Explainable AI (XAI): Core Ideas, Techniques, and Solutions

RUDRESH DWIVEDI, Netaji Subhas University of Technology (formerly NSIT)
DEVAM DAVE, HET NAIK, and SMITI SINGHAL, Pandit Deendayal Petroleum University
RANA OMER, Cardiff University
PANKESH PATEL, University of South Carolina
BIN QIAN, ZHENYU WEN, TEJAL SHAH, GRAHAM MORGAN, and RAJIV RANJAN, Newcastle University

As our dependence on intelligent machines continues to grow, so does the demand for more transparent and interpretable models. In addition, the ability to explain the model generally is now the gold standard for building trust and deployment of artificial intelligence systems in critical domains. Explainable artificial intelligence (XAI) aims to provide a suite of machine learning techniques that enable human users to understand, appropriately trust, and produce more explainable models. Selecting an appropriate approach for building an XAI-enabled application requires a clear understanding of the core ideas within XAI and the associated programming frameworks. We survey state-of-the-art programming techniques for XAI and present the different phases of XAI in a typical machine learning development process. We classify the various XAI approaches and, using this taxonomy, discuss the key differences among the existing XAI techniques. Furthermore, concrete examples are used to describe these techniques that are mapped to programming frameworks and software toolkits. It is the intention that this survey will help stakeholders in selecting the appropriate approaches, programming frameworks, and software toolkits by comparing them through the lens of the presented taxonomy.

CCS Concepts: • **Computing methodologies → Knowledge representation and reasoning;**

Additional Key Words and Phrases: Explainable artificial intelligence, interpretable AI, programming framework, software toolkits

ACM Reference format:
Rudresh Dwivedi, Devam Dave, Het Naik, Smiti Singhal, Rana Omer, Pankesh Patel, Bin Qian, Zhenyu Wen, Tejal Shah, Graham Morgan, and Rajiv Ranjan. 2023. Explainable AI (XAI): Core Ideas, Techniques, and Solutions. *ACM Comput. Surv.* 55, 9, Article 194 (January 2023), 33 pages.
https://doi.org/10.1145/3561048

Authors' addresses: R. Dwivedi (corresponding author), Netaji Subhas University of Technology (formerly NSIT), Dwarka, Delhi, India; email: rudresh.dwivedi@nsut.ac.in; D. Dave, H. Naik, and S. Singhal, Pandit Deendayal Petroleum University, Raysan, Gandhinagar, India; emails: {devam.dce18, het.nce18, smiti.sce18}@sot.pdpu.ac.in; R. Omer, School of Computer Science and Informatics, Cardiff University, Cardiff, UK; email: RanaOF@cardiff.ac.uk; P. Patel, AI Institute, University of South Carolina, SC; email: PPANKESH@mailbox.sc.edu; B. Qian, School of Computing, Newcastle university, UK; email: B.Qian3@newcastle.ac.uk; Z. Wen, Institute of Cyberspace Security, and College of Information Engineering, Zhejiang University of Technology, Hangzhou 310023, China; email: zhenyuwen@zjut.edu.cn; T. Shah, G. Morgan, and R. Ranjan, School of Computing, Newcastle university, UK; emails: {B.Qian3, Tejal.Shah, Raj.Ranjan}@newcastle.ac.uk.

0360-0300/2023/01-ART194 $15.00
https://doi.org/10.1145/3561048

1 INTRODUCTION

As a society, our dependence on intelligent machines is on a continuous upswing. From driverless cars to flexible email filters to forward-looking and preemptive law maintenance models, **machine learning (ML)**-based systems are increasingly being deployed across several domains. With the frequent utilization of complex deep learning architectures, it requires urgent attention to understand the inner workings and to get insights into the outcomes. This is a core motivation of **explainable artificial intelligence (XAI)** [14]. The prime reason for rapid growth in XAI is the increased robustness of **artificial intelligence (AI)** systems in business, enterprise computing, and critical industries [13, 14]. For companies such as Google, a false prediction can lead to the application user being shown a wrong recommendation [62]. However, in critical sectors such as healthcare, finance, and the military, inaccurate predictions can have serious consequences on human life. Hence, it is crucial to understand how these systems make their decisions. As AI permeates into these critical areas, the human limitation to understand complex AI models is a major roadblock. The main reason for this is that the data insights and the tasks solved by machines remain out of sight in increasingly complex models. The user would need access to tons of numbers to explain a deep neural network, and moreover there is no concrete way to understand the model completely. In addition to the black box nature of a model, *bias* can creep in when dealing with data [60]. Model performance metrics (e.g., model accuracy) do not always exhibit true prediction decisions [9]. Just having a highly accurate model is not sufficient to trust and deploy the model in real-world applications.

XAI has been receiving much attention across multiple application domains [21]. Consequently, an increasing number of XAI tools and techniques are being proposed both in industry and academia. The current XAI systems exhibit a diverse set of dimensions and functionalities for simple exploratory data analysis to understanding complex AI models. Therefore, selecting the correct method(s) for given requirements necessitates a clear understanding of the methods and basic differences among the different XAI approaches. However, the state-of-the-art analysis with respect to existing approaches for building XAI-enabled applications has been investigated to a limited extent. In the work of Došilović et al. [15], general interpretability and explainability have been discussed but limited to supervised ML models. However, Bhatt el al. [7] discuss how explanation schemes are adapted for ML engineers, end users, or other stakeholders. They provide recommendations for organizations to achieve real-time explanation and to improve performance. Arrieta et al. [4] present concepts, taxonomies, opportunities, and challenges toward responsible AI. The focus of their survey is on schemes for fairness and discrimination in ML models. Our survey focuses on model and data explainability and on model performance metrics. In addition, case studies are used to explain the significance of feature importance. Furthermore, design considerations for XAI frameworks along with software systems have been discussed to support fairness and accountability.

We believe that a comparative analysis is needed to counsel various stakeholders involved in XAI-enabled application development. The focus of this survey is on approaches to develop XAI applications, covering tools, and technologies for XAI and related concepts to aid implementation in AI-based systems.

Outline. The rest of this article[1] is structured as follows. The different stages of building a typical ML model and an XAI-based model are compared in Section 2. The need for XAI in AI-based applications and in the explanation phases of the ML process is then discussed in Section 3, followed by the taxonomy of XAI techniques in Section 4—discussed in the subsequent

[1]Some of the content is derived from our unpu blished technical report: https://arxiv.org/abs/2011.03195.

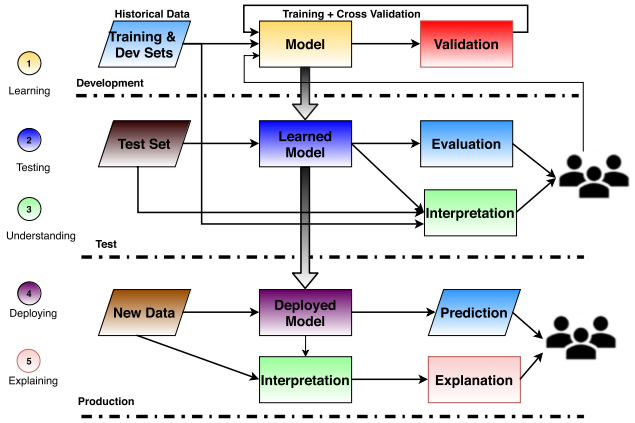

Fig. 1. A pipeline for building ML models with explanation.

sections. Sections 5 and 6 cover the major pillars of XAI system: data explainability and model explainability, respectively. The focus in Section 5 is on basic data visualization and dimensionality techniques along with the associated software libraries, and the focus in Section 6 is on model explainability techniques and the associated software libraries. Using examples, feature-based and example-based XAI techniques are discussed in Sections 7 and 8, respectively. Open source and commercially available toolkits for building XAI models are covered in Section 9, with an analysis on requirements during implementation of XAI presented in Section 10. The article concludes in Section 11.

2 DEVELOPING XAI-BASED APPLICATIONS

Although an ML pipeline can provide accurate predictions, it lacks two important phases: understanding and explaining. The *understanding phase* involves the training and quality assurance of an AI model, whereas the *explaining phase* is important when an ML model is deployed and used in real-world applications. Figure 1 illustrates a revised ML life cycle with the additional steps [58].

2.1 Understanding Phase

The objective of the understanding phase is to improve the model during the training phase prior to deploying it. The stakeholders, as described in Table 1, cross check to make sure that the final model is as precise as it can be and works *as intended* in the real world. The activities involved in this phase are interpreting the important features and how they interact with each other, interpreting what patterns have been learned by the trained model, analyzing biases in data, and ensuring that they are not propagated into the trained model.

In the following, we present use cases of XAI at the understanding phase of a ML process.

Debugging and enhancing AI models. An AI model goes through iterations before it is fully developed [33] (e.g., to improve performance incrementally). During this process, sources of model error are found and removed after meticulously checking and testing. Explanations can shorten this process by helping with the recognition of model error sources.

Detecting bias. A decision-making process can be fully or partially automated with AI models [33]. However, if these models are trained on biased historical data, then such bias can permeate the entire system by negatively impacting the decision outcomes. XAI is a valuable tool that can help

Table 1. Stakeholders During the Understanding Phase

Stakeholders	Description
Developers	They build the AI applications.
	Their primary motive for seeking explainability/interpretability is quality assurance—that is, to aid system testing, debugging, and evaluation, and to improve the robustness of their applications.
Theorists	They understand and advance AI theory.
	The motive to better understand fundamental properties of deep neural networks has led to interdisciplinary research being labeled "artificial neuroscience."
	Membership of this group overlaps with the developer community—for example, in the case of an industry researcher who carries out theoretical work on deep neural network technology (theorist) while also applying the technology to build systems (developer).
Data scientists	The data scientist should be aware of and understand every aspect of the AI system, from the data used for training, the model implemented, and why the predictions developed by the model.
	A data scientist should also be address errors faced by the AI system.

identify biases in AI models. For example, feature importance (discussed in Section 6.1), a common type of explanation, shows the comparative significance of input features for any specific model prediction(s).

Scientific understanding. The Automated Statistician project [56] explains its predictions by breaking down complex datasets into manageable interpretable sections and communicating its results to a user. This enables researchers to enhance their understanding of data features [33].

Building a robust model. Models less likely to be impacted by small changes in inputs are referred to as "robust," and models that are explainable also tend to be more robust [33, 35]. In a way, this is somewhat intuitive: having a coherent explanation for predictions implies that the reasoning is logical, and logical reasoning is less likely to be affected by noise.

AutoML. AutoML has made explainability more important, as it enables the entire data pipeline into a black box (as opposed to just the ML model) [27]. When using AutoML, a user does not have the ability to engineer or select features, or has visibility into a model's decision-making process.

2.2 Explaining Phase

This is the phase where an ML model is deployed and used in real-life applications. The purpose of this phase is to interpret how predictions on real-world data are made by the model. This would expose human-readable explanations to end users, describing how the prediction results are derived—especially important in mission-critical applications. Table 2 describes stakeholders in this phase and why explainability is important for them. In the following, we present use cases of XAI at the explaining phase of the ML process.

Better decision making. The ability to understand why AI systems make certain predictions or generate outcomes can help organizations make appropriate decisions. To illustrate, in churn prediction, an AI model can accurately predict customers likely to leave in the future, and the business is alerted—however, no solution is presented. XAI can shed more light on this decision-making process and answer the "why." Armed with this knowledge, businesses can formulate an appropriate goal-directed plan.

Discrimination. Datasets often contain inherent discrimination. For example, in 2015, searching for "CEO" on Google Images returned women personalities only 11% of the time. This did not match with the real-world representation of 27%. Unlike black box models, XAI models can present the reasoning behind the result, which can help identify the source for the discrimination and address it accordingly.

Justifiability. As legal questions may not addressable with black box models, modern AI systems will have no choice except to add explainability. For example, in the area of individual rights,

Table 2. Stakeholders at the Explaining Phase

Stakeholders	Description
Users	People who use AI systems.
	Members of this group need explanations to help them decide whether/how to act given the outputs of the system, and/or to help justify those actions.
	An example is an insurance company that uses an AI tool to help decide whether and at what cost to sell policies to clients. The end users of the tool, the director of the company, and the clients are all members of the user community.
Consumers	Consumers are recipients of products and services.
	Explanations need to be simple and clear, enabling users with limited understanding to make use of the information without assistance.
	This provides trustworthiness as well as increases the transparency of AI systems.
Businesses	A business stakeholder is someone who wants to deploy an AI system within their product.
	A business stakeholder can include policemen, judges, bank associates, government officials, and doctors, among others.
	Business stakeholders should understand how the model makes a decision. This ensures fairness and also protects users from false decisions taken by the model.
Regulators	An expert or a regulator consistently monitors and audits the AI system.
	In case of a false output/decision made by the model, this group follows the decision trail.
	Another job of regulators is to ensure that the model is up to date, by training it with new data as and when required.

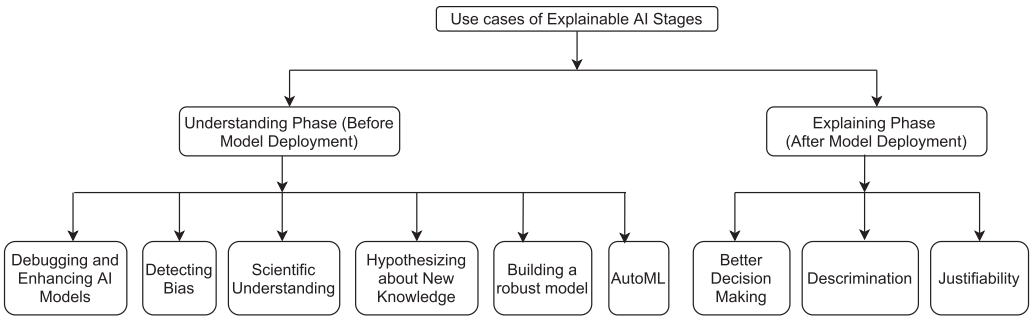

Fig. 2. Use cases of XAI at different phases [4].

regulators will demand more explainability from AI models. People who are adversely impacted by an AI system's decision (e.g., those rejected for a loan) would be interested to know why the system chose not to approve the loan.

3 TAXONOMY OF XAI TECHNIQUES

This section explores different approaches for interpreting ML models and lays the foundations of interpretability techniques. Table 3 presents our comparative analysis and a taxonomy of different XAI techniques covered in subsequent sections (Sections 4 through 7). Figure 3 provides an overview of different XAI techniques.

3.1 White Box Versus Black Box Model Techniques

Black box models are non-transparent in nature, whereas white box models are transparent and comparatively easy to understand. The *black box model* is also termed *intrinsic*, as it is achieved by limiting the complexity of an AI model, whereas a *white box model* is also termed *post hoc*, as it is applied on the model after training.

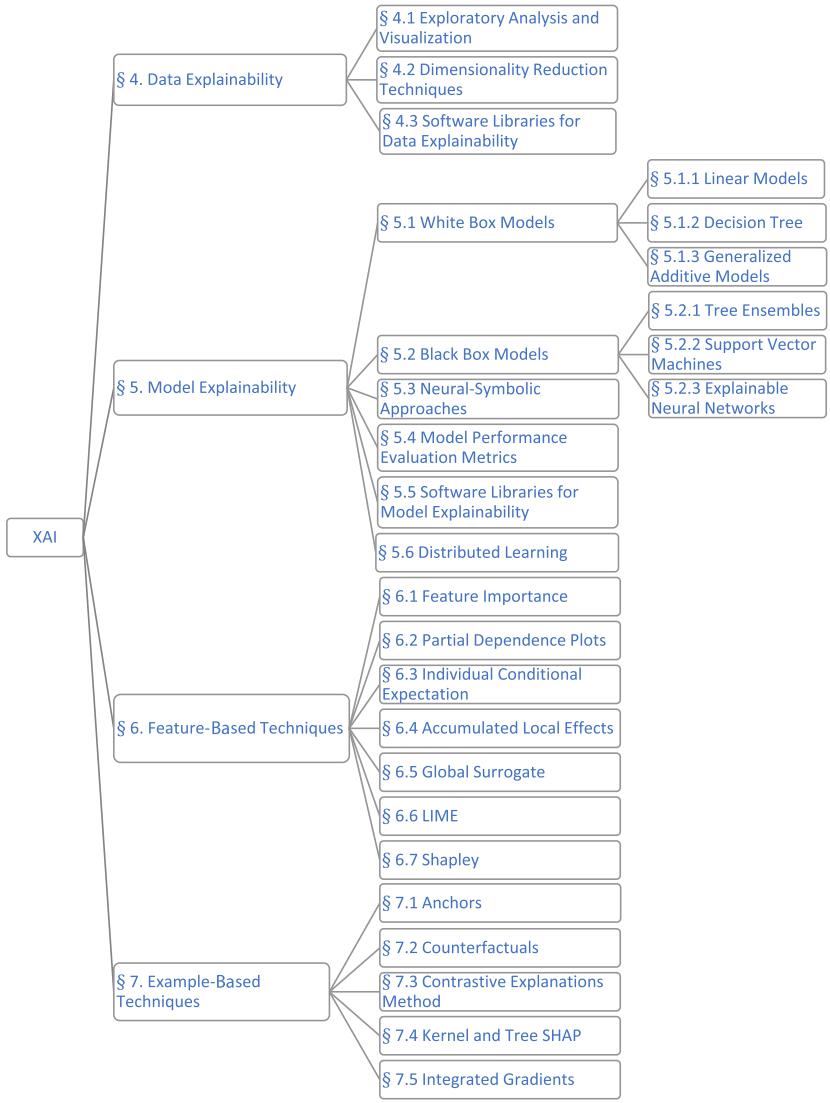

Fig. 3. A taxonomy of XAI techniques applies to data and the model.

White box model techniques. Some AI models are simple and self-explanatory. For example, the predicted outcome y can be mathematically expressed as a weighted sum of all of its features x. It is visualized as a a straight line graph, with a as the slope of the line and b as the intercept on the y-axis. A linear model is a white box model because its mechanism is transparent and simple (as opposed to a black box whose mechanism is not readily understood). Although simple, these are less capable of representing a larger dataset featuring complex interactions. Therefore, for higher accuracy, we require more complex and expressive models.

Black box model techniques. Black box models such as neural networks or complex ensembles of much lower complexity (like a gradient boosting model based on decision trees). The architecture of these models is hard to decipher, as it is not clear how important a role any given feature plays

Table 3. XAI Techniques Versus Taxonomy of XAI Techniques

Classification	XAI Techniques	Global	Local	Model Specific	Model Agnostic	White Box	Black Box
Data explainability	Commonly used data visualization plots	✓	✗	✗	✓	N.A.	N.A.
	Dimensionality reduction techniques	✓	✗	✗	✓	N.A.	N.A.
White box models	Linear model (Section 5.1)	✓	✗	✗	✓	✓	✗
	Decision tree (Section 5.1)	✓	✗	✗	✓	✓	✗
	Generalized additive models (GAMs) (Section 5.1)	✓	✗	✗	✓	✓	✗
	Tree ensembles (Section 5.1)	✓	✗	✗	✓	✓	✗
Artificial neural networks	Neural networks (Section 5.2)	✓	✗	✗	✓	✗	✓
	Neural-symbolic (Section 5.3)	✓	✓	✓	✓	✓	✗
Evaluation metrics	Model evaluation metrics (Section 5.4)	✓	✗	✗	✓	✓	✗
Feature-based XAI techniques	Feature importance (Section 6.1)	✓	✗	✗	✓	✗	✓
	Partial dependence plots (Section 6.2)	✓	✗	✗	✓	✗	✓
	Individual conditional expectation (Section 6.3)	✓	✗	✗	✓	✗	✓
	Accumulated local effects (ALE) (Section 6.4)	✓	✗	✗	✓	✗	✓
	Global surrogate (Section 6.5)	✓	✗	✗	✓	✗	✓
	Local interpretable model-agnostic explanations (LIME) (Section 6.6)	✗	✓	✗	✓	✗	✓
	Shapley value (Section 6.7)	✓	✓	✗	✓	✗	✓
Example-based XAI techniques	Counterfactuals (Section 7.2)	✗	✓	✗	✓	✗	✓
	Anchors (Section 7.1)	✗	✓	✗	✓	✗	✓
	Contrastive explanations method (Section 7.3)	✗	✓	✗	✓	✗	✓
	Prototype counterfactuals (Section 7.2)	✗	✓	✗	✓	✗	✓
	Integrated gradients (Section 7.5)	✗	✓	✗	✓	✓	✗
	Kernel SHAP (Section 7.4)	✓	✓	✗	✓	✗	✓
	Tree SHAP (Section 7.4)	✓	✓	✓	✗	✓	✗

N.A., not applicable.

in the prediction model or how it interacts with other features. For example, in a fully connected neural network, tracing the output features rendered by a model against a specific causative input feature remains a challenge.

3.2 Model-Specific Techniques Versus Model-Agnostic Techniques

There is another dimension to understanding the interpretability of models. It depends on the model being examined. *Model-specific techniques* deal with inner working of a model to interpret its results. They involve examining the structure of an algorithm, including its intermediate representations. *Model-agnostic techniques* deal with analyzing features, their relationship with outputs, and the data distribution. In the following, we briefly present these two techniques [44].

Model-specific techniques. Model-specific interpretation tools are designed purely to interpret models with specific features and capabilities. They can be used only for a single algorithm class. We present various model-specific techniques in Section 5.

Model-agnostic techniques. The interpretation techniques classified as model agnostic can be used on any ML model. The widely used **local interpretable model-agnostic explanations (LIME)** technique is model agnostic and can be used to analyze and interpret any set of ML inputs and corresponding predictions (outputs).

3.3 Global Interpretation Versus Local Interpretation

This classification is based on the scope of the interpretation [44]. The *global interpretation* analyzes the decision-making process at a broader level and is goal oriented. The *local interpretation* gives detailed explanations for every decision made.

Global interpretation. Global interpretation methods involve an overall analysis of a model and its general behavior. The process of defining variables, their dependency, and their interactions goes alongside with the process of assigning importance to these components. Two global interpretation techniques—feature importance and **partial dependence plots (PDPs)**—are described in Section 6.

Local interpretation. Local interpretation involves an analysis of individual predictions and decisions made by the model, to clarify why the model suggested a particular course of action. When a data point prediction/decision is analyzed, the focus is on the subregion around that data point. It enables us to understand the contextual importance of the data point output in that space. LIME and the Shapley value are two such techniques used to understand individual predictions (Section 6).

4 DATA EXPLAINABILITY

The first step of explainability is data visualization, which provides an idea and insight to the dataset. This is where all of the model validation and explanation kick off. This section presents commonly used data explainability techniques—a first step for validating, explaining, and trusting a model. Section 4.3 presents programming frameworks, which can be used to implement these data explainability techniques.

4.1 Exploratory Analysis and Visualization

Visual analysis is crucial for interpretable ML, as knowing its contents is vital for setting a baseline expectation for how models behave and what they create. For a long time, exploratory data analysis and visualization has been a major tool for gleaning meaningful information from data.

4.2 Dimensionality Reduction Techniques

Visualizing is crucial for interpretable ML, but datasets are sometimes hard to visualize due to too many variables and sizes. Although multiple dimensions can be plotted, their interpretation can be complex and error prone. Dimensionality reduction techniques such as **principal component analysis (PCA)**, ICA, isometric mapping (Isomap), ***t*-distributed stochastic neighbor embedding (t-SNE)**, LDA, UMAP (uniform manifold approximation and projection for dimension reduction), LDS, **locally linear embedding (LLE)**, and autoencoders can be used to improve visualization and interpretation. PCA converts observations of correlated features into a set of linearly uncorrelated features through various orthogonal transformations, whereas ICA extracts independent components equivalent to the number of dimensions or features present in the original dataset. Isomap is used to preserve the geodesic distance in the lower dimension, whereas t-SNE produces slightly different results each time on the same dataset, preserving the structure of neighboring points. LDA provides the highest possible discrimination between multiple classes. However, UMAP and LLE are manifold learning methods based on Riemannian geometry and algebraic topology. MDS (multidimensional scaling) represents measures of similarity/dissimilarity among pair of objects by computing distances between points in a low-dimensional space. Autoencoders contain an abstract representation of data, unlike other non-linear dimension reduction algorithms such as LLE or MDS.

4.3 Software Libraries for Data Explainability

Table 4 describes frameworks to implement data explainability techniques. A checkmark (✓) denotes affirmative, and an ✗ denotes unrealizability of the technique. The methodologies provide exploration as well as explanation of the observation data. It gives insights into a dataset by

Table 4. Programming Frameworks for Data Explainability

XAI Techniques	`numpy, pandas`	`matplotlib`	`seaborn`	`sklearn`	`wordcloud`	`networkX`
Data visualization plots	✓	✓	✓	✓	✗	✗
Kernel density estimation (KDE)	✓	✓	✓	✓	✗	✗
Box and whisker plots	✓	✓	✓	✗	✗	
Correlation matrix	✓	✓	✓	✓	✗	✗
WordCloud	✗	✗	✗	✗	✓	✗
Network diagram	✗	✗	✗	✗	✗	✓
Principal component analysis (PCA)	✓	✓	✓	✓	✗	✗
HeatMaps	✓	✓	✓	✓	✗	✗
t-Distributed stochastic neighbor embedding (t-SNE)	✓	✓	✓	✓	✗	✗

expressing the features visually—enabling trends and anomalies to be identified. Although data visualization plots give a good data explainability, plots such as the histogram are much less flexible than kernel density estimations. The latter give estimates of an unknown density function wherein we can not only differ the bandwidth but also implement kernels with various shapes and sizes. However, PCA and t-SNE are purely dimensionality reduction techniques, for example, to 2D or 3D representations.

Software ibraries such as Sklearn and NetworkX can be used for building ML models and network models and provide algorithms for classification and regression. Sklearn provides ease in interpretation for deep learning models. Along with many supervised and unsupervised learning algorithms, Sklearn also consists of cross-validation methods to assess model performance/prediction on unseen data. Using NetworkX, different kinds of networks such as random, weighted, symmetric, and asymmetric networks can be created. WordCloud supports visualization of frequent words in a given text—identifying the most *important* words in the text.

5 MODEL EXPLAINABILITY

In this section, we outline commonly used model explainability techniques to understand AI models, laying the foundation for techniques discussed in Sections 6 and 7. In Section 5.1, we first describe techniques used for the *white box model*, whose internal mechanisms lend themselves to be interpreted in a direct manner. Section 5.2 outlines techniques used in *black box models*, whereas Section 5.3 discusses the application of knowledge representation techniques for explainability. Section 5.4 describes *model performance evaluation metrics*. Section 5.5 presents programming frameworks, which can be used to implement the presented model explainability techniques.

5.1 White Box Models

In this section, we present white box models and programming methods used for interpretation.

5.1.1 Linear Models. Explainability of linear models involves a linear combination of feature values, adjusted by the coefficients of the model. For example, in $y = mx + c$, m is coefficient of feature x^1 and c is coefficient of x^0 so polynomial of degree 1 is a linear polynomial. Similarly, *logistic regression* is one of the most interpretable linear ML models for certain class of events. The seaborn, matplotlib, sklearn, and **accumulated local effects (ALE)** libraries can be used to unfold and visualize a logistic regression model.

5.1.2 Decision Tree. A decision tree predicts the value of a target variable against multiple input variables. The terminal node, also called the *leaf node*, depicts the value of the target variable based on the input variable. A key benefit of decision trees lies in establishing the input and target variable relationship with a logic similar to Boolean. The scikit-learn library includes methods that can be used for interpretation of trees, such as, sklearn.tree.export_text, sklearn.tree.plot_tree,

sklearn.tree.export_graphviz, and dtreeviz and graphviz package. Sklearn also provides a way to evaluate feature importance—the total decrease entropy due to splits over a given feature.

5.1.3 Generalized Additive Models. **Generalized additive models (GAMs)** are an extension of generalized linear models with a smoothing function. GAMs offer a trade-off between simple, interpretable models such as logistic regression and more complex, sophisticated models such as neural networks, which (usually) offer better accuracy and predictive power as compared to simple models. Overfitting is unlikely in GAMs due to the regularization of prediction functions.

5.2 Black Box Models

Black box models cannot be understood or interpreted by themselves. Black box models include tree ensembles, support vector machines, and a variety of neural networks. Inclusion of several layers in a neural network makes it difficult for designers to explain how the algorithm has reached a particular prediction outcome [1].

5.2.1 Tree Ensembles. The tree ensembles method is a learning technique that focuses on integrating several decision trees to create an output. It determines which features to choose at each split. Since a single decision tree may not be enough to yield optimal performance, several decision trees can be combined together to get an optimal performance model. However, the model's complexity increases due to multiple decision trees. As a result, it becomes a lot more difficult to understand model behavior. The following methods are developed for interpreting complex tree ensembles:

- *Simplified tree ensemble learner (STEL)*: The "InTrees" (Interpretable Trees) package coverts a complex tree ensemble into a rule-based learner known as STEL. The ensemble method averages over the variance of multiple models, which in turn deprives the interpretations of individual models.
- *Tree interpreter*: Tree interpreter provides interpretations of decision trees and random forests. It decomposes every prediction result to a sum of feature contributions and bias. This enables a better interpretation of how a feature led to a particular prediction.

5.2.2 Support Vector Machines. Support vector machine is a supervised ML algorithm used for classification and regression. A hyperplane is calculated based on the data points that are plotted in an N-dimensional space. This hyperplane is oriented in such a way that it differentiates between classes by maximizing the distance between the data points of different classes. It works well in high-dimensional spaces and scenarios where the number of dimensions is greater than the number of samples.

5.2.3 Explainable Neural Networks. Inclusion of several layers in large and complex neural nets makes it difficult to explain how a specific prediction or conclusion [1] was reached. This black box explainability refers to assessing predictions made by a model for any given input without having the knowledge of its inherent working. The design of an explainable neural network provides insights into the model. These post hoc explanation schemes are being used for both single and multilayer neural networks, convolutional neural networks, and recurrent neural networks.

5.3 Neural-Symbolic Approaches

Subsymbolic approaches, such as those based on artificial neural networks, are growing in popularity due to their robustness against noisy data and ability to perform complex tasks not otherwise manually possible. However, their black box nature is a major hindrance to explainability. Yet symbolic AI has long been known to be explainable natively [3] since it represents knowledge using

meaningful symbols such as language that is understandable by humans as well as interpretable by machines. However, the lack of scalability of symbolic systems and their inability to handle noisy data has limited their use. More recently, a combination of symbolic and subsymbolic approaches based on artificial neural networks, variously known as neuro-symbolic or neural-symbolic (henceforth NS) systems, has shown significant promise in exploiting the complementary strengths of individual approaches [20, 28]. Consequently, such systems often perform better—for example, Mao et al. [41] present a Neuro-Symbolic Concept Learner, which incorporates symbolic reasoning for better interpretation of visual concepts, and can be trained on less data (10% of the original dataset) while being explainable.

The symbolic element of NS systems commonly consists of knowledge representation and reasoning techniques, such as ontologies and knowledge graphs [28, 47]. These represent domain/expert knowledge as concepts and relationships between them, which together provide semantically rich background context for the domain. Knowledge graphs represent facts, usually in the W3C standard Resource Description Framework [5] or using graph databases and can be linked to ontologies that represent background or contextual knowledge commonly in the Web Ontology Language [29]. Ontologies can thus perform deductive reasoning to derive new, deeper knowledge from existing knowledge in a traceable manner.

Doran et al. [13] postulate that for a system to be comprehensible (i.e., not only be interpretable by experts but also understandable by non-experts), it must "emit symbols" to enable the user to understand how a conclusion is reached. Symbols are closer to how humans understand compared to vectors or numerical encodings as seen in neural networks. However, to achieve this explainability, the authors argue the critical need for reasoning to understand the *why* and *how* behind a particular outcome [13]. Ontologies enable reasoning not only to deduce new knowledge and facts but also for causal inference to understand the cause and effect behind the outcomes, such as through transitivity [23]. For example, in the ontology SNOMED CT [32], the following concepts are represented in a transitive relationship:

$$Acute\ Rheumatic\ Arthritis \xrightarrow{due\ to} Streptococcus\ pyogenes\ Infection \xrightarrow{causative\ agent} Streptcoccus\ pyogenes.$$

Based on transitivity, $\forall x, y, z(R(x, y) \wedge R(y, z) \implies R(x, z))$, it can be inferred that Acute Rheumatic Arthritis (effect) is associated with *Streptococcus pyogenes* (cause). Using further knowledge from the ontology, such as type relationships (classification), a natural language explanation can be generated, such as "`Acute Rheumatic Arthritis`, an `Autoimmune Disorder`, is caused due to an `Infectious Disease`, `Streptococcus pyogenes Infection`. This `Disease` has `causative agent` a `Bacterium`, `Streptococcus pyogenes`. Hence, `Streptococcus pyogenes` is a `causative agent` for `Acute Rheumatic Arthritis` as well." Causal inference has other practical applications such as in personalized recommendations—for instance, drug contraindications [38].

It therefore follows that outputs from subsymbolic approaches when linked to background and contextual knowledge can generate user-specific and understandable explanations [30]. Sarker et al. [47] provide a useful pipeline to demonstrate how background knowledge made available through ontologies can be used to explain the classification behaviors of artificial neural networks. They use DL-Learner [8], a system for supervised ML based on inductive learning, to automatically create class expressions from a **knowledge base (KB)**, which can be used toward explaining the classification behavior. Furthermore, researchers have exploited advances in natural language processing, specifically natural language generation approaches, for further linguistic refinement of explanations [16, 49].

Figure 4 shows an overview of the interaction between neural and symbolic approaches within an NS system [3]. We explain the process with the help of our previous example. Consider a neural

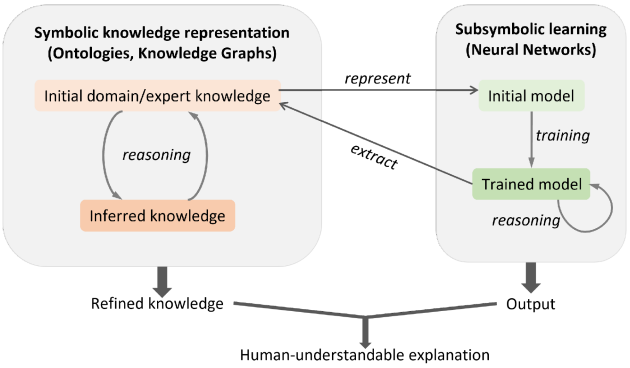

Fig. 4. High-level view of a neural-symbolic system [3].

network model trained for clinical prediction of *Streptococcus pyogenes* infection in patients. The symbolic knowledge consisting of a KB—represented, for example, as a formal ontology—is used to determine the subtypes of the infection to include those in the prediction. The neural model is trained on patient data from several sources such as clinical notes, electronic records, and laboratory results. The output from the model is used to refine the KB by incorporating new associations for causal analysis to encode the rationale behind the neural network's decisions. This and the other encoded knowledge in the KB is now used to construct human-understandable explanations as exemplified earlier. Moreover, the system can be questioned for further analysis, such as *What other conditions do these patients risk developing?* or *What other conditions are caused by S. pyogenes?* In practice, however, only partial elements of such NS integration are utilized [3], and many NS systems are still based on non-logical or "flat" approaches prompting Sarker et al. [48] to argue for the need to focus more on the logical aspects of NS systems to realize improved explainability.

What we can infer from contemporary works is that the combination of neural and symbolic approaches brings together "two most fundamental aspects of intelligent cognitive behavior: the ability to learn from experience and the ability to reason from what was learned" [59]. Thus, where neural systems excel at learning, symbolic approaches can be employed to explain what has been learned. In fact, deep understanding of cognitive science will most certainly benefit the field of NS-AI particularly for the integration of neuro-symbolic approaches [6] to develop a mature and robust integrated model and, consequently, for the design of explainable systems that are closer to how humans understand, think, and communicate.

5.4 Model Performance Evaluation Metrics

In any data science life cycle, model performance evaluation is an important phase before the optimal model is chosen. Evaluation can lead to adjusting model hyperparameters. Evaluation metrics can be both problem specific and agnostic. We present model performance metrics that are incorporated to interpret a model.

5.5 Software Libraries for Model Explainability

In the following, we present some of the software libraries, which can be used for model explainability. Table 5 describes frameworks, which can be used to implement data explainability techniques. It explains if the particular explainable ML algorithm on the left can be carried out with the help of the framework specified at the top. A checkmark (✓) denotes the affirmative case. Similarly, an ✗ signifies that the stated framework cannot be used to implement the technique.

Table 5. Programming Frameworks for Model Explainability

XAI Techniques	Basic Libraries*	TensorFlow	Keras	PyTorch	PyGAM
Linear model coefficients (Section 5.1)	✓	✓	✓	✓	✗
Decision tree (Section 5.1)	✓	✓	✗	✓	✗
Generalized additive models (GAMs) (Section 5.1)	✓	✗	✗	✗	✓
Neural networks (Section 5.2)	✓	✓	✓	✓	✗
Tree ensembles (Section 5.2)	✓	✓	✓	✓	✗
Model performance evaluation metrics (Section 5.4)	✓	✓	✓	✓	✗

*Basic Libraries: `numpy`, `pandas`, `matplotlib`, `seaborn`, `sklearn`.

The mentioned algorithms for generation of models such as linear models, decision trees, GAMs, neural networks, and tree ensembles belong to a family of models that are accounted as explainable. However, some techniques, such as linear model coefficients, do not ensure explainability in practice due to the reason that implementing them on a high-dimensional input may not be explainable. It could only be possible if the number of input features is limited using regularization. Furthermore, the linear model coefficients could be unstable in case of multicollinearlity (multiple correlated features). Decision tree, unlike linear model coefficients, can also be applied to non-linear models. Decision tree visualizations are intuituve and reveal all decisions during model training, thus supporting interpretability and explainability. A GAM is much more flexible than regression. Although GAMs are interpretable, they come at a cost of not fitting every type of data. GAMs are explainable in the sense that the distributions of values for any given feature function can be plotted in 2D, which a domain expert can easily interpret. The most reliable of all neural networks are the most flexible for model explainability. Tree ensembles can be understood as a bunch of decision trees whose results are combined, thus providing the support for explainability the same as decision trees but with the benefit of stringer learning during training.

Ktrain is an interface to Keras to build and train an explainable model. It works for text and image classification. Considering an example image, it focuses on the area of the image over which the classifier can be used for prediction. The visualizations are supported by the eli5 library and based on the GradCAM technique. The text dataset classification is carried out with LIME where the prediction is made using the relative importance of the words.

PyTorch uses Captum for model interpretability, using SmoothGrad, integrated gradients, DeepLIFT, GradientSHAP, and other PyTorch models. It is used for understanding the important neurons and layers of the PyTorch model. It also provides a Web interface "Insights" for data visualization.

Arize AI. Arize is an ML observability framework for model monitoring and assessment, used to diagnose the root cause of a model output and to detect pre- and post-validation checks. In addition, it unveils and explains how models arrive at specific outcomes for any set of predictions.

AIF360. Artificial Intelligence Fairness 360, also referred to as AIF360, is a Python toolkit for the detection and reduction of bias in an ML model to increase the trust in AI, which is the primary vision of XAI. The AIF360 toolkit includes an extensive set of metrics for datasets and models to test for biases along with the explanation of these metrics. These metrics are an integral part of the explanation of biases of an ML model. It also has algorithms to mitigate bias in datasets or/and in models.

AIX360. AI Xplainability 360 is an open source Python toolkit developed by IBM similar to AIF360. This Python toolkit includes several algorithms whose primary motive is the explanation and interpretation of ML models and datasets. Due diligence should be performed before choosing the most appropriate algorithm for explanation of a particular ML model. A decision tree for choosing the appropriate algorithm for a specific condition is given in the following.

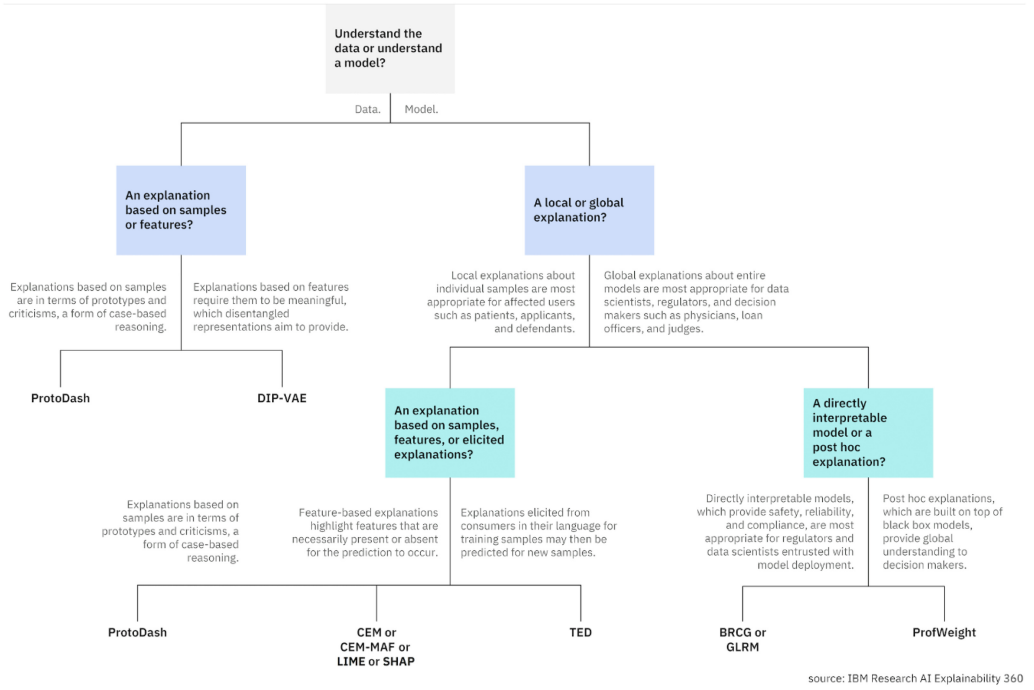

Fig. 5. Decision tree for AIX360 [63].

InterpretML. InterpretML is an open-source Python toolkit by Microsoft for training intelligible models and explaining the black box systems. InterpretML is based on human interpretation of the global and local explanations of the model. It also helps in debugging the model to be able to understand the predictions. InterpretML has the feature of comparing different XAI methods using the same function class. The different XAI methods include decision tree, decision rule list, linear regression, logistic regression, SHAP Kernel Explainer, SHAP Tree Explainer, LIME, Morris sensitivity analysis, and partial dependence. It includes Explainable Boosting Machine, which is an algorithm for explaining models with higher accuracy.

Amazon SageMaker. Similar to Microsoft Azure, Amazon SageMaker is a service offered by Amazon Web Services (AWS) to prepare, build, train, and deploy an ML model. A subcomponent of Amazon SageMaker, called *Amazon SageMaker Clarify*, is used to interpret the ML model by explaining the predictions made by the model. The approach can also be used to detect bias at various stages of the model (i.e., during data cleaning, model training, and deployment).

Fairlearn. Fairlearn is an open source project to make ML models more transparent. Various kinds of biases and other factors pertaining to unfairness exist in an ML model. Fairlearn believes that fairness cannot be introduced by technical toolkits alone. The objective of Fairlearn is to detect and reduce this unfairness, often in the form of biases.

5.6 Distributed Learning

Small ML models lack the representation capability for expressing complex data patterns. To train for large models and data volumes, the limitation of the computing power and storage of a single machine calls for distributing the ML workload across multiple machines and aggregating the results for a coherent model. We describe various technical issues to consider during

implementation of distributed models and report their impact on QoS metrics (e.g., scalability, latency, and convergence speed).

Data parallel, model parallel, and pipeline parallel. Data parallel partitions and places data onto worker nodes, with each one applying the same model for the training process. The worker nodes and central server communicate periodically to ensure that the model is synchronized across all nodes. In the model parallel approach [42, 43], the same model is split and placed on different worker nodes while processing the same copy of the dataset. The result is then aggregated from all submodels. Model parallel depends heavily on the model structure, as not all can be split up easily. For example, during training of the sequential models, only one GPU is utilized at a time, as it has to wait for results from the previous/subsequent GPU. To alleviate this problem, pipeline parallel [31] splits the data into smaller batches and pipelines them for better utilization of the available resources.

Topology. One important design consideration of distributed learning is the organization and synchronization of worker nodes within a system. There are several considerations before actual implementation. First, the system has to be scalable for large number of worker nodes so that the dispatching and aggregating of worker information are efficient and constant. Second, the system communication should be efficient and easy to set up. Parameter Server (PS) [37] is most prominent for centralized aggregation of data parallel training. In PS, worker nodes periodically upload their model parameter updates to the central server. In a decentralized setting where there is no central server, the worker nodes communicate via an AllReduce approach for exchanging model updates. Overall, the communication topology has a very significant impact on the training performance. RingAllReduce [10] usually achieves better scalability due to efficient use of network bandwidth. PS can suffer from network congestion at the server side due to the model aggregation approach.

Synchronization and asynchronization SGD. In single machine training, SGD is used for updating the parameters of a single model. In distributed learning, a global model is updated with the aggregate of all worker gradients computed with SGD. This updated model is then sent to all worker nodes. Parameter aggregation between worker nodes impacts several system metrics: loss in model training quality and convergence speed. Overall, there are three key approaches: synchronous SGD [51], stale asynchronous SGD [61], and asynchronous SGD [11]. In asynchronous SGD, the delayed gradients create noise in the global model and delay the convergence speed. However, it provides better model generalization capability and is fast in training. Synchronous training usually takes more time, as it requires waiting for idle nodes. But compared to an asynchronous approach, it often generates models that produces better performance.

Distributed learning system. Apart from the commonly used computation frameworks that are used for distributed computation, such as MapReduce and Apache Spark, there are many frameworks that are dedicated for deep learning. These frameworks include generic deep learning frameworks such as TensorFlow, PyTorch, and MXNet that incorporate different distributed learning approaches. However, some frameworks are dedicated for distributed learning, as summarized in Table 6. For example, frameworks such as PaddlePaddle include support for both sync and async training, PS, and AllReduce topology. Theano is an optimizing compiler instead of a development framework. We aim to list the trending frameworks that the users/developers could choose from, and thus Theano is not included in the table.

6 FEATURE-BASED TECHNIQUES

This section presents feature-based model explainability techniques, which describes how input features contribute to model output. There are many feature-based methods available: permutation

Table 6. Distributed Learning Frameworks

	Synchronization			Architecture	
	Sync	Async	Stale	Parameter Server	RingAllReduce
TensorFlow	●	●		●	
PyTorch	●	●			●
MXNet	●	●		●	
PaddlePaddle	●	●		●	●
Caffe2	●				●
Baidu AllReduce	●				●
Horovod	●				●
CNTK	●			●	
Distbelief		●		●	
Petuum			●	●	
DMTK	●	●		●	

Weight	Feature
0.3135	worst perimeter
0.2900	worst radius
0.1162	mean concave points
0.0679	worst concave points
0.0420	concave points error
0.0402	worst concavity
0.0226	worst texture
0.0223	worst area
0.0148	mean smoothness
0.0131	mean texture
0.0109	worst smoothness

Fig. 6. An example output of feature importance.

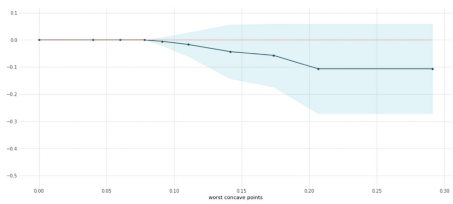

Fig. 7. An example of PDPs.

feature importance, PDPs, **individual conditional expectation (ICE)** plots, ALE plot, global surrogate models, LIME, and **Shapley additive explanations (SHAP)**.

6.1 Feature Importance

Feature importance [17, 19] refers to a class of techniques used to assign scores to input features. It shows each feature's relative importance when a prediction is made. It also defines the basis for dimensionality reduction and feature selection to improve the efficiency of a predictive model.

Permutation importance is a widely used feature importance technique that measures importance by looking at how reshuffling of each predictor randomly impacts the performance of the model. It is considered as a computationally expensive technique.

Table 6 presents a feature importance example of a cancer dataset. An example output of feature importance is a matrix (shown in Figure 6 for the cancer dataset), listing weights and features of the datasets. The topmost values indicate the features that are most important, and the numbers in the bottom represent the least important features. The randomness in our permutation importance evaluation is measured by repetition with several shuffles.

6.2 Partial Dependence Plots

PDPs depict scenarios in which a feature affects predictions. For example, the following questions can be answered with PDP [18, 25]: *What would be the impact of longitude and latitude on prices of houses? How would houses of similar size be priced in different geographic areas?* Figure 7 illustrates a simple PDP example. The x-axis plots the value of feature $f0$ (i.e., worst concave points), and the y-axis plots the predicted value. The solid line drawn in the shaded area represents the variation

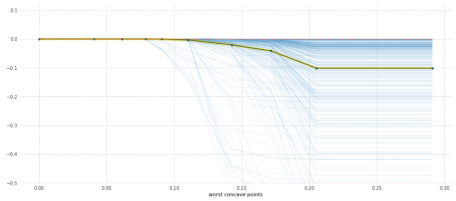

Fig. 8. An example of an ICE plot.

of average prediction when the value of $f0$ changes. The y-axis is represented as a change in the prediction from what it would be predicted as a baseline point (or the leftmost value). The blue shaded area denotes the level of confidence. In our example, the "worst concave points" feature has initially an increasing negative influence, and thereafter it remains neutral. Hence, we can say that for less values, we can initially predict that there are less chances of malignant condition.

Benefits. First, there is an ease in implementation [44]. Second, the computation part is quite intuitive when it comes to PDPs. The partial dependence function at the point of a specific feature value would represent the average prediction. It is easy for a layman to understand the logic of PDPs.

Limitations. The concern associated with PDPs is the assumption of their independence [44]. The feature(s) for which the partial dependence is computed are assumed to be not correlated with other features. Second, the maximum number of features in a partial dependence function is realistically just two. The underlying reason lies with their 2D representation (paper or screen), or lack of ability to visualize more than three dimensions. Third, heterogeneous effects can be uncovered by analyzing the ICE (outlined in Section 6.3) curves, ignoring the aggregated line.

6.3 Individual Conditional Expectation

Instead of average plotting in PDP, ICE [24] shows one line per instance. ICE scores outperform PDP on intuitiveness, as each line stands for the predictions for one instance. As with partial dependence, ICE explains what happens to model predictions when a specific feature varies. Figure 8 presents an ICE plot for feature "worst concave points" of a cancer dataset. The worst concave points plot is decreasing in nature (i.e., lower value (<0.10)) of concave points). It is a factor for a higher value of the target variable (malignant cases). Between 0.10 and 0.20, the graph decreases, and after 0.21, it is constant (has lower y value—benign cases).

Benefits. They are far more intuitive to understand when compared to PDP [44]. A single line plots the predictions for one instance if we change the feature of interest. Distinct from PDP, ICE curves can unveil heterogeneous relationships.

Limitations. First, too many ICE curves could lead to an overcrowded plot without the ability to assess anything. Second, ICE curves and PDPs share the same concern: if the interest feature correlates with the other features, then a few points in lines could be invalid data points. Third, it is difficult to find the average in ICE plots. One solution is to group together the ICE curves with the PDP.

PDP versus ICE versus feature importance. Table 7 presents a comparative analysis among PDP, ICE, and feature importance. PDP demonstrates global effects, concealing the heterogeneous effects. ICE unravels the heterogeneous effects but makes it difficult to find the average. In addition, all three methods consider the features as an independent entity. Hence, if features are correlated, it will result in the creation of unlikely data points.

Table 7. PDP Versus ICE Versus Feature Importance [53]

Approach	Advantages	Disadvantages
PDP	• Intuitive • Easy to implement • Shows global effects	• Assumption of independence • Heterogeneous effects may be hidden
ICE	• Intuitive • Easy to implement • Can uncover heterogeneous relationships	• Can only display one feature meaningfully • Assumption of independence • Not easy to see the average
Feature importance	• Provides a highly compressed global insight • Comparable across problems • Automatically takes into account all interactions	• Not additive • Shuffling the feature added randomness • Need access to true data • Assumption of independence

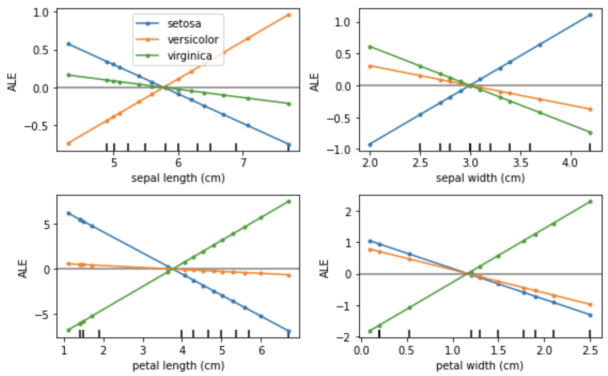

Fig. 9. ALE plots on Iris classification datasets.

6.4 ALE Plot

The ALE plot [2, 44] handles the inherent limitations of PDP. In few cases, PD plots produce erroneous results when the features of the dataset are highly correlated. This is where ALE plots come into the picture. The ALE plot works on a model-agnostic algorithm that provides global explanations for classification and regression models on tabular data. ALE plots are preferred over PD plots, as they produce optimal results despite the correlation between features and are less computationally expensive. ALE plots visualize the effect on the predictions of the model for each feature isolated from all other features.

Example. Figure 9 shows implemented ALE on the Iris classification dataset from Sklearn that has four features (`sepal length`, `sepal width`, `petal length`, `petal width`) and three target values (`setosa`, `versicolor`, `virginica`). The ALE plots visualize feature effects linearly in logit space. Looking at the plot for petal length (bottom right plot), it can be observed that the three lines overlap at the 3.8-cm mark. From that, it can be concluded that the effects of the petal length on each class will be one and the same at 3.8 cm. It can also be observed that the more the petal length is, the more is the chance of the flower being from the *Virginica* species, and the less the petal length is, the more is the chance of it being from the *Setosa* species.

6.5 Global Surrogate

A global surrogate is an interpretable model, developed for approximating black box model predictions. Figure 10 explains a simple three-step process. In step 1, the data has to be fed into the black box model for making a prediction. In step 2, the model type needs to be determined, whether it can be trained as a surrogate model—for instance, linear regression or decision trees. In step 3, the surrogate model is trained. The surrogate training is performed with the use of independent

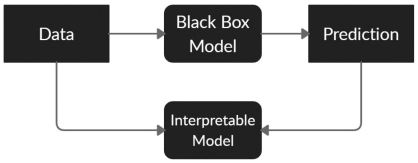

Fig. 10. A process of creating a surrogate model [53].

variables from the input data and dependent variables from the black box prediction. It is noteworthy that the surrogate model can be any interpretable model such as the linear model, decision tree, or human-defined rules. At the end, the prediction error of the surrogate model can be evaluated and compared with the black box predictions. A smaller error means that the explanation of the black box outperformed the surrogate model.

Benefits. The benefits of the surrogate model [44, 53] lie in flexibility to interchange the interpretable model and the underlying black box model. Two surrogate models (the linear model and decision tree) can be trained for the original black box model. This aids to provide two types of explanations.

Limitations. By creating a surrogate model [44, 53], we derive conclusions about an ML model, not about the data. It is possible that an interpretable model is close for one subset but broadly disparate for a different subset of a dataset. In this scenario, how the simple model is interpreted cannot be replicated across all data points. Moreover, this method would not work well to understand how a single prediction was made for a given observation.

6.6 Local Interpretable Model-Agnostic Explanations

LIME [46] is different from the global surrogate in the sense that LIME does not try to explain the whole model. By perturbing the input of data samples and comprehending how the predictions change, LIME tries to understand the model. LIME enables local model interpretability. A single data sample is modified by adjusting some feature values, and the resultant output impact is observed. This is often linked to what human interests are when the output of a model is observed.

A computer vision example. Figure 11 illustrates how LIME is used for image classification. For example, a classifier has to be explained that predicts the likelihood of the image containing an Egyptian cat. The image (Figure 11(b)) is acquired and split into easily interpretable components. As seen in Figure 11(c), a dataset of perturbed instances is generated by turning "off" (turning them gray here) some of the interpretable components. For every instance of perturbation, we compute the probability of an Egyptian cat being in the image as per the model. We then try to understand a locally weighted simple (linear) model on the dataset. The emphasis is more on erring in perturbed instances that best match the original image. Ultimately, we provide the super-pixels showing the highest positive weights as our explanation (Figure 11(d)). We distinguish it by turning everything else gray.

An example: explanations by LIME. Figure 12 illustrates the explanations generated by the LIME technique, using the ML-based price prediction model. It generates price recommendations with explanations, describing how the recommended price is derived at with explanations, instead of just predicting the price. The output of Figure 12 is a list of explanations, considering the contribution of each feature to a predicted price. The left part shows the range of a maximum (1,487.18) and minimum (240.07) value, which is predicted by the ML-based price prediction module. The middle part shows the features (i.e., WT and PPK), which contribute the most in the predicted price of an animal. It can be observed that when the weight is in a range between $308.00 < WT <= 327.00$, it

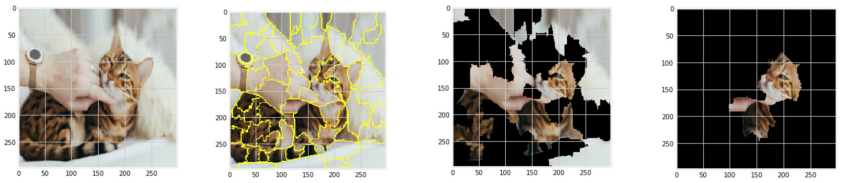

Fig. 11. Explaining a prediction with LIME. (a) Original image of Egyptian cat to be read by LIME. (b) Generated super-pixels in the image using mark boundaries. (c) Perturbed image. (d) Area of the image that produced the prediction of the Egyptian cat.

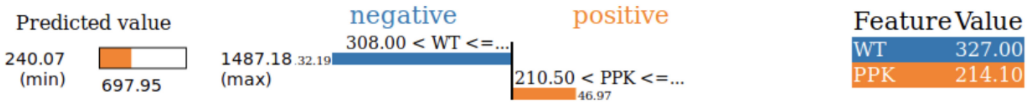

Fig. 12. Model explanations generated by LIME.

is contributing in a negative direction of the prediction. It can also be assessed that when PPK is in a range between $210.50 < PPK <= 214.10$, it is contributing to the positive side of the total price. The right part shows the actual value of a particular feature (i.e., $Weight = 327.00, PPK = 214.10$).

Potential pitfalls. Trust is important for the efficient interaction between humans and ML systems. The explanation of individual predictions is the ideal way to assess trust. Even though LIME seems superior in terms of ease in implementation and computation cost, there are a couple of potential limitations, such as the following. First and foremost, only linear models are used to approximate local behavior in the current implementation. This assumption is correct to some extent when only a small region around the data sample is considered. However, when this region is expanded, there is a possibility that the linear model might be impotent to explain the original model's behavior. There would be non-linearity at local regions for datasets requiring complicated, non-interpretable models. The inability to deploy LIME in such settings is a major pitfall.

6.7 Shapley Value

SHAP is a game-theoretic approach to explain the output of any ML model. The SHAP value [40] is a great tool similar to LIME where interpretations measures the impact of having a certain value for a given feature in comparison to the prediction. This section discusses the Shapley value and how the SHAP value arises from the concept of Shapley. It also demonstrates how SHAP values increase the transparency of the model.

An example. Figure 13 shows an output SHAP plot of an instance of a cancer dataset.[2] The *output value* is the prediction for that observation. The *base value* is the value that would be predicted if the features are unknown for the current output (i.e., mean prediction). *Features (red/blue)* that push the prediction higher (to the right) are shown in red, and those pushing the prediction lower are in blue.

The explanation in Figure 13 displays features that contribute to push the model output from the base value to the model output. The *basevalue* is the model output average over the passed training dataset. Features that push the prediction higher are color coded in red. Their size shows the extent of the effect of the feature visually. Features that push the prediction lower are depicted

[2]Breast Cancer Dataset: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic).

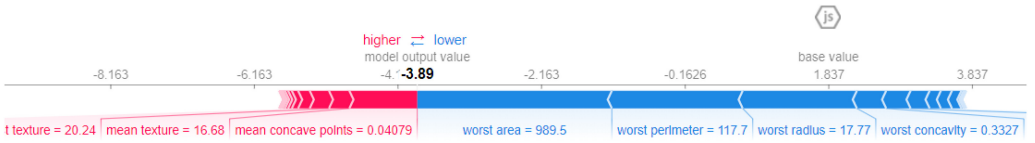

Fig. 13. Example: output of a SHAP plot.

Table 8. Programming Frameworks for Advanced XAI Techniques

Feature-Based XAI Techniques	Basic Libraries*	Keras, TensorFlow, PyTorch	Lime	Shap	Skater	eli5	PDPbox	XAI
Feature importance (Section 6.1)	✓	✓	✗	✗	✗	✓	✗	✓
Partial dependence plots (Section 6.2)	✓	✓	✗	✗	✗	✗	✓	✗
Individual conditional expectation (Section 6.3)	✓	✓	✗	✗	✗	✗	✓	✗
Global surrogate (Section 6.5)	✓	✓	✗	✗	✗	✗	✗	✗
LIME (Section 6.6)	✓	✓	✓	✗	✓	✗	✗	✗
Shapely value (Section 6.7)	✓	✓	✗	✓	✓	✗	✗	✗
Accumulated local effects plot (Section 6.4)	✓	✓	✗	✗	✗	✗	✗	✗

*includes elementary libraries such as numpy, scipy, pandas, scikit-learn etc.

in blue color. We predicted −3.89, whereas the *basevalue* is 1.837. The biggest impact comes from worst area (i.e., 989.5), although the mean concave points value has high effect (0.04079) increasing the prediction. If we subtract the length of the blue bars from the length of the pink bars, it equals the distance from the base value to the output (here, $3.89 + 3.837 - 5.89 = 1.837$).

Benefits. An important advantage of the SHAP value lies in its transparency and interpretability locally, which implies that each observation can have its own SHAP value set. It is possible to explain why an instance gets its prediction and the contributions of its predictors. Example scenarios include the following: when a model denies a loan to an individual and the bank is legally bound to explain why the loan was rejected, or a physician wants to determine the factors responsible for each patient's disease risk so that he can directly look at those risk factors with directed health interventions. The traditional variable importance algorithms provide results for the entire population ignoring individual scenarios. The local interpretability facilitates to locate and compare the effect of these factors.

Shapley value versus LIME. The Shapley value provides local and global interpretation including explanation with theoretical inferences. However, it is computationally expensive to calculate the Shapley value in comparison to LIME. This problem may be resolved with the recently developed kernel SHAP method that applies a fast kernel approximation. However, crunching large background data still incurs high computational costs.

LIME is an optimal alternative for few models such as the KNN model in terms of computation cost. However, it fails to work out-of-the-box on all models. For example, LIME cannot handle the requirement of XGBoost to use xgb.DMatrix() on the input data. On the contrary, SHAP provides fast, reliable evaluations and incorporates Tree Explainer, which optimally estimates Shapley values through XGBoost.

6.8 Software Libraries for Feature-Based XAI Techniques

In the following, we present a few software libraries that can be used to implement feature-based XAI techniques. Table 8 describes frameworks that can be used to implement the following techniques.

ELI5.[3] ELI5 is an abbreviated form of *Explain Like I'm 5*—a popular Python package that provides explanation and a visualization of the predictions of an ML model, and aids in debugging an ML classifier. There are two primary ways to assess how an ML model works using ELI5. First, *Global* ELI5 provides the method `show_weights()` to explain how parameters are acting with respect to the entire model. Second, *Local* ELI5 provides `show_prediction()` to look at a specific instance of a prediction, and an explanation for the model prediction for that instance.

SHAP.[4] SHAP is a popular library used for model explainability, based on Shapley values that measure the influence of various features (i.e., the contribution of every feature of the dataset toward the prediction). It is capable of visualizing both *local* and *global interpretations*. Local explanation consists of why prediction on an individual instance was made, and global explanation provides a summary of feature importance over the entire dataset. SHAP offers a method to visualize the overall data patterns and understand the model in a global context. SHAP has a number of *explainers*: deep (based on the DeepLIFT algorithm [40, 52]), gradient, kernel (to estimate SHAP for regression and classification models), linear (to compute the SHAP values for a linear model with independent features), tree (to calculate SHAP values for decision tree models), and sampling (computes SHAP values by using random permutation of features). SHAP is a robust method that provides the integration of several methods, such as feature importance, feature dependence, interactions, clustering, and summary plots, all included in a single library. SHAP is computationally expensive on certain models such as KNN but runs fast on trees, such as gradient-boosted trees from XGBoost.

XAI.[5] XAI is an explainability toolbox for ML that is specifically designed for data analysis and model evaluation as follows. With regard to data analysis, it facilitates the user to balance the class by up-sampling or down-sampling and thereby splitting a testing and training dataset. It has the ability of visualizing the correlation matrix, thereby explaining the model's behavior. With regard to model evaluation, the interaction between the predictions and the input features can be analyzed by building a deep learning model.

PDPbox.[6] The PDP toolbox, abbreviated as PDPbox, is similar to ICEbox. It is used to compute and visualize the impact or effect of features on the prediction of target variable for any scikit-learn algorithm, thereby explaining the prediction of the model. This library is one step ahead of random forest, as PDPbox provides the direction in which the feature is influencing the prediction.

Skater.[7] Skater is a Python library used for interpreting and identifying relationships between data or features that act as input to the model and the final prediction the model makes. It is used to reveal the interpretations of black box models globally as well as locally. With Skater, one can perform global interpretations using PDPs and feature importance techniques, such as how a loan prediction model uses the customer's credit history, account status, and income to approve or deny his loan. Skater can also measure how a model's performance alters over time after its deployment in a production environment. Skater has the ability to interpret, allowing the practitioner to measure how feature interactions vary across different versions of models.

Tf-explain.[8] Tf-explain is a library used for Keras API (TensorFlow v2.0 workflow), and primary features include tensorboard integration and callbacks. It supports visualization of heatmaps and

[3]ELI5 documentation: https://eli5.readthedocs.io/.
[4]Shap documentation: https://github.com/slundberg/shap.
[5]An eXplainability Toolbox for Machine Learning: https://github.com/EthicalML/xai.
[6]Python partial dependence plot toolbox: https://pdpbox.readthedocs.io/.
[7]Python library for model interpretation/explanations: https://oracle.github.io/Skater/.
[8]Interpretability methods for TensorFlow 2.0: https://tf-explain.readthedocs.io/.

```
Person has heart disease
Anchor generated feature(/s) ['thalach <= 138.00', 'ca > 1.00']
```

Fig. 14. Example: an output of an anchor.

gradients for hyperparameter tuning or confusion matrix generation, for explanation and visualization of the prediction. The methods include GradCAM [50] and SmoothGrad [54].

7 EXAMPLE-BASED TECHNIQUES

Sections 7.1 through 7.5 describe commonly used example-based XAI techniques. Section 7.6 presents programming frameworks to implement these techniques and their comparative analysis.

7.1 Anchors

Anchors capture limit/sufficient conditions of features at which the model gives a high-precision prediction. Anchors support model-agnostic approaches for classification models of text, image, or tabular data. The anchor takes into account the global dataset and then gives the anchor feature values, using `If-Then` rules for finding features associated with input instances responsible for prediction. Anchors are similar to LIME, as they both provide local explanations linearly. However, LIME only covers a local region and may not be generalizable. If the same perturbation space is allotted to both LIME and Anchors, the latter will build a more valid region of instances that better describe the model's behavior.

Example. Consider an example of the Heart Disease dataset, with a feature instance suggesting heart disease for an individual. The value of `thalach` of this person is 131 and `ca` is 3, and therefore the AnchorTabular method reaches the following conclusions (Figure 14). Further, we apply the `AnchorTabular` method to assess which features contribute significantly for such type of prediction.

The person's maximum heart rate is 131 (which is less than 138). The blood vessels colored by `fluoroscopy` are 3 (greater than 1). As the maximum heart rate of a person should be high and the blood vessels colored by `fluoroscopy` should be low, the preceding features act as Anchors for the patient and deduce that the person has heart disease.

7.2 Counterfactual Explanations

Determining *what should be the change in features to switch prediction* when an ML-based model is applied to real-world data (along with the rationale behind its outcome) is important. Counterfactual explanations is a model-agnostic XAI technique that provides changes that can be made to feature values to change the current output to a predefined output. The counterfactual explainer method works on black box models and is best suited for binary datasets. It can also be applied on classification with more than three target values, but the performance degrades as compared to binary classification.

Heart Disease dataset example. A value of `condition` field 0 or 1 signifies the presence or absence of heart disease. From this dataset, we have considered a specific instance where the patient has heart disease. Figure 15 shows this instance. Using input in Figure 15, we generate four different counterfactuals as shown in Figure 16. All of these exhibit the minimum changes to the feature values to change the condition of a patient.

The following are the observations regarding the output. First, the `sex`, `age`, or the type of chest pain (field `cp`) of a person suffering from heart disease cannot be changed. Therefore, these features have to be fixed in each of the counterfactuals. Second, the four different counterfactuals are all different schemes to change the target value from 1 to 0. For example, the second counterfactual on

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	condition
0	67.0	1.0	3.0	120.0	229.0	0.0	2.0	129.0	1.0	2.6	1.0	2.0	2.0	0.99965

Fig. 15. An input to a counterfactual—a specific instance where the patient has heart disease.

Diverse Counterfactual set (new outcome : 0)

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	condition
0	67.0	1.0	3.0	94.0	229.0	1.0	2.0	129.0	0.0	2.6	1.0	0.0	0.0	0.283
1	67.0	1.0	3.0	120.0	186.0	1.0	1.0	124.0	0.0	2.6	2.0	0.0	0.0	0.159
2	67.0	1.0	3.0	120.0	229.0	1.0	1.0	172.0	0.0	3.1	1.0	0.0	2.0	0.315
3	67.0	1.0	3.0	120.0	229.0	0.0	1.0	129.0	0.0	0.6	1.0	0.0	0.0	0.274

Fig. 16. Different counterfactual outputs of the counterfactual explainer.

the list indicates that reduction of cholesterol leads to a decrease in the intensity of heart disease. It also shows that there should be normal results and no defects upon performing the thallium test on the heart. Third, a recurring theme in all counterfactuals is the reduction of `ca` from 2 to `0`; `ca` signifies the number of blocked vessels of the heart—`ca` is the most important feature contributing to having heart disease. Hence, the most important factor in changing the condition is to reduce the number of blocked vessels by using methods like angioplasty.

Counterfactuals guided by prototypes. This refers to explanations described on the basis of a prototype (i.e., using a representative sample of instances belonging to a class). Counterfactuals guided by prototypes is more advanced and accurate, and works on black box models. This method is a model-agnostic approach to interpret results using the prototypes of classes of the target variable. It is much faster than counterfactual, as prototypes speed up the search process significantly by directing the counterfactual to the prototype of a particular class.

7.3 Contrastive Explanations Method

The **contrastive explanations method (CEM)** is an XAI method for generating local explanations on a black box model. CEM defines explanations for classification models by providing insight on preferable features along with unwanted features, such as pertinent positives and pertinent negatives. CEM is the first method that provides the explanations of both what should be minimally present and what should be necessarily absent from the instance to be explained to maintain the original prediction class. Further, it also identifies a minimal set of features that is adequate to differentiate it from the nearest different class. Using CEM, the accuracy of the ML model can be enhanced by looking at cases of mis-classified instances and using the explanations provided by CEM.

The two categories of explanations include pertinent positives and pertinent negatives [12]. The pertinent positives explanation finds the features that are necessary for the model to predict the same output class as the predicted class. For example, this includes the important pixels of an image and the features having a high feature weight, among others. Pertinent positives works similarly to anchors. The pertinent negatives explanation finds the features that should minimally and sufficiently be removed from an instance while maintaining the original output class. Pertinent negatives works similarly to counterfactuals.

An example using the Heart Disease dataset. Figure 17 generates counter explanations in terms of pertinent negatives. The original prediction was 0, which is altered to 1 after applying CEM with pertinent negatives. Pertinent negative explanations work similarly to the counterfactual explanations, as describe previously. The CEM values in the array that are different from the original one

```
Feature names: ['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach', 'exang', 'oldpeak', 'slope', 'ca', 'thal']
Original instance: [[-1.3859065  -1.44454157 -0.16401266 -0.65831955 -0.73753753 -0.41075703
  -1.00172798  1.02001221 -0.695246   -0.90518389  0.64269638 -0.72075958
  -0.87281841]]
Predicted class: [0]
Pertinent negative: [[-1.4474772  -1.4445416   0.24188966 -0.65831953 -0.7375375  -0.41075704
  -1.0017279   1.0200123  -0.5739004  -0.9051839   0.6426964   1.7650422
   0.5487773 ]]
Predicted class: [1]
```

Fig. 17. An output by CEM using the Heart Disease dataset.

Fig. 18. An example of local explanation using the Heart Disease dataset.

change the prediction class. Some of them are cp, ca, and thal. Hence, changes in these features should necessarily be eliminated to retain the original prediction as 0, as they are responsible for flipping the prediction class.

7.4 Kernel and Tree SHAP

The goal of SHAP is to calculate the impact of every feature on the prediction [22, 40]. Compared to Shapley values, Kernel SHAP provides computational efficiency and accurate approximation in higher dimensions. In Kernel SHAP, the full model has been utilized that is already trained instead of retraining models with different feature permutations. Here, the "missing features" are replaced with "samples from the data." This means that the "absent feature values" are equated with "feature value replaced by random feature values selected from data." Now, this modified feature space is fitted to the linear model and the coefficients of this model act as Shapley values.

Local explanation. An example of local explanation using the Heart Disease dataset is illustrated in Figure 18. The base value is the average of all output values of the model on the training data (here, −0.3148). Pink values drag/push the prediction toward 1 (pushes the prediction higher, i.e., toward having heart disease) and the blue toward 0 (pushes the prediction lower, i.e., toward no disease). The magnitude of influence is determined by the length of the features on the horizontal line. The values shown corresponding to the feature are the values of the feature at the particular index (e.g., 2.583 for ca). Here, the highest influence is of ca for increasing the prediction value and of sex for decreasing the value.

Global explanation. Figure 19 plots the impact of features on the prediction class 1. The features are arranged such that the highest influence is of the topmost feature. Thus, ca is the feature that influences prediction the most, followed by thal and so on. The color shades depict the direction in which the feature impacts the prediction. For example, higher SHAP values of ca are shown in red color, which means a high feature value. The higher the value of ca, the higher is the SHAP value (i.e., more toward 1). A high value of ca indicates more chances of heart disease. However, it is the opposite for some features: a high thalach will indicate no heart disease.

Tree SHAP. Tree SHAP is an algorithm to compute exact SHAP values for decision tree based models. The algorithm provides interpretable explanations suitable for regression and classification of models with tree structure applied to tabular data [22, 39]. It attributes the change of a model output with respect to a baseline (e.g., average over a reference set or inferred from node data) to each of the input features. Similar to Kernel SHAP, the Shapley value of each feature is

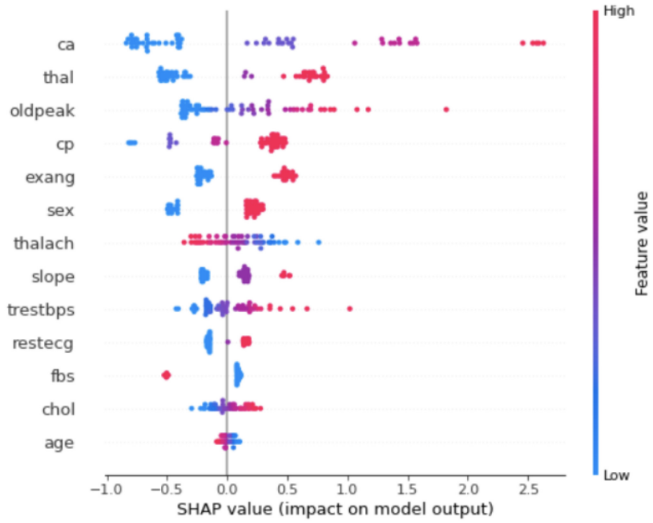

Fig. 19. An example of global explanation using the Heart Disease dataset.

computed by averaging the difference of the model output observed when the feature is part of a group of "present" features.

7.5 Integrated Gradients

Integrated gradients, also known as path-integrated gradients, is an XAI technique that assigns an importance value to each feature of the input using the gradients of the model output [57]. It is a local method that helps explain each individual prediction. This method provides the specific attributions of the feature inputs that are positive attributions and negative attributions. Positive attributions are attributions that contribute or influence a model to make the decision, whereas negative attributions are attributions that contribute or influence a model against a decision made.

An example. Let us understand this more using an example with the MNIST dataset, as presented in positive attributions and negative attributions. Consider an example of the Fashion MNIST dataset, which consists of 70,000 grayscale 28×28 images of different clothing items. The label consists of 10 classes, which denote different kinds of clothing items, such as shirts, hoodies, shoes, and jeans.

The first example (top part of Figure 20) is an image of a shoe. The attributions section shows a melange of positive and negative attributions together. It can be observed from the bar on the right side that green signifies positive attributions and purple signifies negative attributions. The shoe is unique compared to other clothing items, and hence it has a lot more positive attributes than negative ones. The lining, collar, and back part of the shoe are the main pixels that influence the decision of the model. However, the negative attributions are negligible for this particular instance. The second example (bottom part of Figure 20) is an image of a shirt where there is an equal number of positive and negative attributions. The pixels around the collar and the sleeves are the biggest positive attributions. However, the middle portion of the shirt can be mistaken to be a part of a pair of jeans or trousers. Therefore, due to this ambiguity, they are the negative attributions for the prediction. All in all, we can affirm that when the positive attributions outweigh the negative attributions, the model makes the correct prediction.

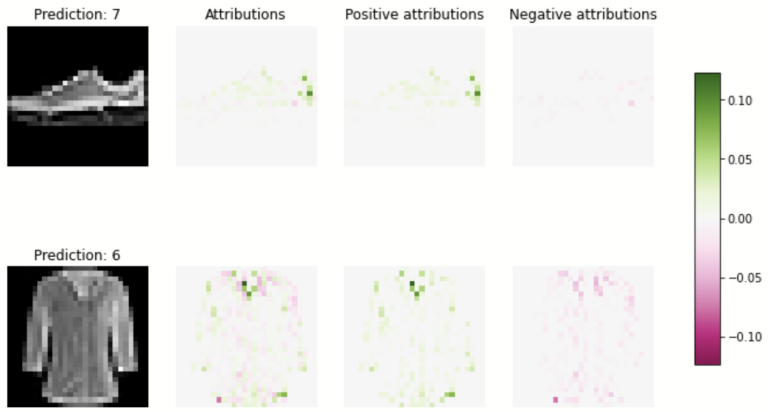

Fig. 20. An example of integrated gradients using the Fashion MNIST dataset.

Table 9. Programming Frameworks for Example-Based XAI Techniques

Example-Based XAI Techniques	Basic Libraries*	Keras, TensorFlow, PyTorch	DiCE	Alibi	Tf-explain
Anchors (Section 7.1)	✓	✓	✗	✓	✗
Counterfactuals (Section 7.2)	✓	✓	✓	✓	✗
Prototype counterfactuals (Section 7.2)	✓	✓	✗	✓	✗
Contrastive explanations method (Section 7.3)	✓	✓	✗	✓	✗
Kernel SHAP (Section 7.4)	✓	✓	✗	✓	✗
Tree SHAP (Section 7.4)	✓	✓	✗	✓	✗
Integrated gradients (Section 7.5)	✓	✓	✗	✓	✓

*includes elementary libraries such as numpy, scipy, pandas, scikit-learn etc.

7.6 Software Libraries for Example-Based XAI Techniques

In the following, we present some of the software libraries that can be used to implement example-based XAI techniques. Table 9 describes frameworks that can be used to implement these techniques.

Diverse counterfactual explanations for ML.[9] **Diverse counterfactual explanations (DiCE)** [45] is a package that gives the "what-if" explanations for the model prediction. The ML models perform prediction based on the values/features present in the data. However, in practical situations, this might not be enough. It is important to know the answer to the question, *What should be the modifications in features so that the prediction flips?* DiCE not only provides the most influential feature for the prediction but also recommends the feature modifications needed for the result. It implements counterfactual explanations that include perturbed features that in turn will lead to the required result. In addition, if some features are difficult to modify (e.g., the financial status is difficult to change than working hours per week), DiCE allows input of relative difficulty by specifying "feature weights." A higher feature weight means that the feature is difficult to modify than that of others.

ALIBI. Alibi [34] is an open source Python library that aims at ML-based model inspection and interpretation. The library consists of a wide range of algorithms, most of which focus on black box models and local interpretation (i.e., interpretation for the prediction to be explained at a particular instance). This library comprises different types of explainers depending on data we are dealing with.

[9]DiCE for ML: https://github.com/interpretml/DiCE.

8 SOFTWARE TOOLKITS

This section presents toolkits available for building XAI applications. Unlike the previous sections (Sections 4–7), we consider the candidates for a toolkit, which include an extensive set of tools and techniques (instead of focusing on one single aspect of XAI), to cover different aspects of XAI techniques such as visualization tools and tools to debug and evaluate ML models, among others. In the following, we present some of the toolkits available both commercially and open source.

8.1 What-If Tool

What-If Tool (WIT)[10] is an attribute of Google's open source TensorBoard web application. It is a user-friendly tool that has the ability to debug and evaluate ML models effectively. This tool aids to understand a model in a simple and intuitive way through a visual interface. It works on both classification and regression models. The prominent feature of WIT is that it allows everyone from ML researchers and developers to non-technical stakeholders for extensive use, as it is free of complex coding. It also provides answers to different what-if scenarios and aids to visualize and explore the counterfactual examples. For example, in a classification model, it returns the instance with the most similar features but different prediction. Thus, it enables and simplifies the task to assess the modifications in model in real time. One of the other features is its visualization of the dataset, which inculcates how diverse is the data, impact on the model's results on altering different feature values, how the hyperparameters should be tuned, and other observations. In addition, it can be beneficial in comparing the results of two different models on the same input data. It is also capable of post-training evaluation.

WIT has three tabs with different features: *Datapoint Editor*, *Performance and Fairness*, and *Feature*: The Datapoint Editor shows the prediction for every datapoint passed. Using Datapoint Editor, we are also able to inspect individual input points and create custom visualizations by changing the feature values. Performance and Fairness provides the overall performance using evaluation metrics such as the confusion matrix and ROC curve. It provides a way to slice data of different features and then applies different strategies for accuracy and fairness enhancement. The Features tab shows the balance in the dataset for every feature by assigning the range of every feature prior to the training.

8.2 TensorBoard

TensorBoard is a visualization tool that helps in inspecting the inner workings of the model, as training a deep neural network can be complex and difficult to comprehend. It is used for measuring the evaluation metrics like loss and accuracy needed during the ML workflow and projecting embeddings to a lower-dimensional space. TensorBoard is also capable of debugging and optimizing TensorFlow programs.

All TensorFlow programs have two basic components: *Operations* and *Tensors*. Tensors are values of a multidimensional array (i.e., data is stored in tensors and operations manipulate the stored data). The data is fed into the model, which consists of a set of operations, and tensors flow between the operations to get the output tensor. The current implementation of TensorBoard allows five visualizations: *scalars*, *images*, *audio*, *histograms*, and *graphs*: TensorBoard's *Scalar Dashboard* visualizes scalar valued tensors that vary with time similar to loss or learning rate. The *Image Dashboard* can display saved images. It can also be used to build an image classifier on arbitrary image data, as it is able to display input images of a network, generated output images of an autoencoder or a GAN. *Audio* enables saving audio that can be embedded and played in the Audio Dashboard using audio widgets. The *Histogram Dashboard* visualizes the distribution of a

[10] https://pair-code.github.io/what-if-tool/.

non-scalar Tensor over different periods of time. It can be used to visualize weight or bias matrices of a neural network. The *Graph Explorer* can visualize a TensorBoard graph, allowing understanding of the TensorFlow model and its operations.

Tensorboard also provides an *Embedding Projector*, which aids in visualizing high-dimensional data and examining embedding layers. For example, in sentiment analysis, it allows searching for specific terms and highlights words that are nearby in the embedding space. It requires a checkpoint through which it reads the input data as well as metadata such as vocabulary files or additional information about the layer to visualize. Another component of Tensorboard is *summary*, a special operation that is required to visualize the model parameters such as weights and biases of a neural network, evaluation metrics, and images such as input images to a network. It feeds in a regular tensor and outputs the summarized data to the disk.

8.3 InterpretML

InterpretML is an open source Python toolkit by Microsoft for training intelligible models and explaining black box systems. It is based on human interpretation of the global and local explanations of the model. It also simplifies debugging the model to be able to understand the predictions. InterpretML has the ability of comparing different XAI methods using the same feature and optimizes real-life datasets. The different XAI methods include the decision tree, decision rule list, linear regression, logistic regression, SHAP Kernel Explainer, SHAP Tree Explainer, LIME, Morris sensitivity analysis, and partial dependence. It includes Explainable Boosting Machine, which is an algorithm for explaining glass box models with higher accuracy. There are currently three features available: (1) *tabular data interpretability*: what-if based interactive visualization is available in the toolkit, where a user can assess the modifications for features of a particular data point in the model's predictions; (2) *interpretability for text data*: a text-specific visualization dashboard is also available for a text classification and sentimental analysis; and (3) *counterfactual example analysis using DiCE*: the demo analysis is also available where it recommends the required modifications to the input features so that the model yields the desired output.

Other features of the dashboard include filtering the data and creating cohorts. A Model performance tab visualizes model performance metrics, as well as the distribution of rejection probability. Overall, model explanations show techniques like feature importance and individual feature importance factors.

9 DESIGN CONSIDERATIONS FOR IMPLEMENTING XAI

Enhancing explainability into AI systems can bring in many positives. However, the implementation of XAI is not the easiest of tasks. In the following, we discuss the considerations while implementing XAI models.

Trade-off: XAI versus model performance. While applying XAI concepts to models, we may have to choose between model *interpretability* and model *performance* [26]. For example, few simple linear or tree-based models can easily explain the model decisions that lead to a specific insight or prediction. Conversely, complex ensemble and deep learning models often produce superlative performance, but they are considered as black box models, as it is quite difficult to assess the model's decisions.

Different users need different forms of explanation in different situations. A decision or recommendation may have to be explained in many ways, depending on audience requirements and the factors in different scenarios [55]. For understanding the system functioning, users may have questions, such as what kind of data was used by the system, the origin of such data, and the reason that data was chosen; how does the model work and what factors impact the decision-making process; and why a specific result was achieved. To figure out what kind of

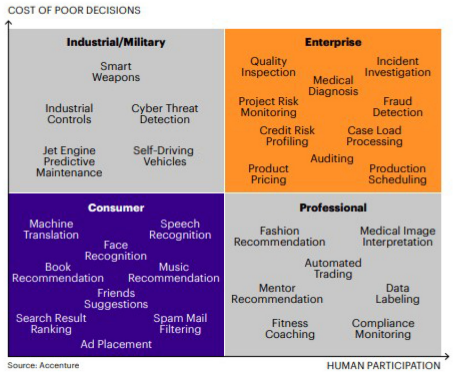

Fig. 21. XAI: cost of poor decisions versus human participation [36].

explanations are required, it is necessary to engage with all stakeholders and build a robust system design.

System design often needs to balance competing requirements. XAI technologies utilize various schemes and approaches, each with its own pros and cons. These methods when used in different applications, the interpretability, accuracy, and privacy varies. For instance, in healthcare and finance applications, AI systems are privy to confidential data for making decisions and offering recommendations. Explainability in such cases differs as businesses must keep in mind to which extent they need to be transparent. This puts a question mark on the suitability of such systems for applications where the knowledge of the decision-making process is vital for purposes of general acceptance and overall accountability.

The quality of data is part of the XAI. Modern AI methods rely on huge volumes of data for making predictions and decisions [55]. Being aware of data quantity and its *provenance* in AI systems ensures system explainability. For instance, image data might be biased against minorities, social media data might be restricted to a particular demographic, or city sensor data might only represent a particular neighborhood.

Explainability may not always be the priority while designing an AI system. Explainability needs must be looked at within the overall objectives of the system. Figure 21 shows various AI applications based on human participation and cost of poor decisions. The extent of requisite explainability differs from one experiment to another. In the example of shopping recommendations, the customer may not want an explanation for the items recommended. However, in situations where the ML model is utilized for making crucial decisions, explainability is paramount. The bottom left quadrant represents the most successful AI use cases, where both potential cost and human participation are low. Decisions listed in the top right quadrant, such as for credit risk profiling and medical diagnosis, represent exponentially increasing cost. The top left quadrant features decisions where any errors can result in disastrous consequences. If AI cannot explain itself in these domains, then its risk of making a wrong decision may override its accuracy and decision-making efficacy.

10 CONCLUSION

In this survey, we have covered a variety of XAI techniques currently in use, ranging from white box models, such as linear models, decision trees, and GAMs, which are ML-based models. The techniques that are intrinsically more explainable and interpretable as compared to their black box counterparts to feature-based techniques such as LIME, SHAP, and global surrogate models, in addition to plots such as PDP, ICE, and ALE, have been presented along with their role in the

progress and development of AI. In essence, this survey seeks to facilitate structured and acute information for the coupled researchers. We have explored our discussions beyond what has been gone so far in the XAI domain toward the idea of distributed AI, a paradigm that imposes storage of data on multiple nodes when implementing AI models in practice, including privacy, transparency, and fairness. We have also investigated the implications of espousing XAI techniques in the context of different applications such as livestock mart and healthcare, unveiling the potential of XAI to compromise the privacy of a user.

Our cogitations toward the future of XAI, articulated in the contrasting discussions carried out throughout this work, capitulate on the cogent need for apt understanding of the capability and limitations emerged up by XAI techniques. It is our prevision that model interpretability must be considered in conjunction with constraints and requirements associated with data explainability, model explainability, fairness, and accountability. A progenitive implementation and deployment of AI schemes in institutions and organizations worldwide will be only assured upon disquisition of all AI axioms jointly.

REFERENCES

[1] Plamen Angelov and Eduardo Soares. 2020. Towards explainable deep neural networks (xDNN). *Neural Networks* 130 (2020), 185–194. https://doi.org/10.1016/j.neunet.2020.07.010

[2] Daniel W. Apley and Jingyu Zhu. 2016. Visualizing the effects of predictor variables in black box supervised learning models. *arXiv preprint arXiv:1612.08468*.

[3] Sebastian Bader and Pascal Hitzler. 2005. Dimensions of neural-symbolic integration— structured survey. In *We Will Show Them! Essays in Honour of Dov Gabbay, Volume One*, Sergei N. Artëmov, Howard Barringer, Artur S. d'Avila Garcez, Luís C. Lamb, and John Woods (Eds.). College Publications, 167–194.

[4] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador Garcia, et al. 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

[5] David Beckett, Tim Berners-Lee, Eric Prud'hommeaux, and Gavin Carothers. 2014. RDF 1.1 Turtle. *World Wide Web Consortium* 2014 (2014), 18–31.

[6] Tarek R. Besold, Artur S. d'Avila Garcez, Sebastian Bader, Howard Bowman, Pedro M. Domingos, Pascal Hitzler, Kai-Uwe Kühnberger, et al. 2017. Neural-symbolic learning and reasoning: A survey and interpretation. *CoRR* abs/1711.03902.

[7] Umang Bhatt, Alice Xiang, Shubham Sharma, Adrian Weller, Ankur Taly, Yunhan Jia, Joydeep Ghosh, Ruchir Puri, José M. F. Moura, and Peter Eckersley. 2020. Explainable machine learning in deployment. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*. ACM, New York, NY, 648–657. https://doi.org/10.1145/3351095.3375624

[8] Lorenz Bühmann, Jens Lehmann, and Patrick Westphal. 2016. DL-Learner—A framework for inductive learning on the semantic web. *Journal of Web Semantics* 39 (2016), 15–24.

[9] Diogo V. Carvalho, Eduardo M. Pereira, and Jaime S. Cardoso. 2019. Machine learning interpretability: A survey on methods and metrics. *Electronics* 8, 8 (2019), 832.

[10] NVIDIA Corporation. 2015. NVIDIA Collective Communications Library (NCCL). Retrieved July 15, 2021 from https://developer.nvidia.com/nccl.

[11] Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, et al. 2012. Large scale distributed deep networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems (NIPS'12)*. 1223–1231.

[12] Amit Dhurandhar, Pin-Yu Chen, Ronny Luss, Chun-Chen Tu, Paishun Ting, Karthikeyan Shanmugam, and Payel Das. 2018. Explanations based on the missing: Towards contrastive explanations with pertinent negatives. In *Advances in Neural Information Processing Systems*. 592–603.

[13] Derek Doran, Sarah Schulz, and Tarek R. Besold. 2017. What does explainable AI really mean? A new conceptualization of perspectives. In *Proceedings of the First International Workshop on Comprehensibility and Explanation in AI and ML 2017 co-located with 16th International Conference of the Italian Association for Artificial Intelligence (AI*IA 2017), Bari, Italy, November 16th and 17th, 2017 (CEUR Workshop Proceedings, Vol. 2071)*, Tarek R. Besold and Oliver Kutz (Eds.). CEUR-WS.org. http://ceur-ws.org/Vol-2071/CExAIIA_2017_paper_2.pdf.

[14] Finale Doshi-Velez and Been Kim. 2017. Towards a rigorous science of interpretable machine learning. *arXiv:1702.08608* [stat.ML].

[15] Filip Karlo Došilović, Mario Brčić, and Nikica Hlupić. 2018. Explainable artificial intelligence: A survey. In *Proceedings of the 2018 41st International Convention on Information and Communication Technology, Electronics, and Microelectronics (MIPRO'18)*. 0210–0215. https://doi.org/10.23919/MIPRO.2018.8400040

[16] Basil Ell, Andreas Harth, and Elena Simperl. 2014. SPARQL query verbalization for explaining semantic search engine queries. In *Proceedings of the European Semantic Web Conference*. 426–441.

[17] Aaron Fisher, Cynthia Rudin, and Francesca Dominici. 2018. Model class reliance: Variable importance measures for any machine learning model class, from the "Rashomon" perspective. *arXiv preprint arXiv:1801.01489*.

[18] Jerome H. Friedman. 2001. Greedy function approximation: A gradient boosting machine. *Annals of Statistics* 29, 5 (2001), 1189–1232.

[19] Jerome H. Friedman and Bogdan E. Popescu. 2008. Predictive learning via rule ensembles. *Annals of Applied Statistics* 2, 3 (2008), 916–954.

[20] Giuseppe Futia and Antonio Vetrò. 2020. On the integration of knowledge graphs into deep learning models for a more comprehensible AI—Three challenges for future research. *Information* 11, 2 (2020), 122.

[21] Krishna Gade, Sahin Cem Geyik, Krishnaram Kenthapadi, Varun Mithal, and Ankur Taly. 2019. Explainable AI in industry. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'19)*. ACM, New York, NY, 3203–3204. https://doi.org/10.1145/3292500.3332281

[22] María Vega García and José L. Aznarte. 2020. Shapley additive explanations for NO2 forecasting. *Ecological Informatics* 56 (2020), 101039.

[23] Christine Golbreich, Evan K. Wallace, and Peter F. Patel-Schneider. 2009. OWL 2 Web Ontology Language New Features and Rationale. W3C Working Draft 11 June 2009. Retrieved September 9, 2022 from http://www.w3.org/TR/2009/WD-owl2-new-features-20090611/.

[24] 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.

[25] Brandon M. Greenwell. 2017. pdp: An R package for constructing partial dependence plots. *R Journal* 9, 1 (2017), 421–436.

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

[27] Xin He, Kaiyong Zhao, and Xiaowen Chu. 2019. AutoML: A survey of the state-of-the-art. *arXiv:1908.00709* [cs.LG].

[28] Pascal Hitzler, Federico Bianchi, Monireh Ebrahimi, and Kamruzzaman Sarker. 2020. Neural-symbolic integration and the semantic web. *Semantic Web* 11, 1 (2020), 3–11.

[29] Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, and Sebastian Rudolph. 2009. OWL 2 web ontology language primer. *W3C Recommendation* 27, 1 (2009), 123.

[30] Andreas Holzinger, Chris Biemann, Constantinos S. Pattichis, and Douglas B. Kell. 2017. What do we need to build explainable AI systems for the medical domain? *arXiv preprint arXiv:1712.09923*.

[31] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, et al. 2019. GPipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in Neural Information Processing Systems* 32 (2019), 103–112.

[32] SNOMED International. 2022. Data Analytics with SNOMED CT. Retrieved June 29, 2022 from https://confluence.ihtsdotools.org/display/DOCANLYT/Data+Analytics+with+SNOMED+CT.

[33] Bahador Khaleghi. 2020. An Explanation of What, Why, and How of eXplainable AI (XAI). Retrieved June 9, 2022 from https://towardsdatascience.com/an-explanation-of-what-why-and-how-of-explainable-ai-xai-117d9c441265.

[34] Janis Klaise, Arnaud Van Looveren, Giovanni Vacanti, and Alexandru Coca. 2019. Alibi: Algorithms for Monitoring and Explaining Machine Learning Models. Retrieved September 9, 2022 from https://github.com/SeldonIO/alibi.

[35] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. 2016. Adversarial machine learning at scale. *CoRR* abs/1611.01236.

[36] Accenture Labs. 2020. Understanding Machines: Explainable AI. Technical Report. Retrieved September 9, 2022 from https://www.accenture.com/_acnmedia/pdf-85/accenture-understanding-machines-explainable-ai.pdf.

[37] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. 2014. Scaling distributed machine learning with the parameter server. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI'14)*. 583–598.

[38] Joanne S. Luciano, Bosse Andersson, Colin Batchelor, Olivier Bodenreider, Tim Clark, Christine K. Denney, Christopher Domarew, et al. 2011. The Translational Medicine Ontology and Knowledge Base: Driving personalized medicine by bridging the gap between bench and bedside. *Journal of Biomedical Semantics* 2 (2011), 1–21.

[39] Scott M. Lundberg, Gabriel Erion, Hugh Chen, Alex DeGrave, Jordan M. Prutkin, Bala Nair, Ronit Katz, Jonathan Himmelfarb, Nisha Bansal, and Su-In Lee. 2020. From local explanations to global understanding with explainable AI for trees. *Nature Machine Intelligence* 2, 1 (2020), 2522–5839.

[40] Scott M. Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*. 4765–4774.
[41] Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B. Tenenbaum, and Jiajun Wu. 2019. The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. *arXiv preprint arXiv:1904.12584*.
[42] Azalia Mirhoseini, Anna Goldie, Hieu Pham, Benoit Steiner, Quoc V. Le, and Jeff Dean. 2018. A hierarchical model for device placement. In *Proceedings of the International Conference on Learning Representations*.
[43] Azalia Mirhoseini, Hieu Pham, Quoc V. Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, and Jeff Dean. 2017. Device placement optimization with reinforcement learning. In *Proceedings of the International Conference on Machine Learning*. 2430–2439.
[44] Christoph Molnar. 2019. Interpretable Machine Learning. Retrieved September 9, 2022 from https://christophm.github.io/interpretable-ml-book/.
[45] Ramaravind Kommiya Mothilal, Amit Sharma, and Chenhao Tan. 2019. Explaining machine learning classifiers through diverse counterfactual explanations. *CoRR* abs/1905.07697.
[46] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2018. Anchors: High-precision model-agnostic explanations. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*. 1527–1535.
[47] Kamruzzaman Sarker, Ning Xie, Derek Doran, Michael L. Raymer, and Pascal Hitzler. 2017. Explaining trained neural networks with semantic web technologies: First steps. In *Proceedings of the Twelfth International Workshop on Neural-Symbolic Learning and Reasoning, NeSy 2017, London, UK, July 17-18, 2017 (CEUR Workshop Proceedings, Vol. 2003)*, Tarek R. Besold, Artur S. d'Avila Garcez, and Isaac Noble (Eds.). CEUR-WS.org. http://ceur-ws.org/Vol-2003/NeSy17_paper4.pdf.
[48] Kamruzzaman Sarker, Lu Zhou, Aaron Eberhart, and Pascal Hitzler. 2021. Neuro-symbolic artificial intelligence: Current trends. *arXiv preprint arXiv:2105.05330*.
[49] Arne Seeliger, Matthias Pfaff, and Helmut Krcmar. 2019. Semantic web technologies for explainable machine learning models: A literature review. *PROFILES/SEMEX@ ISWC* 2465 (2019), 1–16.
[50] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. 2017. Grad-CAM: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International Conference on Computer Vision*. 618–626.
[51] Alexander Sergeev and Mike Del Balso. 2018. Horovod: Fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799*.
[52] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. 2017. Learning important features through propagating activation differences. *arXiv preprint arXiv:1704.02685*.
[53] Two Sigma. 2019. Interpretability Methods in Machine Learning: A Brief Survey. Retrieved September 9, 2022 from https://www.twosigma.com/insights/article/interpretability-methods-in-machine-learning-a-brief-survey/.
[54] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. 2017. SmoothGrad: Removing noise by adding noise. *arXiv preprint arXiv:1706.03825*.
[55] The Royal Society. 2019. Explainable AI: The Basics, Policy Briefing. Retrieved September 9, 2022 from https://royalsociety.org/topics-policy/projects/explainable-ai/.
[56] Christian Steinruecken, Emma Smith, David Janz, James Lloyd, and Zoubin Ghahramani. 2019. The automatic statistician. In *Automated Machine Learning: Methods, Systems, Challenges*, Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren (Eds.). Springer, 161–164. https://doi.org/10.1007/978-3-030-05318-5_9
[57] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic attribution for deep networks. *arXiv preprint arXiv:1703.01365*.
[58] Ajay Thampi. 2020. *Interpretable AI, Building Explainable Machine Learning Systems*. Manning Publications.
[59] Leslie G. Valiant. 2003. Three problems in computer science. *Journal of the ACM* 50, 1 (2003), 96–99.
[60] Danding Wang, Qian Yang, Ashraf Abdul, and Brian Y. Lim. 2019. Designing theory-driven user-centric explainable AI. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–15.
[61] Eric P. Xing, Qirong Ho, Wei Dai, Jin Kyu Kim, Jinliang Wei, Seunghak Lee, Xun Zheng, Pengtao Xie, Abhimanu Kumar, and Yaoliang Yu. 2015. Petuum: A new platform for distributed machine learning on big data. *IEEE Transactions on Big Data* 1, 2 (2015), 49–67.
[62] Yongfeng Zhang and Xu Chen. 2018. Explainable recommendation: A survey and new perspectives. *CoRR* abs/1804.11192.
[63] Vijay Arya, Rachel K. E. Bellamy, Pin-Yu Chen, Amit Dhurandhar, Michael Hind, Samuel C. Hoffman, Stephanie Houde, Q. Vera Liao, Ronny Luss, Aleksandra Mojsilović, Sami Mourad, Pablo Pedemonte, Ramya Raghavendra, John Richards, Prasanna Sattigeri, Karthikeyan Shanmugam, Moninder Singh, Kush R. Varshney, Dennis Wei, Yunfeng Zhang. 2019. One Explanation Does Not Fit All: A Toolkit and Taxonomy of AI Explainability Techniques. https://arxiv.org/abs/1909.03012.

Received 20 October 2021; revised 8 July 2022; accepted 21 August 2022