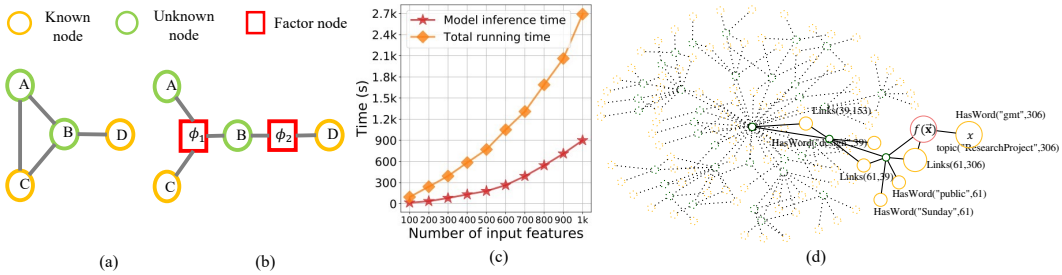

Fig. 3. Examples of a Markov Network (a) and the factorized Markov Network (b). (c) shows the XAI running time on a PGM application, text classification. (d) shows a slice of the PGM of the text classification application.

(3) We **design and implement PXAI** using C++. We extensively **evaluate PXAI through representative case studies of distinct ML models and application domains**: probabilistic graphical models (visual question answering [4] and text classification [17]), multi-layer perceptron (credit score classification [30]), and k-means clustering (ML deletion [22]). Our evaluation results demonstrate that PXAI effectively and efficiently explains black-box AI/ML models compared to prior approaches.

2 Motivation: Applying Existing XAI tools on a PGM Application

We motivate the need for PXAI using an application of explaining a probabilistic graphical model and demonstrate the shortcomings of existing XAI tools.

2.1 Probabilistic Graphical Model

The probabilistic graphical model (PGM) [32] is an AI/ML model that uses a graph-based representation where *nodes* represent random variables and *edges* represent probability dependencies to encode complex probability distributions. There are two types of graphs, a directed graph (Bayesian Network) and undirected graph (Markov Network). Both graphs "break up" the probability distribution into smaller pieces (termed *factors*), then define the joint probability as the product of factors. For example, Figure 3 (a) presents a Markov Network that consists of four nodes. Notably, in Markov Networks, each fully connected subgraph (e.g., node A, B, C and node B, D) forms a factor, and the joint probability of this Markov Network is defined as $\mathrm{P}(A, B, C, D) = \frac{1}{Z}\,\phi_1(A, B, C)\,\phi_2(B, D)$ where the factor ϕ is a user-defined function (termed *potential function*) and Z is a scalar.

The model inference of probabilistic graphical models normally refers to a process that computes the marginal probability distribution of an *unknown node* when conditioned on the *known nodes*, whose probabilities are evidenced. A common and efficient solution is to convert a PGM to a factor graph [32], a bipartite graph that interconnects a set of factor nodes (e.g., ϕ_1 and ϕ_2) to a set of variable nodes (e.g., Figure 3 (b)), then run a belief propagation [35] [21] for Bayesian Networks or loopy belief propagation [41] for Markov Networks.

2.2 Running Example

We use a running example of text classification [17] from Alchemy [1, 18], an open-source AI software that implements probabilistic graphical models. The goal of this application is to classify the topic of a document given a series of input features of what words or hyperlinks appear in the document. Alchemy builds a Markov Network that consists of known nodes as input features and unknown nodes as outputs, then estimates the probability of each topic of each document using model inference algorithms.

As it is challenging to explain the model inference outputs computed from complex belief propagations, we run XAI tools on this application. In particular, we run the individual conditional expectation (ICE) algorithm [23] to estimate the influences of all input features on one output. Following the sample-then-inference procedure, we change the value of an input feature, then run loopy belief propagation, one by one. To evaluate the performance, we sample from testing data containing more than 50k input features. Figure 3 (c) shows Alchemy's total run time (orange curve) and the loopy belief propagation run time (red curve) with respect to different sample sizes. Both curves grow super-linearly, which means that it will take *more than 30 hours* to finish the ICE computation on the whole testing data set.

We now consider why the XAI computation is inefficient:

Challenge C1 (Irrelevant Changes): Figure 3 (d) presents a slice of the PGM of this application. The yellow circles represent known nodes, the green circles represent unknown nodes, and the red circle represents the output $f(\vec{\mathbf{x}})$ to explain, which is labeled by `topic("Research Project",306)`. In this figure, a small group of solid nodes and edges represents a subgraph where the values of intermediate computation results differ when the value of the rightmost known node x is changed. That indicates there are irrelevant variables and computations when running model inference.

Challenge C2 (Insignificant Changes): The size of a known node represents its influence on $f(\vec{\mathbf{x}})$ estimated by ICE. The larger node indicates the larger influence. In this case, only two input features have higher influences than 0.01, and the influences of most input features are lower than 0.0001. That indicates that, given an output to explain, there are insignificant input features and corresponding computations.

Intuition: In summary, challenges **C1** and **C2** drive the development of PXAI, emphasizing that computational efficiency can be significantly enhanced by omitting irrelevant and insignificant variables and intermediate computations.

3 PXAI Overview

We now present an overview of PXAI's design and application in Figure 4. The upper part of this figure shows how AI developers can use PXAI to compute explanations for users, and the lower part of this figure presents PXAI's design and workflow. AI developers train and deploy ML models as an AI service to users. After giving an input instance $\vec{\mathbf{x}}$ to the model, the user asks for explanations of a model inference output $f(\vec{\mathbf{x}})$. Instead of taking advantage of existing XAI tools, we design and develop **p**rovenance-enabled e**x**plainable **AI** (PXAI) that accomplishes the following design goals:

- **Goal G1 (Decoupling):** PXAI should decouple the XAI toolkit and reasoning capabilities from the original AI/ML model to enable a generalizable approach across models.
- **Goal G2 (Efficiency):** PXAI should improve the computational efficiency of XAI tools to enable practical and timely explanations.

The goals are accomplished through the following stages:

Provenance Maintenance When a user gives an input instance $\vec{\mathbf{x}}$, PXAI builds and maintains a provenance graph during the process of model inference that computes the output $f(\vec{\mathbf{x}})$. For example, Figure 4 presents a provenance graph (the upper left graph) that is denoted by $G = (V, E)$. The provenance graph is a directed acyclic graph that records the values and the computation dependencies of the model inference outputs $f(\vec{\mathbf{x}})$ and $g(\vec{\mathbf{x}})$ that are represented by green rounds (e.g., the probabilities of unknown nodes in a PGM). The outputs rely on several intermediate variables that are represented by orange rounds, and originate from the input features represented by yellow rounds, as well as model parameters by purple rounds. Between the variables, there are vertices (blue rectangles) that represent the operators (e.g., Sum, Mul) connecting the input variables and output variables. We formally define and describe PXAI's provenance model in Section 4.1.

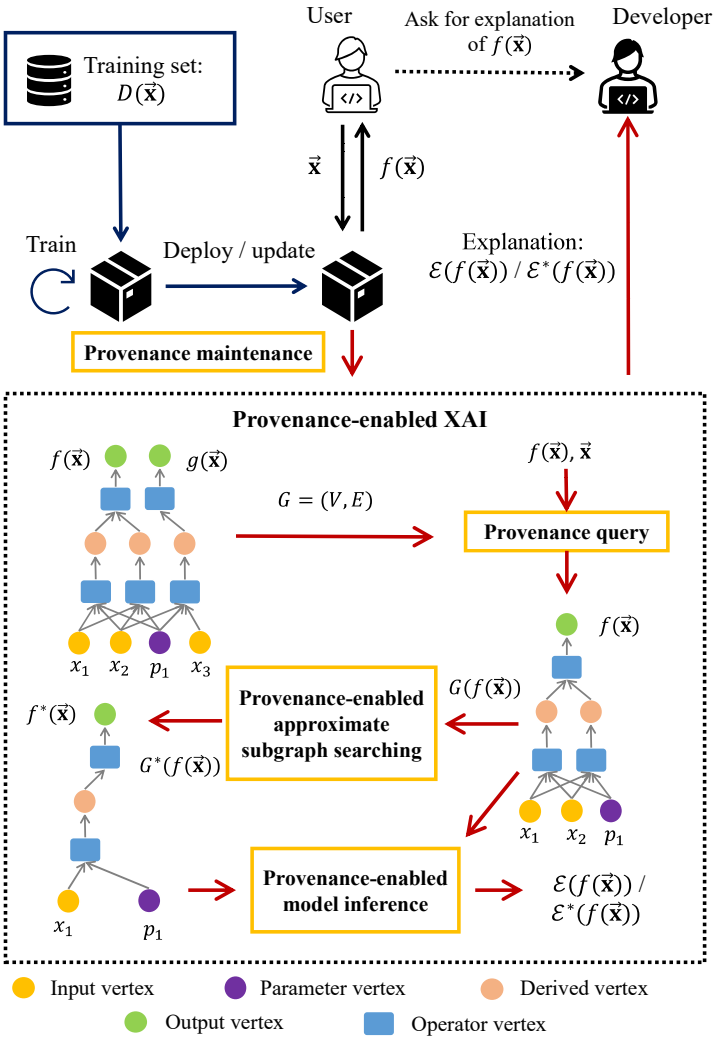

Fig. 4. Overview of PXAI's design and application.

PXAI decouples XAI computations from the original model (goal **G1**) because the provenance graphs record all information required by XAI tools. Both the model outputs and explanations can be derived from the provenance graphs instead of from the models.

Based on the provenance graphs, PXAI improves the computation efficiency of XAI tools (goal **G2**) as follows:

Provenance Query When a user requests the explanations of a model inference output $f(\vec{\mathbf{x}})$, PXAI allows AI developers to trace the full computation dependencies of $f(\vec{\mathbf{x}})$ (i.e., a *backward trace*). For example, in Figure 4, the lower right graph presents the backward trace from $f(\vec{\mathbf{x}})$, which is denoted by $G(f(\vec{\mathbf{x}}))$. That backward trace is a subgraph of G that excludes the computation dependencies of other outputs, such as $g(\vec{\mathbf{x}})$; therefore, it improves the computation efficiency of XAI (addressing challenge **C1**). In addition, PXAI supports a *forward trace* that traces how a variable (e.g., an input

Table 1. Vertex classes and descriptions.

Vertex class	Description
Variable	*Definition*: A vertex that represents a variable in a process of ML model inference *Properties*: ID, value, contribution, derivative *Subclasses*: Input vertex, Parameter vertex, Derived vertex, Output vertex
Operator	*Definition*: A vertex that represents an operator in a process of ML model inference *Properties*: ID, params *Subclass examples* (user-defined): Sum vertex, Mul vertex, Inv vertex, Exp vertex, Scale vertex, ReLU vertex, Sigmoid vertex, NearestCentroid vertex

Table 2. Edge classes and descriptions.

Edge class	Description
Variable-to-Operator	*Definition*: An edge that represents that a variable is one of the inputs of an operator *Properties*: ID, contribution, derivative
Operator-to-Variable	*Definition*: An edge that represents that a variable is an output of an operator *Properties*: ID

feature x_1) contributes to other variables. We provide formal descriptions and several examples in Section 4.1.

Provenance-Enabled Approximate Subgraph Searching Sometimes the backward trace result is too large or too dense, which leads to a large XAI computation overhead. To mitigate this, PXAI searches for an approximate subgraph, denoted by $G^*(f(\vec{\mathbf{x}}))$. For example, the lower left graph in Figure 4 presents an approximate subgraph of $G(f(\vec{\mathbf{x}}))$. The approximate subgraph improves the computation efficiency of model inference and XAI by excluding insignificant variables and computation dependencies (addressing challenge **C2**).

We say that the approximate subgraph is *XAI-approximate* when the outputs and explanations derived from the approximate subgraph are approximate to the outputs and explanations derived from the original graph. We provide formal definitions of the approximate subgraphs and the algorithms of provenance-enabled approximate subgraph searching in Section 5.

Provenance-Enabled Model Inference Following the provenance query and provenance-enabled approximate subgraph searching, PXAI computes the explanations $\mathcal{E}(f(\vec{\mathbf{x}}))$ that are derived from $G(f(\vec{\mathbf{x}}))$, or the approximate explanations $\mathcal{E}^*(f(\vec{\mathbf{x}}))$ that are derived from $G^*(f(\vec{\mathbf{x}}))$ to the output $f(\vec{\mathbf{x}})$.

Similar to previous XAI tools, PXAI adopts XAI features (e.g., feature attributions and counterfactual explanations) to compute explanations following the sample-then-inference procedure. However, during the inferring phase, PXAI improves the computation efficiency by excluding irrelevant variables and computations (addressing challenge **C1**). For example, in $G(f(\vec{\mathbf{x}}))$ from Figure 4, when a sample of the input instance only differs in one input feature x_1, repeating the computation not involving x_1 is not necessary and can be avoided. We provide the algorithms of provenance-enabled model inference in Section 4.2.

4 PXAI Provenance Model

We formally define a *provenance model* that tracks the creation and transformation of all data within AI/ML models. We develop a provenance-enabled model inference approach that accelerates XAI computations by avoiding irrelevant computations of model inference.

4.1 Definitions

PXAI's *provenance graph*, denoted by $G = (V, E)$, is a directed acyclic graph (DAG) where the vertices represent variables and operations, and the edges represent computation dependencies of model inference. The provenance graph maintains variable values, contributions and derivatives of vertices and edges. Based on the provenance graph, we can analyze the derivations of a variable (i.e., backward trace) and how a variable contributes to other variables (i.e., forward trace).

Table 1 shows the definitions and descriptions of the vertex classes. Each vertex $v \in V$ belongs to either the Variable vertex or the Operator vertex. Among the Variable vertex subclasses, the Input vertex and Parameter vertex respectively represent input features and parameters of AI/ML models—they are the provenance graph's root vertices that do not have ancestry vertices. The Derived vertex represent the intermediate computation results in ML models—they are the internal vertices that have both ancestry and child vertices. The Output vertex represent the outputs in ML models—they are leaf vertices in the provenance graph that do not have child vertices. Subclasses of the Operator vertex differ in operator types (e.g., summation, multiplication, inverse, exponentiation and scaling).

Table 2 shows the definitions and descriptions of the edge classes. Each edge $e \in E$ belongs to either the Variable-to-Operator (V2O) edge class or the Operator-to-Variable (O2V) edge class. The provenance model allows only one O2V edge for each operator (i.e., each operator only has one output), and allows multiple V2O edges for each variable (i.e., each variable can contribute to multiple variables through an operator).

In Table 1 and 2, the Variable vertices and V2O edges are associated with properties contribution and derivative that play pivotal roles in approximate subgraph searching. We formally define them here as follows:

Definition 4.1 (contribution). A *contribution*, denoted by Con, of a Variable vertex v or a V2O edge e is the difference between an output $f(\vec{\mathbf{x}})$ and the output when the vertex or edge is excluded from the provenance graph:

$$\mathrm{Con}(e) = f(\vec{\mathbf{x}}) - f(\vec{\mathbf{x}}|G \setminus e) \tag{1}$$

$$\mathrm{Con}(v) = f(\vec{\mathbf{x}}) - f(\vec{\mathbf{x}}|G \setminus v) \tag{2}$$

Definition 4.2 (derivative). A *derivative*, denoted by Der, of a Variable vertex v is the partial derivative of an output $f(\vec{\mathbf{x}})$ of v, and the derivative of a V2O edge e is the partial derivative carried by e (chain rule):

$$\mathrm{Der}(e) = \frac{\partial}{\partial v_e}[f(\vec{\mathbf{x}})]\frac{\partial v_e}{\partial v_s} \tag{3}$$

where v_e represents the end vertex of e and v_s represents the source vertex of e.

$$\mathrm{Der}(v) = \sum_{e \in E^+(v)} \mathrm{Der}(e) \tag{4}$$

where $E^+(v)$ represents the set of out edges of v.

Based on the provenance graph, we can retrieve the computation dependencies of a vertex through backward and forwarding tracing, which we formally define as follows:

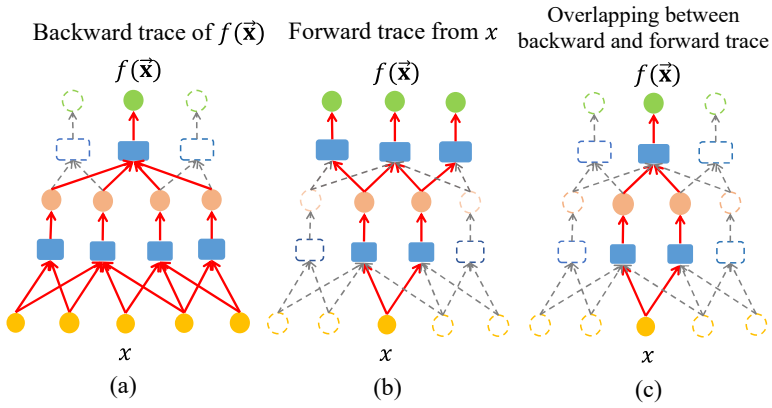

Fig. 5. Examples of backward trace, forward trace and provenance-enabled model inference.

Algorithm 1 Provenance-enabled Model Inference

```
1: function ModelInference(G, s, v_set)
2:     G' ← ∅
3:     G(s) ← backward trace from s
4:     for v ∈ v_set do
5:         Ĝ(v) ← forward trace from v
6:         G' ← G' ∪ (G(s) ∩ Ĝ(v))
7:     Change values of Input vertices in v_set
8:     f'(x⃗) ← compute the value of s in G'
9:     return f'(x⃗)
```

Definition 4.3 (backward trace). A *backward trace* result of a vertex v, denoted by $G(v)$, is a subgraph of G that records all derivations of v. Starting from v, a backward trace traverses the provenance graph by recursively calling v.predecessors() until it reaches an Input vertex or Parameter vertex.

Definition 4.4 (forward trace). A *forward trace* result of a vertex v, denoted by $\hat{G}(v)$, is a subgraph of G that v contributes to. Starting from v, a forward trace traverses the provenance graph by recursively calling v.successors() until it reaches a Output vertex.

Figure 5 (a) presents a backward trace example of an output $f(\vec{\mathbf{x}})$. Figure 5 (b) presents a forward trace example of an input x. In the figures, the solid rounds and red arrows represent the vertices and edges in $G(f(\vec{\mathbf{x}}))$ and $\hat{G}(x)$.

4.2 Provenance-Enabled Model Inference

Based on the provenance model, we develop a provenance-enabled model inference approach to accelerate XAI computation. The key intuition is that when the values of some input features are changed by XAI tools, not all variables within the provenance graph necessarily need to be updated.

Figure 5 shows an example. Given an output $f(\vec{\mathbf{x}})$ to explain, an XAI tool generates a sample $\vec{\mathbf{x}}'$ where only one input feature x differs from the original input instance $\vec{\mathbf{x}}$. By overlapping the backward trace from a source Output vertex and the forward trace from the Input vertex x, we find that only the subgraph shown in Figure 5(c) should be updated, and the values of adjacent vertices of the subgraph can be directly utilized.

Algorithm 2 Fast Provenance-Enabled Model Inference

```
1:  function FastModelInference(G, s, v_set)
2:      if s is an Input vertex or Parameter vertex then
3:          if s is in v_set then
4:              Change s.value
5:          return s.value
6:      if s.input_set ∩ v_set = ∅ then
7:          return s.value
8:      l ← ∅
9:      v_opt ← s.predecessors()
10:     for each v ∈ v_opt.predecessors() do
11:         if v is not visited then
12:             l.append(FastModelInference(G, v, v_set))
13:         else
14:             l.append(v.value)
15:     s.value ← v_opt.compute(l)
16:     return s.value
```

Our approach of provenance-enabled model inference is formalized in Algorithm 1. The inputs of this algorithm include a provenance graph G, the Variable vertex s (standing for "source") for the algorithm to infer (e.g., an output of an ML model $f(\vec{\mathbf{x}})$), and v_set, a set of Input vertices whose values have been changed. We first initiate an empty graph as the subgraph to update G' (Line 2). Then, we perform a backward trace from s (Line 3). Next, for each variable v in v_set, we do a forward trace from v and compute an intersection between the backward trace subgraph $G(s)$ and the forward trace subgraph $\hat{G}(v)$. The subgraph to update is unionized by the intersection subgraph (Lines 4–6). Finally, we change the values of vertices in v_set (Line 7), compute and update all variables in G', including the output variable $f'(\vec{\mathbf{x}})$ as the provenance-enabled model inference result (Line 8).

To optimize the performance of provenance-enabled model inference, we add "shortcuts" from a Variable vertex to all Input vertices that contribute to the vertex as a property of Variable vertices: input_set. For example, in Figure 5(a), we record that all five Input vertices contribute to the Output vertex of $f(\vec{\mathbf{x}})$ during the backward trace from $f(\vec{\mathbf{x}})$.

Taking advantage of the shortcuts, we design a fast provenance-enabled model inference algorithm, shown in Algorithm 2. The algorithm is a backward depth-first search (DFS) algorithm that recursively traverses each vertex in the provenance graph. We first define two termination conditions of DFS traversing:

- **Condition 1.** The current vertex s is either Input vertex or Parameter vertex that do not have predecessors (Lines 2–5). If s is in the v_set, we change its value (Line 4).
- **Condition 2.** No vertex in v_set is in s.input_set (Lines 6–7). It indicates s does not depend on the input features whose values have been changed; therefore, we do not need to update its value.

We initiate an empty list l (Line 8) and get the Operator vertex v_{opt} that is the predecessor of s (Line 9). Then, we iteratively visit each predecessor of v_{opt}, recursively run Algorithm 2 once a predecessor is not visited, and append the returned values to l (Lines 10–14). The operator vertex updates s.value based on l (Line 15).

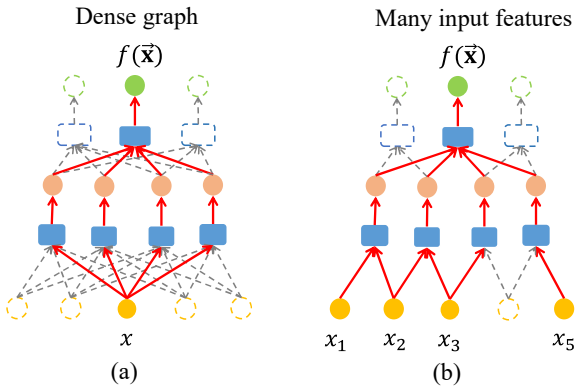

Fig. 6. Examples of least contribution prune and least derivative prune.

5 Approximate Subgraph

Based on PXAI's provenance model, provenance-enabled model inference improves the computation efficiency of XAI tools by excluding irrelevant variables and computation dependencies.

However, the performance efficiency of provenance-enabled model inference is susceptible to the **density** of the provenance graph and the **number of input features** whose values are changed. For example, Figure 6(a) shows a dense provenance graph where each input vertex is connected to all derived vertices. Figure 6(b) shows a provenance graph where the values of many input vertices are updated. In each case, the overlap between the backward and forward trace is only slightly smaller than the whole provenance graph, and the improvement over the original AI/ML model inference is limited.

Inspired by prior approaches in approximate provenance [46, 58], we explore an approach that excludes *insignificant* input features and computation dependencies to further improve PXAI's performance. We first formally define an approximate subgraph that is XAI-approximate. We also introduce two heuristics-bsaed provenance-enabled approximate subgraph searching algorithms.

5.1 Definitions

The goal of approximate subgraph searching is to simultaneously minimize the size of the subgraph and minimize the difference between approximate explanation $\mathcal{E}^*(f(\vec{\mathbf{x}}))$ that is derived on the approximate subgraph, and the original explanations $\mathcal{E}(f(\vec{\mathbf{x}}))$ that is derived on the original provenance graph (i.e., XAI-approximate). The intuition behind XAI-approximate is that both the outputs and derivatives on the approximate subgraph should be approximate to the original graph within a neighborhood of an input instance.

The intuition is demonstrated in Figure 7. In this figure, the x-axis represents an input feature, and the y-axis represents corresponding output of an ML model. The approximate curve in Figure 7 (a) performs the worst because it fails to approximate outputs or derivatives. In Figures 7 (b) and (c), the approximate curves either fail to approximate the derivatives or the outputs. In the end, the approximate curve in Figure 7 (d) is the best approximation concerning both outputs and derivatives.

We formally define an approximate subgraph as follows:

Definition 5.1 (approximate subgraph). Given a provenance graph $G = (V, E)$, an instance of input $\vec{\mathbf{x}}$, and the corresponding output $f(\vec{\mathbf{x}})$, the *approximate subgraph* $G^*(f(\vec{\mathbf{x}}))$ is a subgraph of

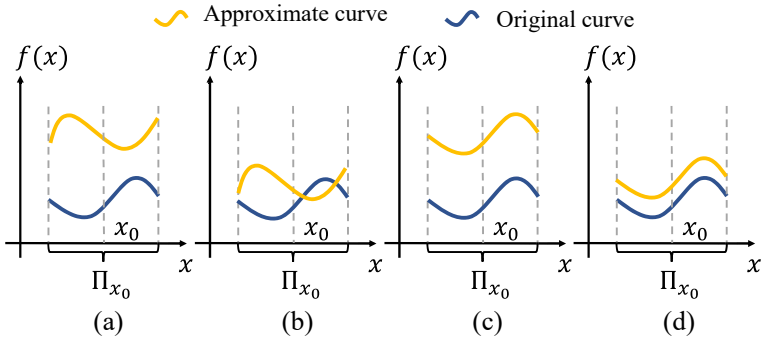

Fig. 7. Examples of $f(x)$ approximation.

$G(f(\vec{\mathbf{x}}))$ that minimizes the following objective:

$$G^*(f(\vec{\mathbf{x}})) = \underset{G'(f(\vec{\mathbf{x}}))\in G(f(\vec{\mathbf{x}}))}{\arg\min} \quad \alpha\, L_1(f'(\vec{\mathbf{x}}), f(\vec{\mathbf{x}}), \Pi_{\vec{\mathbf{x}}}) + \beta\, L_2(\triangledown f'(\vec{\mathbf{x}}), \triangledown f(\vec{\mathbf{x}}), \Pi_{\vec{\mathbf{x}}}) + \gamma\, |G'(f(\vec{\mathbf{x}}))| \tag{5}$$

In Equation 5, the optimization objective consists of three terms: L_1 and L_2 represent the losses that evaluate the approximation of outputs and derivatives, and $|G'(f(\vec{\mathbf{x}}))|$ represents the size of the subgraph. In addition, $\Pi_{\vec{\mathbf{x}}}$ denotes the neighborhood of $\vec{\mathbf{x}}$, $\triangledown f(\vec{\mathbf{x}}) = \left[\frac{\partial f(\vec{\mathbf{x}})}{\partial x_1} \quad \dots \quad \frac{\partial f(\vec{\mathbf{x}})}{\partial x_n}\right]$, the partial derivatives of the output of the input features (i.e., derivatives), and α, β, γ are user-specific parameters that balance three terms. In our practical implementation, we typically configure the three parameters to be equal, foregoing a detailed exploration of parameter optimization.

We approximate L_1 and L_2 by uniformly drawing random samples $\vec{\mathbf{x}}' \backsim \Pi_{\vec{\mathbf{x}}}$ around the input instance $\vec{\mathbf{x}}$:

$$L_1(f'(\vec{\mathbf{x}}), f(\vec{\mathbf{x}}), \Pi_{\vec{\mathbf{x}}}) = \frac{1}{N} \sum_{\vec{\mathbf{x}}'\backsim\Pi_{\vec{\mathbf{x}}}} |f'(\vec{\mathbf{x}}') - f(\vec{\mathbf{x}}')| \tag{6}$$

$$L_2(\triangledown f'(\vec{\mathbf{x}}), \triangledown f(\vec{\mathbf{x}}), \Pi_{\vec{\mathbf{x}}}) = \frac{1}{N} \sum_{\vec{\mathbf{x}}'\backsim\Pi_{\vec{\mathbf{x}}}} \|\triangledown f'(\vec{\mathbf{x}}') - \triangledown f(\vec{\mathbf{x}}')\|_2 \tag{7}$$

where N denotes the number of samples. In Equation 6, L_1 is defined as the mean absolute difference between the approximate outputs and the original outputs. In Equation 7, L_2 is defined as the mean Euclidean distance between the approximate derivatives and the original derivatives.

Taking advantage of L_1 and L_2, we evaluate to what extent an approximate subgraph is XAI-approximate as follows:

Definition 5.2 ((ϵ, λ, r)**-approximate).** An approximate subgraph $G^*(f(\vec{\mathbf{x}}))$ and the approximate explanation $\mathcal{E}^*(f(\vec{\mathbf{x}}))$ derived on $G^*(f(\vec{\mathbf{x}}))$ are (ϵ, λ, r)-*approximate* when the provenance-enabled model inference results $f^*(\vec{\mathbf{x}}')$, $\vec{\mathbf{x}}' \backsim \Pi_{\vec{\mathbf{x}}}$, satisfy $L_1 \leq \epsilon$, $L_2 \leq \lambda$ and $\|\vec{\mathbf{x}}' - \vec{\mathbf{x}}\|_2 \leq r$.

Additionally, we define a special case of (ϵ, λ, r)-approximate when the radius r of the neighborhood $\Pi_{\vec{\mathbf{x}}}$ is 0 (i.e., in Equation 6 and 7, $N = 1$, L_1 and L_2 only depend on $\vec{\mathbf{x}}$):

Definition 5.3 ((ϵ, λ)**-approximate).** An approximate subgraph $G^*(f(\vec{\mathbf{x}}))$ and the approximate explanation $\mathcal{E}^*(f(\vec{\mathbf{x}}))$ derived on $G^*(f(\vec{\mathbf{x}}))$ are (ϵ, λ)-*approximate* when the provenance-enabled model inference result $f^*(\vec{\mathbf{x}})$ satisfies $L_1 \leq \epsilon$ and $L_2 \leq \lambda$.

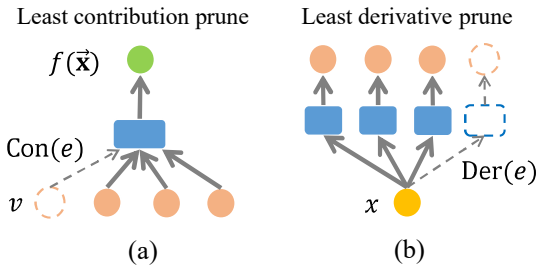

Fig. 8. Examples of the least contribution prune and the least derivative prune.

5.2 Provenance-Enabled Approximate Subgraph Searching

Although an approximate subgraph provides a more efficient approach for provenance-enabled XAI, searching for a globally optimal solution to the problem defined in Equation 5 is intractable. Based on PXAI's provenance model, we introduce two provenance-enabled approximate subgraph searching algorithms: **provenance pruning**, which is model-agnostic; and **input subset searching**, which is designed for probabilistic graphical models.

5.2.1 Provenance Pruning. Provenance pruning takes advantage of two properties of vertices and edges in PXAI's provenance model: contribution and derivative. The intuition behind this algorithm is that we heuristically prune the least "important" edges, therefore, minimizing the impact of pruning.

Figure 8 presents two strategies of provenance pruning: the least contribution prune and the least derivative prune. In Figure 8(a), the leftmost edge, connecting from a Derived vertex v to the Output vertex $f(\vec{\mathbf{x}})$, has the least contribution, which indicates that pruning this edge affects the output the least. In Figure 8(b), the rightmost edge, connecting from a Input vertex x, has the least derivative, which indicates that pruning this edge affects the derivative of the output of x the least. To simultaneously minimize the impact on the outputs and derivations, we design a mixed strategy of the least contribution prune and the least derivative prune based on a new metric, the *importance* of edges:

Definition 5.4 (importance). The *importance* of an edge, denoted $\text{Imp}(e)$, is defined as:

$$\text{Imp}(e) = \alpha\,|\text{Con}(e)| + \beta\,|\text{Der}(e)| \tag{8}$$

where $\text{Con}(e)$ is defined in Equation 1, $\text{Der}(e)$ is defined in Equation 3, and α and β are user-specific weighting parameters.

We present the provenance pruning approach in Algorithm 3. The inputs of this algorithm include $G(s)$, the backward tracing result from a source vertex s, ϵ, λ, r, that are three parameters defined in Definition 5.2 and 5.3, and k, a user-specific parameter that determines how many edges to prune in one iteration. In each iteration, we first update the contributions and derivatives of edges in $G^*(s)$ (Line 5). Next, we compute a list l of importances (Equation 8) of m edges (Line 6). Then, we get the top-k edges from l in ascending order (i.e., the least important k edges), and prune the edges (Lines 8–9). In the end, we check whether the pruned subgraph satisfies (ϵ, λ, r)-approximate. If so, we continue pruning; otherwise, we stop and return the approximate subgraph from the previous iteration (Lines 10–13).

5.2.2 Input Subset Searching. Input subset searching reduces the problem defined in Equation 5 to a problem that searches for the optimal subset of Input vertices. Instead of pruning an existing provenance graph (e.g., provenance pruning in Section 5.2.1), input subset searching builds

Algorithm 3 Provenance Pruning

```
1: function ProvenancePruning(G(s), ϵ, λ, r, k)
2:     G*(s) ← G(s)
3:     while true do
4:         V, E ← G*(s)
5:         ∀e ∈ E, update Con(e) and Der(e)
6:         l ← [Imp(e_1), ..., Imp(e_m)]
7:         G′ ← G*(s)
8:         for each e ∈ top-k(l, k,order=ascending) do
9:             G′ ← G′ \ e
10:        if G′ satisfies (ϵ, λ, r)-approximate then
11:            G*(s) ← G′
12:        else
13:            Break
14:    return G*(s)
```

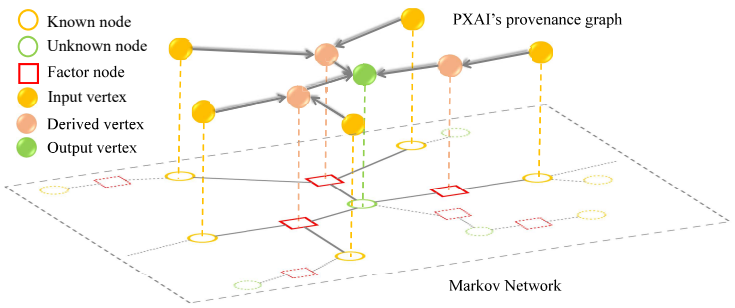

Fig. 9. An example of the PXAI provenance graph of a probabilistic graphical model.

approximate subgraphs based on the input subsets through backward trace and forward trace similar to Algorithms 1 and 2. Although it is relatively easier than searching the optimal subgraphs, enumerating all possible subsets of Input vertices is still intractable. To solve this problem, we design a heuristic searching algorithm for probabilistic graphical models that takes advantage of the graphical structures.

The intuition behind this algorithm is that, in a probabilistic graphical model, the probability inference result of an unknown node is mostly impacted by its adjacent neighbors (e.g., Markov blanket [49] and local Markov property [20]). For instance, Figure 9 shows a factorized Markov Network (in the bottom plane) and a simplified provenance graph (in the upper plane) that records the variables and computation dependencies in the process of loopy belief propagation. In the provenance graph, Input vertices correspond to the known nodes, Derived vertices correspond to the factor nodes, and the Output vertex corresponds to the unknown node in this Markov Network. In the Markov Network, compared to other known nodes (dotted yellow circles), the adjacent known nodes (solid yellow circles) directly contribute to the unknown node (solid green circle). Therefore, the corresponding Input vertices should be taken into account by the input subset searching algorithm earlier.

We present the input subset searching approach in Algorithm 4. We use a priority queue to heuristically search for the subset of Input vertices that lead to the smallest loss in Equation 5. We initialize the approximate subgraph $G^*(s)$ (Line 2), the optimal loss $\mathcal{L}^*$ (Line 3), and a set of PGM nodes $\mathcal{N}$_set (Lines 4–5). Next, we initialize a priority queue pq of 3-tuples, and in each tuple,

Algorithm 4 Input Subset Searching

```
1:  function InputSubsetSearching(G(s))
2:      G*(s) ← ∅
3:      ℒ* ← ∞
4:      𝒩_s ← the PGM node that corresponds to s
5:      𝒩_set ← {𝒩_s}
6:      pq ← ∅
7:      pq.push((𝒩_set, ∅, ∞))
8:      while pq is not empty do
9:          𝒩_set, G, ℒ ← pq.pop()
10:         if ℒ < ℒ* then
11:             G*(s) ← G
12:             ℒ* ← ℒ
13:         for each 𝒩_i in 𝒩_set's adjacent known nodes do
14:             𝒩_set' ← 𝒩_set.insert(𝒩_i)
15:             if 𝒩_set' has not been visited then
16:                 G' ← build subgraph of G(s) based on 𝒩_set'
17:                 ℒ' ← compute loss of G' using Equation 5
18:                 pq.push((𝒩_set', G', ℒ'))
19:     return G*(s)
```

Table 3. A list of PXAI's Provenance Maintenance APIs.

API	Description
addVariable(ID,V,T)	Add a Variable vertex to the provenance graph. V stands for value of this variable. T stands for the vertex subclass (e.g., Input)
addOperator(ID,T,P)	Add an Operator vertex to the provenance graph. T stands for the vertex subclass (e.g., Sum), P stands for a list of parameters of this operator (e.g., the base number of Exp operator)
addEdge(S,E)	Add an edge that connecting two vertices. S and E stand for the IDs of the source vertex and the end vertex.

the elements represent $\mathcal{N}$_set, provenance subgraph and loss (Lines 6–7). While pq is not empty, we pop the 3-tuple that has the smallest loss $\mathcal{L}$ (Lines 8–9), then check whether it is the current optimal solution (Lines 11–12). Next, for each adjacent known node $\mathcal{N}_i$ of $\mathcal{N}$_set, we extend $\mathcal{N}$_set by incorporating $\mathcal{N}_i$, and build a new set $\mathcal{N}$_set$'$ (Lines 13–14). If it has not been visited, we build subgraph G' of $G(s)$ based on a subset of Input vertices that correspond to PGM known nodes in $\mathcal{N}$_set$'$. Based on G', we compute the corresponding loss $\mathcal{L}'$ using Equation 5, and finally push them to pq (Lines 15–18). To avoid enumerating all subsets, we add an early termination condition that pq's size can not surpass a certain threshold.

6 Implementation

PXAI is programmed in C++. We implemented classes and subclasses of vertices and edges that are defined in Table 1 and 2. We implemented a class of provenance graphs, as well as the provenance tracers on top of the Boost Graph library [2]. PXAI provides a list of provenance maintenance APIs, which are shown in Table 3, for AI developers to call in their model inference codes (e.g., belief propagation of probabilistic graphical models and the forward pass of neural networks). Each vertex and edge is unique and is identified by the ID. PXAI allows developers to define new

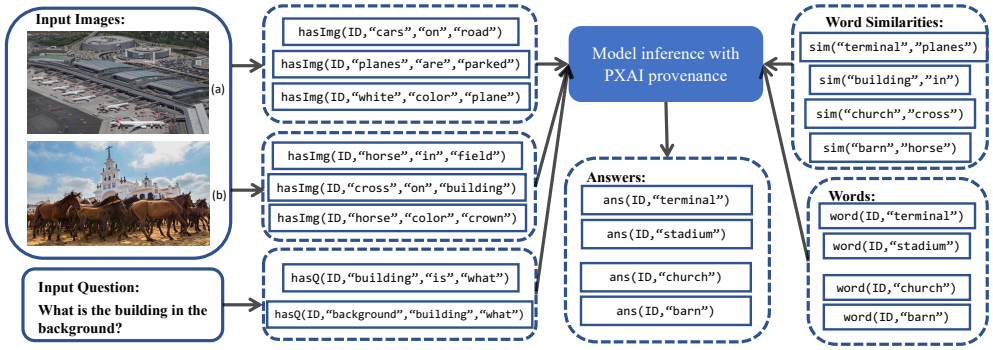

Fig. 10. Demonstration of two VQA test cases “terminal” (a) and “church” (b)

```
r1 w1: hasImgAns(V,Z,X1,R1,Y1) <= word(V,Z) ∧hasImg(V,X1,R1,Y1) ∧
       sim(Z,X1) ∧ sim(Z,Y1).
r2 w2: candidate(V,Z) <= word(V,Z).
r3 w3: candidate(V,Z) <= word(V,Z) ∧ hasQ(V,X,R,Y) ∧
       hasImgAns(V,Z,X1,R1,Y1) ∧ sim(R,R1) ∧ sim(Y,Y1) ∧ sim(X,X1).
r4 w4: ans(V,Z) <= candidate(V,Z) ∧ hasQ(V,X,R,"WHAT") ∧
       hasImg(V,Z1,R1,X1) ∧ sim(Z,Z1) ∧ sim(R,R1) ∧ sim(X,X1).
```

Fig. 11. The VQA program from PSL.

subclasses of Operator vertices so that the developers control the granularity of the provenance graph. For example, developers can use a Convolution operator, which is coarser, or decompose it into a series of Mul and a Sum operator, which are finer.

7 Evaluation

We assess PXAI’s effectiveness across diverse ML models through targeted case studies:

- Visual question answering [4] with a PGM (Section 7.1)
- Credit score classification [30] with an MLP (Section 7.3.1)

demonstrates PXAI’s capability to produce comprehensible approximate explanations that are consistent with the original explanation, and

- Text classification [17] with a PGM (Section 7.2)
- Credit score classification with an MLP (Section 7.3.2 and 7.3.3)
- ML deletion [22] with k-means clustering (Section 7.4)

Our evaluation offer a thorough insight into PXAI, illustrating its utility as an optimization toolkit in ML workflows that require flexible and efficient model updates. In short, our evaluation results demonstrate that PXAI significantly improves the computational efficiency of XAI by up to five orders of magnitude, at an acceptable trade-off involving moderately increased space and time complexity in provenance computations.

The experiments in Section 7.1 and 7.2 were conducted on a Dell PowerEdge R730 server with dual Intel Xeon E5-2640 CPUs and 64GB memory using Ubuntu 18.04.6 LTS. The experiments in Sections 7.3 and 7.4 were conducted in Alibaba Cloud, utilizing an AliServer equipped with 128 Intel(R) Xeon(R) Platinum 8369B CPUs and 2 TB of memory running CentOS.

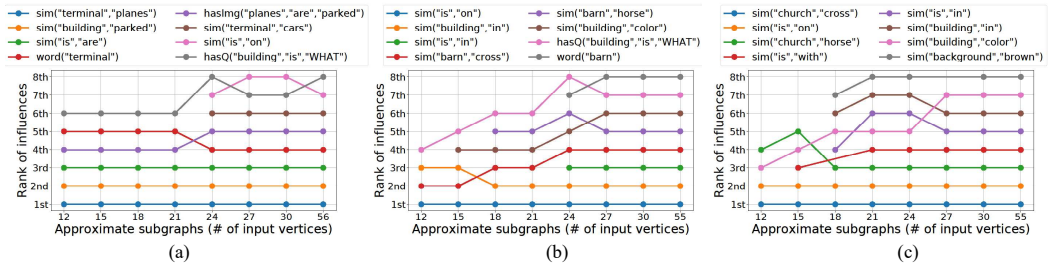

Fig. 12. Ranks of the top-8 most influential **Input vertices** of the **Output vertices**: `ans("terminal")` (a), `ans("barn")` (b) and `ans("church")` (c).

7.1 Case Study 1: Visual Question Answering (Probabilistic Graphical Model)

Visual question answering (VQA) [4] is an application of a probabilistic graphical model provided by PSL [6]. Figure 10 demonstrates an overview of two VQA test cases. In this application, we are given known nodes (i.e., the input features in PGM) from model-modality machine learning systems:

(1) `hasImg` captures image information and are extracted by computer vision models, such as `hasImg(ID,"planes","are","parked")` of "terminal" in Figure 10 (a).
(2) `hasQ` contains keywords from a question and are extracted by NLP models, such as `hasQ(ID, "building","is","what")` based on a question "What is the building in the background?"
(3) `sim` estimates the similarities between words and are collected from language models.
(4) `word` represents the candidate answers with prior confidence scores.

Combining all inputs, Aditya et al. [4] provide a PSL program in Figure 11 that builds a Markov Network from four weighted first-order logic formulas. In this program, `r1` extends more `hasImg` by replacing synonyms of keywords of `hasImg`. Formula `r2` indicates that every word can be a possible candidate for the final answer. Formula `r3` provides another way of determining a candidate answer which is derived from similarities of keywords of `hasImgAns` and `hasQ`. Finally, `r4` combines all information from images, questions and similarities, then generates the unknown node `ans`, which represents the output in PGM.

7.1.1 Explaining Correct Output. In evaluating the "terminal" test case depicted in Figure 10 (a), the model accurately predicts `ans(ID,"terminal")`. To explain this output, we applied ICE on PXAI's provenance and its approximate subgraphs to evaluate the influences of Input vertices on the Output vertex `ans(ID,"terminal")`. In Figure 12 (a), the x-axis represents the number of Input vertices (i.e., the scale of provenance graphs), and the y-axis represents the rank of the most influential Input vertices. It compares the original explanation (the rightmost of the x-axis) with the approximate explanations, derived from the provenance graph and its (0.01,0.01)-approximate subgraphs. It is observed that with approximate subgraphs encompassing more than 21 Input vertices, the top-ranked vertices maintain a similar order to those derived from the original graph.

Among the top-8, `sim("terminal","planes")` is the most influential Input vertex, which is reasonable because planes are highly correlated with terminals. The most influential `hasImg` tuple is `hasImg(ID,"planes","are","parked")`, which provides critical evidence that planes are parked around the building. Finally, `word(ID,"terminal")` is also an important Input vertex for the answer `ans(ID,"terminal")` as it includes the candidate "terminal".

7.1.2 Debugging Undesired Output. PXAI's debugging capability was tested with a misclassified image of a "church" in Figure 10 (b), where the model wrongly inferred "barn" as the building in

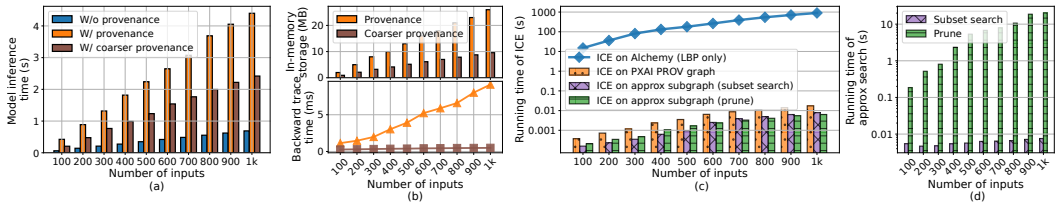

Fig. 13. PXAI performance of provenance maintenance (a), query and storage (b), XAI (c) and approximate subgraph searching (d) when applied to PGM.

Table 4. Two approximate counterfactual explanations to `ans(ID,"church")`.

Input	Expl A	Expl B
`sim("church","cross")`	0.097 → 0.497	0.097 → 0.249
`sim("barn","cross")`	0.301 → 0.253	0.301 → 0
`sim("building","in")`	0.135 → 0.147	0.135 → 0.152
`sim("is","on")`	0.144 → 0.148	0.144 → 0.168

the background. To investigate the misprediction, we applied ICE to both the incorrect output `ans(ID,"barn")` and the correct `ans(ID,"church")` using PXAI on the provenance graph and its (0.01,0.01)-approximate subgraphs. The results are illustrated in Figure 12 (b) and (c), where we show that once the number of Input vertices exceeds 24, the rankings of the approximate subgraphs are consistent with the original provenance graph (the rightmost of the x-axis).

We identified that `sim("church","cross")` is the most influential Input vertex of `ans(ID, "church")` and `sim("barn","cross")` is also among the top-8 for `ans(ID,"barn")`. In the word-similarity data set, we notice that the word "barn" has a higher similarity with "cross" (0.301) compared to the one between "church" and "cross" (0.097), which is counter-intuitive and suspicious. Next, we ran counterfactual explanations on approximate subgraphs for intuition on how to debug this output. Table 4 shows two approximate counterfactual explanations (Expl A and B) in which values of only four Input vertices are modified. These counterfactual explanations demonstrated how altering the probability of `sim("church","cross")` could effectively shift the model's prediction toward the correct answer. For example, increasing the similarity score of `sim("church","cross")` by 0.4 in one counterfactual scenario (Expl A) significantly improved the likelihood of the desired outcome.

7.2 Case Study 2: Text Classification (Probabilistic Graphical Model)

We first evaluate PXAI's performance using the motivation case study text classification [17] presented in Section 1. The Alchemy [1] provides a MLN program that consists of 5365 weighted first-order logic formulas and a testing data set that consists of 50617 `HasWord` input features and 2153 `Links` input features. We systematically sampled subsets of this dataset in increments of 100, ranging from 100 to 1000 instances, adhering to a fixed 3:7 proportion of `Links` to `HasWord` inputs to ensure diversity in feature representation.

7.2.1 Provenance Maintenance and Query. First, we measure the overhead of provenance maintenance. We compare the running time of model inference (i.e., loopy belief propagation) without provenance and with provenance. The evaluation results are shown in Figure 13 (a). We observe that the running times increase linearly as the number of input features increases. Provenance maintenance executes in a constant time and does not affect the asymptotic complexity and scalability. In addition, we note that the performance of provenance maintenance can be optimized by adopting

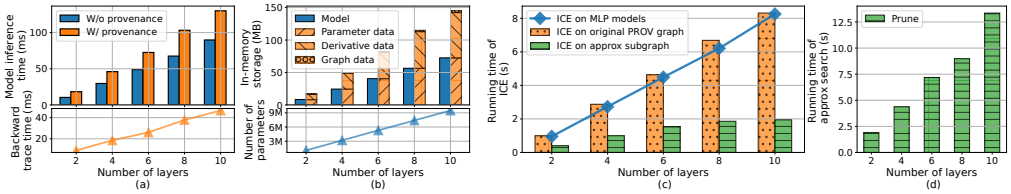

Fig. 14. PXAI performance of provenance maintenance and query (a), storage (b), XAI (c) and approximate subgraph searching (d) when applied to MLP.

coarser provenance where the Sum and Mul operators are replaced by coarser operators, such as sum of multiplications (i.e., the factor-to-variable message passing in belief propagation [52]).

In analyzing PXAI's in-memory storage demands and the provenance query overhead, Figure 13 (b) depicts a linear increase in storage requirements for both the detailed and coarser provenance graphs corresponding to the rise in input features. Subsequently, we calculated the average runtime for backward tracing from all Output vertices across different input feature counts. The outcome, represented by the orange and brown curves, also demonstrates a linear escalation with growing input features. Notably, the additional time incurred by the provenance query is minimal (measured in milliseconds), significantly less than the time taken for model inference, owing to PXAI's ability to omit irrelevant variables and calculations.

7.2.2 XAI Acceleration. To evaluation PXAI's ability to accelerate XAI, we compute the average running time of ICE computation (i.e., computing the influences of all input features for an output), approximate subgraph searching and ICE on approximate subgraphs of all Output vertices. Figure 13 (c) compares the running time of ICE on Alchemy as a baseline (blue curve), PXAI's provenance graph (orange bars), approximate subgraphs from input subset searching (purple bars), and approximate subgraphs from provenance pruning (green bars). Compared to the baseline, PXAI imposes significantly lower overheads *by approximately 5 orders of magnitude*, and the computation is more efficient on the the approximate subgraphs. Figure 13 (d) shows the average running time of input subset searching (purple bars) and pruning (green bars). All approximate subgraphs satisfy (0.01,0.01)-approximate. Pruning is more time-consuming due to the expensive computation of contributions of the edges. However, as shown in Figure 13 (d), pruning creates smaller approximate subgraphs for large sample sizes.

7.3 Case Study 3: Credit Score Classification (Multi-Layer Perceptron)

We next apply PXAI to a Multi-Layer Perceptron (MLP) using a case study: credit score classification. This case study, drawn from a Kaggle competition [30], targets the prediction of creditworthiness based on financial attributes. After preprocessing, which includes data cleaning, one-hot encoding, and data scaling, the final dataset comprises 46 input features representing various financial indicators, such as "Monthly In-hand Salary" and "Number of Loans". The task is to classify individuals into one of three credit score categories: "Good", "Standard", and "Poor". In the upcoming sections, we conduct a comprehensive evaluation of PXAI. The assessment includes qualitative analysis (Section 7.3.1) of approximate explanations and quantitative analysis (Sections 7.3.2 and 7.3.3) of PXAI runtime performance.

7.3.1 Explaining Output of Poor Credit Score. Credit score classification has been managed by automated decision-making systems, specifically ML models. Recipients of these systems may occasionally require explanations for the decisions made. For example, Figure 15 presents a snapshot of the financial attributes of an individual who has been automatically categorized as having a

Monthly In-hand Salary	Number of Loan	Number of Bank Accounts	Delay from Due Date	Payment of Min Amount	Student Loan	Mortgage Loan	Model Output
1480.6	9	7	37	True	True	True	Poor

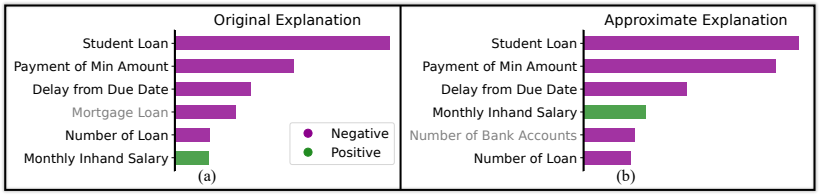

Fig. 15. An example of credit score classification explanations.

"Poor" credit score by an MLP model. And we aim to address the question: “Why the credit score is not good?” by utilizing ICE with PXAI.

Figure 15 (a) and (b) show the top-6 influential input features, assessed via normalized ICEs, in relation to the "Good" credit score output. This evaluation is conducted using both the original provenance graph and a (0.1, 0.1)-approximate subgraph where 50% of edges have been pruned. Negative influences are represented by magenta bars, whereas positive influences are illustrated by green bars. It is demonstrated that five out of six input features (denoted in black text) remain included in the approximate explanation, and the top-3 input features do not change in the approximate explanation. The findings highlight that PXAI’s approximate provenance method is capable of generating reasonable explanations.

7.3.2 Provenance Maintenance and Query. To evaluate the scalability of PXAI when applied to MLP, we conducted the training and subsequent evaluation of the PXAI on a series of MLP models with varying depths. Specifically, MLPs with 2, 4, 6, 8, and 10 layers were analyzed, each consisting of 1024 neurons per layer. First, we evaluate the running time of the MLP model with and without the inclusion of PXAI’s provenance maintenance. The performance are summarized in Figure 14 (a) where the provenance maintenance introduces approximately 40%+ overhead. Furthermore, the execution time for backward tracing from the Output vertices back to Input vertices offers evidence of PXAI’s scalability.

Next, we investigate the space complexity of PXAI, in comparison with the original MLP models, by measuring the in-memory storage requirements for provenance data as more features are processed through the graph. Figure 14 (b) shows the decomposed provenance storage, and the scaling of model parameters. Specifically, the provenance consists of model parameter data, which is nearly equivalent to the model itself, derivative data, and graph data, including vertices and edges. The results indicate that PXAI introduces a linear storage overhead.

7.3.3 XAI Acceleration. To evaluate PXAI’s ability of XAI acceleration, in this subsection, we compute the average time of ICE computations, identify and utilize approximate subgraphs, and execute ICE on these subgraphs. The performance comparison is visualized in Figure 14 (c) and reveals the time efficiency of PXAI’s approximate methods compared to a baseline of ICE on the original, non-approximated MLP structure. As depicted in Figure 14 (c), the execution time on the original provenance graph (orange bars) aligns closely with the established baseline (blue curve), which is reasonable given the minimal presence of irrelevant computations in MLP inference (See Figure 6). Notably, the employment of approximate subgraphs facilitates at most 4x acceleration, notwithstanding a one-time overhead attributed to the approximate subgraphs searching, which is presented in Figure 14 (d). In this figure, the searching time grows linearly as the number of layers increases.

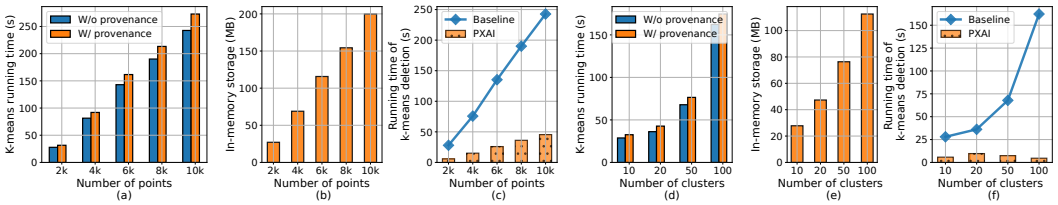

Fig. 16. PXAI performance of provenance maintenance (a and c), query (b and e), and ML deletion (c and f) when applied to k-means clustering.

Algorithm 5 k-means Deletion on Provenance Graph

```
1:  function K-MeansDeletion(G, n)
2:      C^i ← ∅                                          ▷ The set of clusters to update
3:      for i ← 1 to G.iteration do
4:          v_p ← G.getVertex(n, i)                      ▷ The vertex of point n
5:          v_c ← v_p.successors()                       ▷ The centroid of point n
6:          Remove the edge from v_p to v_c               ▷ Delete the point
7:          C^i.insert(v_c)
8:          for v_c ∈ C^I do
9:              Update v_c.values through backward trace
10:             for v_d ∈ v_c.successors() do
11:                 Update v_d.values[v_c.ID]
12:                 if The nearest neighbor changes then
13:                     Add/remove the edges
14:                     Add corresponding centroid vertices to C^{i+1}
15:         if converged then return
16:     if ! converged then
17:         Continue to run the original k-means algorithm
```

7.4 Case Study 4: ML Deletion (k-Means Clustering)

Continuing our evaluation of PXAI, we delve into ML deletion with a focus on k-means clustering [22]. ML deletion seeks to efficiently updating a trained model subsequent to the removal of data points from the training dataset. The provenance-based approach promises performance enhancements in this context *without any approximation of ML deletion*. The provenance-enabled k-means deletion algorithm, outlined in Algorithm 5, is designed to minimize irrelevant computations . The algorithm works by identifying the data point's associated centroid and the distances to the centroids, then updating the centroid's value, distances, and its dependent data points' assignments, through provenance tracing.

To conduct a thorough evaluation, we leveraged the widely recognized MNIST Database of Handwritten Digits [14]—a benchmark challenge in the field of machine learning. Our experimental analysis proceeded in a twofold manner. First, we quantified the scalability of provenance maintenance, query performance, and k-means deletion as a function of the point number (the number of clusters is set 10). The results are presented in Figure 16 (a)-(c). Next, we extended our analysis to examine the effects of cluster numbers on the same performance metrics (the number of points is set 2000). The results are presented in Figure 16 (d)-(f).

To assess the overhead costs associated with provenance maintenance, we track and compare the execution times of k-means clustering with and without provenance. In Figure 16 (a) and (d), PXAI

Table 5. Applicability of PXAI to ML models (○ stands for unessential and not suitable, ◐ stands for essential but not suitable, ◑ stands for unessential but suitable, ● stands for essential and suitable).

ML models	Supervised learning								Unsupervised learning		Reinforcement learning	
	Linear			Tree-based		Naive Bayesian	PGM		Clustering		Model-based	Model-free
	Linear regression	Logistic regression	Neural network	Decision tree	Random forest		Bayesian network	Markov network	k-means	GMM	Deep Q-learning	Q-learning
PXAI applicable	◑	◑	●	○	◐	○	●	●	●	◐	●	○

Table 6. Applicability of PXAI to XAI tools (○ stands for not applicable and ● stands for applicable).

XAI	Post-hoc								Ante-hoc
	Feature attribution				Counterfactual explanation		Surrogate-based		Henricks et al. [27]
	LRP [5]	ICE [23]	SHAP [36]	PDA [62]	Wachter et al. [57]	Tolomei et al. [55]	LIME [47]	Anchor [48]	
PXAI applicable	○	●	●	●	●	○	●	●	○

incurs less than 15% overheads. In addition, Figure 16 (b) and (e) show a linearly growth of PXAI's memory utilization. We did not measure provenance query because Algorithm 5 are performed on the whole provenance graph.

Our primary goal with this case study, as demonstrated in [22] (c) and (f), is to evaluate the effectiveness of a provenance-aware k-means deletion algorithm (orange bars) compared to retraining the k-means model from scratch after data point removal (blue curves). The algorithm's efficiency stems from its ability to target only the computation paths affected by the deletion, thereby reducing the overall calculation load. In Figure 16 (c), the running times of both the baseline and PXAI grow linearly. PXAI enables more than 5x accelerations of k-means deletion. In Figure 16 (f), the execution time of PXAI initially increases and then decreases, in contrast to the rapidly escalating running time of the baseline. Notably, at a cluster count of 100, PXAI achieves a substantial 35x acceleration compared to the baseline. This efficiency gain is attributed to the fact that as the number of clusters grows, a larger proportion of clusters and data points become irrelevant to the ML deletion process and are consequently bypassed via provenance tracking.

8 Discussion

Applicability to other ML models Beyond the case studies presented in Section 7, we now consider the applicability of PXAI to other ML models. Table 5 outlines the mainstream ML landscape. We consider the interpretability of ML models and assess PXAI's suitability in terms of intermediate computation. We find that PXAI is not *essential* for inherently interpretable ML models, including (but not limited to) linear regression, logistic regression, and decision trees, because their decision-making processes are already transparent. We also find that PXAI is not *suitable* for ML models that do not generate intermediate computational results during inference, such as Gaussian Mixture Models (GMMs) and Q-learning algorithms.

We discuss PXAI's applicability in Table 6, encompassing categories of various XAI tools. PXAI is applicable to all XAI tools that follow a sample-then-inference procedure, such as ICE [23] of feature attribution and counterfactual explanations [57]. Some XAI tools, such as LRP [5], are computationally efficient; however, they are model-specific (e.g., for neural networks). Ante-hoc XAI tools, such as Henricks et al. [27], are designed as inherently interpretable ML models and do not require PXAI.

Scope of Novelty Our goal with PXAI is not to create a novel XAI algorithm but rather to improve the computational efficiency of existing XAI algorithms. For example, the objective

of XAI-approximate is not to enhance the explainability of AI/ML models, but rather to reduce computational overhead while maintaining minimal deviations from the original explanations.

Developer Effort PXAI currently requires an ML developer to instrument their source code with the appropriate API calls in Table 3 to maintain adequate provenance collection. Although existing provenance tools can track data derivation in source code, such as noWorkflow [43] or in binary executables, scuh as BEEP [33] and OmegaLog [26], we leave automated provenance tracking in ML applications as future work.

Scalability Challenges In recent years, neural networks have scaled to encompass over one billion parameters, as evidenced by the development of large language models (LLMs). When applied to large-scale neural networks with complicated architectures, despite the linear computational overhead associated with provenance maintenance as demonstrated in Section 7, PXAI encounters the following challenges.

(1) **Storage overhead**: Currently, PXAI supports only in-memory provenance, which may become exceedingly large and unsuitable for large-scale neural networks. However, there have been extensive study on provenance storage and optimizations in database, such as Chapman et al. [11], Hu et al. [29] and Ding et al. [16], providing a comprehensive view of potential solutions to this challenge.
(2) **Parallel acceleration**: The inference of large-scale neural networks often relies on parallel acceleration using GPUs. At present, PXAI does not implement parallel acceleration, including batch inference, which presents a limitation. Future work could incorporate parallel acceleration techniques to enhance PXAI's scalability.

9 Related Work

Reuse of intermediate computation results Clipper [12] caches model inference results to improve the throughput of training. Pretzel [34] converts machine learning pipelines into "model plans", which reduces the model inference latency by reusing the parameters and computations between similar model plans. These approaches do not implement provenance or lineage and are highly coupled with specific ML systems (e.g., ML.Net, Tensorflow and Caffe) that generate machine learning pipelines.

Some prior work has implemented data provenance/lineage for data reuse. nbsafety [38] traces the lineage of intermediate states of the "cells" in Jupyter Notebook to detect and avoid unsafe interactions from the notebook users. KeystoneML [54] and HELIX [60] optimize iterative executions of machine learning workflows, including preprocessing, model training, model inference and post-processing, by reusing computation results from previous iterations. These approaches are coarse-grained; therefore, they cannot trace fine-grained (i.e., pipeline-level) computation dependencies required by XAI tools. LIMA [42] builds a fine-grained lineage inside ML systems, but its performance is susceptible to the density of lineage. In contrast, PXAI decouples XAI tools from AI/ML systems/models, implements a fine-grained provenance model, and further accelerates XAI by searching approximate subgraphs.

AI/ML security and interpretation Data provenance improves AI/ML security by tracing the data in model training. For example, Song and Vitaly [53] design a black-box auditing method that determines whether a user's data are used to train a text-generative model. Baracaldo et al. [7] uses data provenance to track the origin and transformation of data points in the training data set to filter "poisonous" data. PrIU [59] tracks the provenance of the data for incremental training of logistic regression and linear regression models so that the training error can be identified. However, these approaches focus on model training instead of model inference.

Data provenance helps interpret or debug model inference results. noWorkFlow [43] collects provenance that captures all computational steps and data leading to an output from Python scripts. LAMP [37]'s fine-grained provenance model records computation dependencies for graphical ML model inference (i.e., PageRank), adopts automatic differentiation to explain model inference results, and uses provenance to decouple the intensive derivative computation from the models. Deep learning frameworks also builds computational graphs, such as PyTorch [3]. In contrast, PXAI offers a more general approach applicable to various AI/ML models. It enhances the computational efficiency of model-agnostic XAI tools through provenance-enabled model inference and the introduction of XAI-approximate.

Some studies, such as those by Deutch et al. [15], Milo et al. [39], and P3 [58], have leveraged provenance and approximate provenance to explain inference results within DataLog and ProbLog programs. In contrast, PXAI expands the scope to encompass a broader array of ML models.

10 Conclusion

In this paper, we introduced PXAI, a local and model-agnostic XAI tool, following a sample-then-inference procedure. To decouple XAI from AI/ML models, we create a provenance model that tracks the creation and transformation of all data within AI/ML models. To exclude insignificant variables and computations without affecting the explanations, we define and design searching algorithms for approximate subgraphs that are XAI-approixmate. Our evaluation shows that PXAI derives reasonable explanations and that PXAI significantly improves the computation efficiency of XAI tools.

Acknowledgments

The authors would like to thank the anonymous shepherd and reviewers for their helpful comments and feedback, which improved this work. This material is based upon work supported by the National Science Foundation under Grant No. CNS-1704189.

References

[1] [n. d.]. Alchemy - Open Source AI. http://alchemy.cs.washington.edu/alchemy1.html

[2] [n. d.]. The Boost Graph Library (BGL). https://www.boost.org/doc/libs/1_80_0/libs/graph/doc/index.html

[3] [n. d.]. Overview of PyTorch Autograd Engine. https://pytorch.org/blog/overview-of-pytorch-autograd-engine/

[4] Somak Aditya, Yezhou Yang, and Chitta Baral. 2018. Explicit Reasoning over End-to-End Neural Architectures for Visual Question Answering. In *AAAI*.

[5] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. 2015. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one* 10, 7 (2015), e0130140.

[6] Stephen H. Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. 2015. Hinge-Loss Markov Random Fields and Probabilistic Soft Logic. *Journal of Machine Learning Research* 18 (2015), 109:1–109:67.

[7] Nathalie Baracaldo, Bryant Chen, Heiko Ludwig, and Jaehoon Amir Safavi. 2017. Mitigating Poisoning Attacks on Machine Learning Models: A Data Provenance Based Approach. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security* (Dallas, Texas, USA) *(AISec '17)*. Association for Computing Machinery, New York, NY, USA, 103–110. https://doi.org/10.1145/3128572.3140450

[8] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador Garcia, Sergio Gil-Lopez, Daniel Molina, Richard Benjamins, Raja Chatila, and Francisco Herrera. 2020. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion* 58 (2020), 82–115. https://doi.org/10.1016/j.inffus.2019.12.012

[9] Peter Buneman, Sanjeev Khanna, and Tan Wang-Chiew. 2001. Why and where: A characterization of data provenance. In *International conference on database theory*. Springer, 316–330.

[10] Rich Caruana, Yin Lou, Johannes Gehrke, Paul Koch, Marc Sturm, and Noemie Elhadad. 2015. Intelligible Models for HealthCare: Predicting Pneumonia Risk and Hospital 30-Day Readmission. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Sydney, NSW, Australia) *(KDD '15)*. Association for Computing Machinery, New York, NY, USA, 1721–1730. https://doi.org/10.1145/2783258.2788613

[11] Adriane P. Chapman, H. V. Jagadish, and Prakash Ramanan. 2008. Efficient provenance storage. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data* (Vancouver, Canada) *(SIGMOD '08)*. Association for Computing Machinery, New York, NY, USA, 993–1006. https://doi.org/10.1145/1376616.1376715

[12] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J. Franklin, Joseph E. Gonzalez, and Ion Stoica. 2017. Clipper: A Low-Latency Online Prediction Serving System. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. USENIX Association, Boston, MA, 613–627. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/crankshaw

[13] Susanne Dandl, Christoph Molnar, Martin Binder, and Bernd Bischl. 2020. Multi-objective counterfactual explanations. In *Parallel Problem Solving from Nature–PPSN XVI: 16th International Conference, PPSN 2020, Leiden, The Netherlands, September 5-9, 2020, Proceedings, Part I*. Springer, 448–469.

[14] Li Deng. 2012. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine* 29, 6 (2012), 141–142.

[15] Daniel Deutch, Amir Gilad, and Yuval Moskovitch. 2015. Selective provenance for datalog programs using top-k queries. *Proc. VLDB Endow.* 8, 12 (aug 2015), 1394–1405. https://doi.org/10.14778/2824032.2824039

[16] Hailun Ding, Juan Zhai, Dong Deng, and Shiqing Ma. 2023. The Case for Learned Provenance Graph Storage Systems. In *32nd USENIX Security Symposium (USENIX Security 23)*. USENIX Association, Anaheim, CA, 3277–3294. https://www.usenix.org/conference/usenixsecurity23/presentation/ding-hailun-provenance

[17] P. Domingos and D. Lowd. 2009. . https://doi.org/10.2200/S00206ED1V01Y200907AIM007

[18] Pedro Domingos and Daniel Lowd. 2019. Unifying logical and statistical AI with Markov logic. *Commun. ACM* 62, 7 (2019), 74–83.

[19] Jaimie Drozdal, Justin Weisz, Dakuo Wang, Gaurav Dass, Bingsheng Yao, Changruo Zhao, Michael Muller, Lin Ju, and Hui Su. 2020. Trust in AutoML: exploring information needs for establishing trust in automated machine learning systems. In *Proceedings of the 25th International Conference on Intelligent User Interfaces* (Cagliari, Italy) *(IUI '20)*. Association for Computing Machinery, New York, NY, USA, 297–307. https://doi.org/10.1145/3377325.3377501

[20] Patrick Forré and Joris M Mooij. 2017. Markov properties for graphical models with cycles and latent variables. *arXiv preprint arXiv:1710.08775* (2017).

[21] Víctor Garcia Satorras and Max Welling. 2021. Neural Enhanced Belief Propagation on Factor Graphs. In *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research, Vol. 130)*, Arindam Banerjee and Kenji Fukumizu (Eds.). PMLR, 685–693. https://proceedings.mlr.press/v130/garcia-satorras21a.html

[22] Antonio A. Ginart, Melody Y. Guan, Gregory Valiant, and James Zou. 2019. *Making AI Forget You: Data Deletion in Machine Learning*. Curran Associates Inc., Red Hook, NY, USA.

[23] Alex Goldstein, Adam Kapelner, Justin Bleich, and Emil Pitkin. 2015. Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation. *journal of Computational and Graphical Statistics* 24, 1 (2015), 44–65.

[24] Todd J. Green, Grigoris Karvounarakis, and Val Tannen. 2007. Provenance Semirings. In *Proceedings of the Twenty-Sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (Beijing, China) *(PODS '07)*. Association for Computing Machinery, New York, NY, USA, 31–40. https://doi.org/10.1145/1265530.1265535

[25] David Gunning and David Aha. 2019. DARPA's explainable artificial intelligence (XAI) program. *AI magazine* 40, 2 (2019), 44–58.

[26] Wajih Ul Hassan, Mohammad A. Noureddine, Pubali Datta, and Adam Bates. 2020. OmegaLog: High-Fidelity Attack Investigation via Transparent Multi-layer Log Analysis. *Proceedings 2020 Network and Distributed System Security Symposium* (2020). https://api.semanticscholar.org/CorpusID:211268590

[27] Lisa Anne Hendricks, Zeynep Akata, Marcus Rohrbach, Jeff Donahue, Bernt Schiele, and Trevor Darrell. 2016. Generating visual explanations. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*. Springer, 3–19.

[28] Andreas Holzinger, Randy Goebel, Ruth Fong, Taesup Moon, Klaus-Robert Müller, and Wojciech Samek. 2022. xxAI-beyond explainable artificial intelligence. In *xxAI-Beyond Explainable AI: International Workshop, Held in Conjunction with ICML 2020, July 18, 2020, Vienna, Austria, Revised and Extended Papers*. Springer, 3–10.

[29] Die Hu, Dan Feng, Yulai Xie, Gongming Xu, Xinrui Gu, and Darrell Long. 2020. Efficient Provenance Management via Clustering and Hybrid Storage in Big Data Environments. *IEEE Transactions on Big Data* 6, 4 (2020), 792–803. https://doi.org/10.1109/TBDATA.2019.2907116

[30] Kaggle. [n. d.]. *Credit Score Classification*. https://www.kaggle.com/datasets/parisrohan/credit-score-classification

[31] Been Kim, Rajiv Khanna, and Oluwasanmi O Koyejo. 2016. Examples are not enough, learn to criticize! criticism for interpretability. *Advances in neural information processing systems* 29 (2016).

[32] Daphne Koller and Nir Friedman. 2009. *Probabilistic graphical models: principles and techniques*. MIT press.

[33] Kyu Hyung Lee, Xiangyu Zhang, and Dongyan Xu. 2013. High Accuracy Attack Provenance via Binary-based Execution Partition. In *20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24-27, 2013*. The Internet Society. https://www.ndss-symposium.org/ndss2013/high-accuracy-attack-provenance-binary-based-execution-partition

[34] Yunseong Lee, Alberto Scolari, Byung-Gon Chun, Marco Domenico Santambrogio, Markus Weimer, and Matteo Interlandi. 2018. {PRETZEL}: Opening the black box of machine learning prediction serving systems. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 611–626.

[35] H-A Loeliger. 2004. An introduction to factor graphs. *IEEE Signal Processing Magazine* 21, 1 (2004), 28–41.

[36] Scott M Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. *Advances in neural information processing systems* 30 (2017).

[37] Shiqing Ma, Yousra Aafer, Zhaogui Xu, Wen-Chuan Lee, Juan Zhai, Yingqi Liu, and Xiangyu Zhang. 2017. LAMP: Data Provenance for Graph Based Machine Learning Algorithms through Derivative Computation. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering* (Paderborn, Germany) *(ESEC/FSE 2017)*. Association for Computing Machinery, New York, NY, USA, 786–797. https://doi.org/10.1145/3106237.3106291

[38] Stephen Macke, Hongpu Gong, Doris Jung-Lin Lee, Andrew Head, Doris Xin, and Aditya Parameswaran. 2021. Fine-Grained Lineage for Safer Notebook Interactions. *Proc. VLDB Endow.* 14, 6 (feb 2021), 1093–1101. https://doi.org/10.14778/3447689.3447712

[39] Tova Milo, Yuval Moskovitch, and Brit Youngmann. 2020. Contribution Maximization in Probabilistic Datalog. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. 817–828. https://doi.org/10.1109/ICDE48307.2020.00076

[40] Ramaravind K Mothilal, Amit Sharma, and Chenhao Tan. 2020. Explaining machine learning classifiers through diverse counterfactual explanations. In *Proceedings of the 2020 conference on fairness, accountability, and transparency*. 607–617.

[41] Kevin Murphy, Yair Weiss, and Michael I Jordan. 2013. Loopy belief propagation for approximate inference: An empirical study. *arXiv preprint arXiv:1301.6725* (2013).

[42] Arnab Phani, Benjamin Rath, and Matthias Boehm. 2021. LIMA: Fine-Grained Lineage Tracing and Reuse in Machine Learning Systems. In *Proceedings of the 2021 International Conference on Management of Data* (Virtual Event, China) *(SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA, 1426–1439. https://doi.org/10.1145/3448016.3452788

[43] João Felipe Pimentel, Leonardo Murta, Vanessa Braganholo, and Juliana Freire. 2017. noWorkflow: a tool for collecting, analyzing, and managing provenance from python scripts. *Proc. VLDB Endow.* 10, 12 (aug 2017), 1841–1844. https://doi.org/10.14778/3137765.3137789

[44] Joelle Pineau, Philippe Vincent-Lamarre, Koustuv Sinha, Vincent Larivière, Alina Beygelzimer, Florence d'Alché Buc, Emily Fox, and Hugo Larochelle. 2021. Improving Reproducibility in Machine Learning Research (a Report from the NeurIPS 2019 Reproducibility Program). *J. Mach. Learn. Res.* 22, 1, Article 164 (jan 2021), 20 pages.

[45] Atul Rawal, James McCoy, Danda B. Rawat, Brian M. Sadler, and Robert St. Amant. 2022. Recent Advances in Trustworthy Explainable Artificial Intelligence: Status, Challenges, and Perspectives. *IEEE Transactions on Artificial Intelligence* 3, 6 (2022), 852–866. https://doi.org/10.1109/TAI.2021.3133846

[46] Christopher Ré and Dan Suciu. 2008. Approximate lineage for probabilistic databases. In *PVLDB*. 797–808.

[47] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. " 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*. 1135–1144.

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

[49] Matthew Richardson and Pedro Domingos. 2006. Markov Logic Networks. *Mach. Learn.* 62, 1-2 (Feb. 2006), 107–136.

[50] Wojciech Samek, Grégoire Montavon, Sebastian Lapuschkin, Christopher J. Anders, and Klaus-Robert Müller. 2021. Explaining Deep Neural Networks and Beyond: A Review of Methods and Applications. *Proc. IEEE* 109, 3 (2021), 247–278. https://doi.org/10.1109/JPROC.2021.3060483

[51] Sebastian Schelter, Joos-Hendrik Böse, Johannes Kirschnick, Thoralf Klein, and Stephan Seufert. 2017. Automatically tracking metadata and provenance of machine learning experiments. In *NeurIPS 2017*. https://www.amazon.science/publications/automatically-tracking-metadata-and-provenance-of-machine-learning-experiments

[52] Parag Singla and Pedro M Domingos. 2008. Lifted First-Order Belief Propagation.. In *AAAI*, Vol. 8. 1094–1099.

[53] Congzheng Song and Vitaly Shmatikov. 2019. Auditing data provenance in text-generation models. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 196–206.

[54] Evan R Sparks, Shivaram Venkataraman, Tomer Kaftan, Michael J Franklin, and Benjamin Recht. 2017. Keystoneml: Optimizing pipelines for large-scale advanced analytics. In *2017 IEEE 33rd international conference on data engineering (ICDE)*. IEEE, 535–546.

[55] Gabriele Tolomei, Fabrizio Silvestri, Andrew Haines, and Mounia Lalmas. 2017. Interpretable predictions of tree-based ensembles via actionable feature tweaking. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. 465–474.

[56] Paul Voigt and Axel Von dem Bussche. 2017. The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing* 10, 3152676 (2017), 10–5555.

[57] Sandra Wachter, Brent Mittelstadt, and Chris Russell. 2017. Counterfactual explanations without opening the black box: Automated decisions and the GDPR. *Harv. JL & Tech.* 31 (2017), 841.

[58] Shaobo Wang, Hui Lyu, Jiachi Zhang, Chenyuan Wu, Xinyi Chen, Wenchao Zhou, Boon Thau Loo, Susan B. Davidson, and Chen Chen. 2020. Provenance for Probabilistic Logic Programs. In *Extending Database Technology*. 145–156.

[59] Yinjun Wu, Val Tannen, and Susan B Davidson. 2020. Priu: A provenance-based approach for incrementally updating regression models. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 447–462.

[60] Doris Xin, Stephen Macke, Litian Ma, Jialin Liu, Shuchen Song, and Aditya Parameswaran. 2018. HELIX: Holistic Optimization for Accelerating Iterative Machine Learning. *Proc. VLDB Endow.* 12, 4 (dec 2018), 446–460. https://doi.org/10.14778/3297753.3297763

[61] Hao Yuan, Jiliang Tang, Xia Hu, and Shuiwang Ji. 2020. XGNN: Towards Model-Level Explanations of Graph Neural Networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (Virtual Event, CA, USA) *(KDD '20)*. Association for Computing Machinery, New York, NY, USA, 430–438. https://doi.org/10.1145/3394486.3403085

[62] Luisa M. Zintgraf, Taco S. Cohen, Tameem Adel, and Max Welling. 2017. Visualizing Deep Neural Network Decisions: Prediction Difference Analysis. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=BJ5UeU9xx

Received April 2024; revised July 2024; accepted August 2024