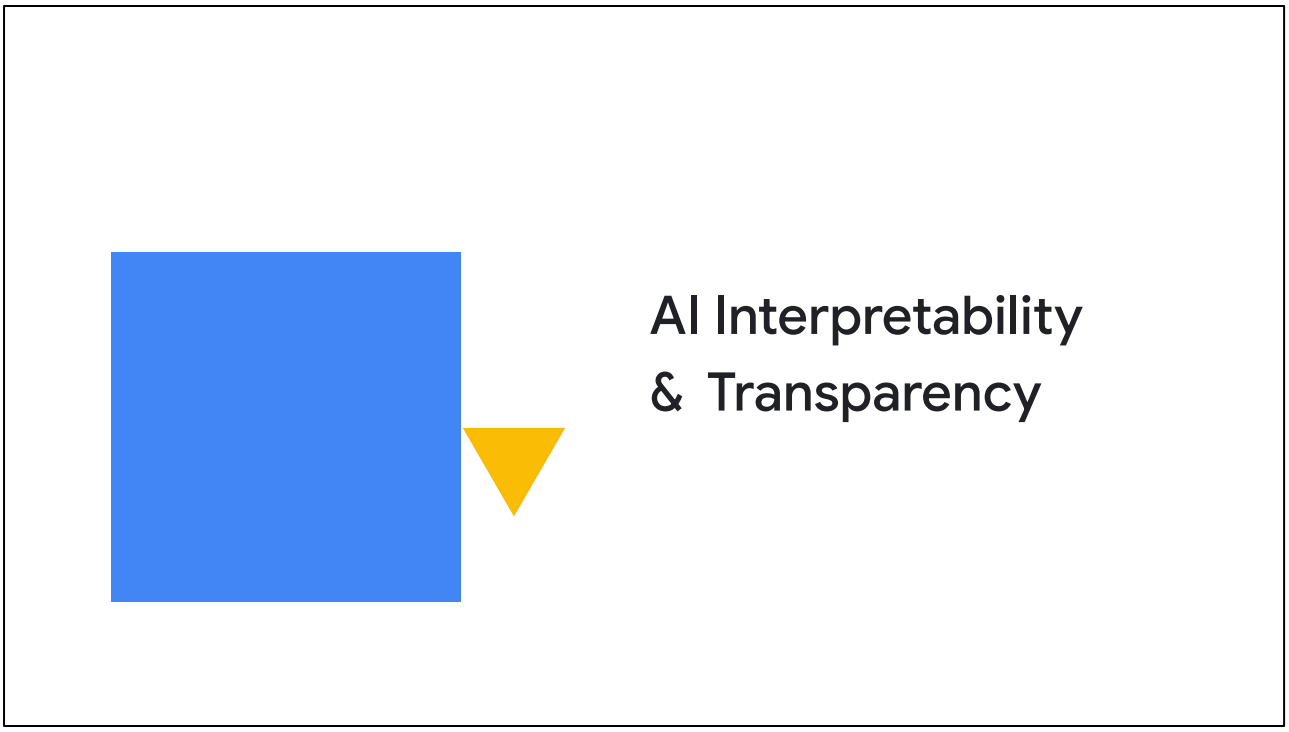


Welcome to AI Interpretability & Transparency

01 Overview of interpretability and transparency

02 Overview of interpretability techniques

03 Feature based explanations: Model agnostic

04 Feature based explanations: Model specific

05 Concept-based and example-based explanations

06 Tools for Interpretability

07 Data and Model Transparency

08 Lab: Vertex Explainable AI

This module consists of 8 lessons.

In this course, you learn to ...

01	Explain what interpretability and transparency mean in machine learning
02	Identify various interpretability techniques used to understand model behaviors
03	Discuss the various interpretability techniques that can be applied to models
04	Describe tools you can use to apply interpretability techniques to your machine learning models
05	Describe toolkits you can use to ensure data and model transparency

Today, you will learn to -

- Explain what interpretability and transparency mean in machine learning.
- Identify various interpretability techniques used to understand model behaviors
- Discuss the various interpretability techniques that can be applied to models
- Describe tools you can use to apply interpretability techniques to your machine learning models.
- Describe toolkits you can use to ensure data and model transparency

01 Overview of interpretability and transparency

02 Overview of interpretability techniques

03 Feature based explanations: Model agnostic

04 Feature based explanations: Model specific

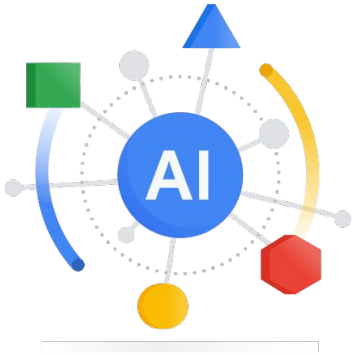
05 Concept-based and example-based explanations

06 Tools for Interpretability

07 Data and Model Transparency

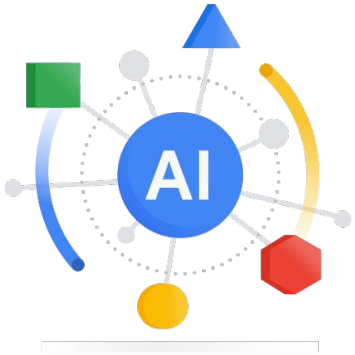
08 Lab: Vertex Explainable AI

Let’s start with an overview of interpretability and transparency.



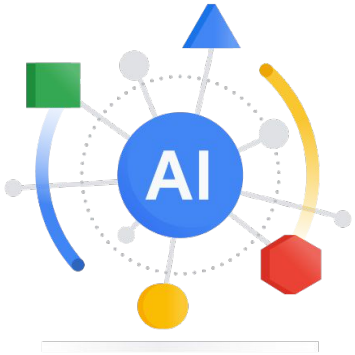
1. Be socially beneficial.
2. Avoid creating or reinforcing unfair bias.
3. Be built and tested for safety.
4. Be accountable to people.
5. Incorporate privacy design principles.
6. Uphold high standards of scientific excellence.
7. Be made available for users that accord with these principles.

Interpretability and transparency relate to the fourth Google's AI principle:



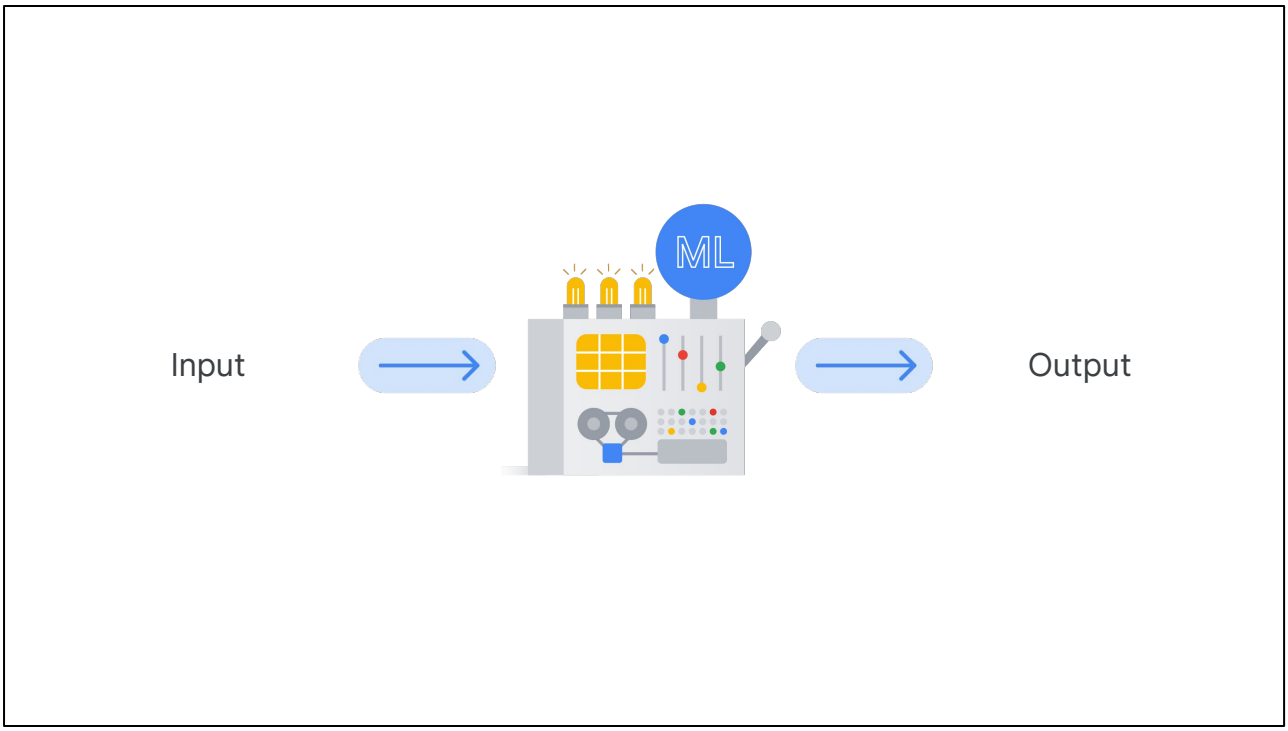
1. Be socially beneficial.
2. Avoid creating or reinforcing unfair bias.
3. Be built and tested for safety.
4. Be accountable to people.
5. Incorporate privacy design principles.
6. Uphold high standards of scientific excellence.
7. Be made available for users that accord with these principles.

“Be accountable to people”;



1. Be socially beneficial.
2. Avoid creating or reinforcing unfair bias.
3. Be built and tested for safety.
4. Be accountable to people.
5. Incorporate privacy design principles.
6. Uphold high standards of scientific excellence.
7. Be made available for users that accord with these principles.

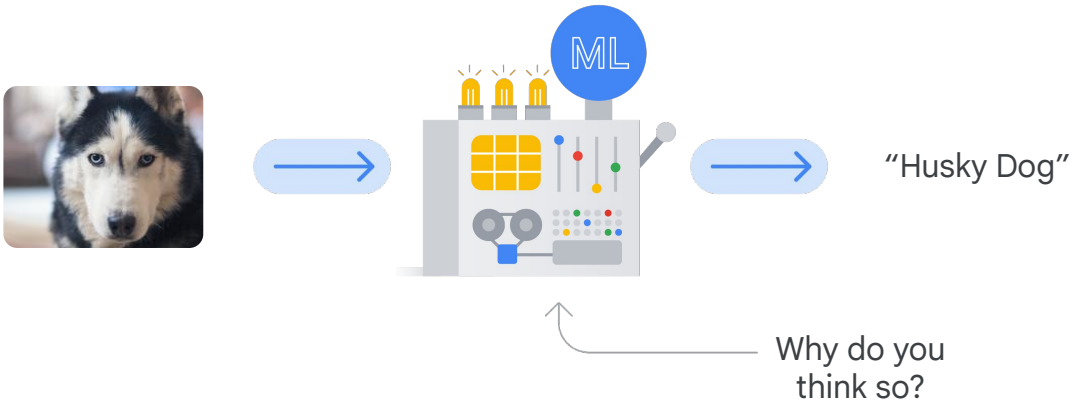
Now, it's worth noting that principles often intersect, and interpretability and transparency are also essential to the second principle, because interpretation is the key to mitigating unfair biases in AI.



Interpretability aims to understand machine learning model behaviors in many ways.
A machine learning model is a function that converts inputs to outputs.

Let’s say that in this image classification example, our machine learning model receives an image and output classification result of a “Husky Dog”.

Interpretability



It seems good! But why did our model say that it was an image of a husky dog? What kind of characteristics are used to judge an image of a husky dog? How can we understand the reasoning behind it?

As we progress through this module, we explore multiple techniques to investigate these questions.

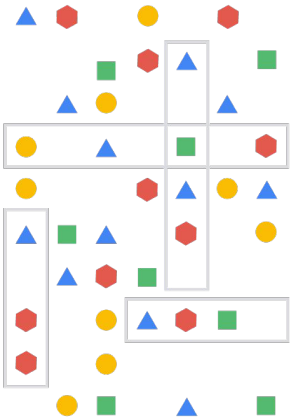
In this course,
interpretability and explainability
are used interchangeably.

It should be noted that in some cases, the term “explainability” is used instead of “interpretability”. Other times, the two terms have slightly different definitions with “interpretability” meaning a deeper and wider understanding of models in holistic systems, and “explainability” is used for more specific and technical methods. In this course, we define and use interpretability and explainability as having similar definitions.

Transparency

Data

- What kind of dataset we use for model training?
- How was it collected?
- How was it preprocessed?
- Does it have known bias problem?

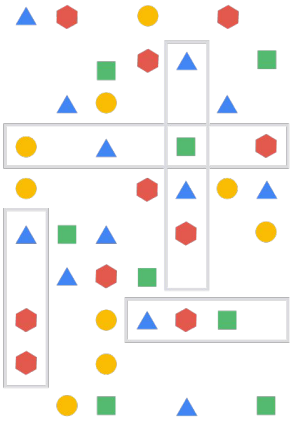


Now when it comes to transparency, it's very important to obtain trust from stakeholders, including users.
Usually, machine learning applications involve multiple modules, such as data collection system, data processing pipeline, machine learning model, evaluation system, serving systems and so on.

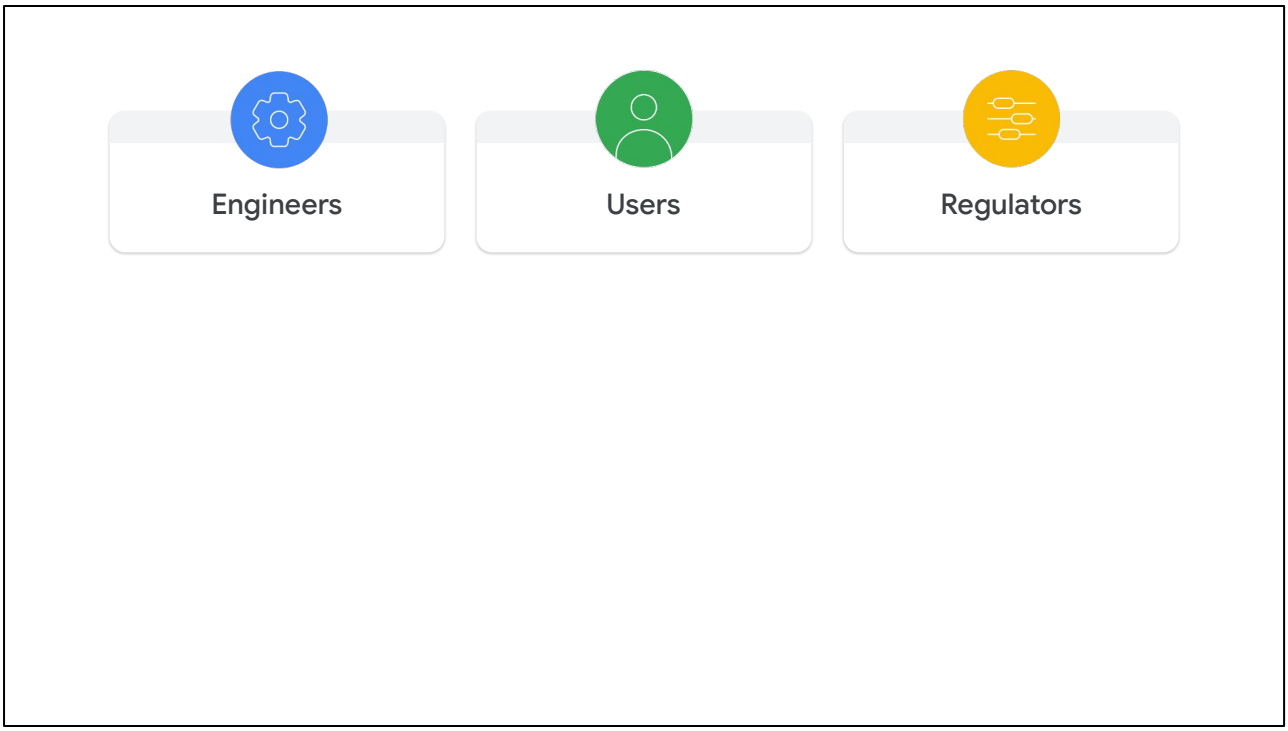
Transparency

Model

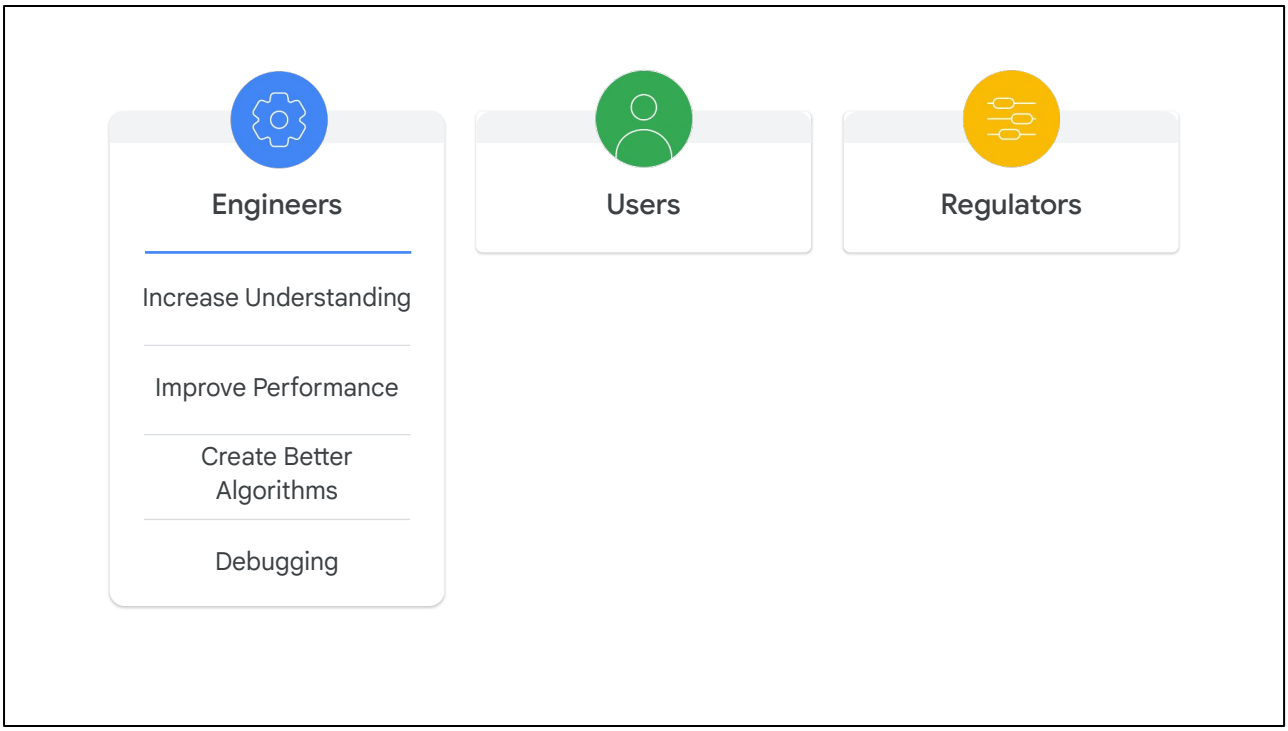
- What kind of Machine Learning algorithm did we used?
- Is there any considerations on the usage?
- Does it have any ethical concerns?



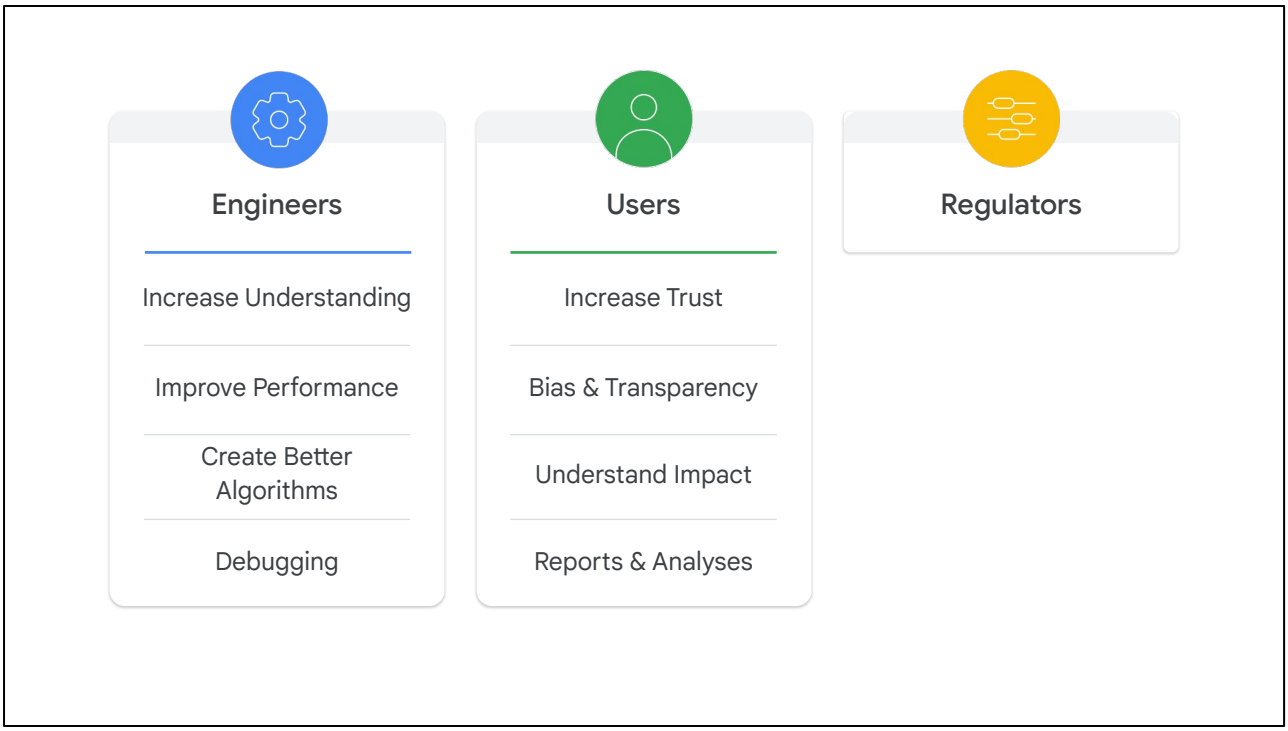
Since each module is dependent on each other, it is always important to make documentation for each system and communicate based on that information. Google offers some useful tools and frameworks for this purpose, which we will discuss in more depth later on.



At a high level, there are 3 stakeholder groups that benefit from interpretability and transparency, as it relates to AI systems.

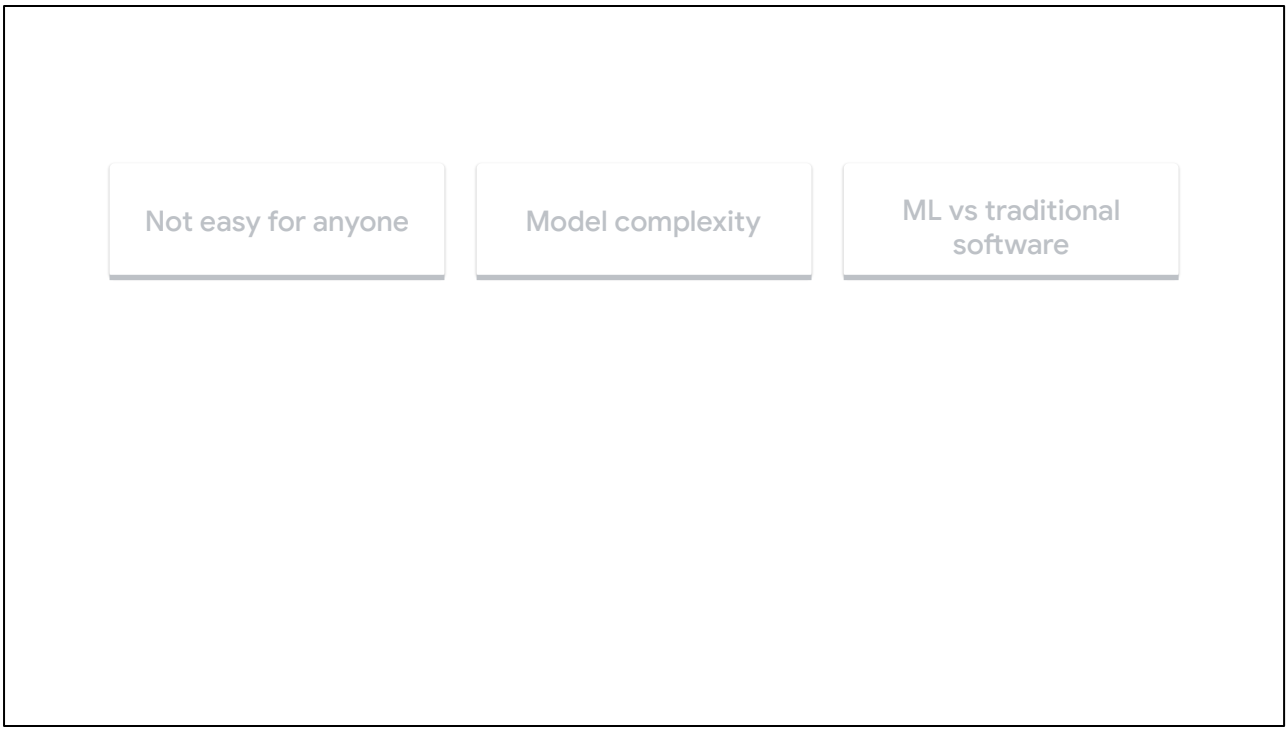


For Engineers: the focus is more on interpretable ML techniques to model understanding and improve performance.
More complex models can be difficult to debug, understand, and control. Interpretability and transparency methods help with this.

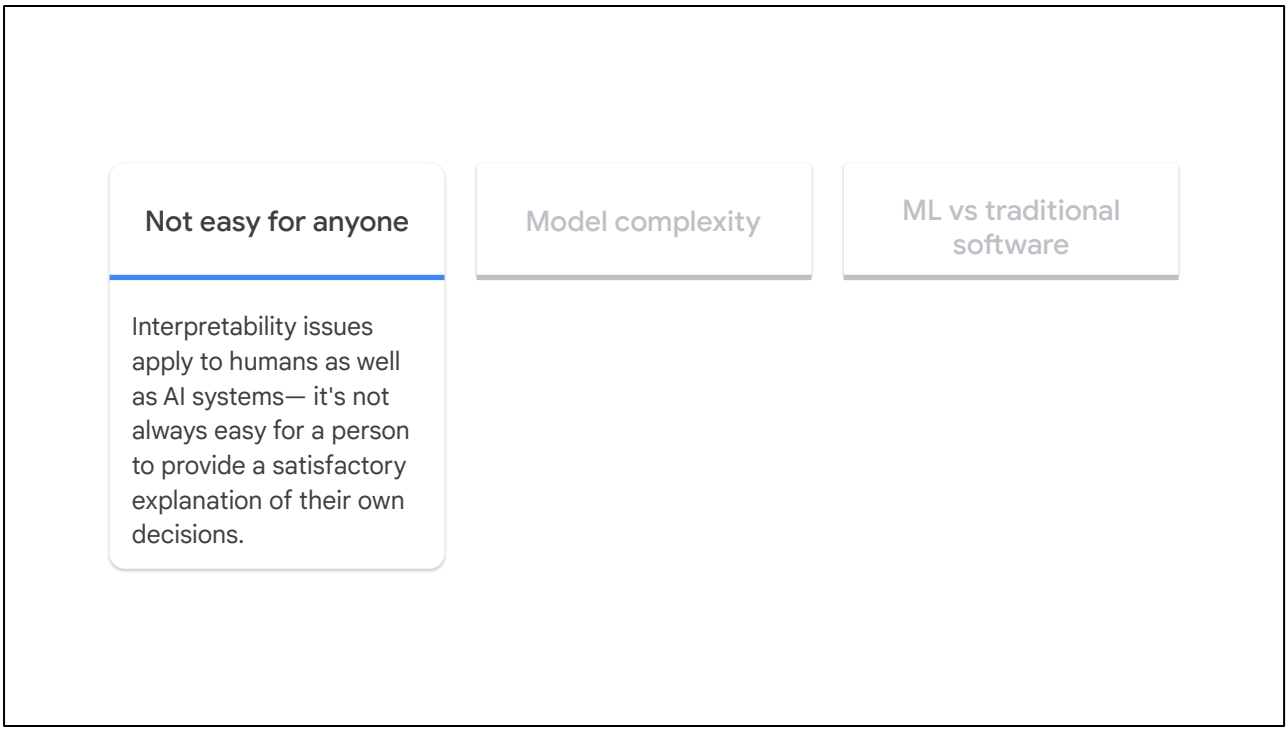


For Users: The focus is on trust for model consumers. Users might not care about internal mode, but are very interested in understanding the impact of model predictions.
Explainable systems build trust with these end users that show how model decisions are reliable and equitable.

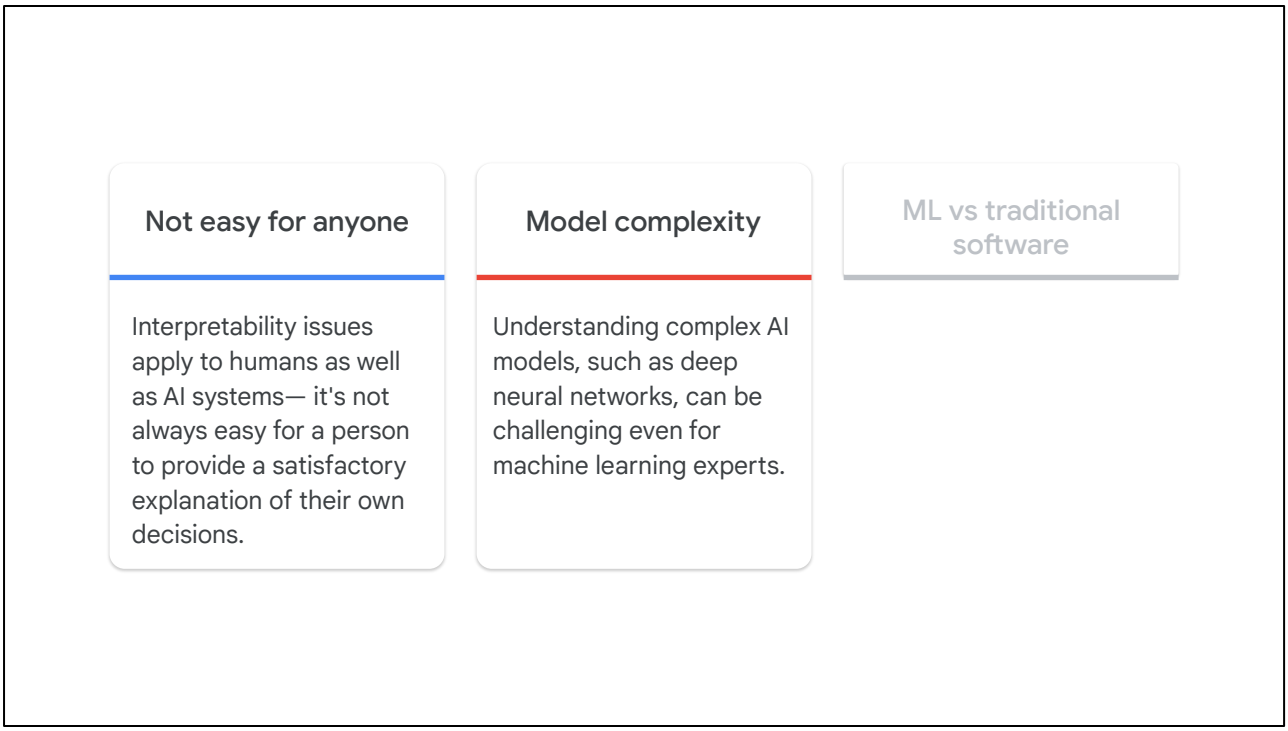
For Regulators: The focus is on ensuring that model decisions are in compliance with laws, and do not amplify undesirable bias from underlying datasets. Interpretable explanations provide auditable metadata. This allows regulators to trace unexpected predictions back to their inputs to inform corrective actions.



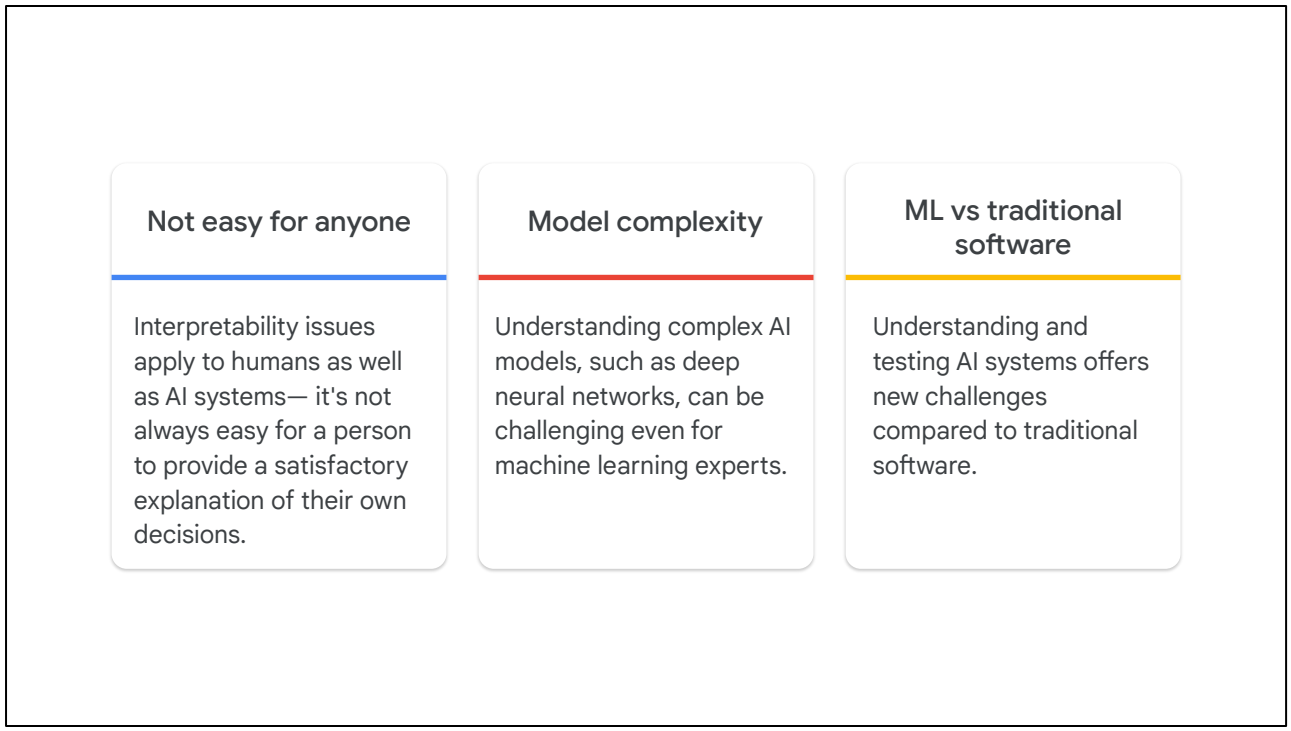
Although interpretability is an active area of research, the task is not easy because:



1. Interpretability issues apply to humans and AI systems. After all, it's not always easy for a person to provide a satisfactory explanation of their own decisions.



2. Understanding complex AI models, such as deep neural networks, can be challenging even for machine learning experts.
There is typically a tradeoff between complexity and interpretability of models.



3. Understanding and testing AI systems offers new challenges compared to traditional software. Traditional software is essentially a series of “if-then” rules that, through some chasing, can be interpreted and debugged; With AI systems that not only have code but also data and models, it is much harder to pinpoint one specific bug that leads to a faulty decision.

01 Overview of interpretability and transparency

02 Overview of interpretability techniques

03 Feature based explanations: Model agnostic

04 Feature based explanations: Model specific

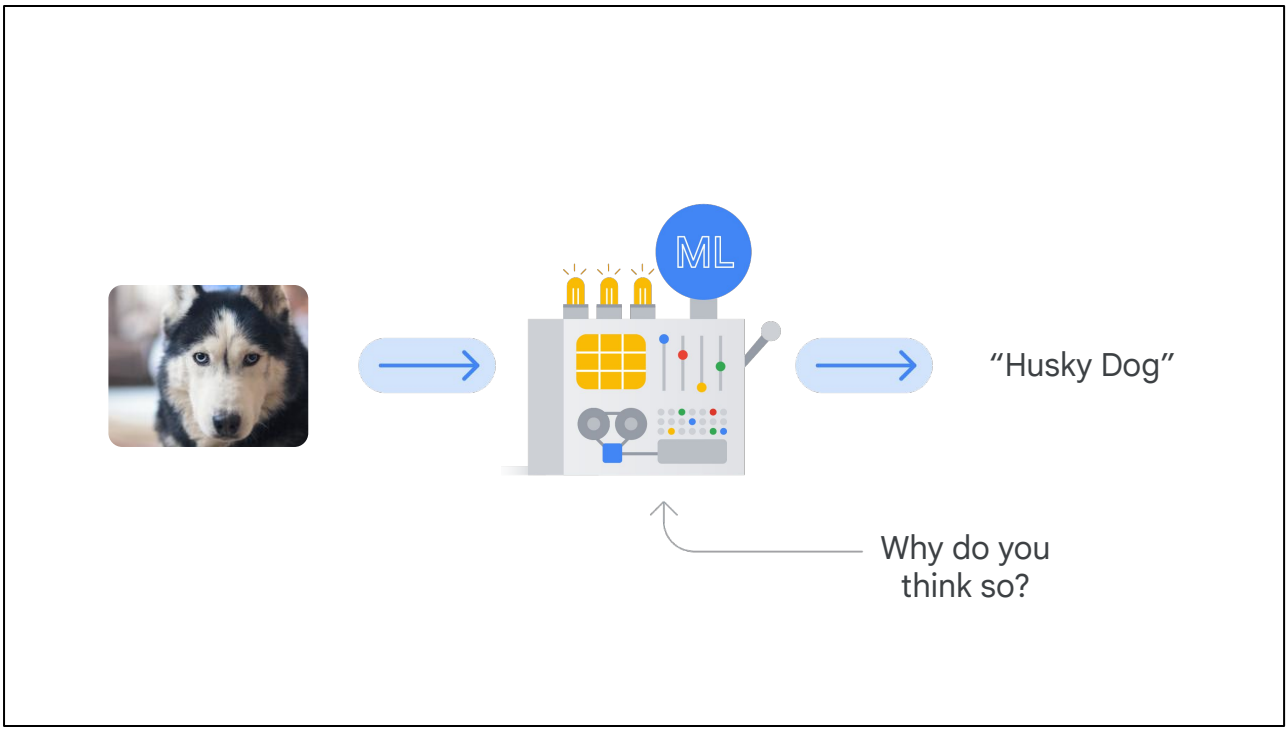
05 Concept-based and example-based explanations

06 Tools for Interpretability

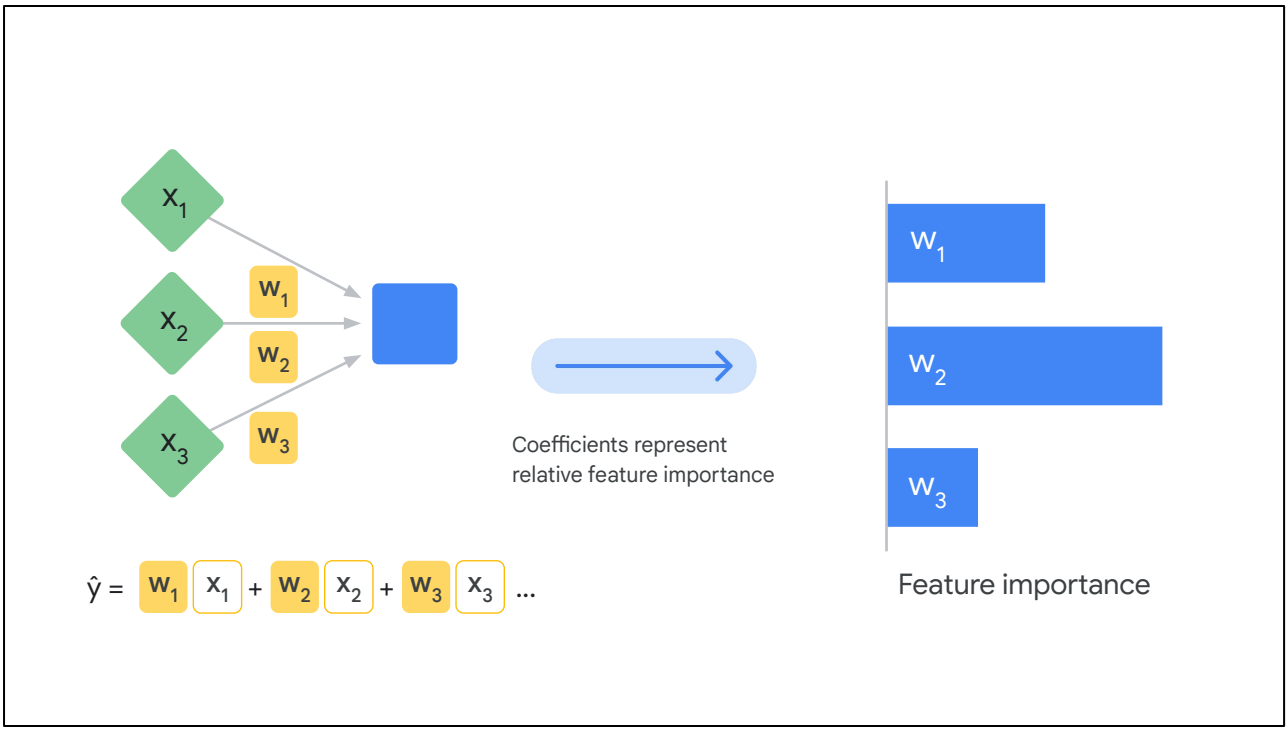
07 Data and Model Transparency

08 Lab: Vertex Explainable AI

Let’s talk about the role interpretability plays in understanding model behaviors.

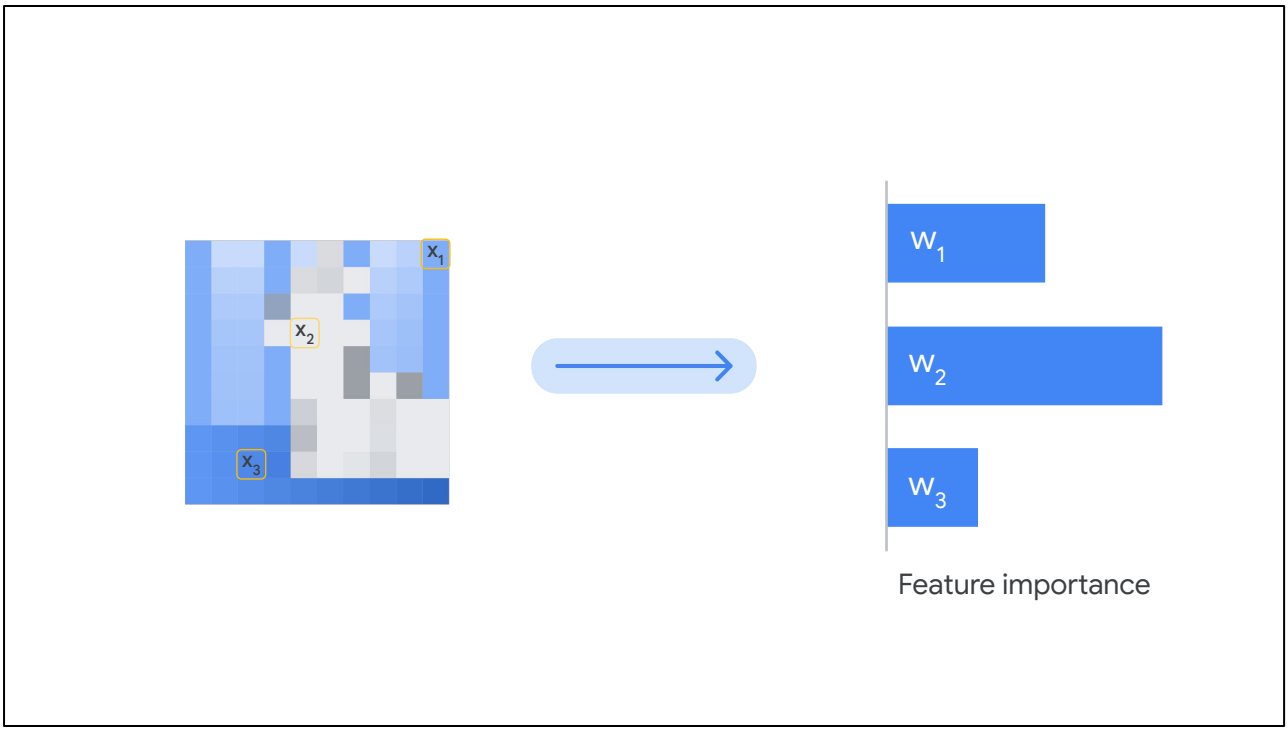


Here is a classification model of a Husky Dog. Now, the machine learning (or ML) model provides a smart prediction, but how do we understand the reasoning behind it?

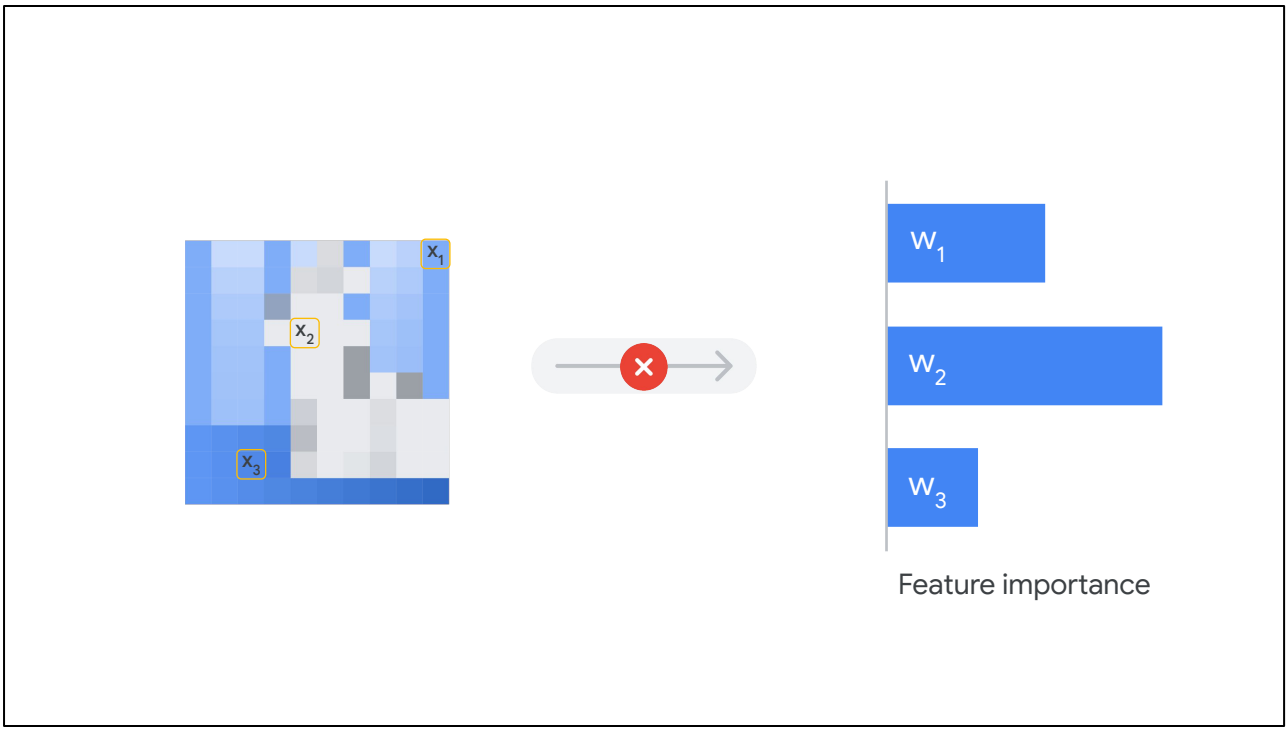


If our model is very simple, like a linear regression model, it is not difficult to understand the reasoning. If we can assume input features of Xs are normalized, we can simply inspect each coefficient of W.

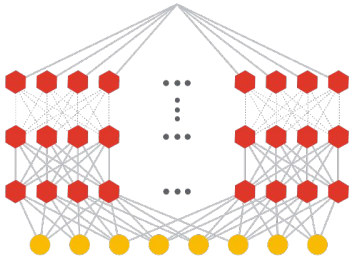
Since linear regression involves a weighted sum of the input features, the magnitude of the coefficients corresponds to the relative importance of those features.



For instance, in image classification, we can analyze the coefficients associated with each pixel.

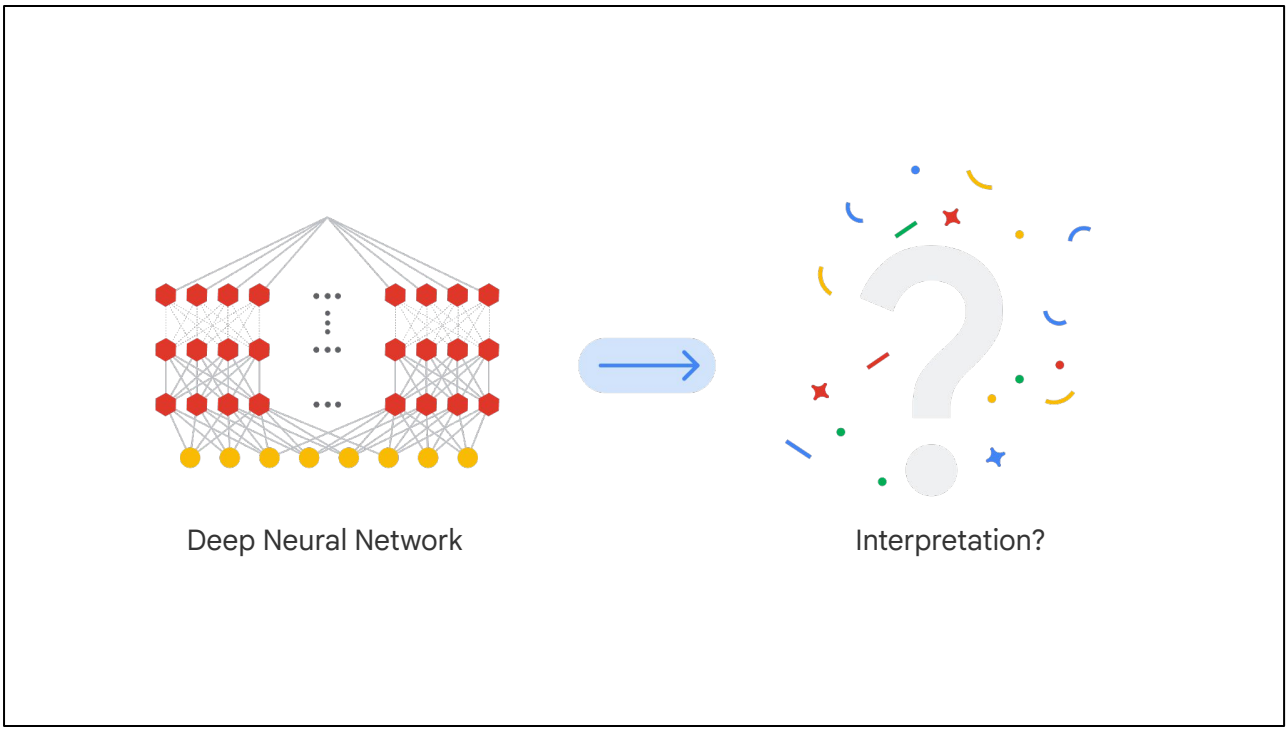


However, for complex tasks like image classification, we typically don't employ such simple models.

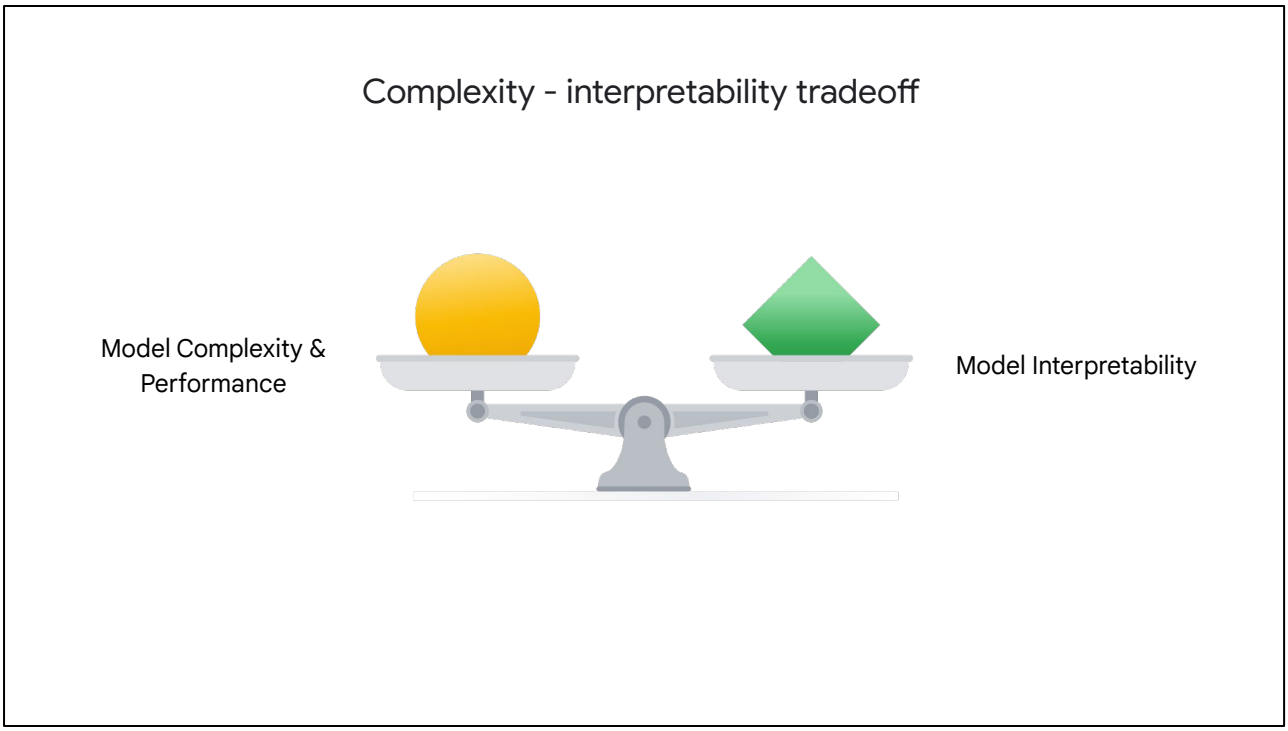


Deep Neural Network

Now, deep neural network (or DNN) models, such as convolutional neural networks (CNNs) and Transformers, generally outperform linear regression models in image classification tasks. This is because DNNs can extract more intricate relationships between the vast number of pixels in an image.
It does this by using many weight parameters.



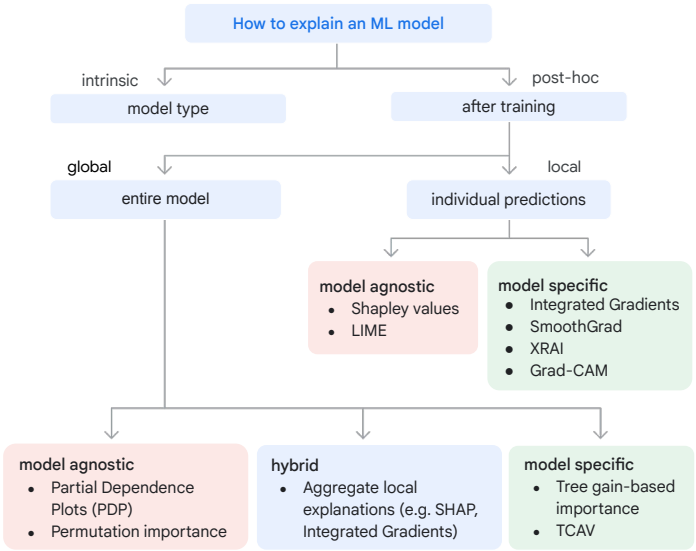
However, the increased complexity of DNNs makes it more challenging to comprehend their behavior in a way that allows for straightforward interpretation.



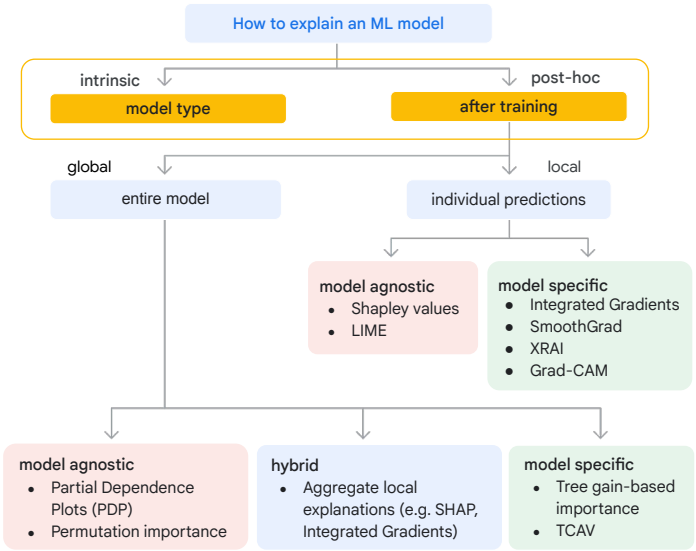
Although DNN is a very popular and common approach, their widespread adoption is hindered by their "black-box" nature, making them less interpretable than simpler models. This inherent trade-off between complexity and interpretability poses a challenge in understanding the reasoning behind DNNs.

It should be noted that complexity doesn’t always lead to better performance, so you should always consider using simple models when the performance gain of using a more complex model is small.

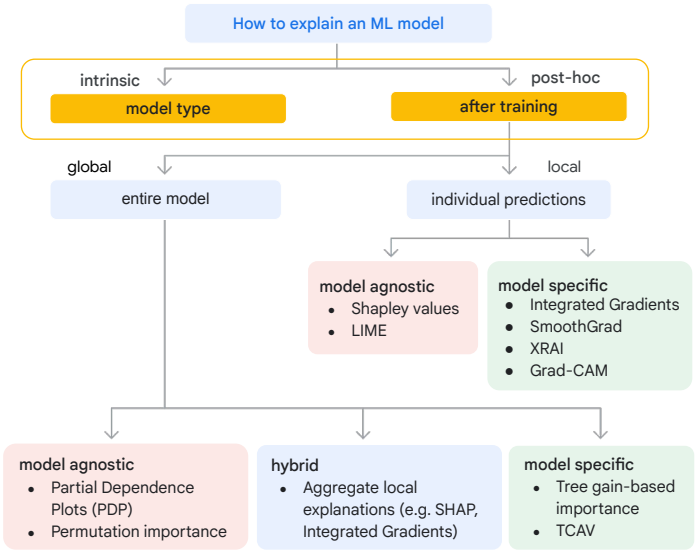
To help understand very deep and complex models, a lot of modern interpretability techniques exist.

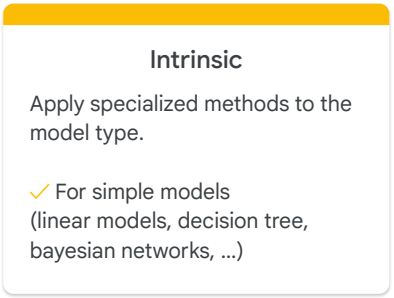


There are many ways to categorize different types of interpretability techniques, so we classified them into several subcategories.



Every interpretability technique first falls into one of two categories: intrinsic or post-hoc.

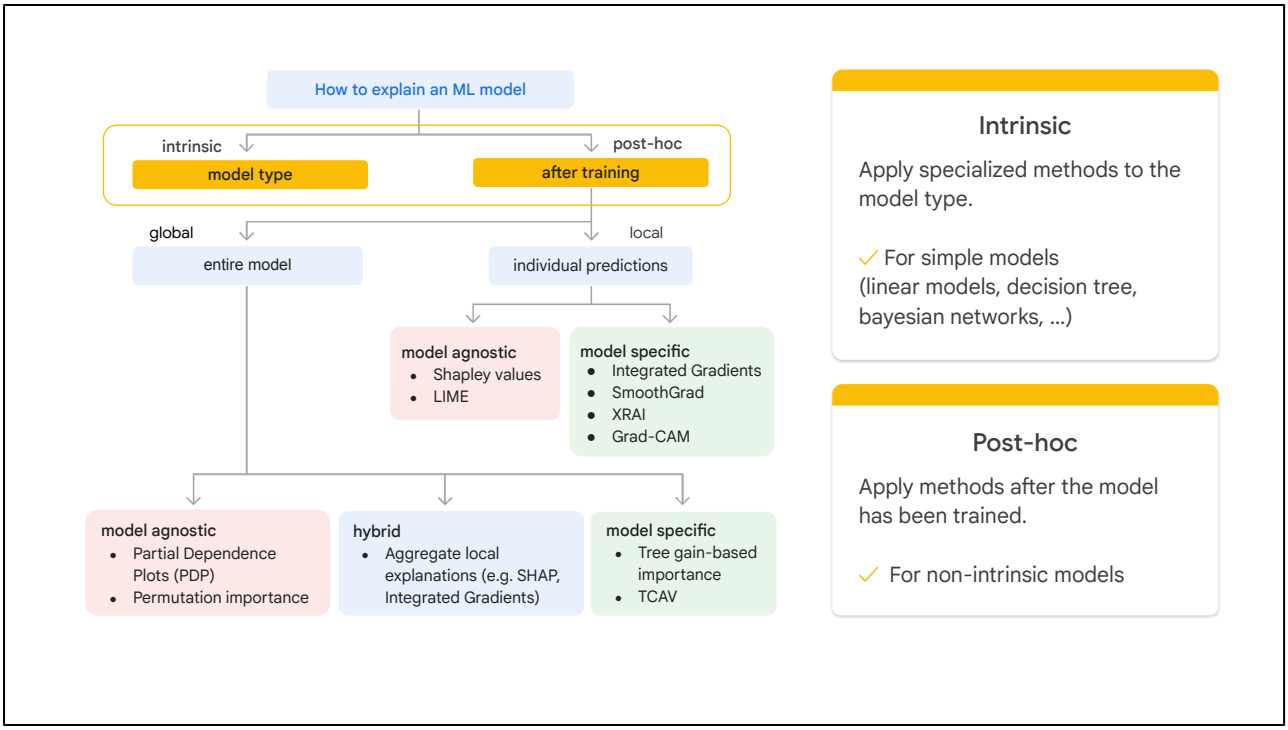




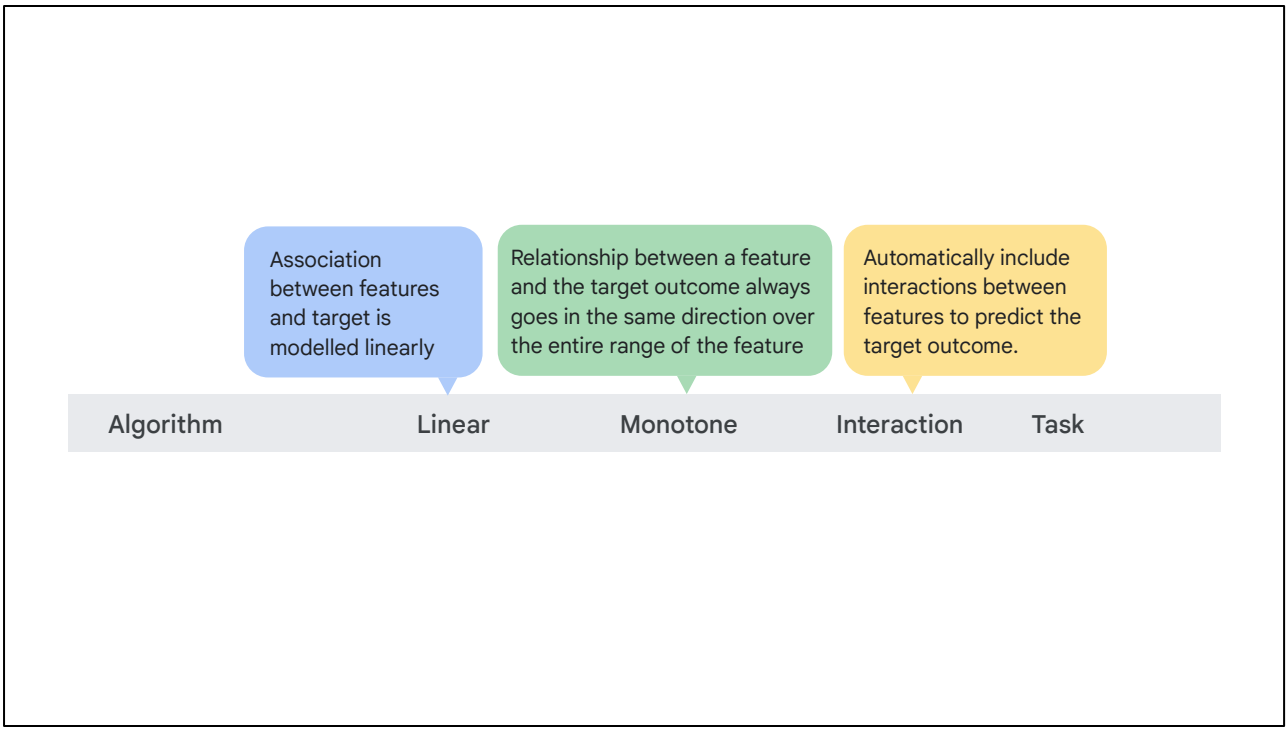
Intrinsic: The ability of a model to be directly interpreted by examining its structure or learning process. This means that the model's internals are inherently transparent and can be understood without the need for additional tools.

This approach is only possible for simple methods such as linear regression models, where you merely need to look at the trained weights for each feature, decision forests, bayesian networks, and so on.

Note that some models still require sophisticated visualization to become interpretable.



Next, Post hoc: Post hoc refers to methods that are applied after a model is trained. These methods aim to provide insights into the model's behavior and explain its predictions.
This approach is necessary for non-intrinsic models, which are often more complex, and it provides a consistent way to assess interpretability across different model types.
Post-hoc methods can be applied to intrinsically interpretable models too.



So, intrinsically interpretable models remain among the most popular interpretability methods, because they are fairly easy to understand.

At the same time, care should be given when describing these models to stakeholders.

Why?

- First, if your model isn't complex enough to accurately describe the underlying system and fit the data well, explanations aren't going to be useful, because the models are not useful.
- Second, ML models learn patterns and correlations, not causal structures.
- Third, model internals and interpretable ML methods tell you about your model, not necessarily your data or the underlying data generation system.
- And lastly, keep explanations to claims about the model only; be careful making inferences on underlying populations of interest.

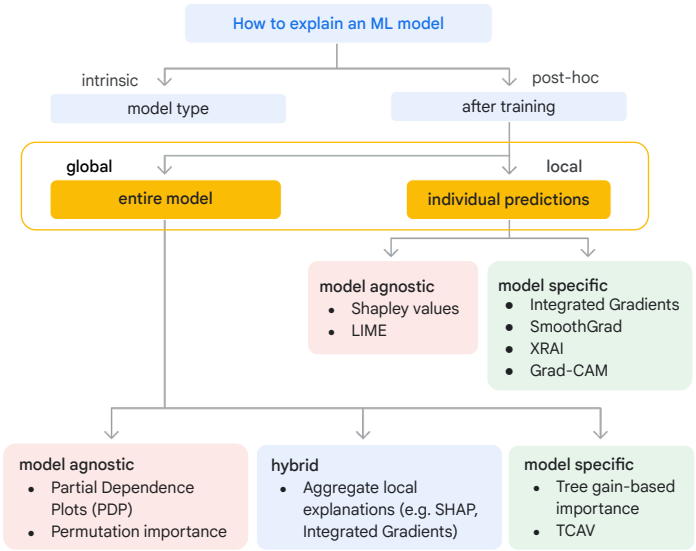
Association between features and target is modelled linearly

Relationship between a feature and the target outcome always goes in the same direction over the entire range of the feature

Automatically include interactions between features to predict the target outcome.

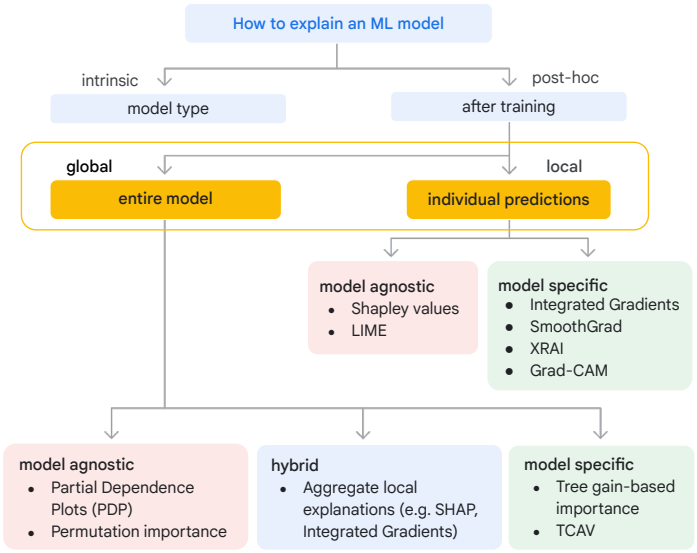
Algorithm	Linear	Monotone	Interaction	Task
Linear regression	Yes	Yes	No	regr
Logistic regression	No	Yes	No	cls
Decision trees	No	Some	Yes	cls,regr
Naive Bayes	No	Yes	No	cls

Intrinsic explanations vary in their characteristics, particularly in terms of linearity, monotonicity, and the ability to consider feature interactions. For example, Intrinsic explanation of linear regression is linear and monotone, but it doesn't consider interactions, because we simply look at coefficients of each feature.



So, now let's focus more on post-hoc techniques, which are more flexible in terms of model selections.

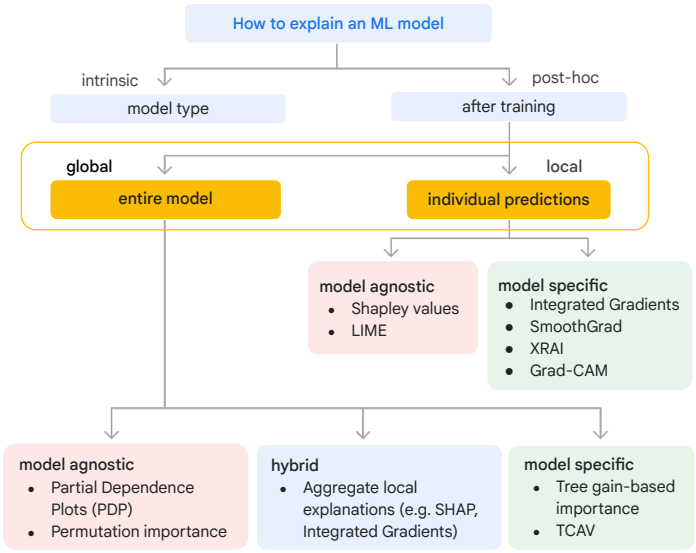
Post-hoc interpretation techniques can provide either local or global interpretability.



Local

Interpretability of individual predictions or a small part of the model's prediction space.

Local techniques provide explanations for each data point. We could ask the model, "Why do you classify this image as a 'Husky Dog'?" Using a local technique, we can then delve into the reasoning behind a specific prediction for a particular data point.



Local

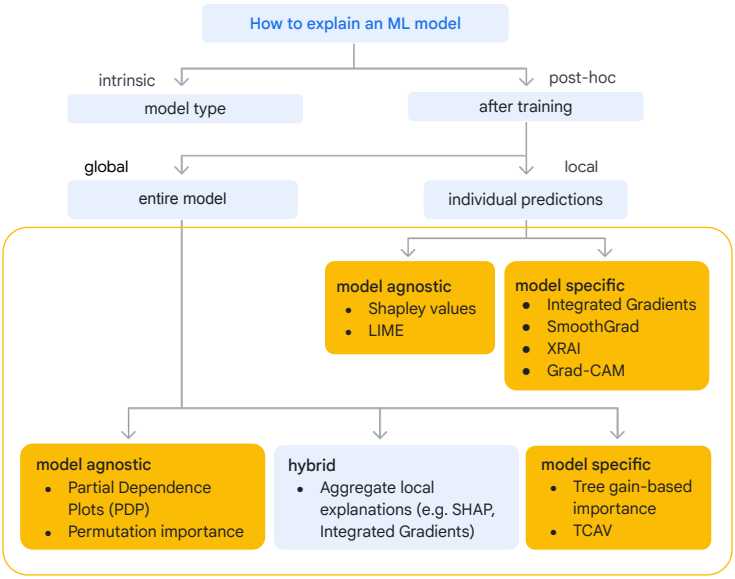
Interpretability of individual predictions or a small part of the model's prediction space.

Global

Aggregated, ranked contributions of input variables for the entire model's prediction space.

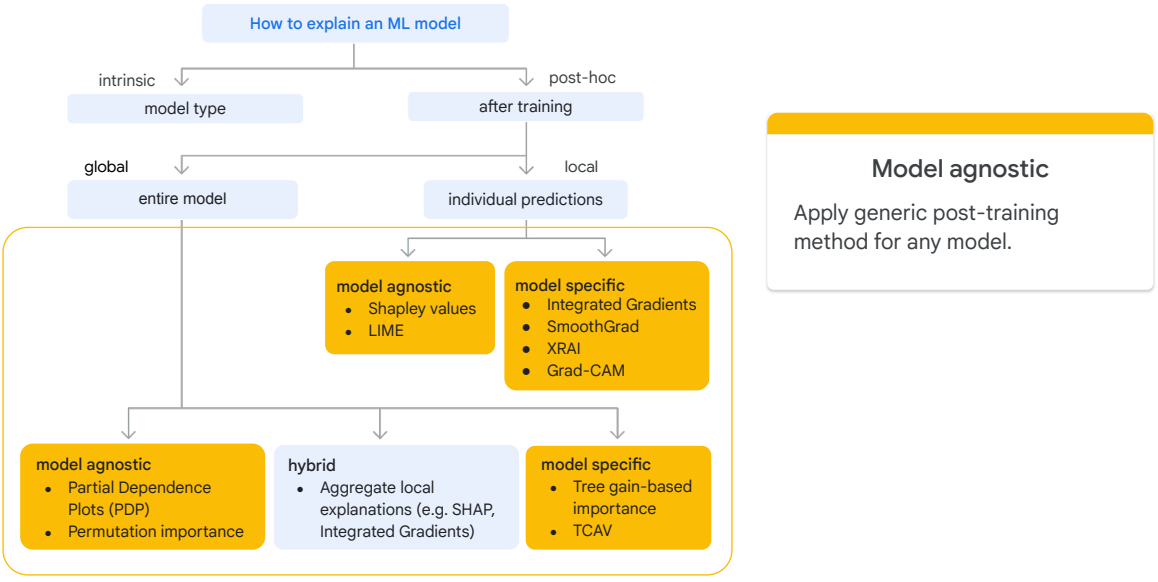
Global techniques provide more aggregated explanations, referring to the entire model's prediction space. Using a global technique, we can ask the model "Which feature do you prioritize most in general?".

Global methods provide a more complete map of the prediction space but can fail to accurately capture the nuances of individual predictions. It can also fail to effectively explain the model's behavior in specific regions of the prediction space.

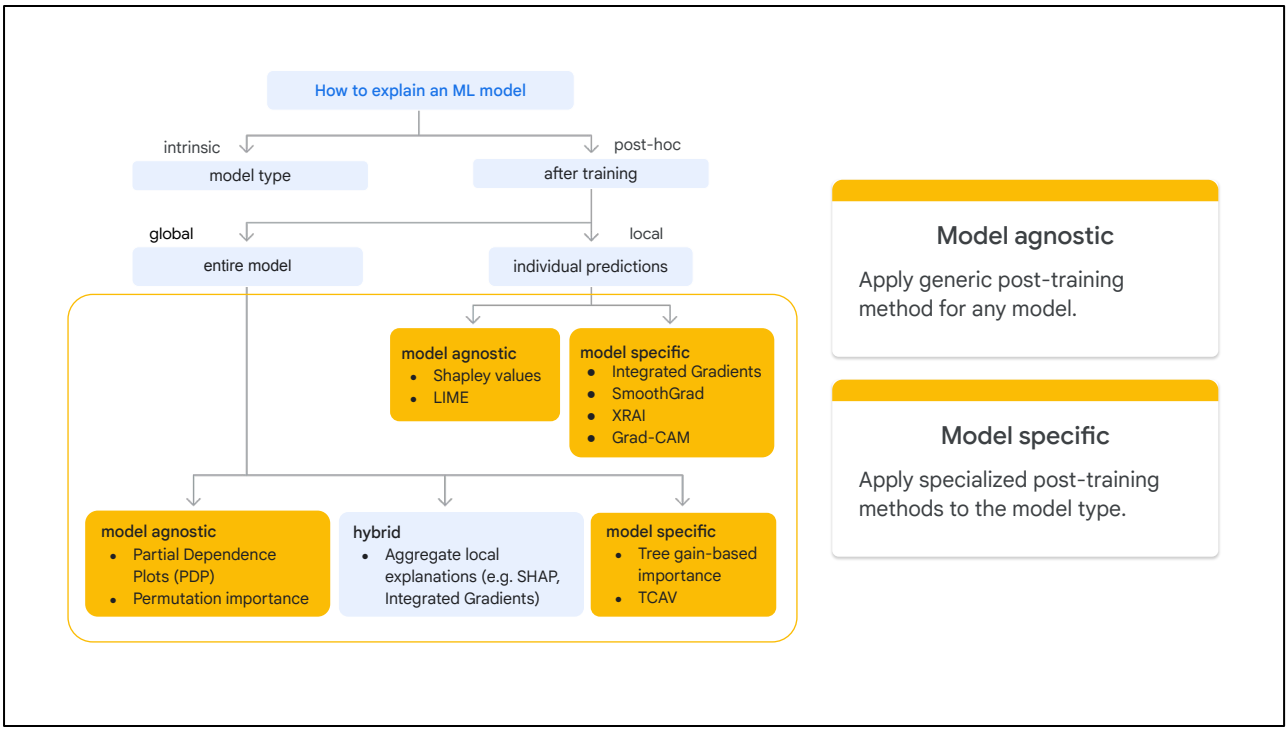


Now let's talk about the terms model agnostic and model specific.

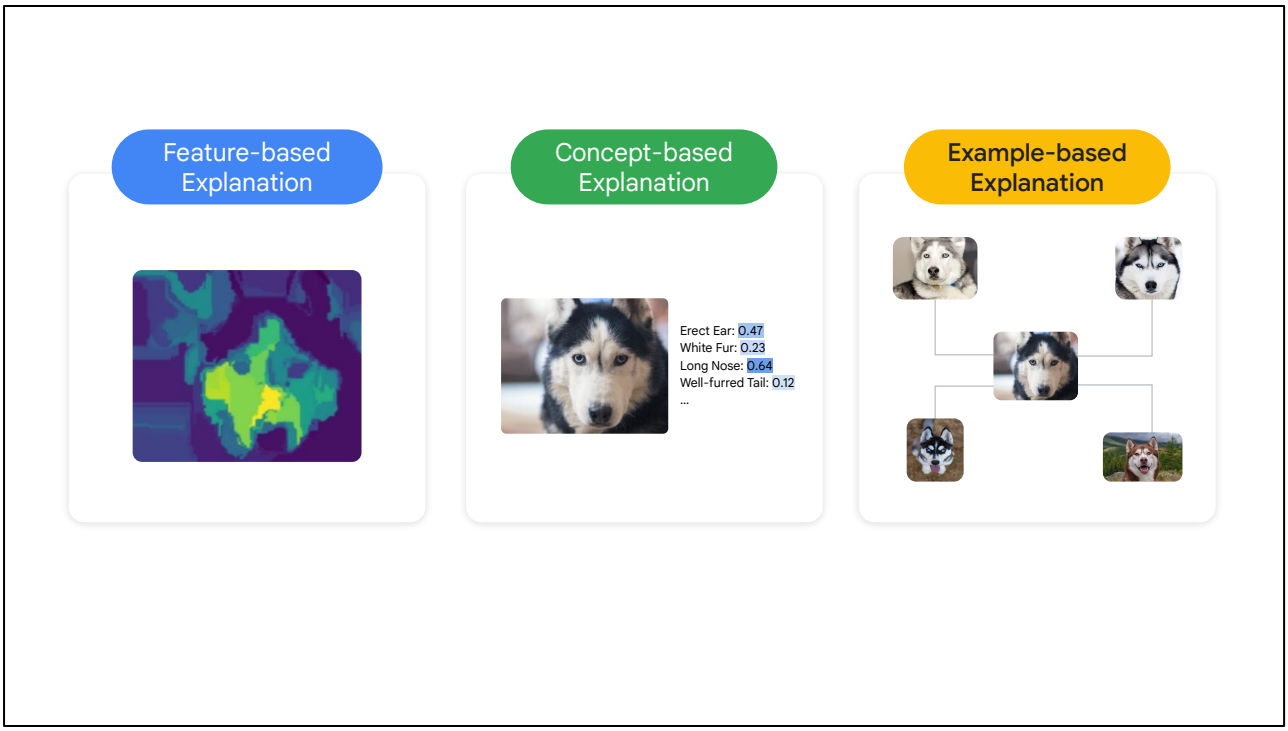
Both local and global interpretation methods fall into one of these two areas. At a high level, model-specific methods use the internal details of the model, whereas model-agnostic methods examine the model's behavior by manipulating the data fed into the model.



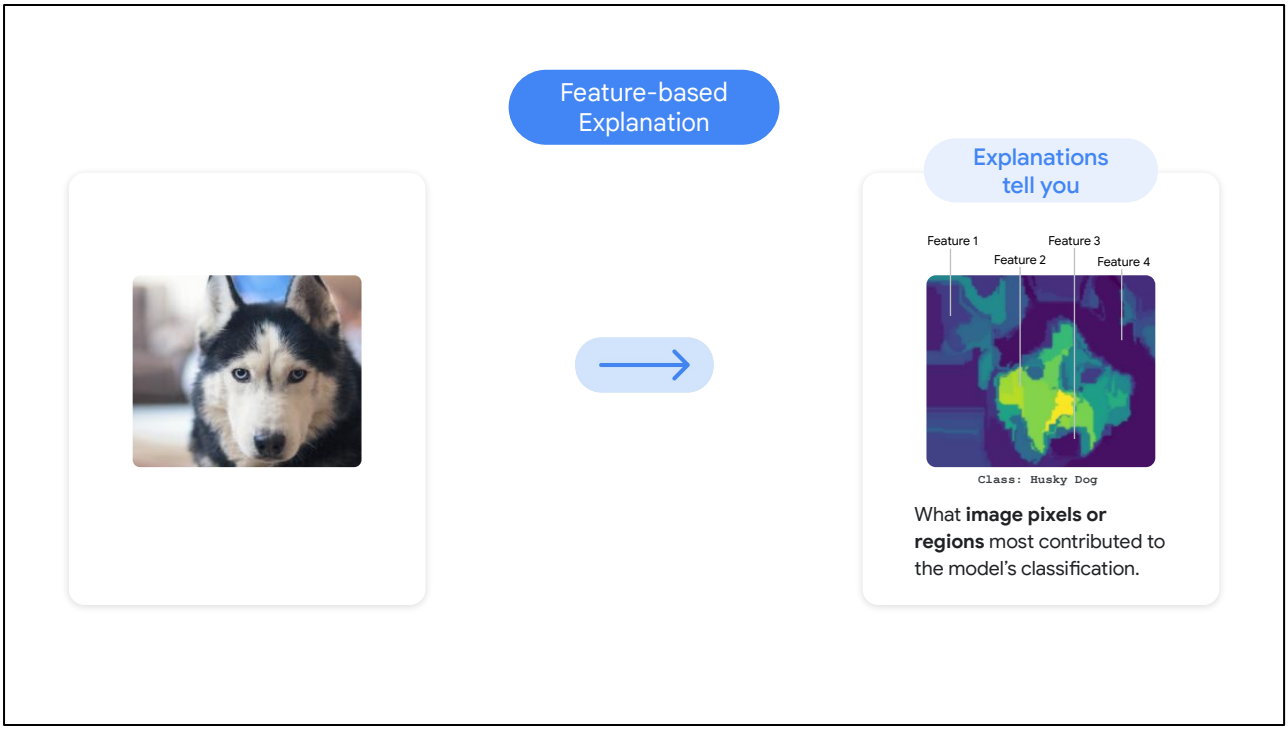
Model agnostic interpretability methods don't rely on model internals. Instead, they analyze how changes in input features affect the model's output predictions. This approach allows these techniques to be applied to a wide range of machine learning models.



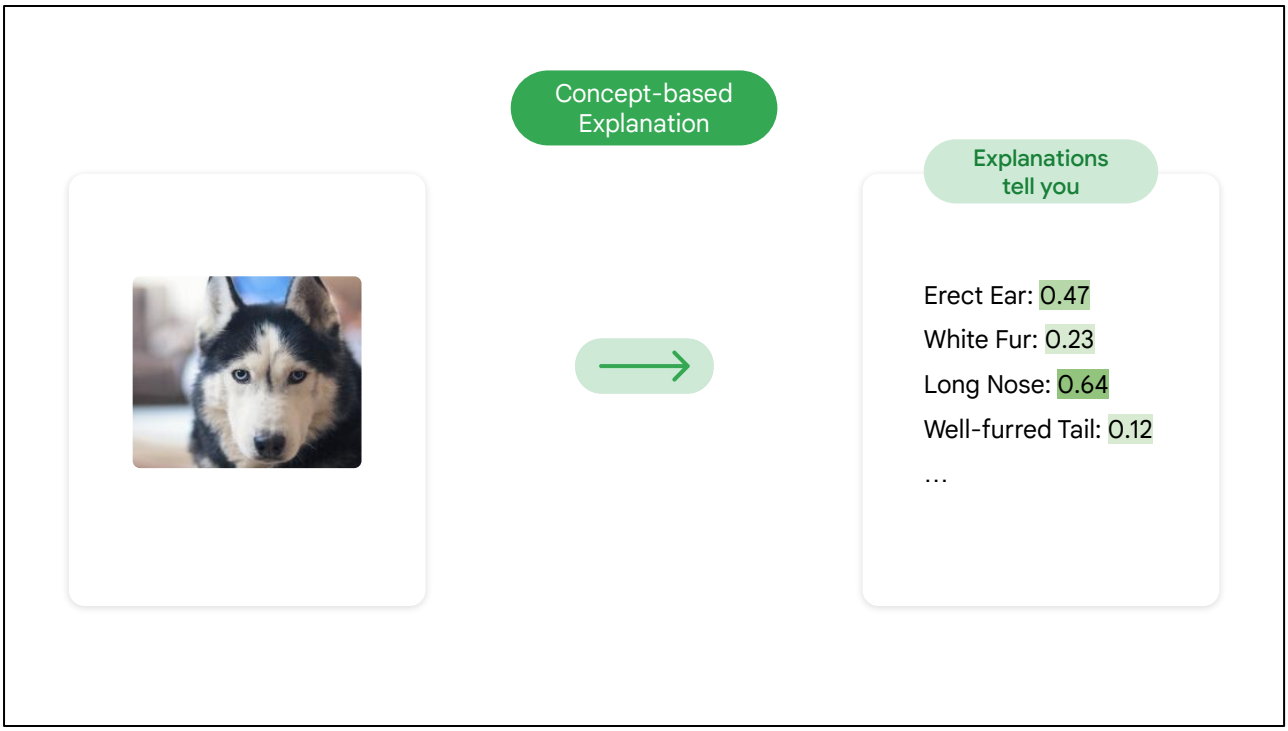
Model specific interpretability method is restricted to usage on specific types of machine learning models. For example, some techniques rely on the gradients of a neural network, and can only be applied to differentiable models like deep neural networks.



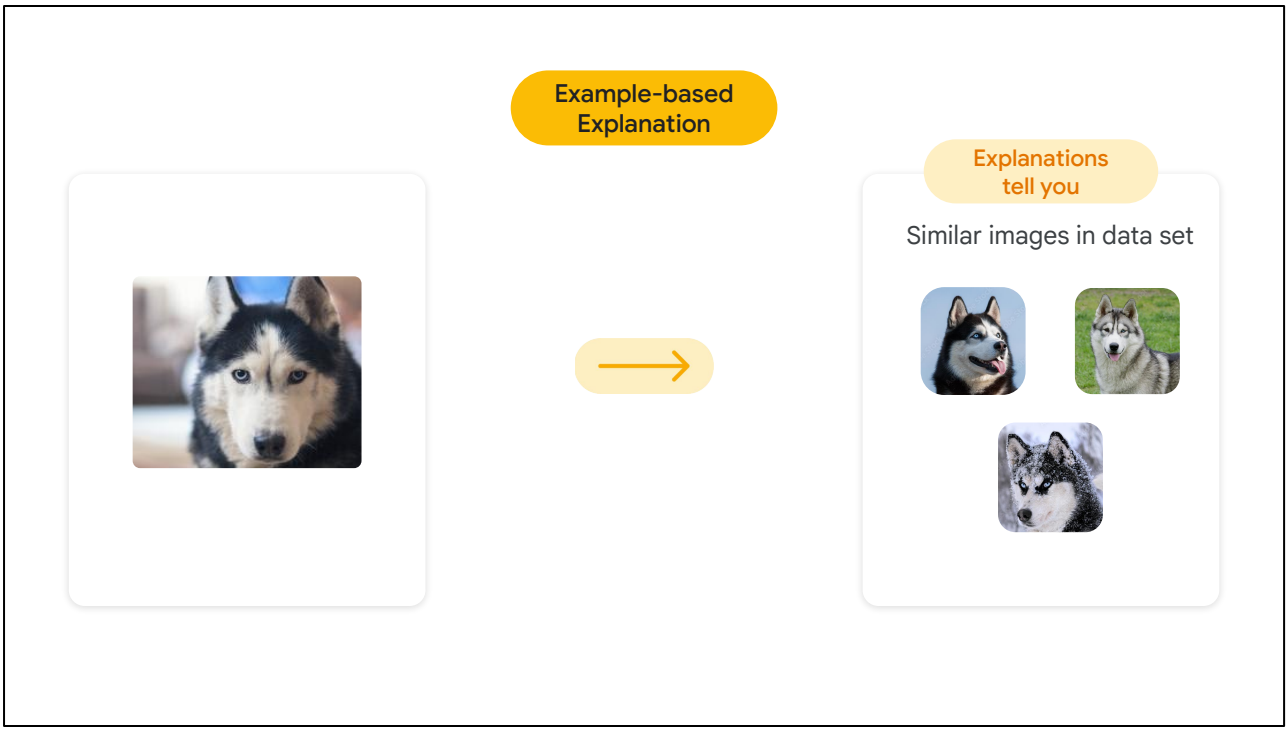
Interpretability techniques can be categorized based on the type of output they produce. Here's a breakdown of the major categories based on output types. Although this is not an exhaustive list, it covers major categories.



1. Most of the major interpretability methods fall into the category called feature-based explanations. These provide attribution scores, masks, or some visualizations that highlight the importance of each input feature for a given task. For example, feature-based explanations might indicate which features are most relevant in classifying an image as a Husky dog.



2. Concept-based explanations are similar to the feature-based explanation, but it outputs relative importance scores to broader different “concepts” instead of each feature. These “concepts” such as “long-nose-ness” or “white-fur-ness” are designed to be more intuitive and interpretable for humans.



3. Example-based explanations provide very different types of output. Instead of returning the importance of feature or concept explanations, it provides data points that our target model thinks is similar to a given query data point. This kind of output is also useful to understand how models interpret data and extract meaningful embeddings from the input.

01 Overview of interpretability and transparency

02 Overview of interpretability techniques

03 **Feature based explanations: Model agnostic**

04 Feature based explanations: Model specific

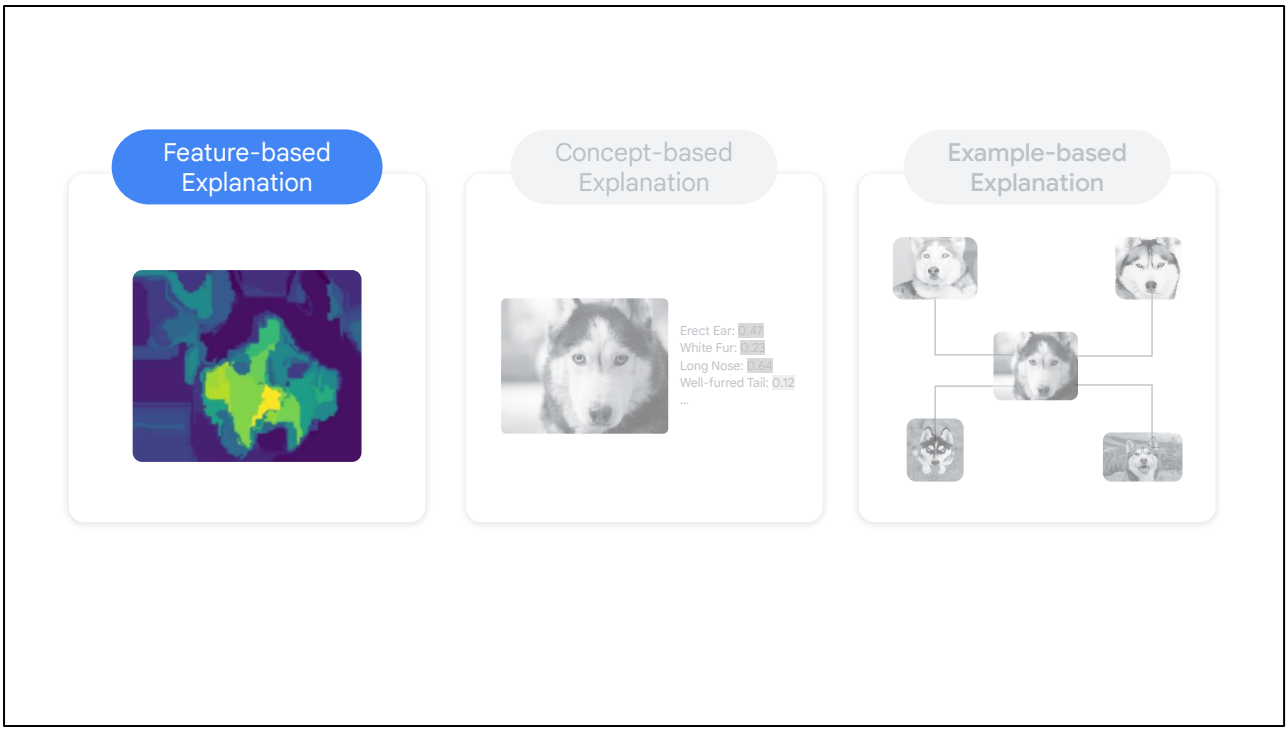
05 Concept-based and example-based explanations

06 Tools for Interpretability

07 Data and Model Transparency

08 Lab: Vertex Explainable AI

Let’s look more in depth into each model agnostic approach.

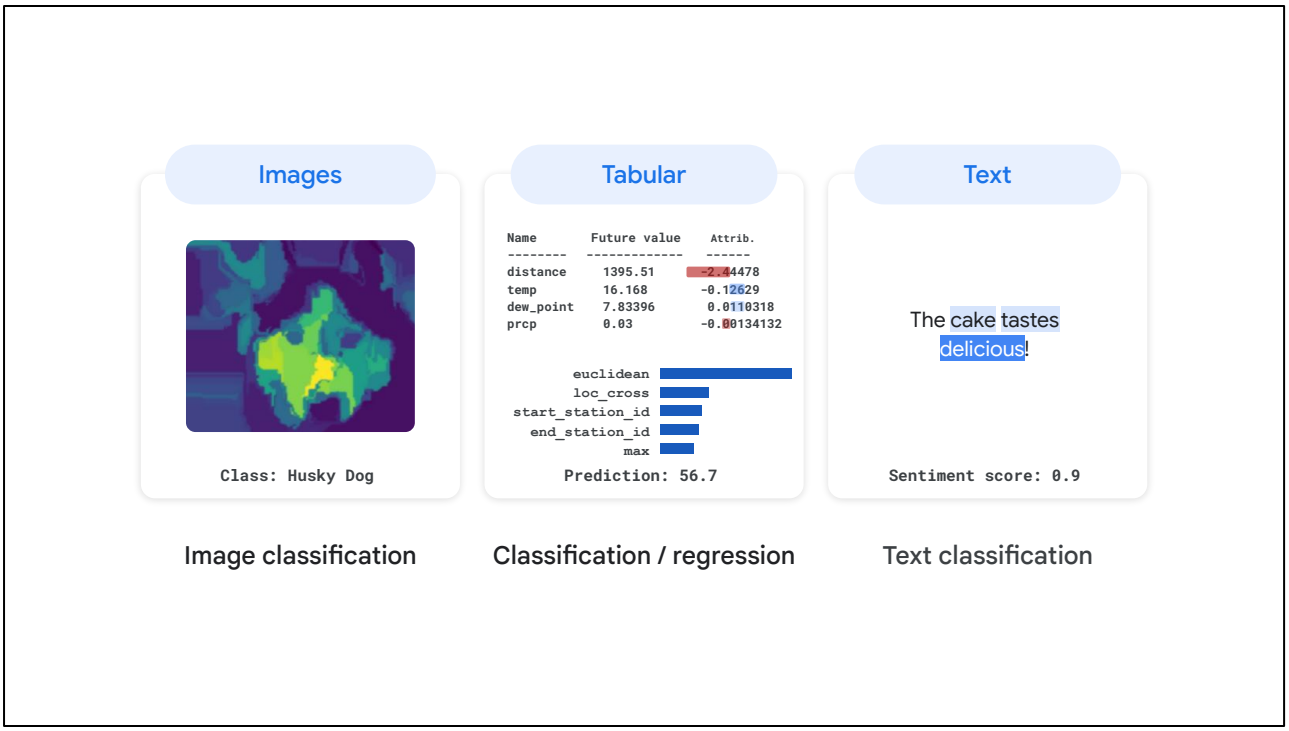


First, feature-based explanations. Again, feature-based explanations provide attribution scores, masks, or some visualizations that highlight the importance of each input feature for a given task. What this looks like depends on the type of data you're using.

For each data modality of images, tabular data, and text, feature attributions look different.

For images, feature attributions will show you which pixels or regions within an image most contributed to the model's classification. For example, for this image of a husky, it looks like the model is focusing mostly on the husky's nose and facial features, which is likely what we would hope for with a model seeking to classify dog breeds.

For tabular data, feature attributions tell you how much each feature column contributed to either a specific prediction or the model overall. Some methods just return the magnitude of impotence as positive values, while others can give you directions of feature contribution to outputs, like positive or negative contribution.



For text, feature attributions tell you how much each word or token in a sentence or utterance contributed to the sentiment score. For example, in this sentence - “The cake tastes delicious”, feature attributions could show you that the high sentiment score of 0.9 is largely attributable to the word ‘delicious’, which had a strongly positive influence on the text’s classification. Whereas the other words in the phrase had significantly less influence.

Feature-based explanation techniques

Let’s look at some popular feature-based explanation techniques:

Feature-based explanation techniques

- Global techniques:
  - Permutation Feature Importance
  - Partial Dependence Plots

There are global techniques like Permutation Feature Importance and Partial Dependence Plots,

Feature-based explanation techniques

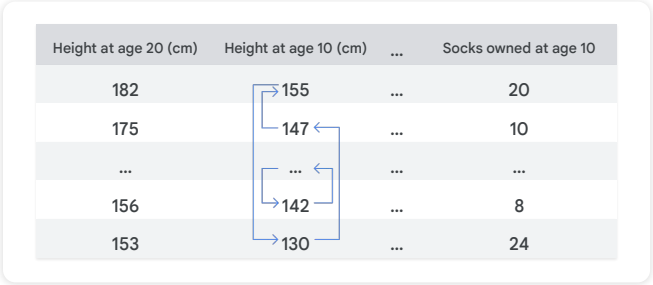
- Global techniques:
  - Permutation Feature Importance
  - Partial Dependence Plots
- Local techniques:
  - LIME
  - Shapley Values
  - Integrated gradients
  - XRAI

and local methods such as LIME, Shapley Values, integrated gradients, and XRAI.

Permutation Feature Importance:

post-hoc, global, model agnostic

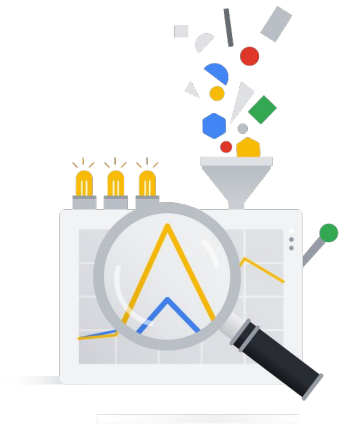
- Randomly shuffles values of a single feature and observes the resulting change in the model's error rate. The higher the increase in error, the more important the feature is considered to be.



First, permutation feature importance. This is a post-hoc, global, model-agnostic method that measures the impact of each feature on a model's performance. It works by randomly shuffling the values of a single feature and observing the resulting change in the model's error rate. The higher the increase in error, the more important the feature is considered to be.

Permutation Feature Importance:

post-hoc, global, model agnostic

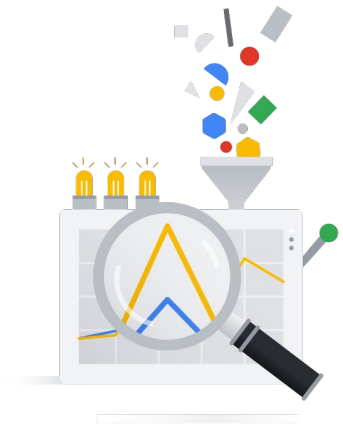


Here is an overview of the process

Permutation Feature Importance:

post-hoc, global, model agnostic

1. Randomly shuffle the feature values.

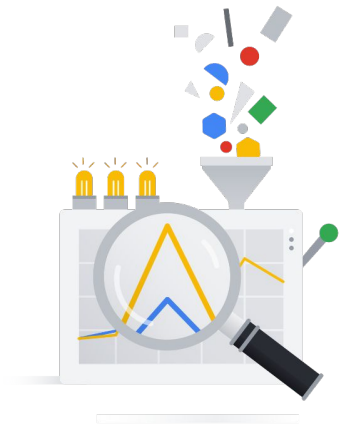


1. For each feature, randomly shuffle its values while keeping all other features unchanged.

Permutation Feature Importance:

post-hoc, global, model agnostic

1. Randomly shuffle the feature values.
2. Make predictions with the shuffled feature.

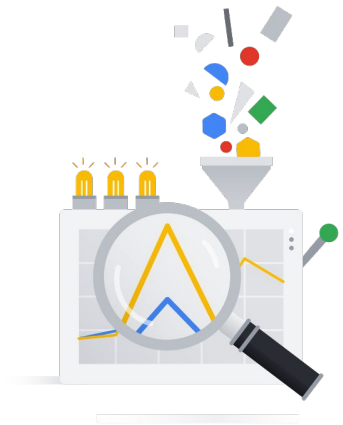


2. Use the shuffled feature values to generate predictions.

Permutation Feature Importance:

post-hoc, global, model agnostic

1. Randomly shuffle the feature values.
2. Make predictions with the shuffled feature.
3. Compute the new error.



3. Calculate the error rate of the model with the shuffled feature.

Permutation Feature Importance:

post-hoc, global, model agnostic

1. Randomly shuffle the feature values.
2. Make predictions with the shuffled feature.
3. Compute the new error.
4. Compute the difference in error.

4. Compare the new error to the original error rate to determine how much the model's performance was affected by shuffling the feature.

Increased model error after shuffling = important
Little to no error difference after shuffling = unimportant

Based on this, then:

A feature is considered “important” if randomly shuffling it increases the model error. This indicates that the model relied heavily on that feature for making accurate predictions.

A feature is considered “unimportant” if randomly shuffling its values results in little to no error differences. This indicates the model wasn’t heavily influenced by the feature in the first place.

So, in summary, the importance of a feature is determined by the magnitude of the error increase caused by shuffling its values.

Permutation feature technique can be intuitive and is easy to implement, but sometimes it can be misleading. Randomly shuffling a feature can create artificial patterns in the data, and potentially cause misleading results. Additionally, the random nature of this method can cause the importance scores to vary significantly across different trials.

Partial Dependence Plots (PDPs):

post-hoc, global, model agnostic

- Used to visualize the relationship between a model's predictions and the values of specific input features

Partial dependence plots (or PDPs) are a post-hoc, global, and model-agnostic technique used to visualize the relationship between a model's predictions and the values of specific input features.

Partial Dependence Plots (PDPs):

post-hoc, global, model agnostic

- Used to visualize the relationship between a model's predictions and the values of specific input features
- Show how the model's predictions change as we vary the values of one input feature while holding all other features constant.

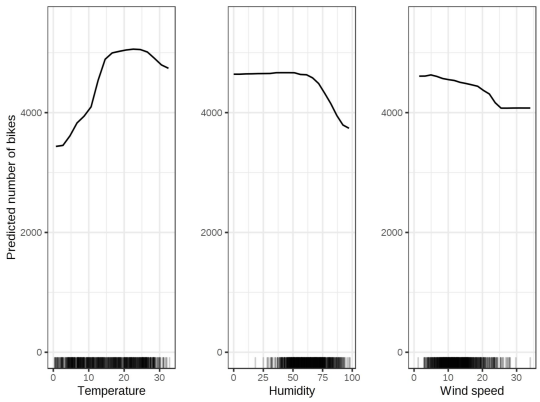
They show how the model's predictions change as we vary the values of one input feature while holding all other features constant.

Partial Dependence Plots (PDPs):

post-hoc, global, model agnostic

- Used to visualize the relationship between a model's predictions and the values of specific input features
- Show how the model's predictions change as we vary the values of one input feature while holding all other features constant.

feature distributions plot



More specifically, the partial dependence function at a particular feature value represents the average prediction if we force all data points to assume that feature value.

PDPs often visualize feature distributions in plots, which can show areas in feature space with limited data.

While it is possible to visualize PDPs for multiple feature interactions, humans can have difficulty interpreting high-dimensional visualizations. Therefore, PDPs are typically limited to two interacting features, one on each axis.

Partial Dependence Plots (PDPs):

post-hoc, global, model agnostic

To do this:

Partial Dependence Plots (PDPs):

post-hoc, global, model agnostic

1. Choose the input feature you want to analyze.

1. Choose the input feature you want to analyze.

Partial Dependence Plots (PDPs):

post-hoc, global, model agnostic

1. Choose the input feature you want to analyze.
2. Systematically change the values of the selected feature

2. Systematically change the values of the selected feature while keeping all other features constant.

Partial Dependence Plots (PDPs):

post-hoc, global, model agnostic

1. Choose the input feature you want to analyze.
2. Systematically change the values of the selected feature
3. Compute the average predicted outcome

3. For each feature value, compute the average predicted outcome across all data points. This represents the partial dependence of the prediction on that feature value.

Partial Dependence Plots (PDPs):

post-hoc, global, model agnostic

1. Choose the input feature you want to analyze.
2. Systematically change the values of the selected feature
3. Compute the average predicted outcome
4. Plot the feature values

4. Plot the feature values on the x-axis and the corresponding average predictions on the y-axis. This visualizes the partial dependence relationship.

PDPs provide a valuable tool for understanding the relationship between input features and model predictions. They can help identify important features, detect nonlinear relationships, and uncover potential biases in the model.

Local Interpretable Model-Agnostic Explanations (LIME):

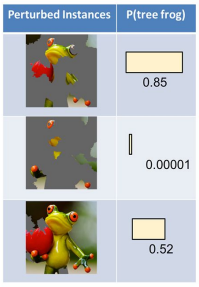
post-hoc, local, model agnostic

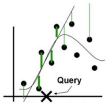
- Creates an explanation by approximating the underlying model locally, with an interpretable one.
- A linear model or a decision tree is often used.

Original Image

Interpretable Components

Original Image
P(tree frog)=0.54





Locally weighted regression

Explanation

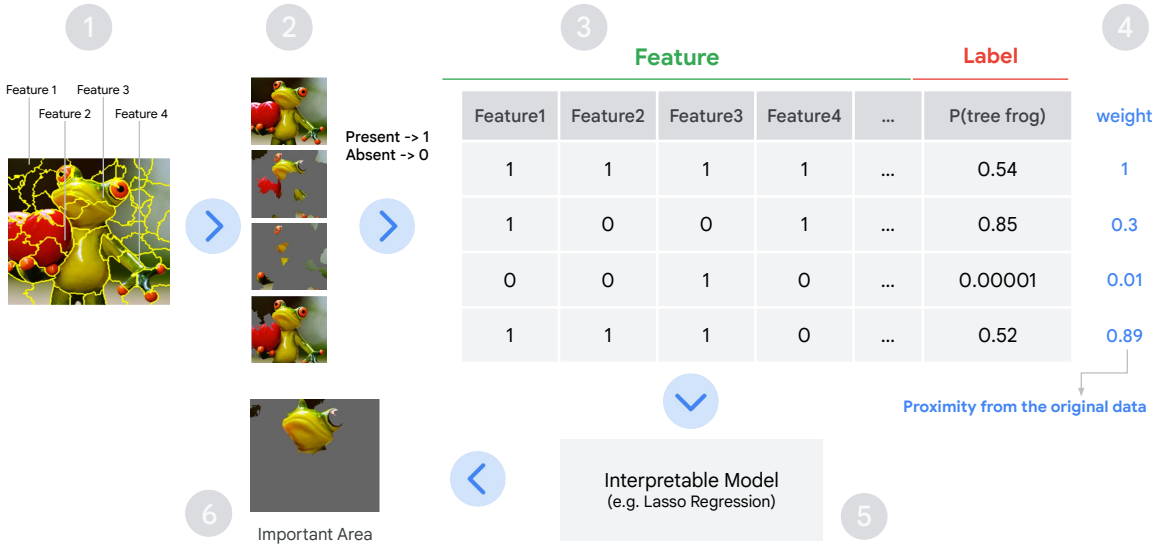
Sources: Marco Tulio Ribeiro, Pixabay

Now let's take a look at local methods.

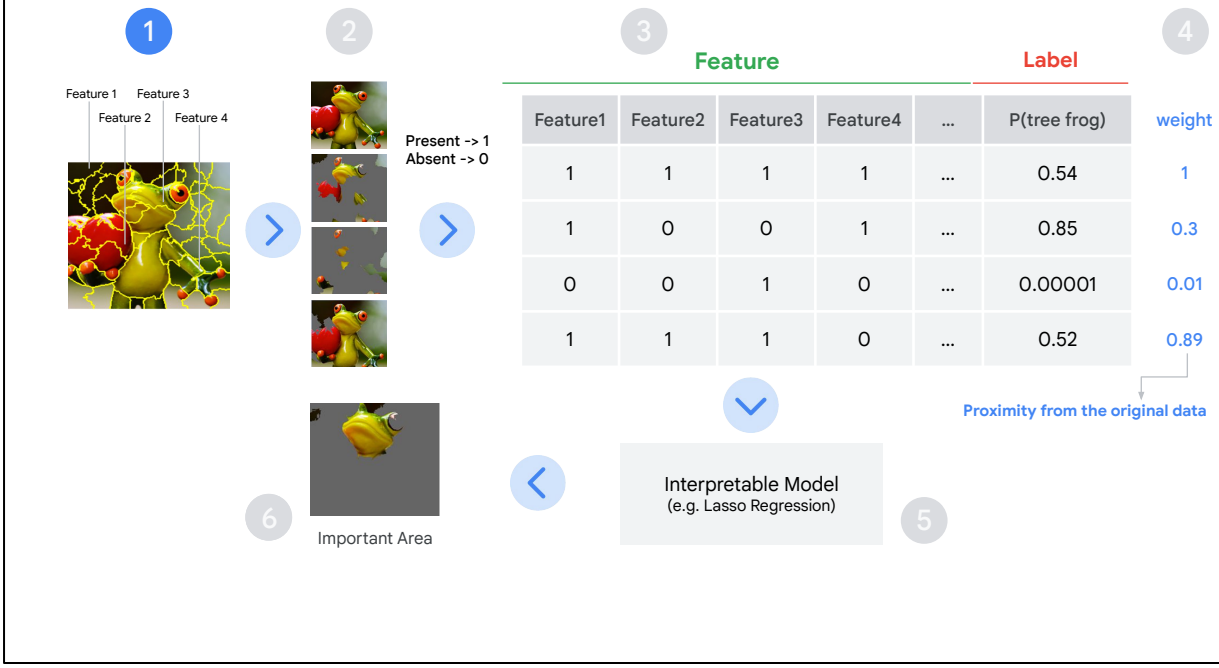
First is LIME, which stands for Local Interpretable Model-Agnostic Explanations. This has been popular for a number of years.

LIME is post-hoc, local, and model-agnostic.

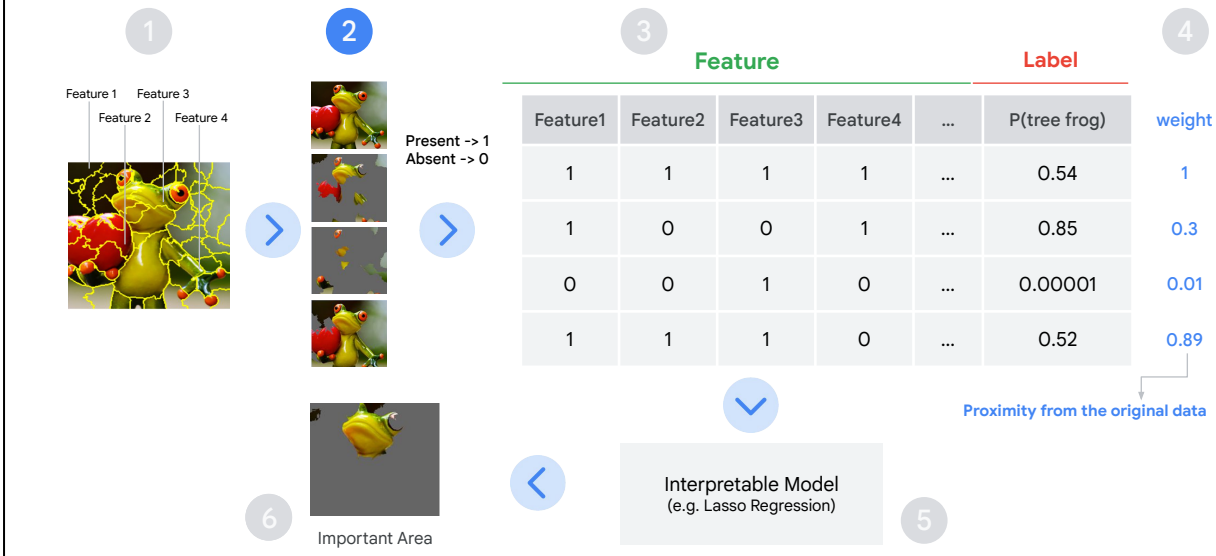
LIME creates an explanation by approximating the underlying model locally, with an interpretable one. We usually use a linear model or a decision tree.



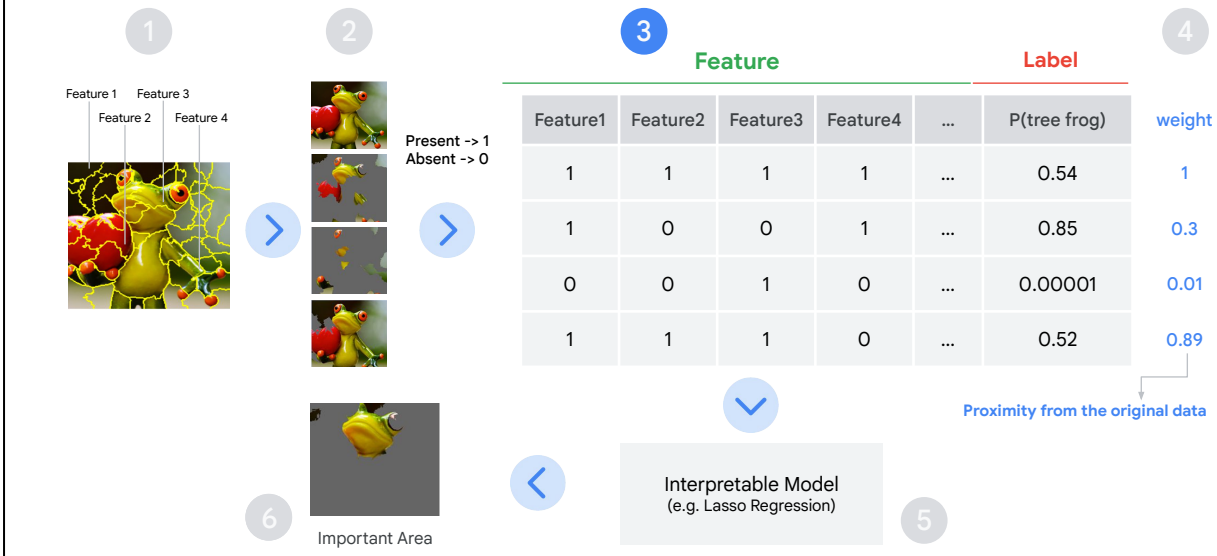
The process for the LIME method looks like this:



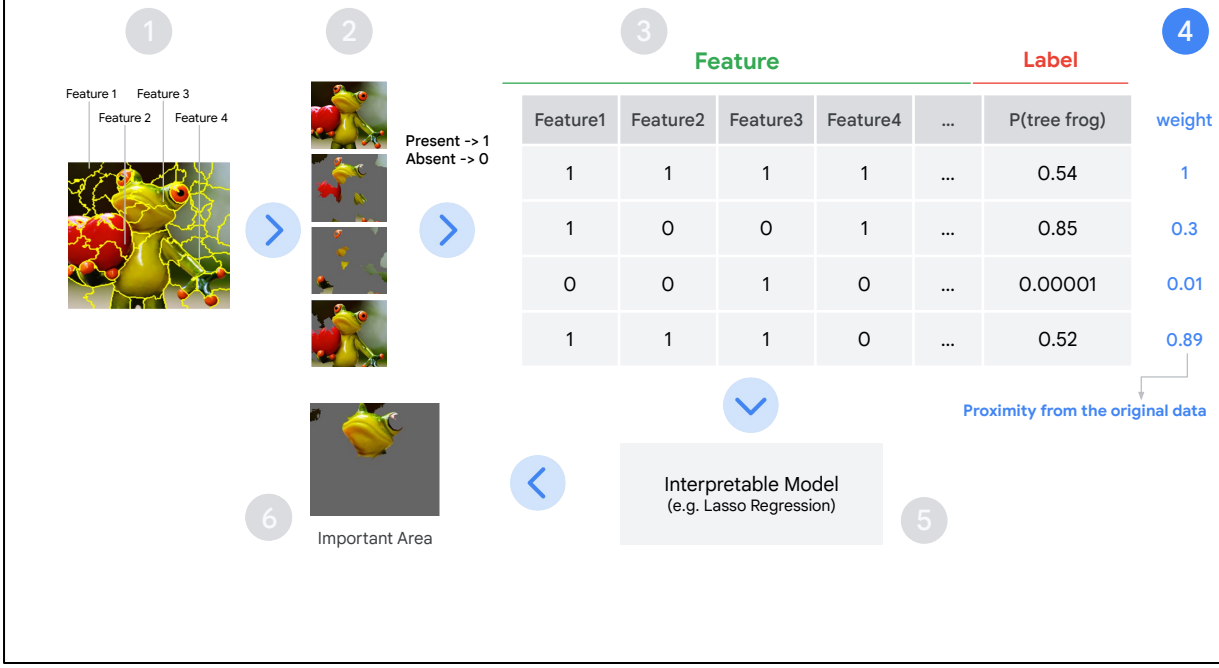
1) Select your instance of interest, for example, an image of a tree frog. In image case, split the image into super-pixels, which are a subset of segments.



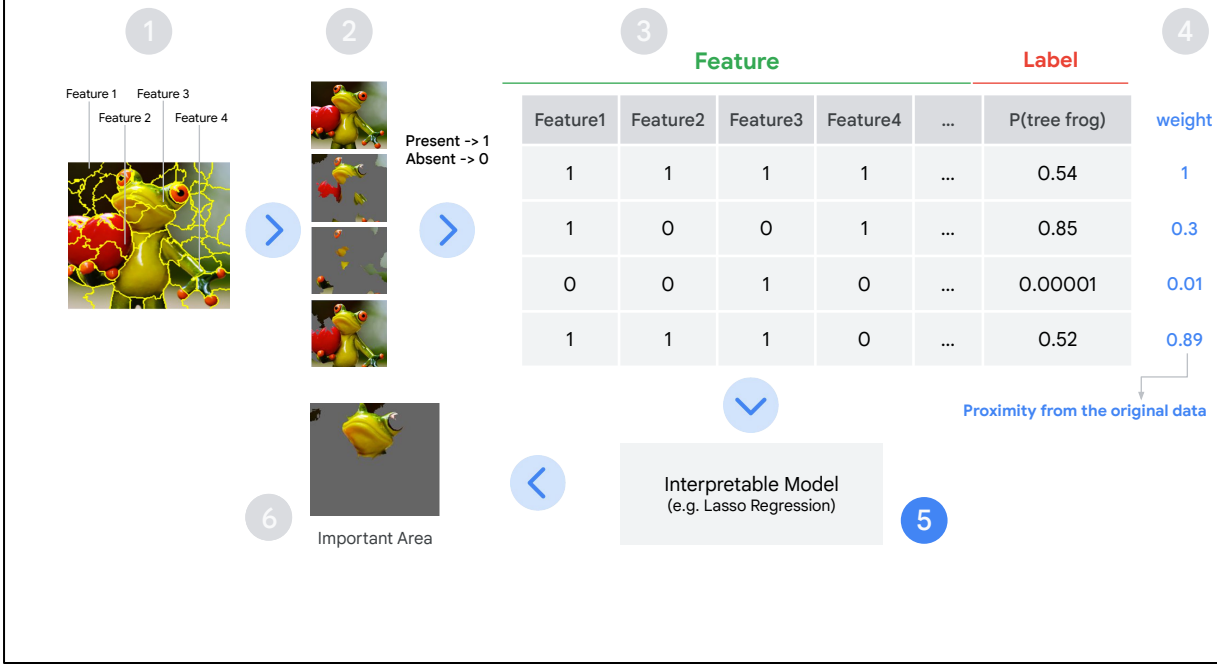
2) Generate perturbations of the input image by “hiding” some features or image super-pixels.



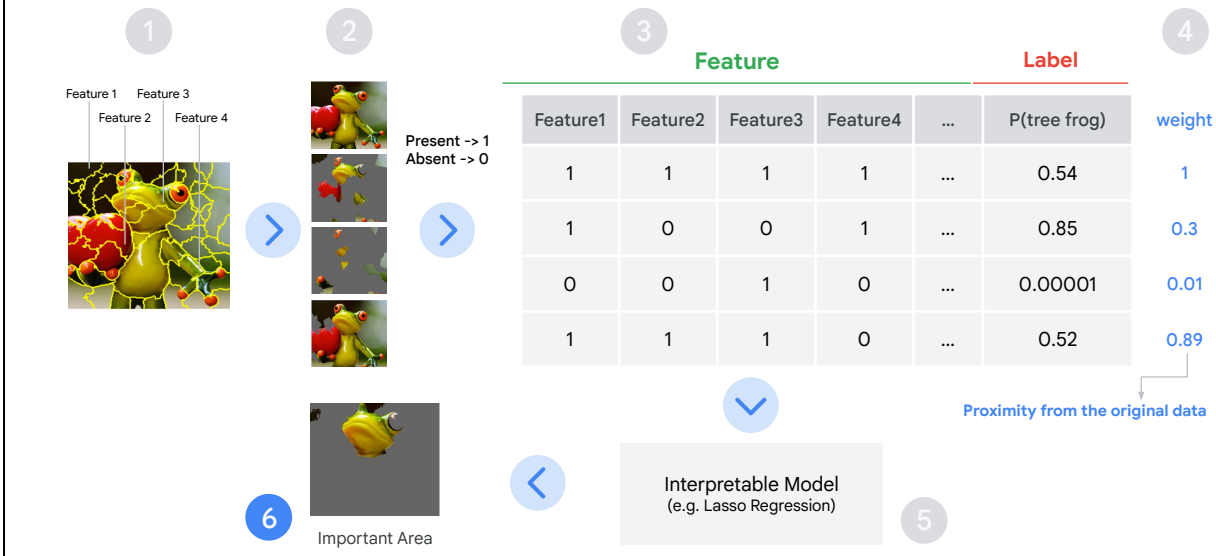
3) For each perturbation, use a target model, such as a deep neural network model we want to interpret, in order to predict the output. Construct a new dataset where each input represents the presence or absence of a feature, and the corresponding output is the predicted probability of the perturbation being a frog.



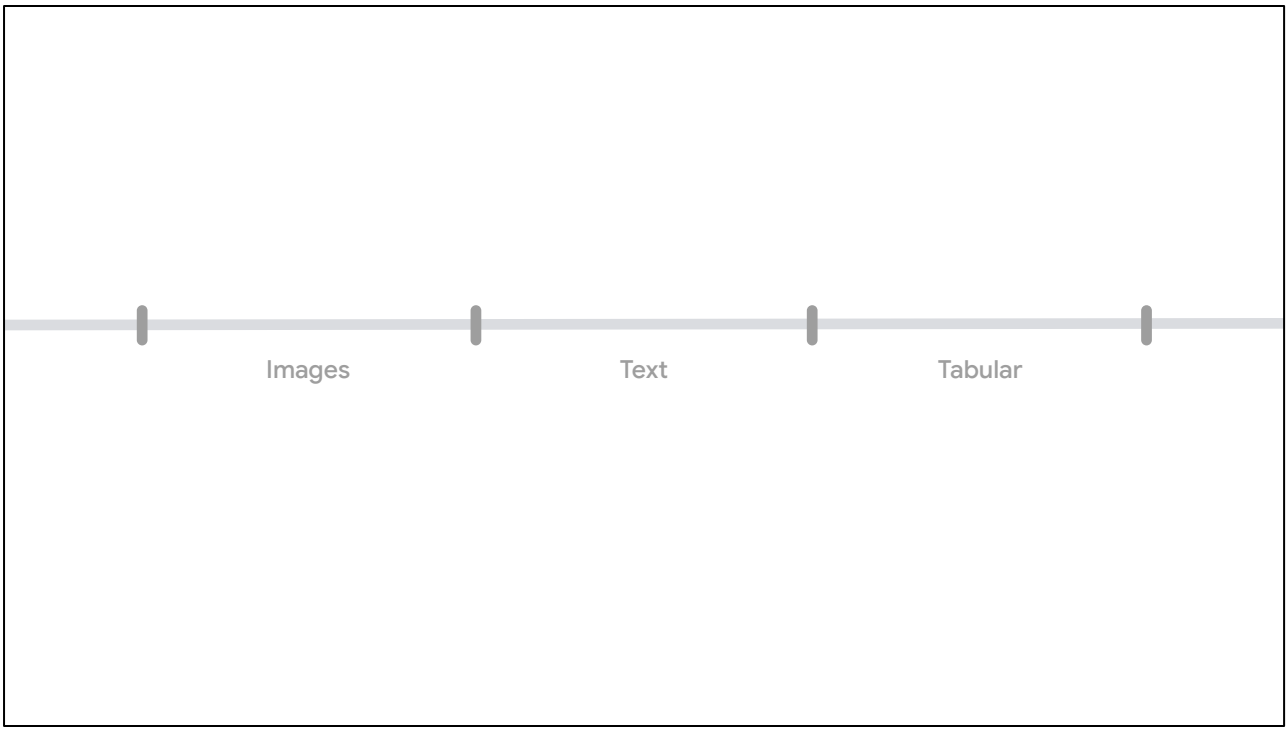
4) Calculate weights for each perturbation based on a distance metric. This typically measures the difference between the perturbed image and the original image, such as the cosine distance, on the dataset constructed in the step above.



5) Train an intrinsically interpretable model, such as a decision tree or linear regression, using the newly constructed dataset of perturbations.



6) Interpret the trained model to identify which areas of the image have a stronger association with the prediction of a tree frog. This can be achieved by examining the model's coefficients or decision rules to understand how it weighs different features in the perturbed images.

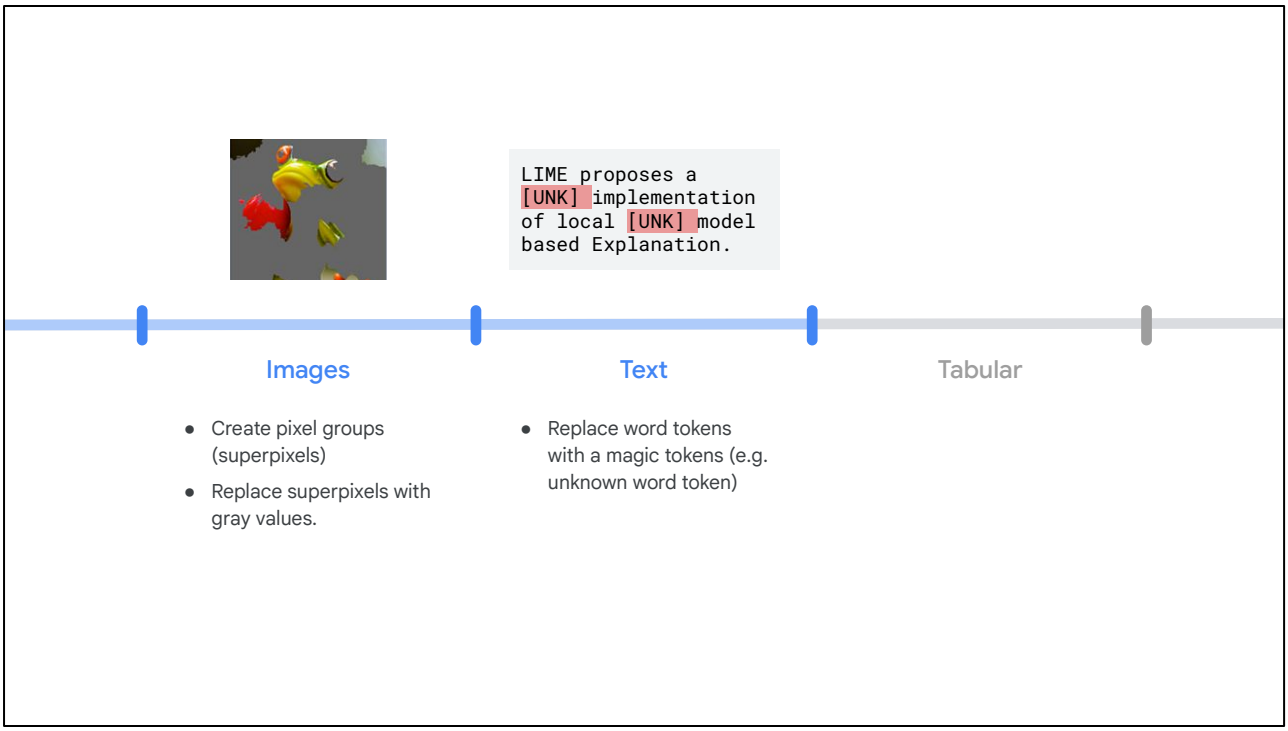


Interpreting a simpler surrogate model instead of a complex one is an intriguing approach. However, the perturbation procedure varies depending on the data type:

Images | Text | Tabular

- Create pixel groups (superpixels)
- Replace superpixels with gray values.

For images: Replace superpixels with meaningless values, such as gray values.



For text: Substitute words with a special token, like an UNKNOWN word token, to eliminate meaning.

```
LIME proposes a
[UNK] implementation
of local [UNK] model
based Explanation.
```

Temp	Humid	Wind
32	52	19
32	74	28

Images

- Create pixel groups (superpixels)
- Replace superpixels with gray values.

Text

- Replace word tokens with a magic tokens (e.g. unknown word token)

Tabular

- Sample from a normal distribution with mean and standard deviation taken from the feature
- Weights (proximity) are computed with exponential smoothing kernel

For tabular data: Sample data from a feature distribution and calculate weights based on similarity, instead of hiding information.

Shapley Values:

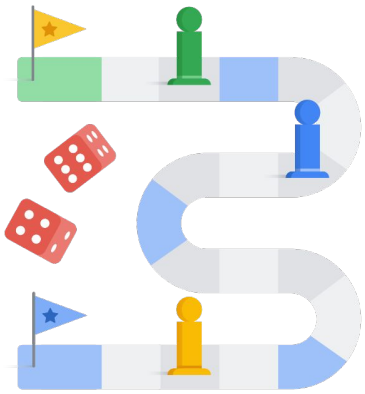
post-hoc, local, model agnostic

Next let's talk about Shapley values.

Shapley Values:

post-hoc, local, model agnostic

- Shapley values come from an area of mathematics known as **cooperative game theory**

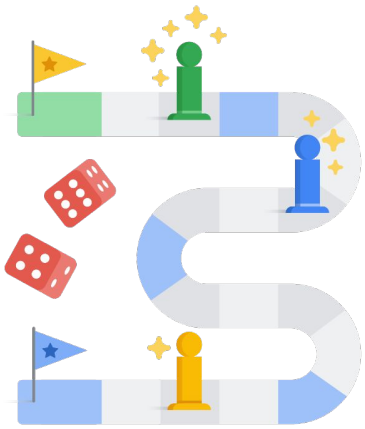


Shapley values are a concept from cooperative game theory. It has been adapted to explain the contribution of individual features to a machine learning model's predictions. In game theory,

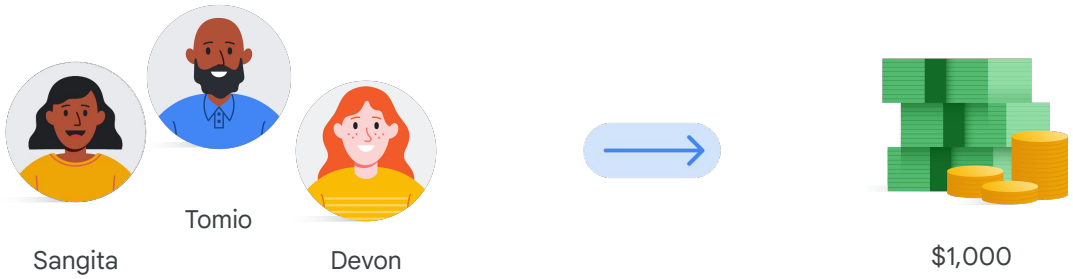
Shapley Values:

post-hoc, local, model agnostic

- Shapley values come from an area of mathematics known as **cooperative game theory**
- This technique tries to quantify the contribution of each player in a cooperative situation.



Shapley values are used to distribute the payoff of a cooperative game among the players based on the magnitude of the contribution to the game. In machine learning, the players are the features, and the payoff is the attribution score for the model's prediction. Shapley values are a post-hoc technique.



For an understanding of this, let’s look at an example.

Imagine Sangita, Tomio, and Devon form a team for an esports competition, collectively winning $1,000. Recognizing their varying skill levels, they decided to split the prize based on their individual contributions to the game instead of an equal division.

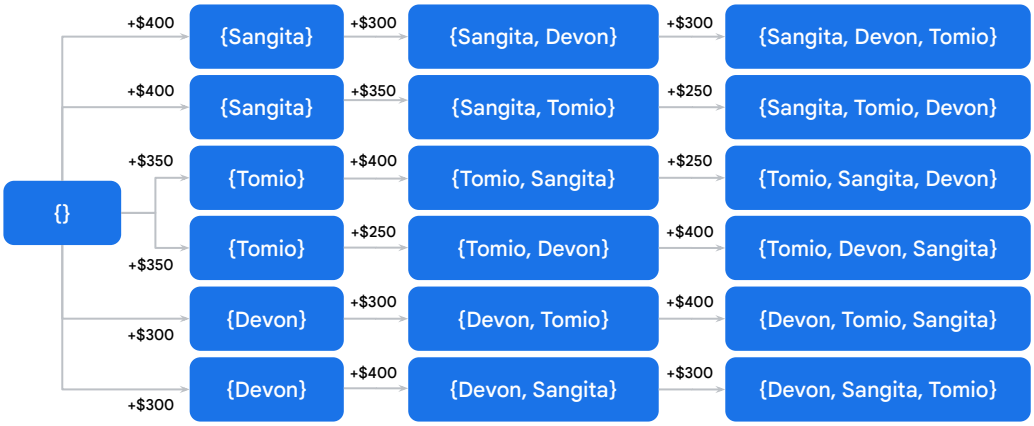
Step 1: Do multiple trials in different team set up

	Team	Result
Case 1	{}	$0
Case 2	{Sangita}	$400
Case 3	{Tomio}	$350
Case 4	{Devon}	$300
Case 5	{Sangita, Tomio}	$750
Case 6	{Sangita, Devon}	$700
Case 7	{Tomio, Devon}	$600
Case 8	{Sangita, Tomio, Devon}	$1,000

Sangita, a mathematician, proposes using Shapley values to quantify each player's contribution. They create a gaming simulator and repeatedly play the same game with different team combinations, resulting in varying prize amounts.

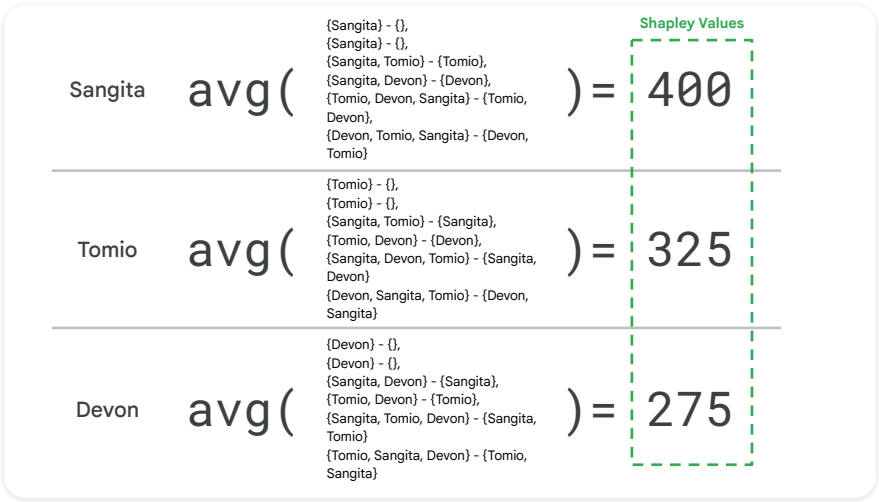
The Shapley value method for this scenario involves changing the combination of feature teams, marginalizing the impact of other features, and calculating the average prediction.

Step 2: Arrange trials as paths



The result tablet is rearranged as a “path” to add players one by one in different orders. In this case, there are six possible paths.
Based on the previous step's result table, the result difference is calculated for each cell along each path. In this case, all differences are positive, indicating that adding a player increases the overall prize. However, in machine learning predictions, differences can be negative, reflecting the influence of features that push predictions towards smaller values.

Step 3: Calculate average contributions



By averaging the result gaps when a player joins a team, we can compute the Shapley value, direction, and magnitude of each player's contribution. Sangita's Shapley value is 400, higher than Tomio and Devon's!

A key characteristic of Shapley values is their "additivity." Thanks to this property, Sangita, Tomio and Devon's Shapley values sum up to $1,000, the amount they earned together. This allows them to easily distribute the prize based on their respective Shapley values.

OK... But what happens if we have 1000 people?

	Team	Result
Case 1	{}	$0
Case 2	{Sangita}	$400
Case 3	{Tomio}	$300
⋮		
Case 1001	{Sangita, Tomio}	$550
Case 1002	{Sangita, Devon}	$700
⋮		
Case 500501	{Sangita, Tomio, Devon}	$950
⋮		
Case 2^{1000}	{Sangita, Tomio, Devon, Angela, Kyle, ...}	$...

Although Shapley values offer a mathematically sound approach to explaining feature importance, their computational complexity poses a practical challenge. Calculating Shapley values for a large number of features becomes computationally challenging as the number of features increases exponentially. For example, computing Shapley values for 1000 features requires running a simulation or making predictions equal to 2 to the power of 1000 times, which is simply not feasible.
So should we avoid using this method if we have many features?

Approximation algorithms of Shapley values

Fortunately, many efficient approximation algorithms of Shapley Values are available, such as:

Approximation algorithms of Shapley values

Sampled Shapley

Approximate Shapley value by sampling (not calculating all the cases) permutations.

Sampled Shapley

Approximation algorithms of Shapley values

Sampled Shapley	Kernel SHAP
Approximate Shapley value by sampling (not calculating all the cases) permutations.	Slightly faster than Sampled sharpley, but it assumes the independence of features.

Kernel SHAP, and

Approximation algorithms of Shapley values

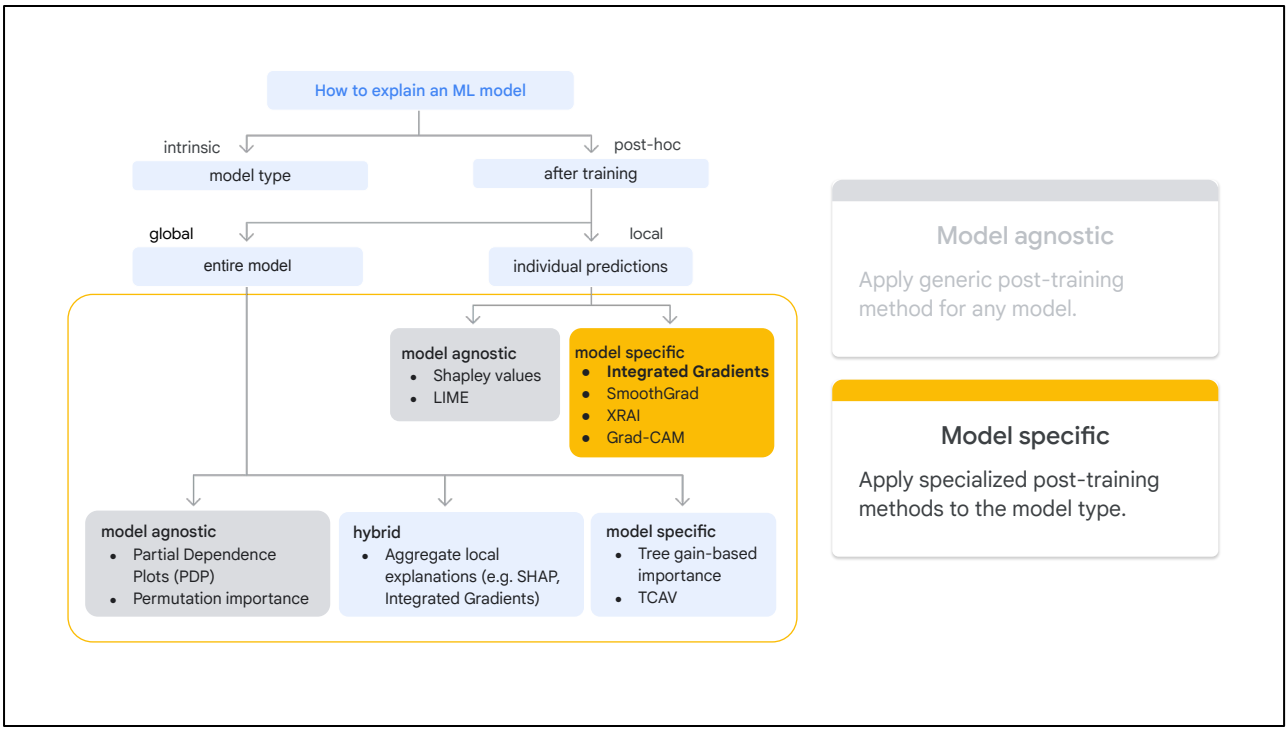
Sampled Shapley	Kernel SHAP	Tree SHAP
Approximate Shapley value by sampling (not calculating all the cases) permutations.	Slightly faster than Sampled sharpley, but it assumes the independence of features.	Faster approximation algorithm, but it can only be applied to Tree-based models. Hence, this technique is model-specific.

Tree SHAP.

Each has slightly different approximation strategies and applications, but are all widely used in many machine learning interpretation use cases.

Let’s look at model-specific approaches, particularly, methods applicable to deep neural network models.

The term “model-specific” might sound inflexible, but, in fact, can provide finer explanations for very complex models.



The first method is Integrated Gradient, or IG. IG is a feature-based post-hoc explanation method that aims to explain the relationship between a model's predictions in terms of its features.

It is designed for differentiable models such as neural networks. IG primarily focuses on explaining individual predictions.

Gradient-based Attribution

Create attribution using gradient of the output with respect to each input feature.

- This gradient is similar to feature weights for linear models
- These attributes can be visualized as a mask on the target image.

$$x_i = x_i \frac{\partial y}{\partial x_i}$$

attribution for feature

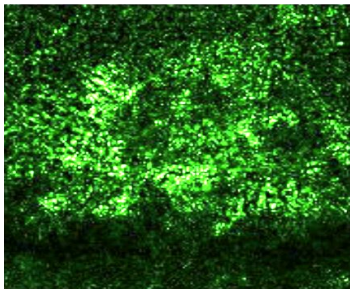
Differentiability is crucial for gradient-based explanations like IG, as it allows us to compute gradients of the model's output with respect to its input features

Neural networks, a common type of differentiable model, employ backpropagation for training. Backpropagation is an algorithm that efficiently propagates the error from the output layer back to the input layer, allowing the network to adjust its parameters in a way that minimizes the error.

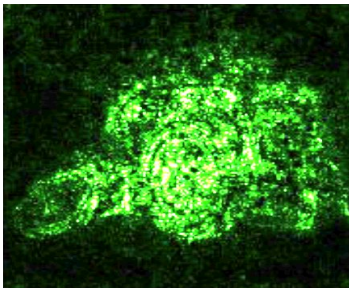
While backpropagation focuses on gradients with respect to model parameters, interpretability techniques like IG require gradients with respect to input features. These gradients represent the sensitivity of the model's output to changes in a particular input feature. A larger gradient indicates a stronger influence of the corresponding feature on the model's output.

In essence, the differentiability of the target model enables gradient-based explanations like IG to quantify the impact of individual input features on model predictions, providing valuable insights into the model's decision-making process.

Original Image (Input)



Vanilla Gradients



Integrated Gradients

Then, why not visualize the gradients over the image? The image in the middle is the gradient mask of a DNN-based image classification model for a camera image.
As you can see, the image appears too blurred to distinguish the classified camera object.

Here lies the issue: As your model "learns" the relationship between the range of an individual pixel and the correct class, the gradient becomes increasingly small. It might even go to zero. This is called saturation.

In comparison, on the right, notice how integrated gradients are much better at identifying the edges of the camera object. In particular, highlighting the pixels around the lens as being important. It captures a better representation of the camera that is more human interpretable.

So how do integrated gradients overcome gradient saturation?

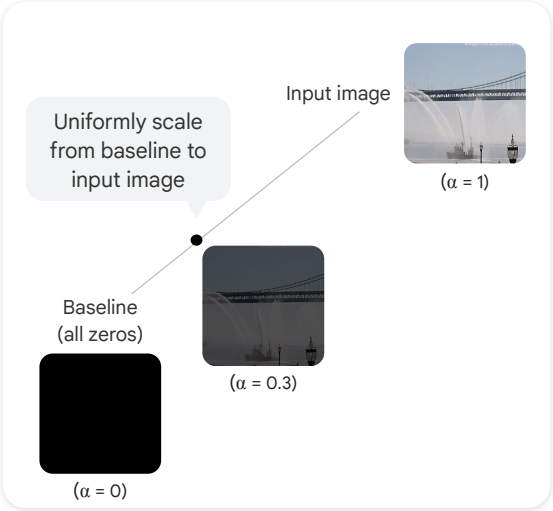
Integrated gradients example

The label for this image was “fire boat”.

Let’s see what integrated gradients will tell us.

Let’s walk through an example.

Let's say we had a model that correctly predicted this image as a fire boat.

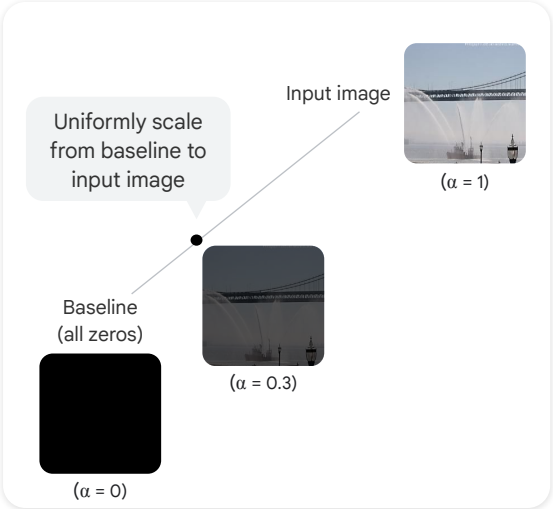


Construct a sequence of images interpolating from baseline (black) to the actual image

Average the gradients across these images

In IG, we first set a baseline image, usually a completely black image.

And we generate a linear interpolation between the baseline and the original image. Interpolated images are small steps (denoted as alpha) in the feature space.



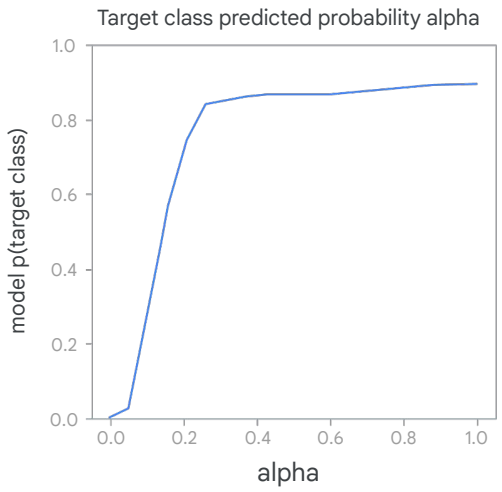
Construct a sequence of images interpolating from baseline (black) to the actual image

Average the gradients across these images

$$IG_i(\text{image}) = \text{image}_i \int_0^1 \nabla F_i(\alpha \cdot \text{image}) d\alpha$$

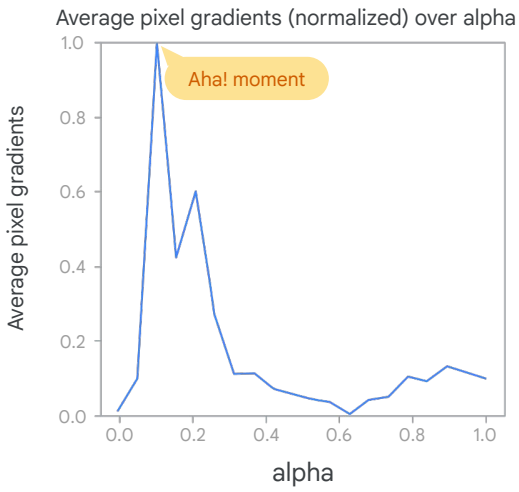
$IG_i(\text{image})$	is the integrated gradient wrt the *i*th pixel i.e. the attribution for the *i*th pixel
F	is the prediction function for the label
image_i	is the intensity of the *i*th pixel

Rather than directly calculating gradients for the original image, we calculate the gradients for images along a path from the original image to a baseline image. Integrating these gradients yields the integrated gradients.

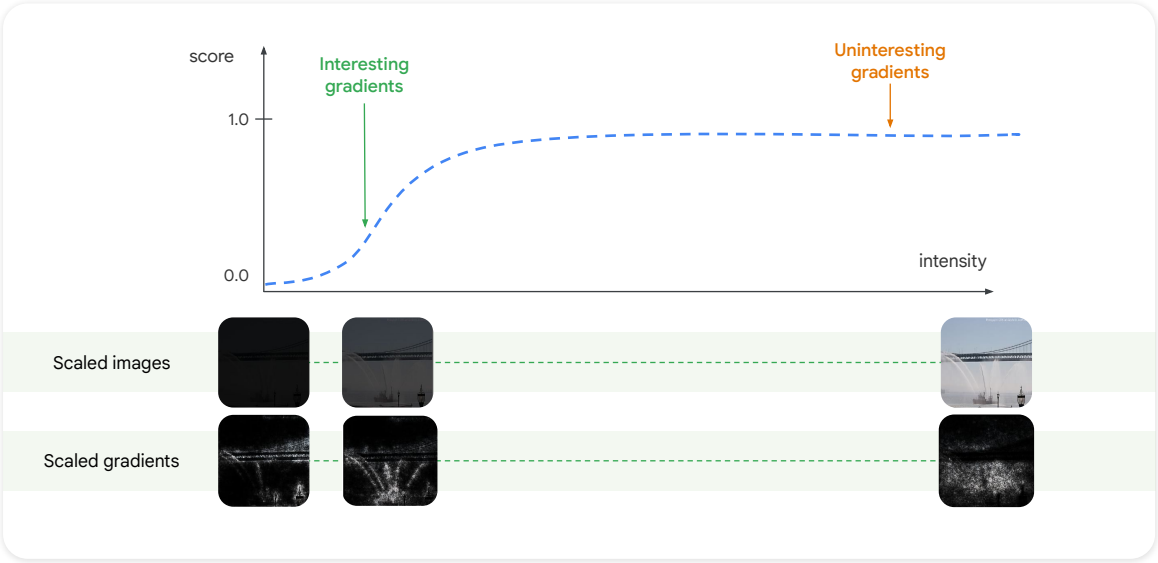


But why do we need to integrate the gradients?
To answer this, let's visualize the gradients from the previous step to connect theory to practice.

The left plot shows how your model's confidence in the "Fireboat" class varies across alphas. Notice how the gradients, or slope of the line, largely flattens or saturates between 0.3 and 1.0 before settling at the final "Fireboat" predicted probability of about 85%.



The right plot shows the average gradient magnitudes over alpha. Note how the values sharply approach and even briefly dip below zero after 0.2. In fact, your model learns the most even at lower values of alpha. Intuitively, you can think of this as your model has learned that the pixels of water cannons to make the correct prediction is at 0.2, sending these pixel gradients to zero after that. However, the model is still uncertain, and is focused on bridge or water jet pixels as the alpha values approach the original input image.



Here are the actual gradient maps on each alpha interpolation, starting with a baseline of an all black image.

We can see the important information, like that of the water canon, is actually captured around alpha 0.2. At alpha 1, large gradients are only looking at the background.

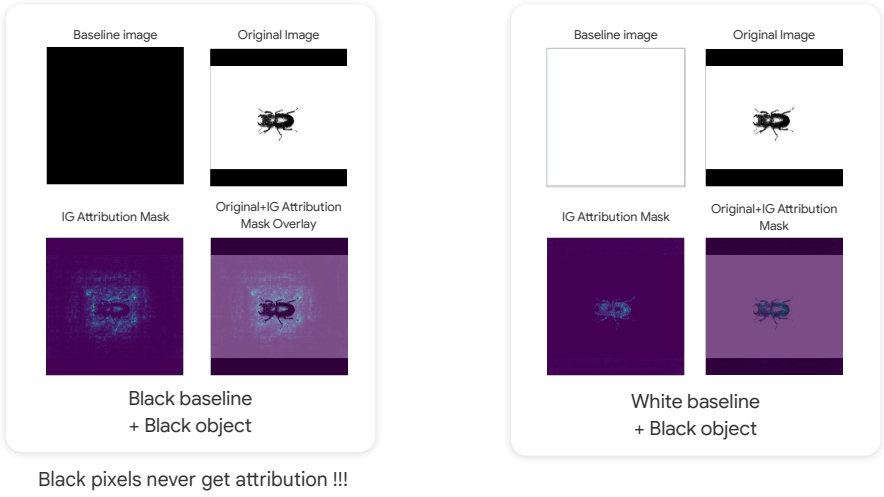
Now we can integrate all the gradients. Practically, we just compute the Riemann sum approximation by using each alpha interpolation, because we don't know this function itself.

The resulting attribution map

Image

Explanation

And here is the result of Integrated Gradient showing the water cannon pixels that captured the “fire boat” class.
However, note that the model is not looking at the boat itself. It's worth checking whether this model works well, even when a fireboat is not using a water cannon.



Black pixels never get attribution !!!

Integrated Gradient also has its drawbacks.

One of the biggest difficulties is related to the baseline image.

The implication of a black baseline is, if black pixels are important to the prediction, they receive no attribution!
As a result, the choice of the baseline plays a central role in interpreting and visualizing IG's pixel feature importances.

On the left, you can see how IG attributions completely miss the solid black beetle in the top image.
On the right, you can see how changing the baseline to a white image corrects the interpretation of the IG attributions.

So how can we improve upon IG and its baseline selection problem?

XRAI improves upon IG

Original image

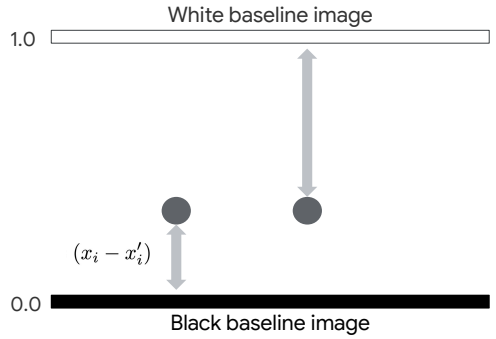
Integrated Gradients

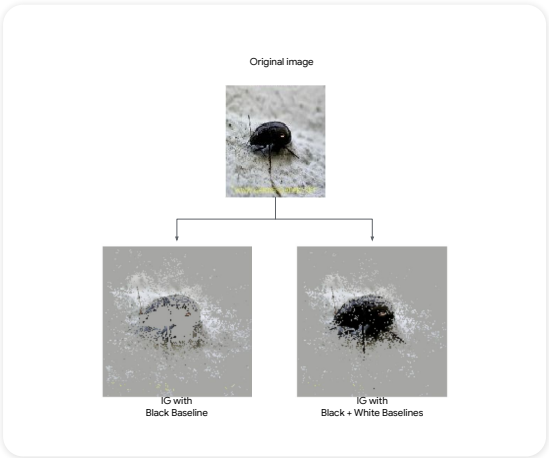
XRAI

Through XRAI, an explainable AI method from Google Research. XRAI solved for the baseline selection problem. XRAI also added a novel region-based attribution method for clearer mask generation.

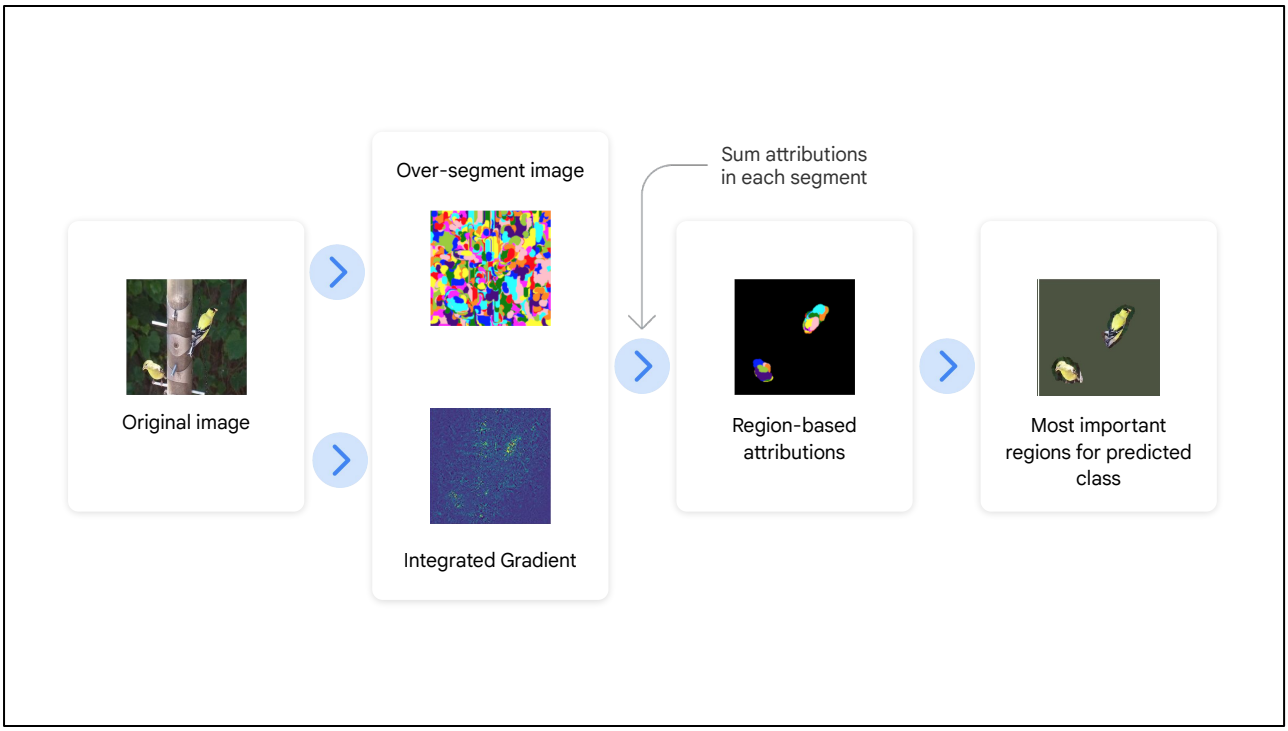
Look at the image on the right. Instead of identifying individually important pixels, XRAI creates and highlights a region of the original image as important.

$$IG_i(x) = (x_i - x_i') \int_{\alpha=0}^{1} \frac{\partial F(x' + \alpha \times (x - x'))}{\partial x_i} d\alpha$$





XRAI overcomes the baseline selection problem by using a black and a white baseline image together to compute IG attributions.



XRAI also improves upon Integrated Gradients (IG) by oversegmenting the image into smaller regions and ranking them based on their IG positive attributions. This approach provides more localized and intuitive explanations for natural images. Although IG remains valuable for domains requiring fine-grained feature explainability such as the medical image domain, XRAI excels in generating intuitive explanations for natural images.

01 Overview of interpretability and transparency

02 Overview of interpretability techniques

03 Feature based explanations: Model agnostic

04 Feature based explanations: Model specific

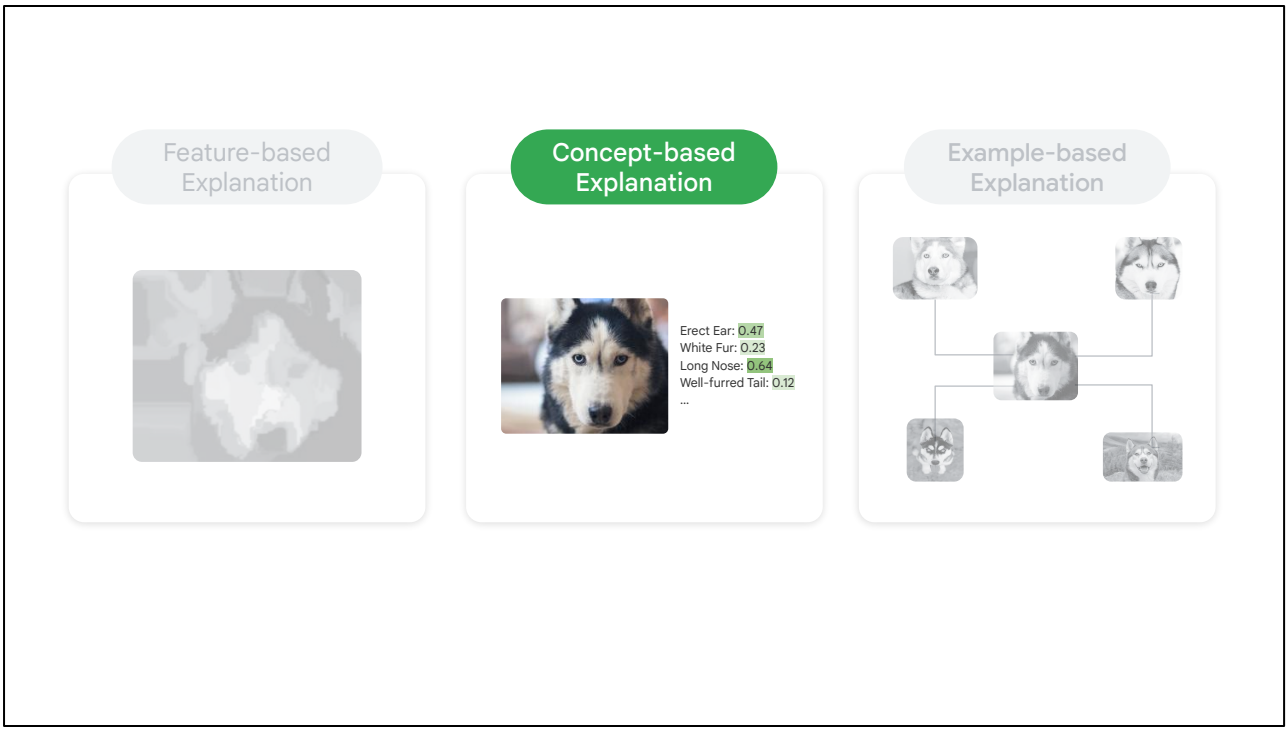
05 Concept-based and example-based explanations

06 Tools for Interpretability

07 Data and Model Transparency

08 Lab: Vertex Explainable AI

Let’s move on to the concept-based explanation.

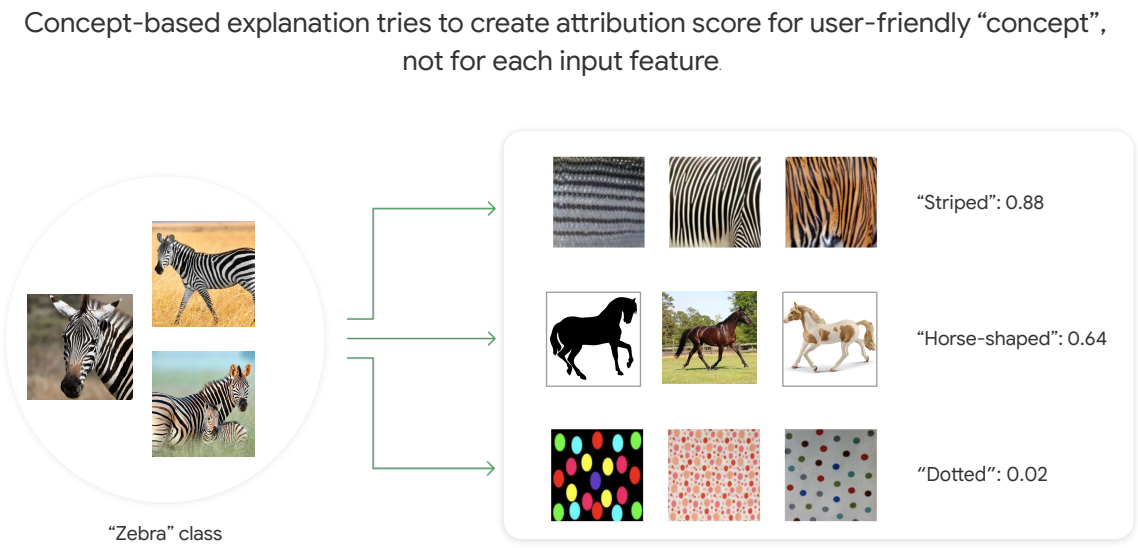


Concept-based explanations are a type of interpretability technique used to understand how machine learning models make predictions through higher-level concepts that are more meaningful to humans, instead of individual input features.

Concept-based explanation tries to create attribution score for user-friendly “concept”, not for each input feature.

“Zebra” class

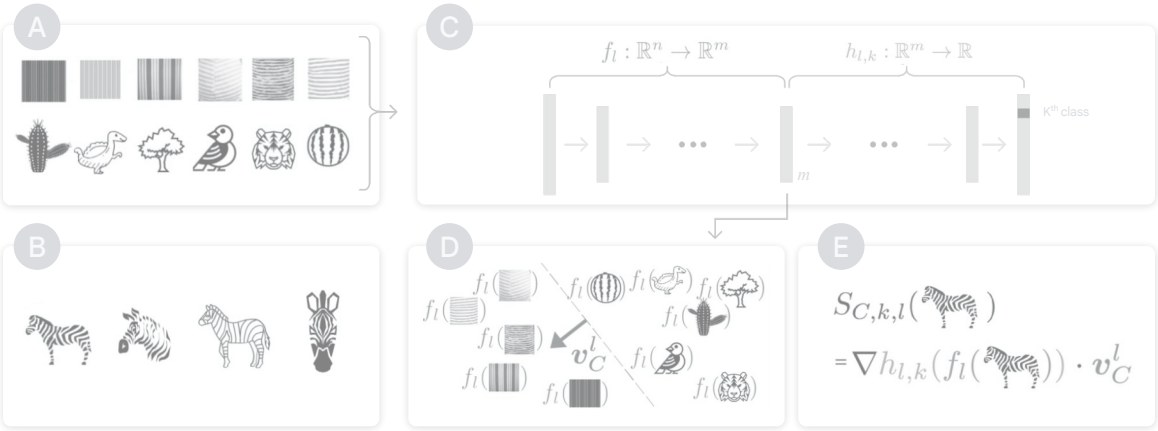
Let’s say we have an image classification model that classifies the “Zebra” class.



If we can get an attribution score for user-friendly concepts, like striped-ness, horse-shaped-ness, that would be intuitive and informative.

Testing with Concept Activation Vectors (TCAV):

post-hoc, global, model-specific

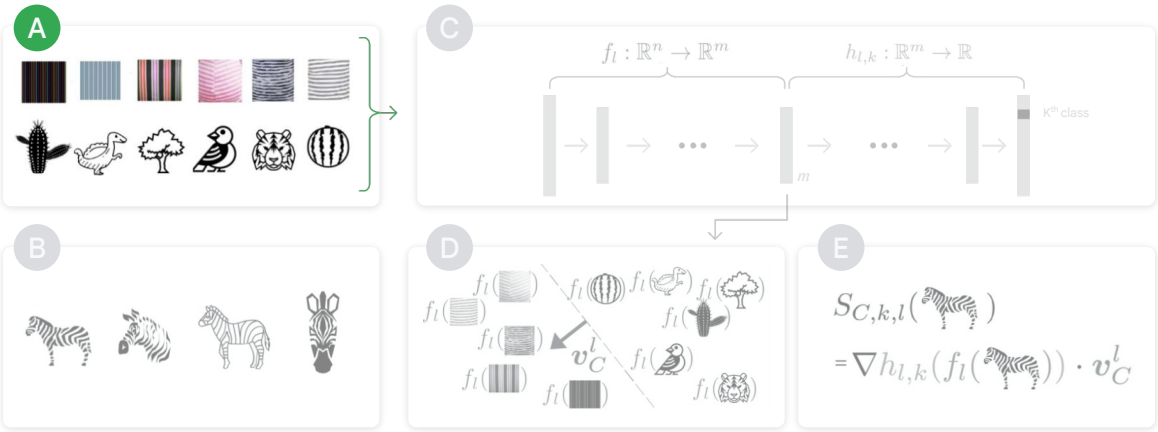


TCAV (Testing with Concept Activation Vectors) stands as a pioneering approach in the field of concept-based explanations. It aims to provide explanations for arbitrary concepts, enabling a deeper understanding of how machine learning models utilize these concepts in their decision-making process.

The TCAV methodology involves the following steps:

Testing with Concept Activation Vectors (TCAV):

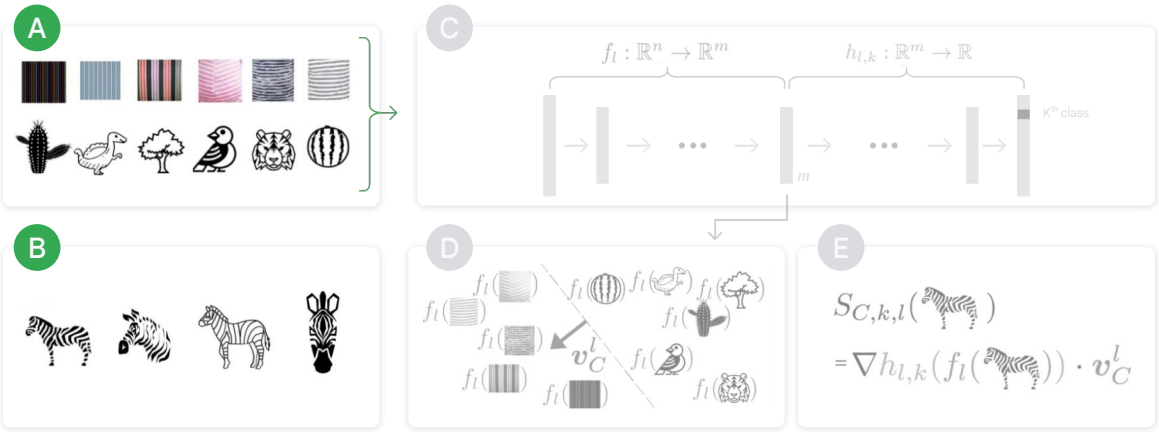
post-hoc, global, model-specific



- Gather a user-defined set of examples that exemplify the concept of interest (e.g., 'striped' patterns) along with a set of random examples.

Testing with Concept Activation Vectors (TCAV):

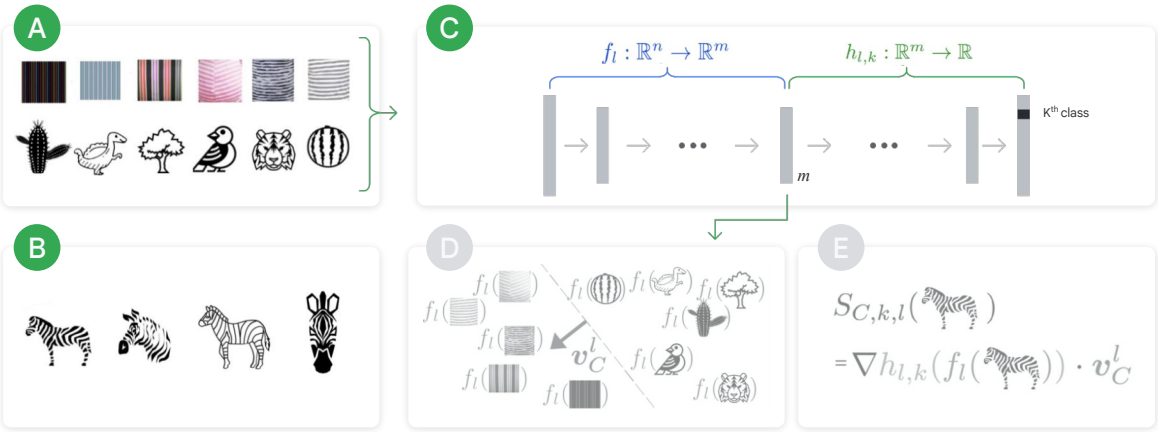
post-hoc, global, model-specific



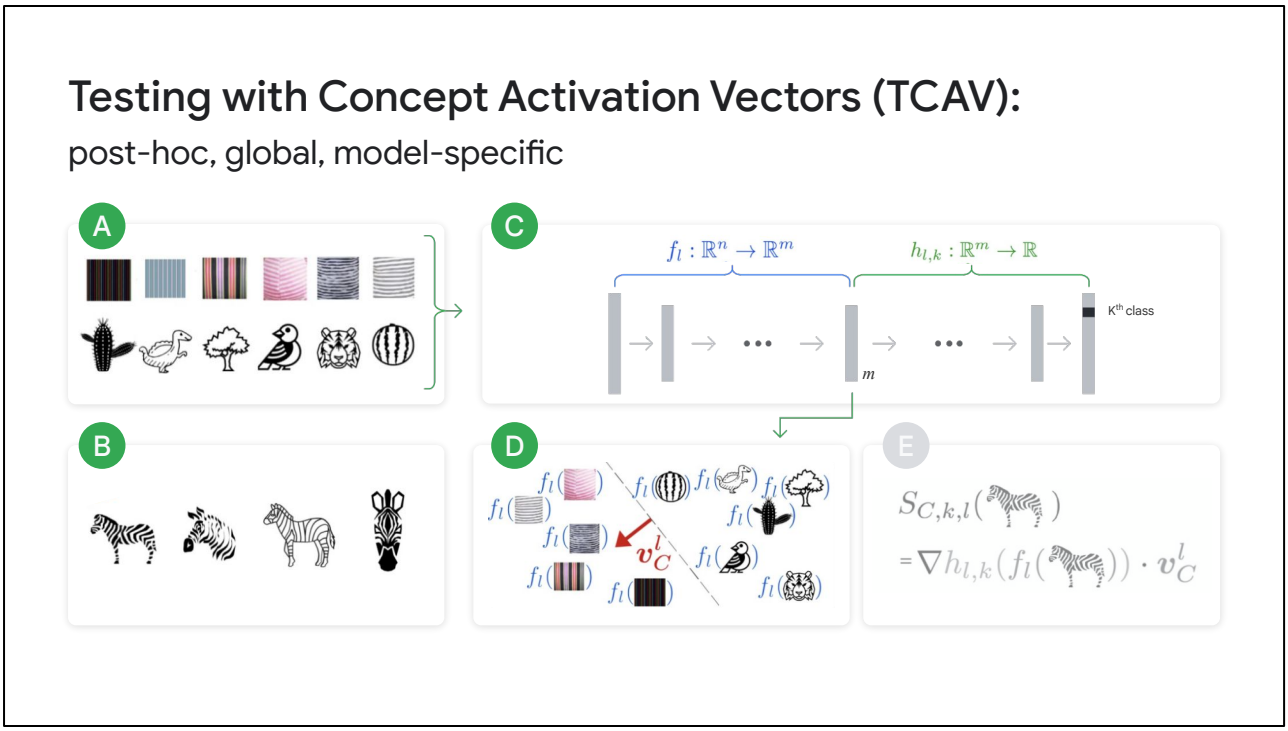
- Collect labeled training data instances representing the target class (for example, “zebras”).

Testing with Concept Activation Vectors (TCAV):

post-hoc, global, model-specific



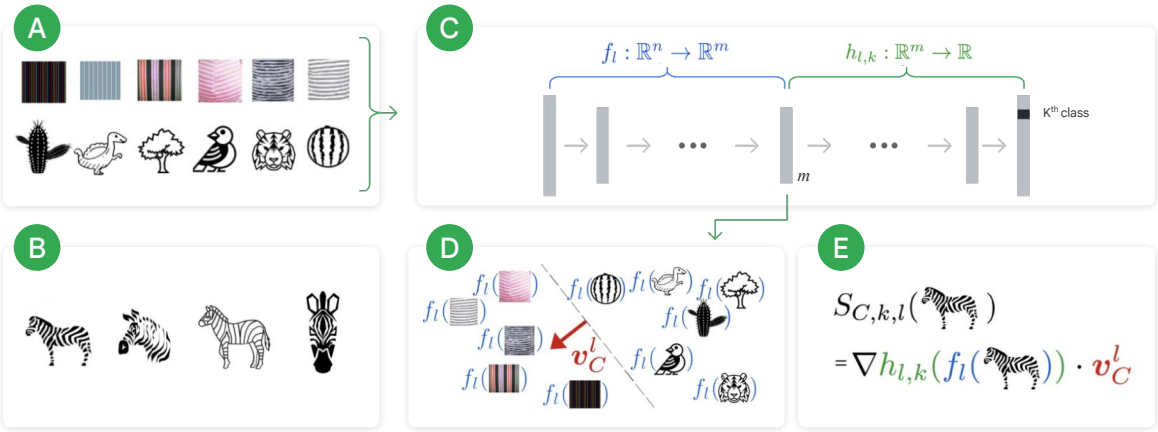
- Employ a trained target neural network to process the prepared concept data.



- In a feature space of an intermediate layer of the model, train a linear classifier t o distinguish between the representations produced by the concept's examples and those from random examples. The Concept Activation Vector (v^l_C) is the vector perpendicular to the classification boundary, and represented by the red arrow.

Testing with Concept Activation Vectors (TCAV):

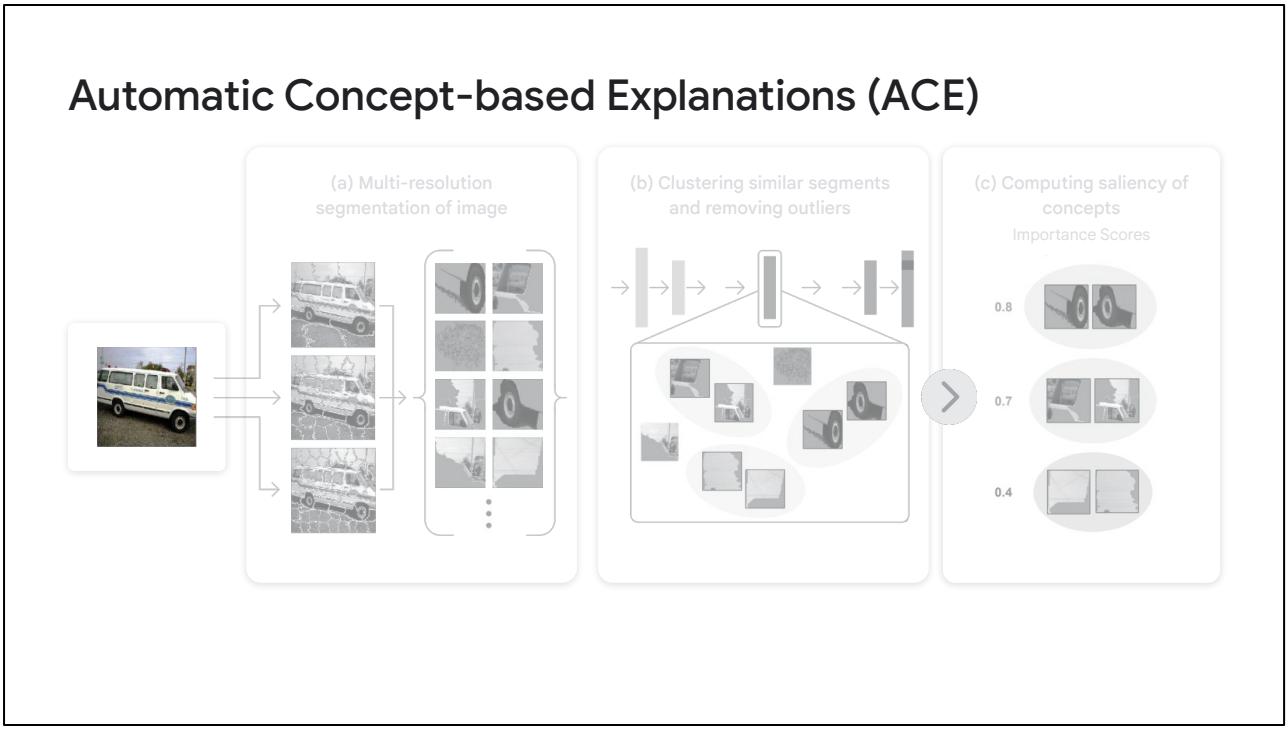
post-hoc, global, model-specific



- For the class of interest (zebras), use the directional derivative to quantify the model's sensitivity to the concept.

By following these steps, TCAV enables the computation of concept attribution for a wide range of concepts, providing valuable insights into how machine learning models rely on these concepts for making predictions.

TCAV is also useful for fairness testing purposes, such as ensuring that a class of “CEO” doesn’t have a stronger concept attribution for “male” concepts over “female” concepts.

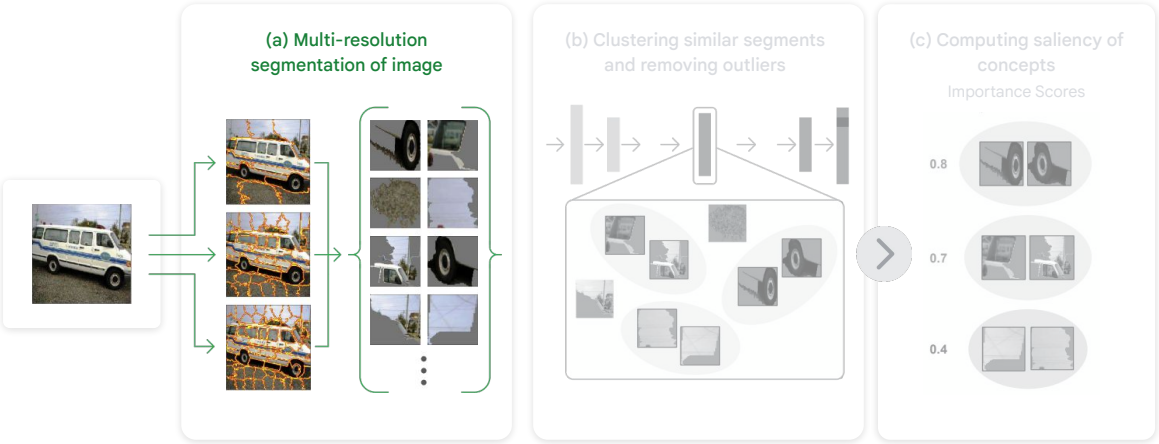


TCAV provides flexibility regarding the choice of concepts. But in some situations, you might want to automate the concept selection process, since creating a dataset for a concept can be complex.

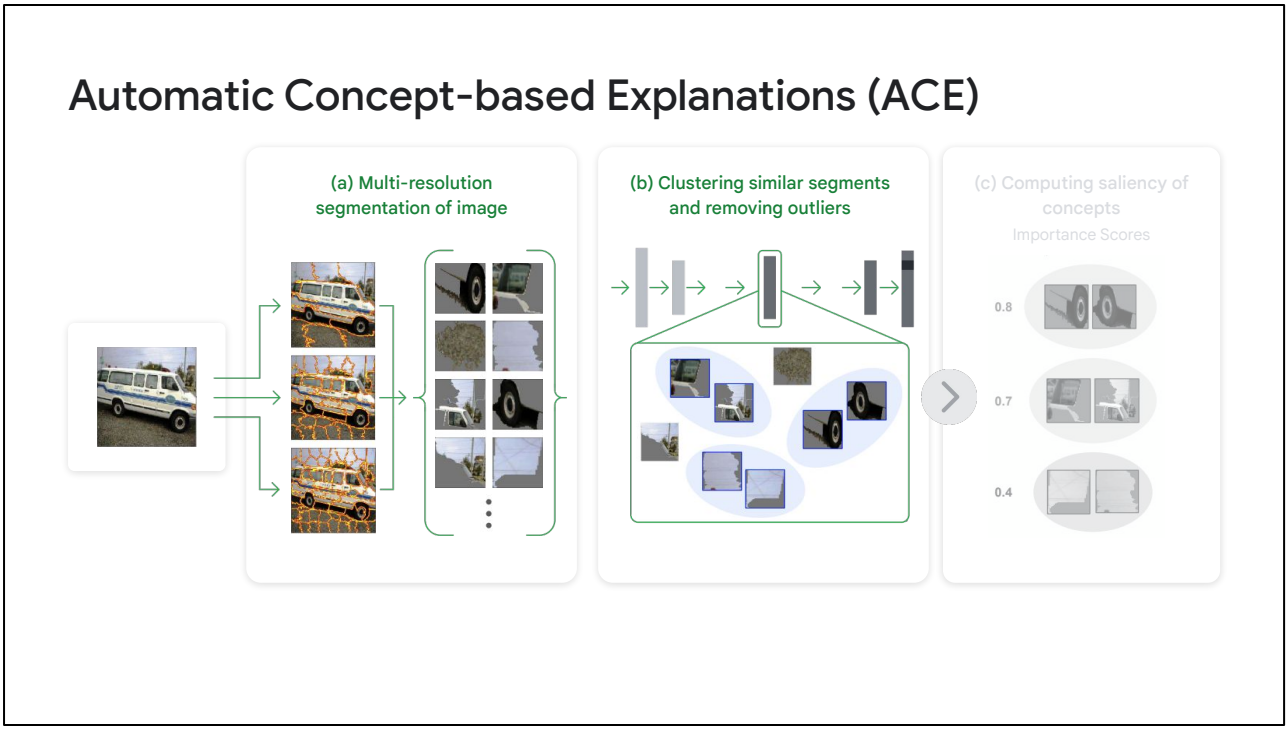
ACE, or Automatic Concept-based Explanation, offers this automation capability.

In this approach, Concept Activation Vectors are automatically created through the following steps:

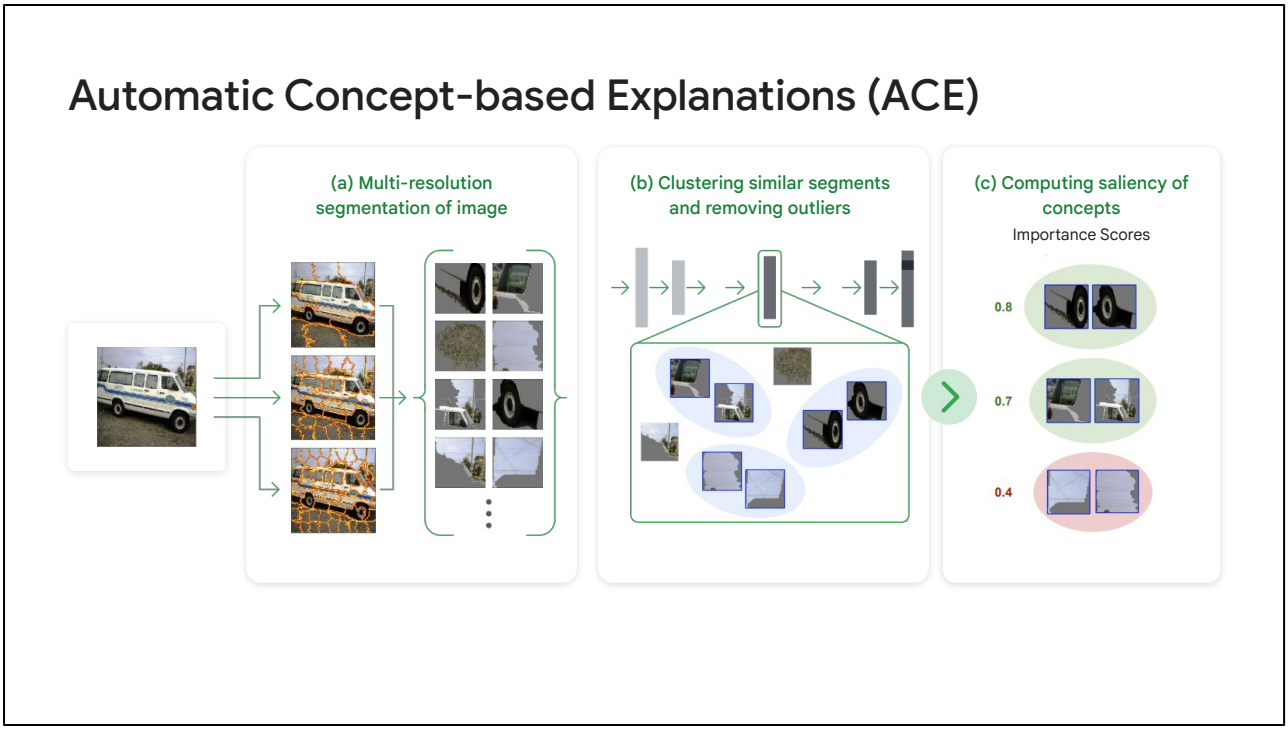
Automatic Concept-based Explanations (ACE)



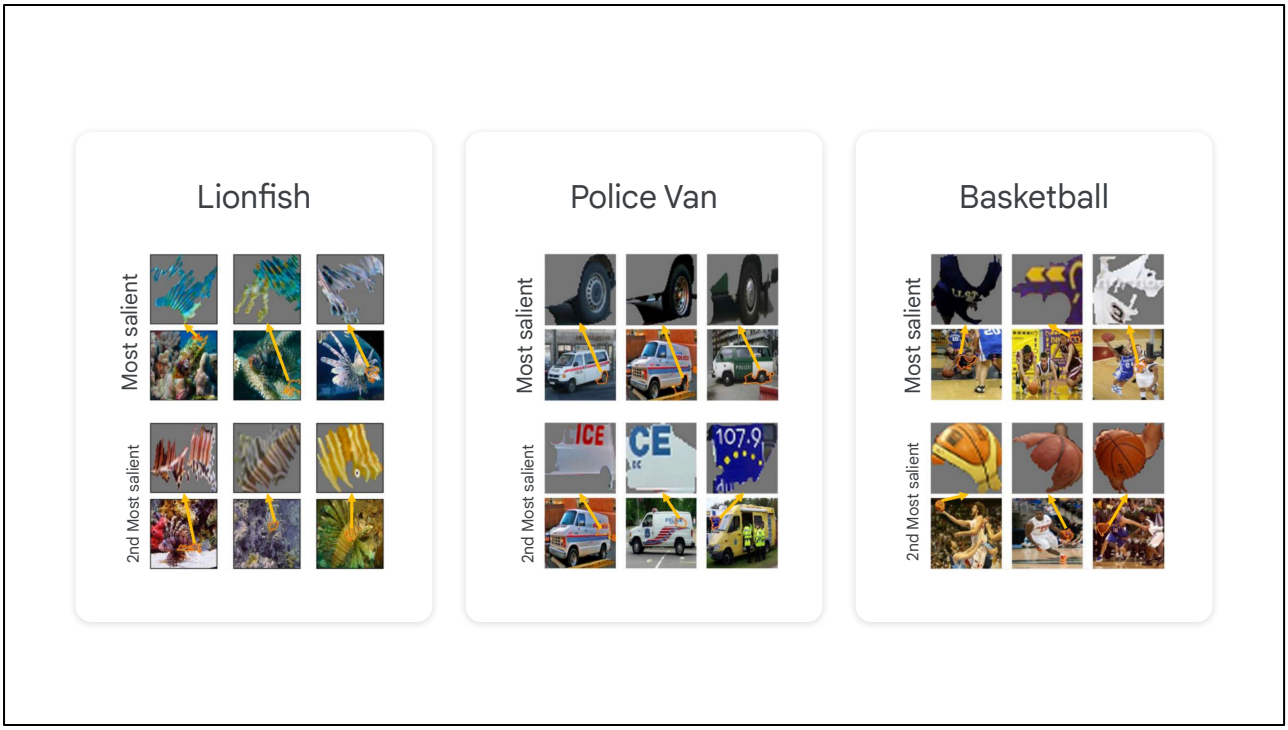
- Split the target image into smaller, cropped segments, then rescale back to the original image resolution to ensure compatibility with the target neural network.



- Apply k-means clustering inside the feature space of an intermediate layer of the target neural network. Each cluster is regarded as an automatically created concept.



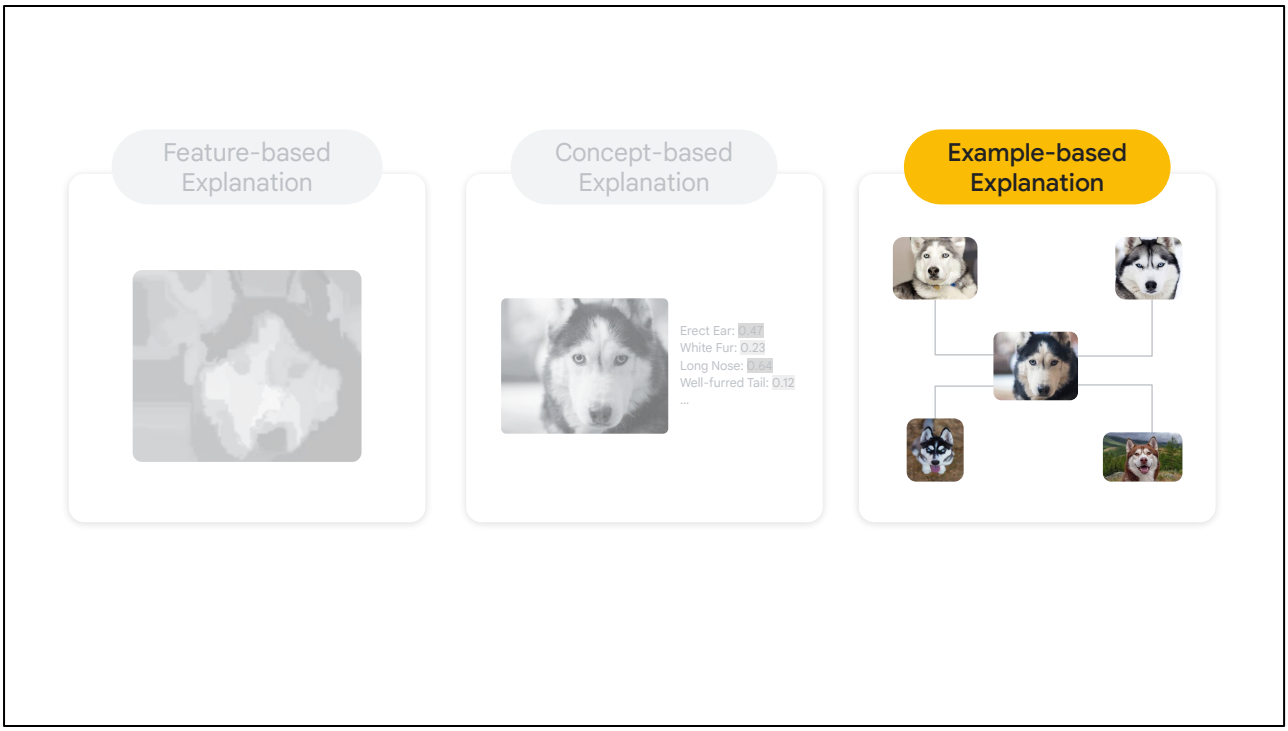
- Calculate concept activation vectors for each cluster to determine the attribution of each concept to the prediction. The calculation of the Concept Activation Vector is the same as TCAV.



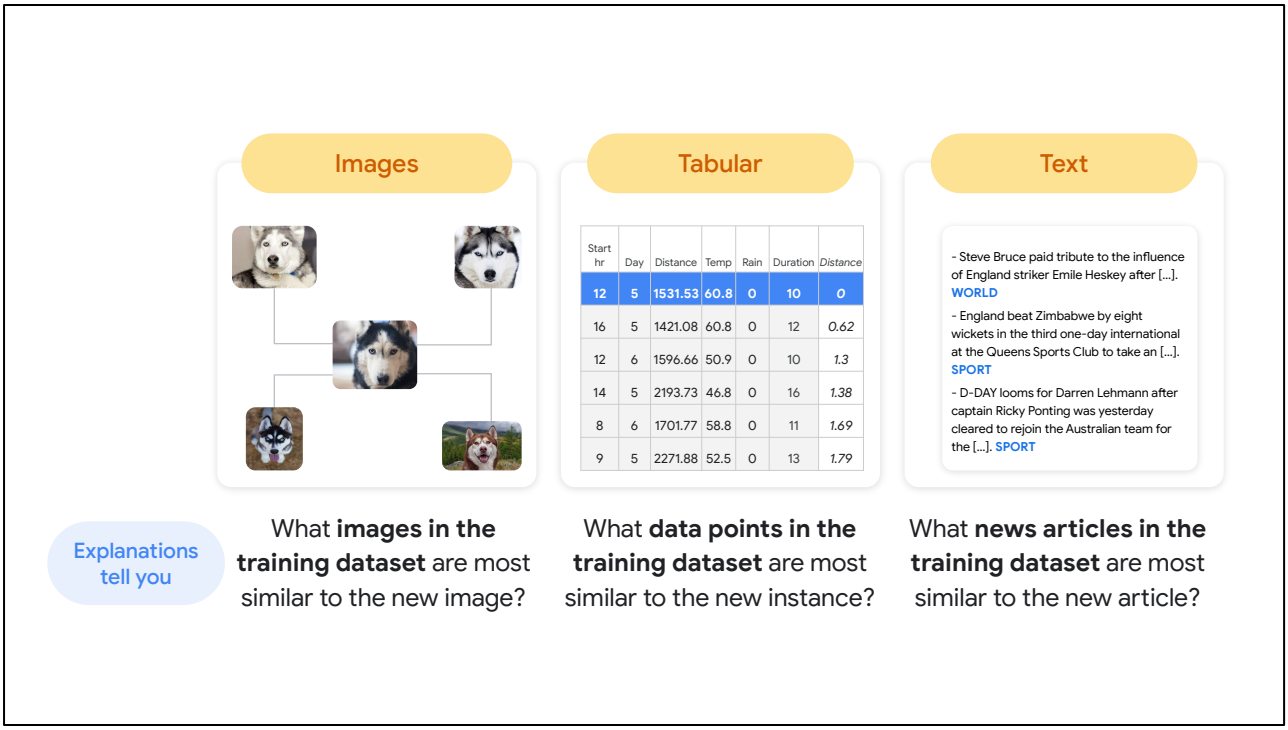
Here are some output examples of the ACE method.

Although it can't provide hand-picked concepts like "striped-ness" or "horse-shaped-ness", it still gives us interpretable results.

For example, it seems like the uniform logo and basketball surface are important concepts for the "basketball" class.



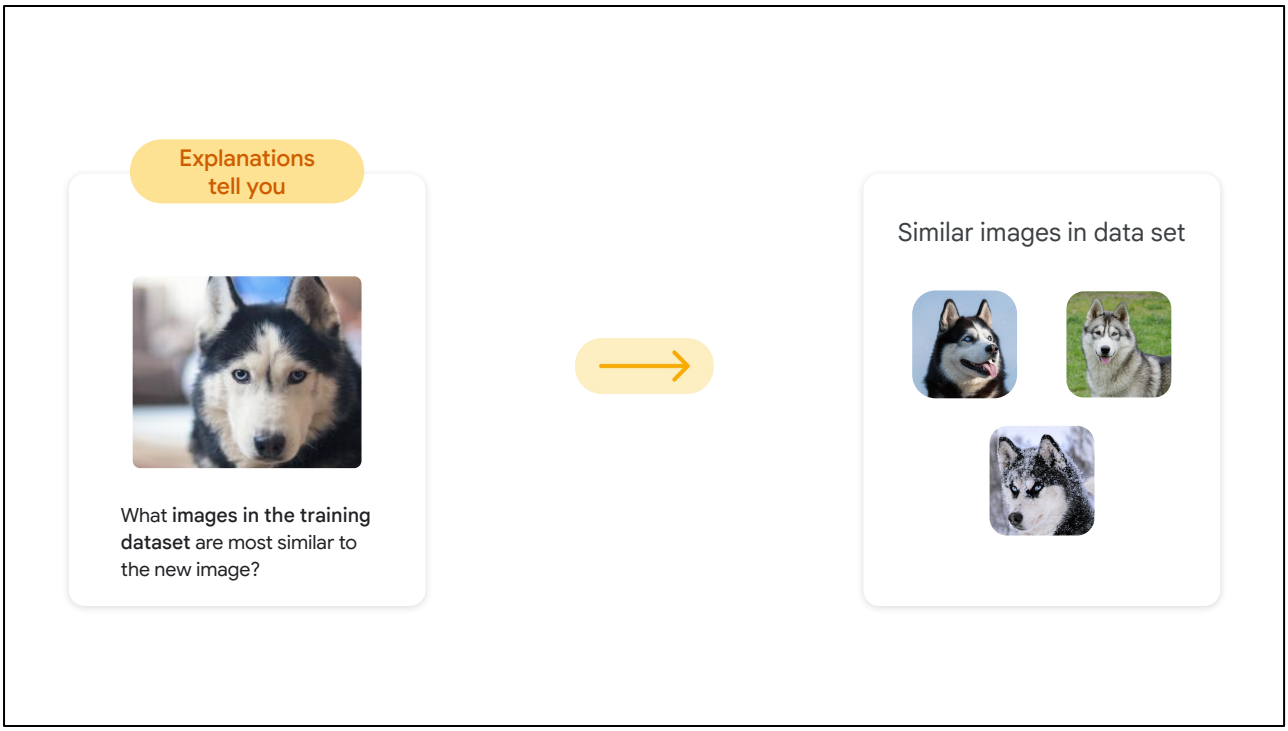
Let’s talk about the example-based explanation.



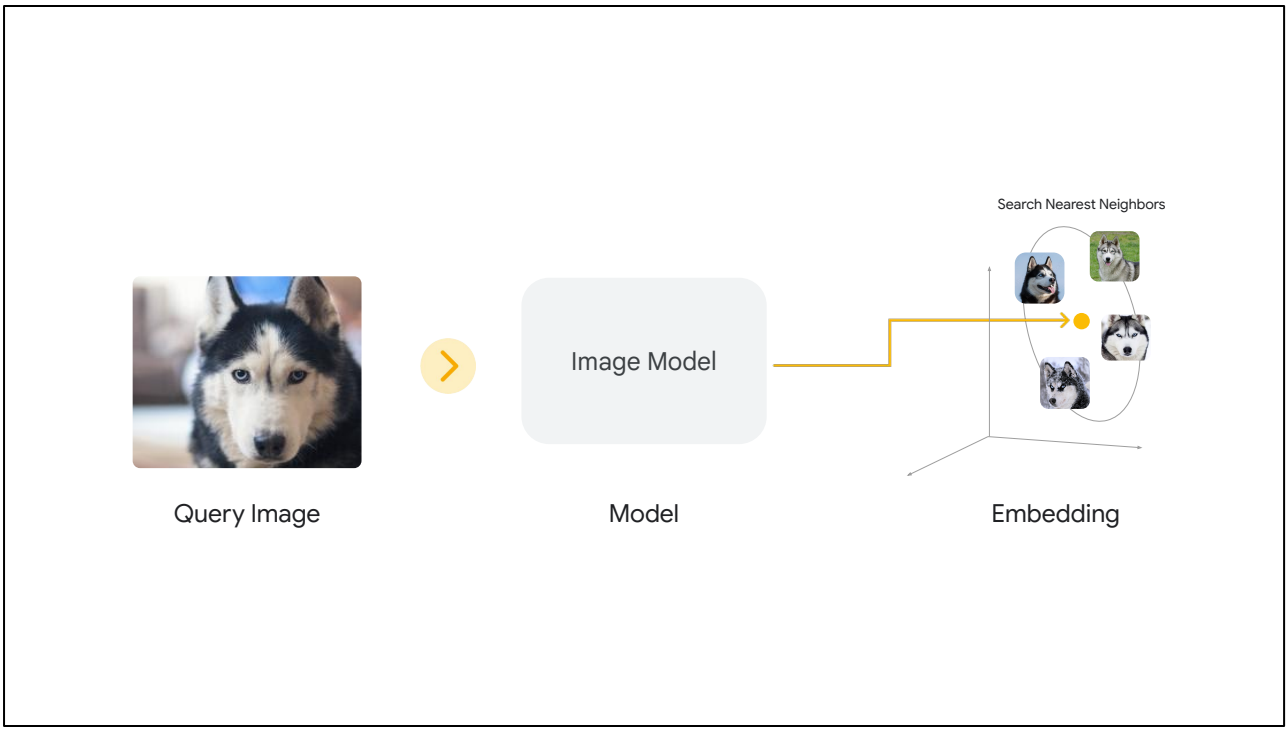
The focus here is on explaining a model's results by looking at the training data.

Example-based explanations is another interpretability technique that allows users to understand model behavior and how predictions are made. It relies on providing approximate nearest neighbor-based explanations.

We can generate example-based explanations for multiple types of data such as image, tabular and text.



For images, example-based explanations will show you which other images in the training data appeared most similar to the new image that you want to classify. For example, for this image of a dog, the model is classifying it as a husky, and all similar examples in the training data were also labeled as huskies.
This works very similar for tabular and text data - the focus is on explaining model results by showing the most similar examples from the training data.

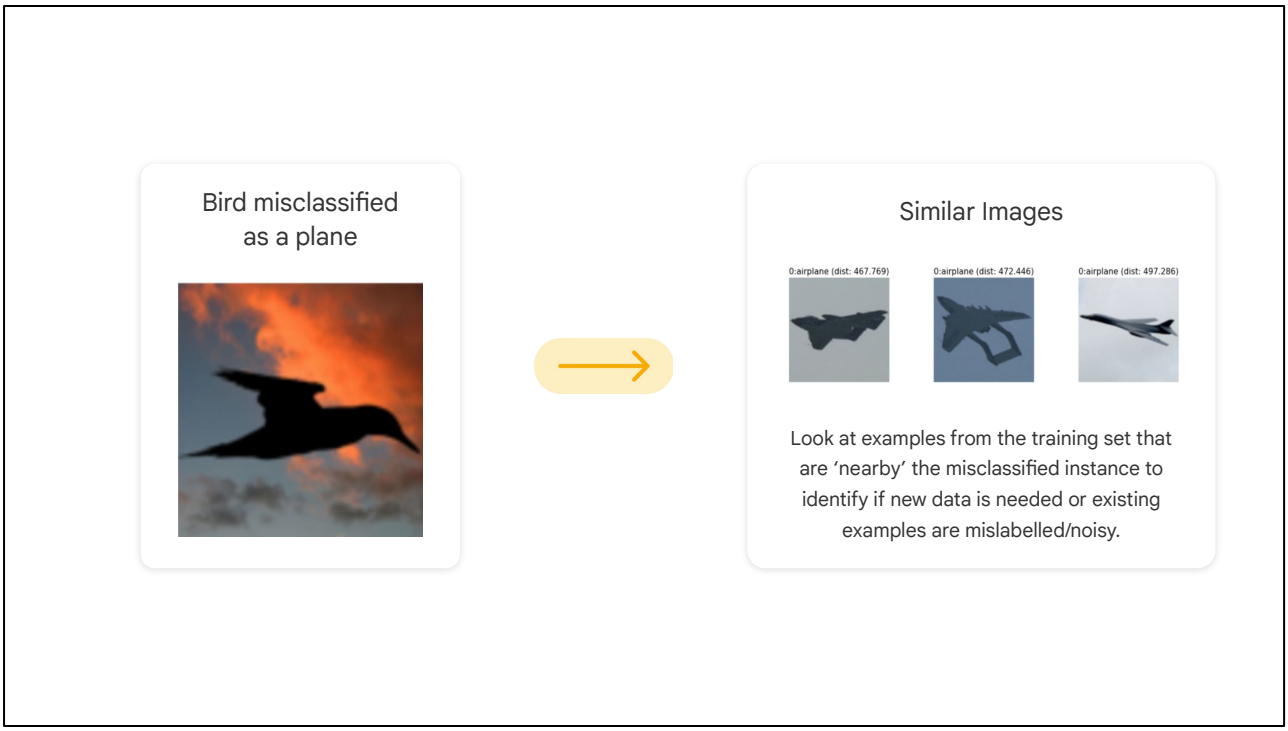


How can we compute the nearest neighbors?
First of all, all neural network models create some embedding as intermediate representations of input, regardless of the type, like image, text, or tabular.

So to compute the nearest neighbors, we can investigate representation of a layer, usually a layer close to the output layer, but before the final output layer.

First, compute embeddings of the training dataset, and then we can query the nearest neighbors by using a target image embedding.

.



We can use this explanation in many ways.

Let's say you have a classification model that misclassified this bird image as a plane.

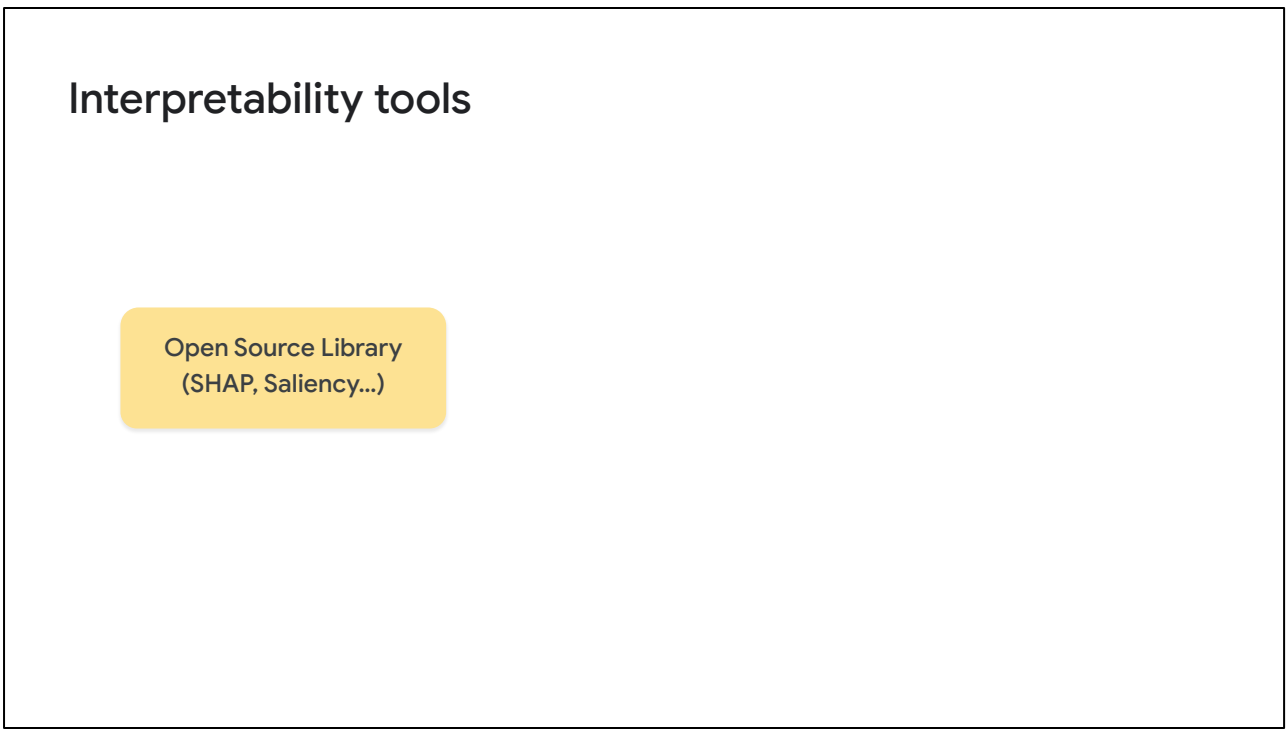
We used example-based explanations to retrieve other images in the training data which appeared most similar to this misclassified bird image for the model. Examining those, we identified that both the misclassified bird image and the similar images were dark silhouettes. This, in turn, signals a potential lack of images of birds with dark silhouettes in the training data, and an immediate action to gather more data with images of silhouetted birds in order to improve the model.

We discussed many interpretability techniques. Now let’s look into useful tools to apply these techniques to your machine learning models.

Interpretability tools

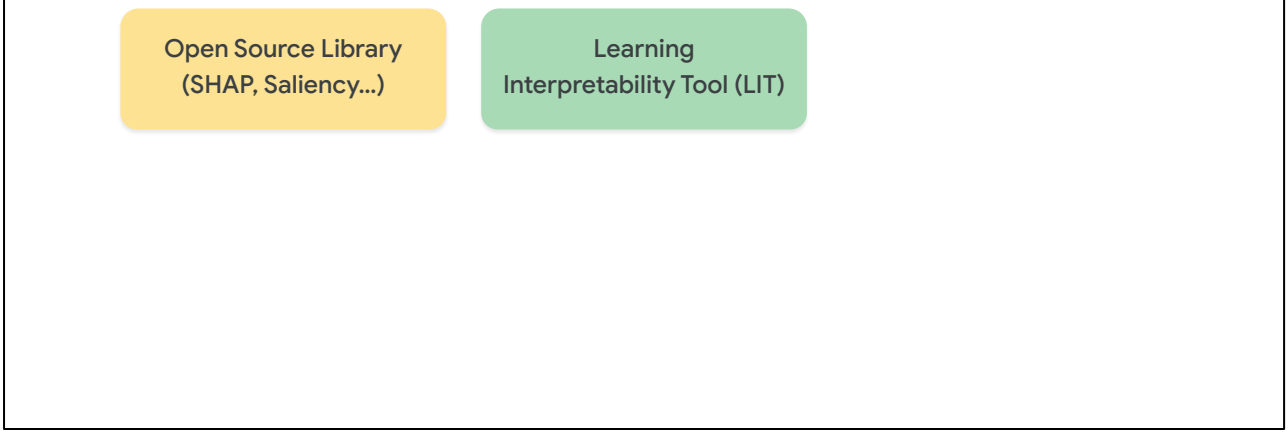
We are going to look at three tools:

Interpretability tools



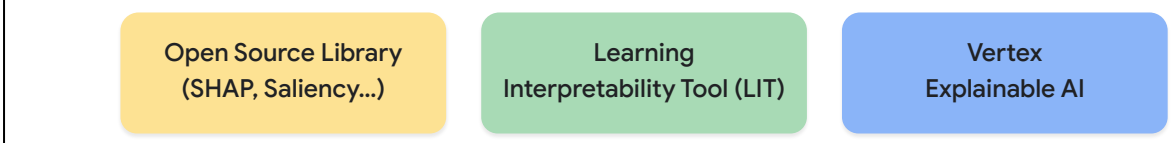
- Open source libraries: SHAP and Saliency

Interpretability tools



- Learning Interpretability Tool: An open-source platform for visualization and understanding of models.

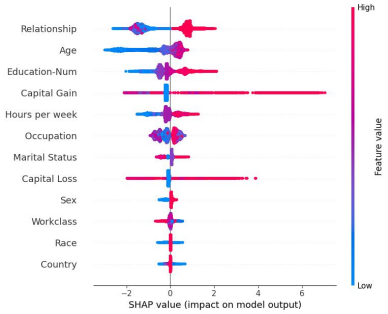
Interpretability tools



- Vertex Explainable AI: A Google Cloud solution for post-hoc explanations.

SHAP library

Visualization Example



Code Sample

```
import shap
...
explainer =
shap.TreeExplainer(model)
shap_values =
explainer.shap_values(X)
shap.summary_plot(shap_values, X)
```

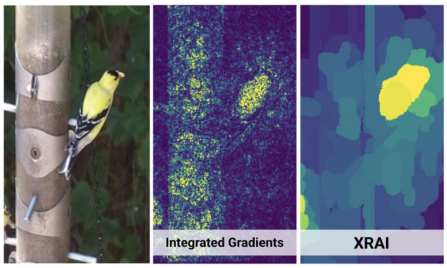
First, SHAP Python library provides popular implementations of approximate Shapley values, including Sampled Shapley, Kernel SHAP, Tree SHAP and so on.

It's a post-hoc technique that generates explanations for individual predictions that can also be aggregated for global model feature importances on tabular and text data.

SHAP's biggest drawback is its computational cost on large feature sets that have limited its application to domains such as images.

Google PAIR

Visualization Example

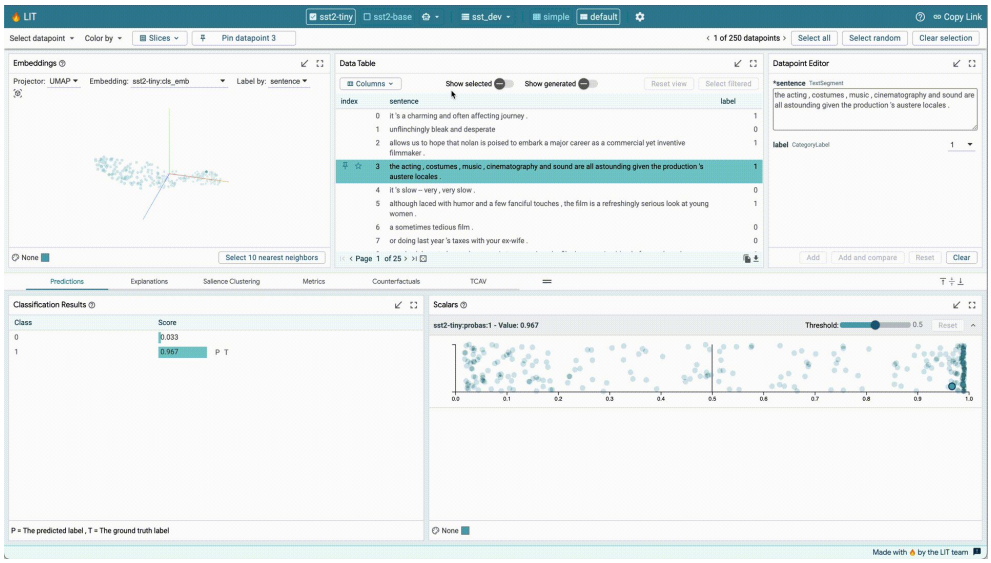


Code Sample

```
import saliency.core as saliency
...
xrai_object = saliency.XRAI()
xrai_attributions =
xrai_object.GetMask(
    im,
    call_model_function,
    call_model_args,
    batch_size=20
)
```

Next is Saliency: Google's People + AI Research team, or PAIR team, offers a saliency Python library that mainly covers gradient-based explanation techniques including Integrated Gradients, XRAI, and a lot of their variants.

Learning Interpretability Tool (LIT)



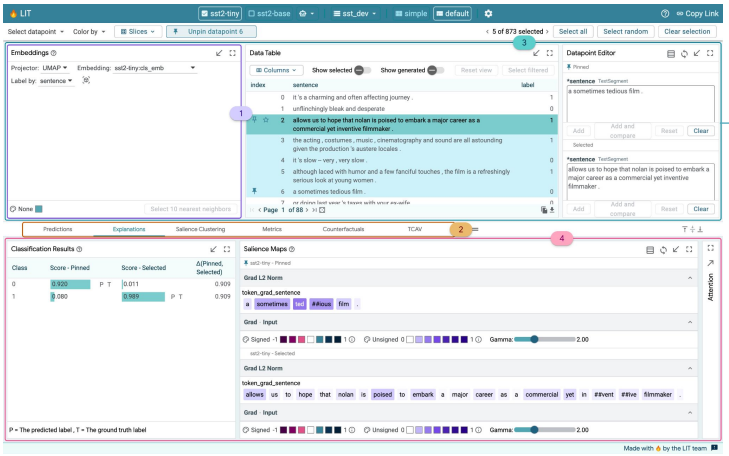
The Learning Interpretability Tool (LIT) is for researchers and practitioners who want to understand model behavior through a visual, interactive, and extensible tool. It mainly supports Natural Language Processing (NLP) with some preliminary support for tabular and image data.

You can use LIT to ask and answer questions such as:

- What kind of examples does my model perform poorly on?
- Why did my model make this prediction? Does the model properly focus on important features, instead of obviously unimportant features like image background?
- Does my model behave consistently if I change things like textual style, verb tense, or pronoun gender?
- And does this method relate to counterfactual analysis in AI fairness and bias?

LIT has lots of functionalities, and it even provides you with the ability to add custom techniques, metrics, visualizations, and more. You can also customize the layout itself to select your modules and groups of interest.

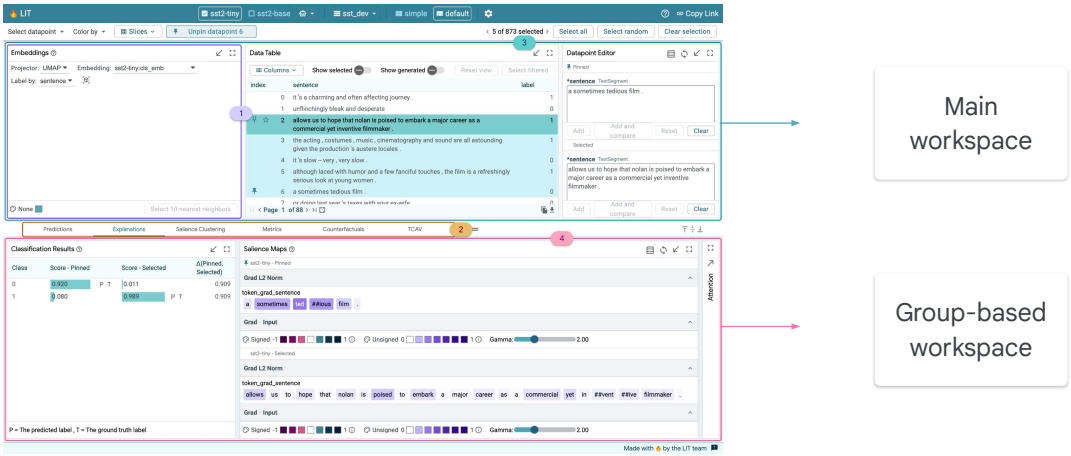
Learning Interpretability Tool (LIT)



Let’s get a general overview of the UI to get a feel of what capabilities it offers.

LIT is divided into two workspaces: a main workspace in the upper half of the interface,

Learning Interpretability Tool (LIT)



and a Group-based workspace in the lower half. Each workspace is a logical combination of groups, each containing different modules. This allows you to view different visualizations and methods side-by-side.

Learning Interpretability Tool (LIT) supports...

- Models:
  - Classification
  - Regression
  - Sequence-to-Sequence
  - ...

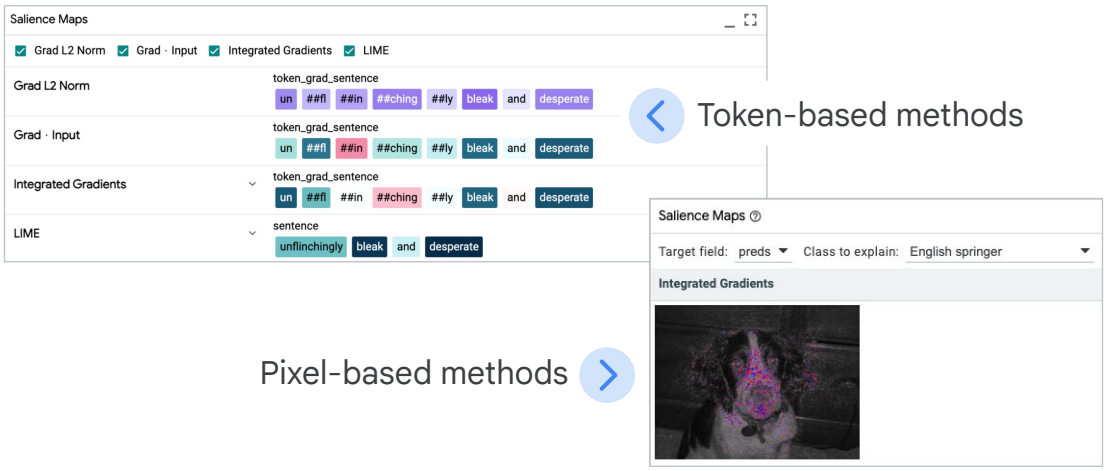
LIT supports many models and features. Models like Classification, Regression, Sequence-to-Sequence, etc.

Learning Interpretability Tool (LIT) supports...

- Models:
  - Classification
  - Regression
  - Sequence-to-Sequence
  - ...
- Features:
  - Embedding Projector
  - TCAV
  - Counterfactual Analysis
  - ...

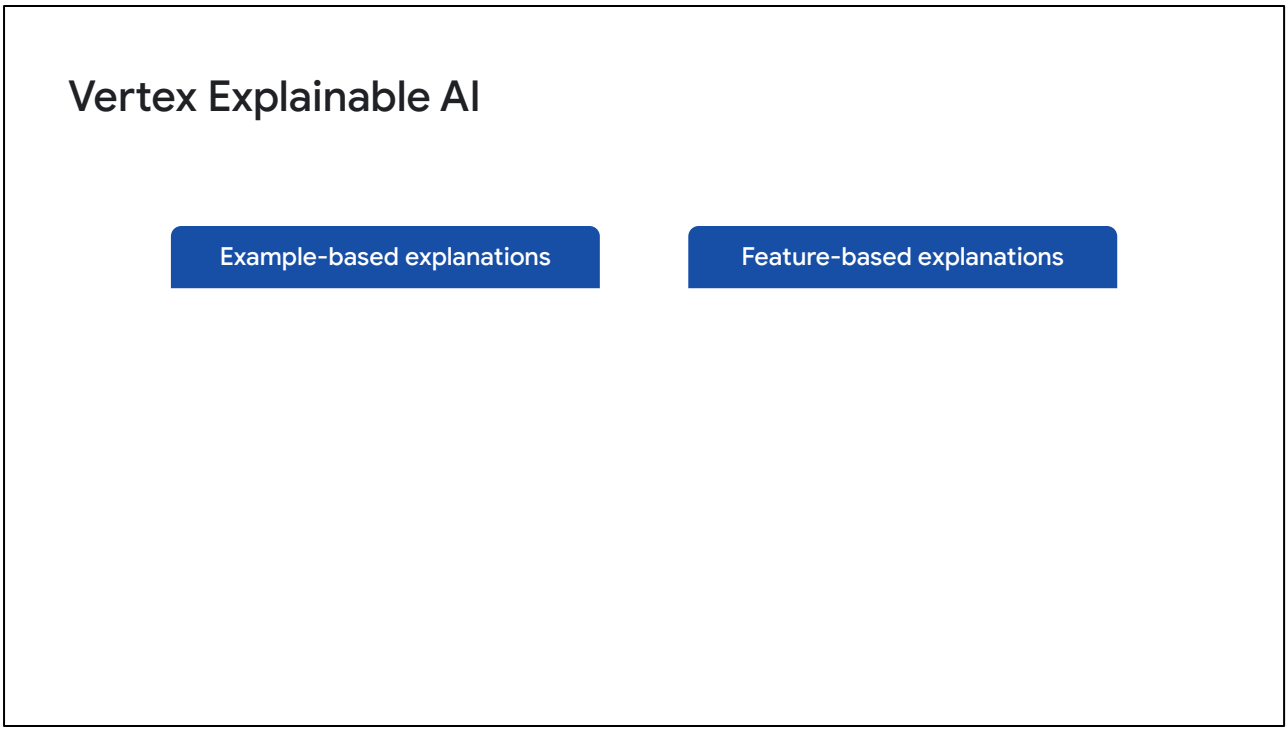
and features like an embedding projector, TCAV, Counterfactual Analysis, etc.

Learning Interpretability Tool (LIT): Key features

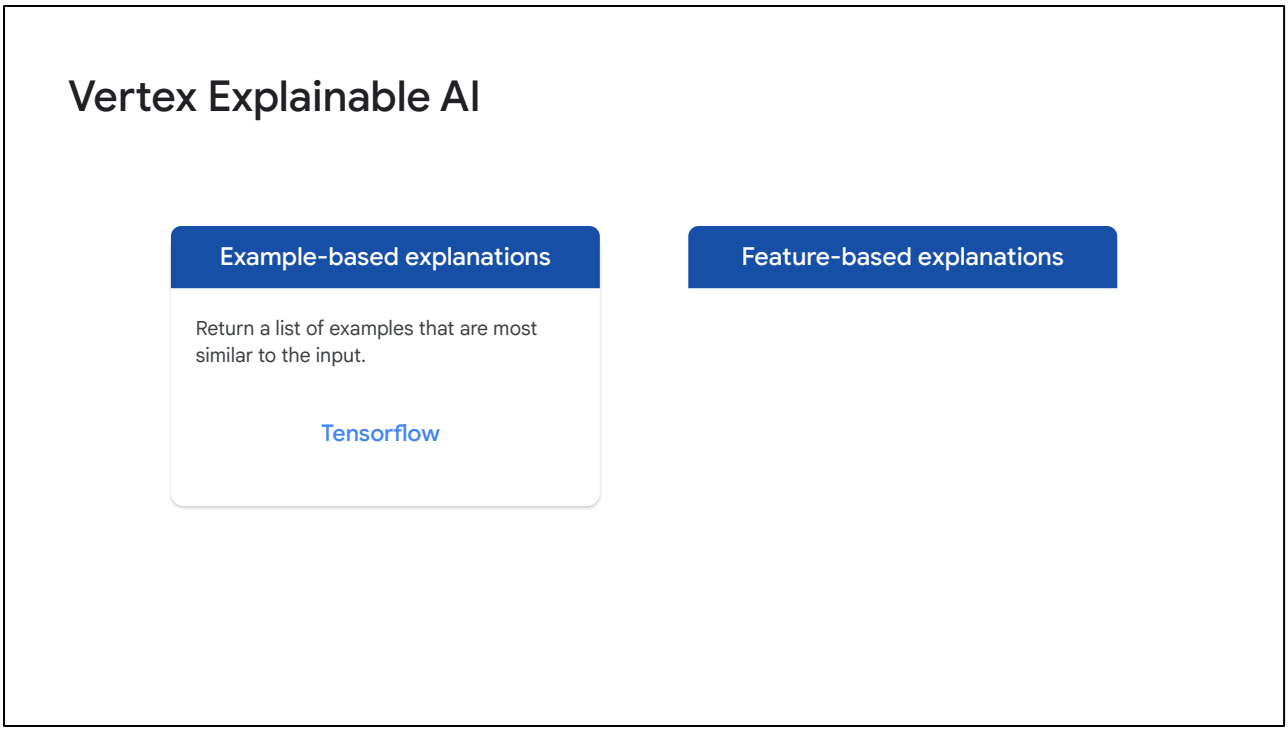


Two of the most important features to mention are token-based input salience methods and pixel-based salience methods.

Token-based input salience methods include gradient-based methods and black-box techniques like LIME.
Pixel-based salience methods are used for models that take images as inputs.
The output for both types of salience methods is rendered in the salience maps module in the LIT UI, which allows for comparison of multiple methods at once.

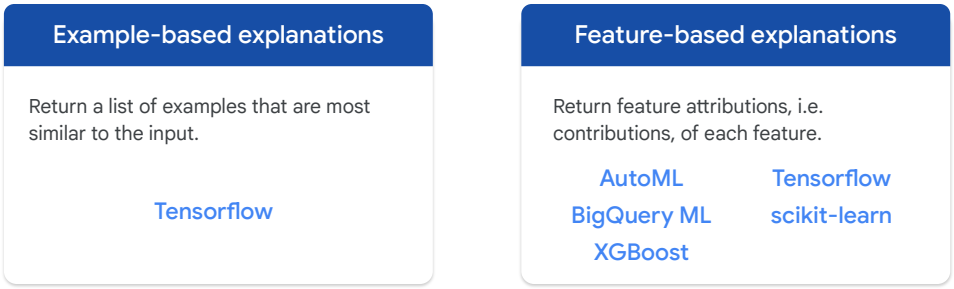


Vertex Explainable AI is Google Cloud managed service for interpretability. It offers feature-based and example-based explanations to provide better understanding of model decision making, even for complex models.



With example-based explanations, Vertex AI uses nearest neighbor search to return a list of examples (typically from the training set) that are most similar to the input. It currently supports only TensorFlow models that can provide an embedding (latent representation) for inputs.

Vertex Explainable AI



With feature-based explanations, Vertex AI uses Sampled Shapley, Integrated gradients, or XRAI . It works on tabular, image, video, and text data. It currently supports all types of models such as AutoML, BQML models, and custom-trained models on Vertex AI, and frameworks such as TF, scikit-learn, and xgboost.

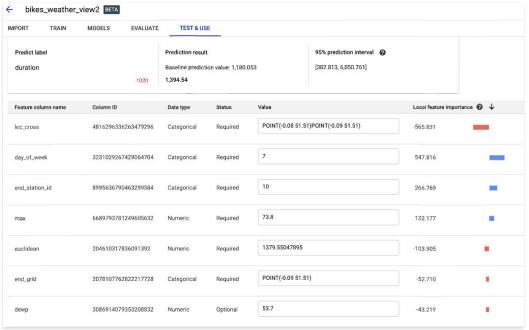
Vertex Explainable AI

BigQuery ML

AI Explanation functions

- ML.EXPLAIN_PREDICT
- ML.EXPLAIN_FORECAST
- ML.GLOBAL_EXPLAIN
- ML.FEATURE_IMPORTANCE
- ML.ADVANCED_WEIGHTS

AutoML



For Bigquery ML and AutoML, Google Cloud provides simple ways to look at predictions within the product.

BigQueryML provides five different AI explanation functions that you can use in SQL.

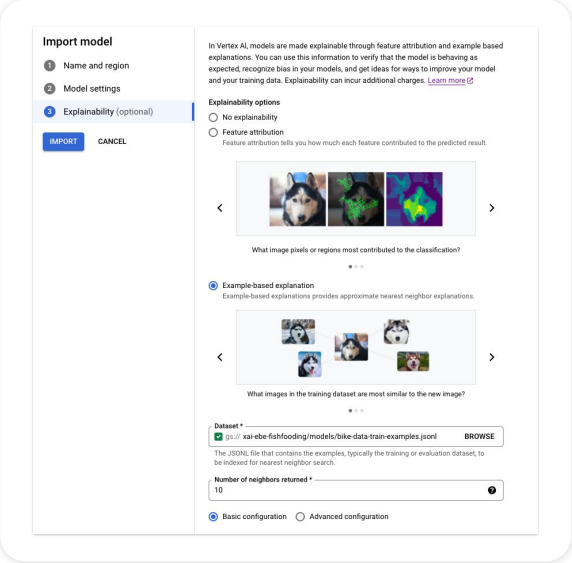
For AutoML, you can simply access explanations from the TEST & USE tab, select and configure the prediction type you want, and get explanations! In the screenshot here, you are seeing an example of online prediction for text data with samples Shapley for local feature importance. Global importance is also supported.

Vertex Explainable AI

Set up explanations for custom models via:

Console gcloud CLI REST Python

Simply import the model in Model Registry, and configure your desired explanations in the Explainability tab!



On Vertex AI, it is also easy to set up explanations.
This is an example to configure it from the console, but remember that you can also set them up through gcloud CLI, REST, and Python.

All you need to do is import the model in Model Registry and configure your desired explanations in the Explainability tab!

01 Overview of interpretability and transparency

02 Overview of interpretability techniques

03 Feature based explanations: Model agnostic

04 Feature based explanations: Model specific

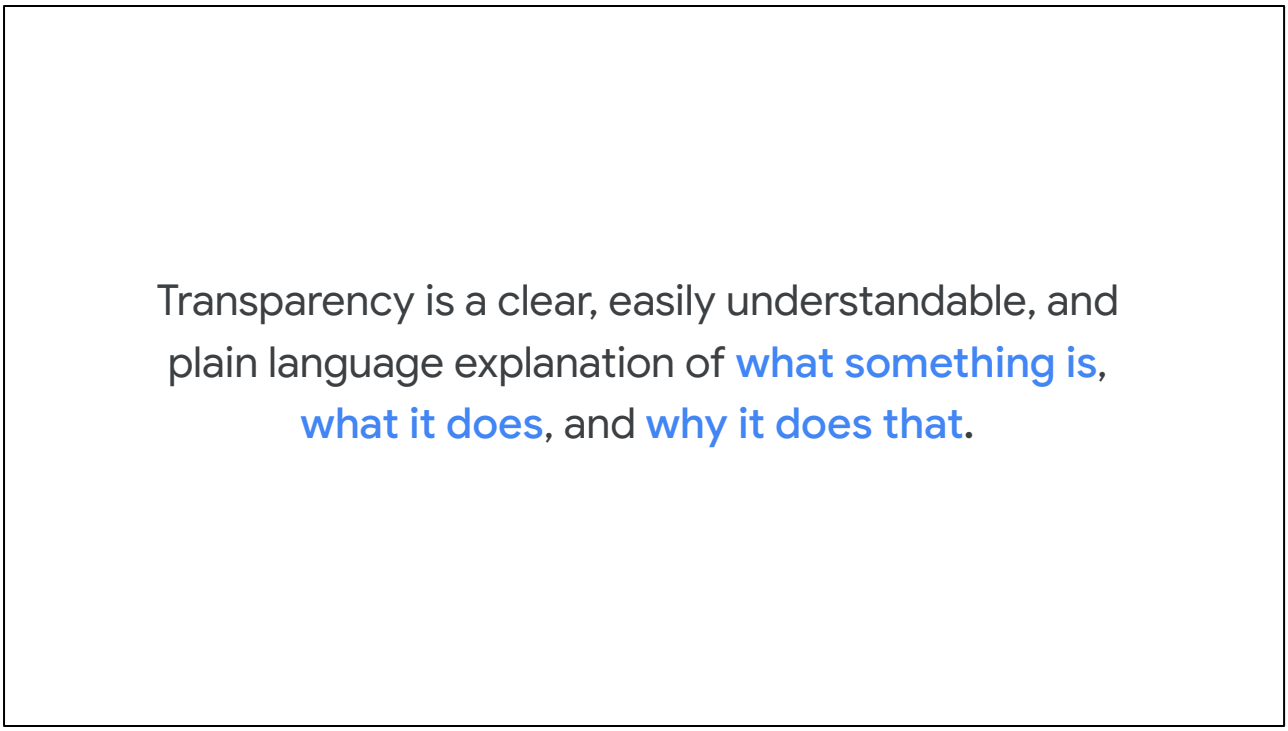
05 Concept-based and example-based explanations

06 Tools for Interpretability

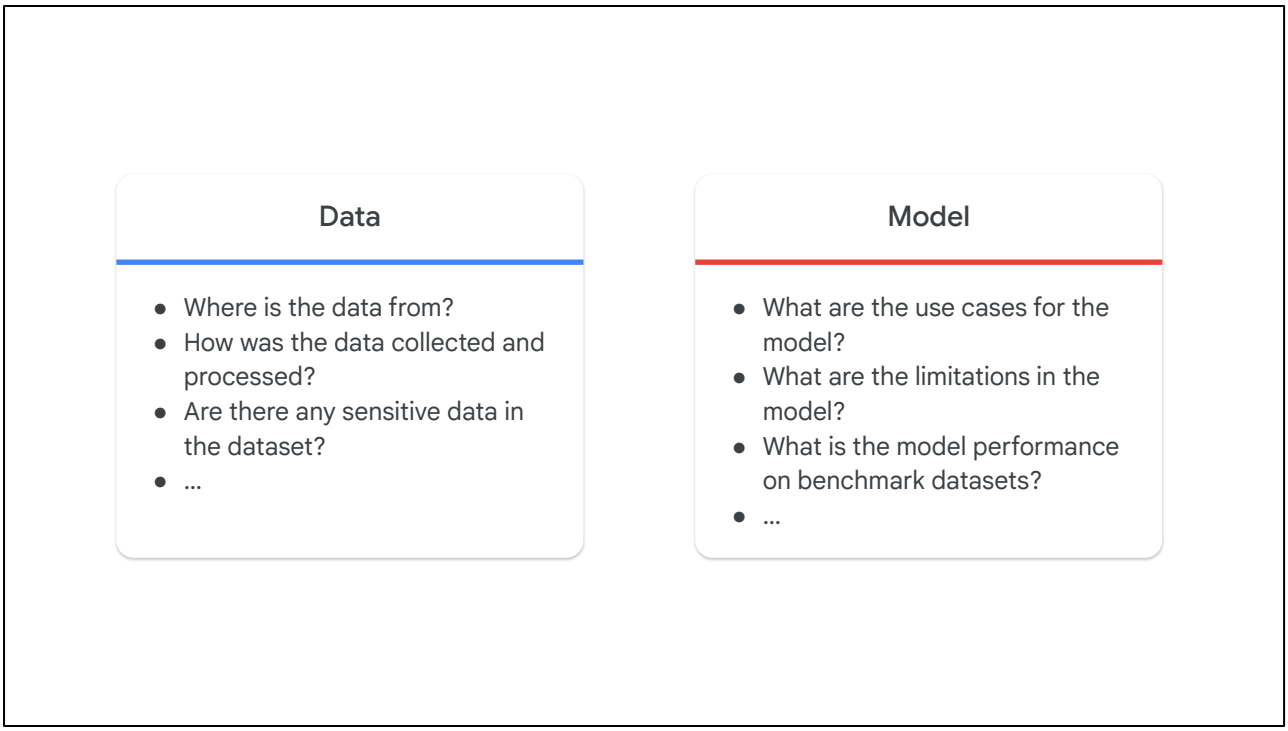
07 Data and Model Transparency

08 Lab: Vertex Explainable AI

Let’s explore transparency in detail, specifically data and model transparency.



Transparency is a clear, easily understandable, and plain language explanation of what something is, what it does, and why it does that.



Machine learning transparency involves sharing information about system behaviors and organizational processes. This might include documenting and sharing how models and datasets were created, trained, and evaluated.
Transparency artifacts are a form of structured information reporting that focuses on transparency during product creation and performance to encourage responsible AI adoption and application. Think of this as being similar to nutritional labels for food products. Data cards and model cards are two types of transparency artifacts.

Data Cards

Data Cards are structured summaries of essential facts about various aspects of ML datasets needed by stakeholders across a project's life cycle for responsible AI development.

Let's first talk about data cards. Data cards are structured summaries of essential facts about various aspects of ML datasets that are needed by stakeholders across a project's life cycle for responsible AI development. You can take advantage of these templates to simply and reliably document your datasets.

Data Cards

- Upstream sources
- Data collection and annotation methods
- Training and evaluation methods
- Intended use
- Decisions affecting model performance

Data cards can include any of the following that are appropriate for your use case: (1) upstream sources, (2) data collection and annotation methods, (3) training and evaluation methods, (4) intended use, and (5) decisions affecting model performance.

Data Cards

When creating documentation for the data that you use, be sure to involve the whole team. This will ensure you ask all the questions that you need to understand about your data, and that you include clear and actionable answers to these questions in the data card. You should also involve the following roles when interacting with the data card

Data Cards

- Producers: The individuals/teams who will be creating the Data Card.

- **Producers:** The individuals or teams who will be creating the data card.

Data Cards

- Producers: The individuals/teams who will be creating the Data Card.
- Consumers: The individuals/teams who will be using the Data Card.

- **Consumers:** The individuals or teams who will be using the data card.

Data Cards

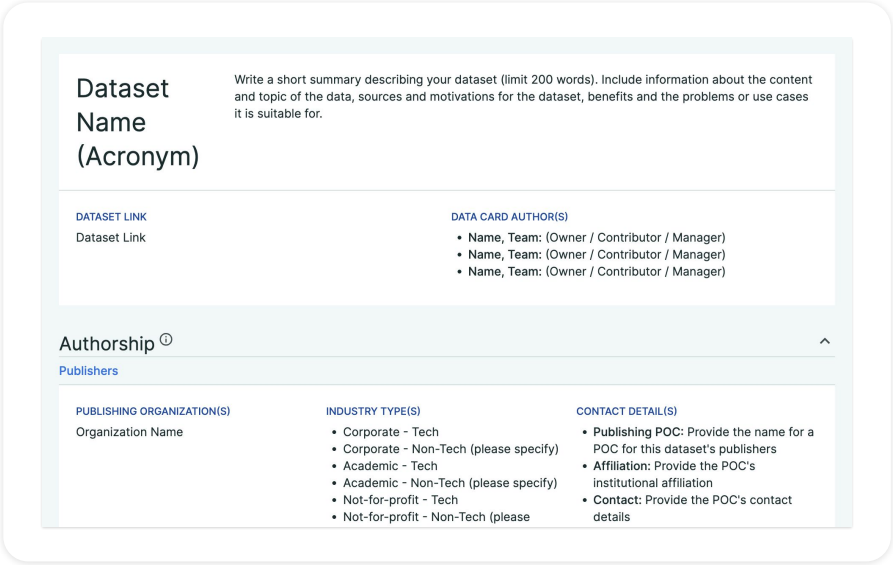
- Producers: The individuals/teams who will be creating the Data Card.
- Consumers: The individuals/teams who will be using the Data Card.
- End users: The individuals who might be taking actions based on the system that is, or will be, built on the dataset.

- **End users:** The individuals who might be taking actions based on the system that is, or will be, built on the dataset.

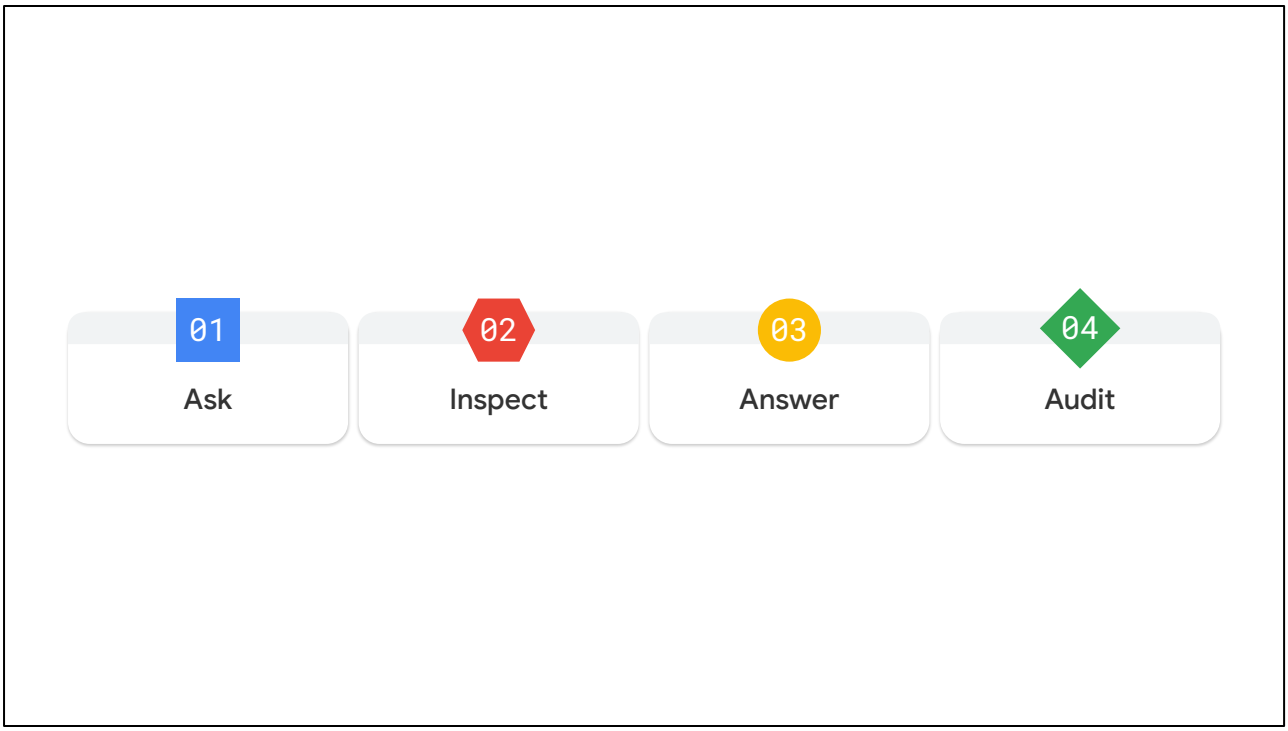
Data Card Template

Summary	Extended Use
Authorship	Transformations
Dataset Overview	Annotations & Labeling
Example of Data Points	Validation Types
Motivations & Intentions	Sampling Methods
Access, Retention & Wipeout	Known Applications & Benchmarks
Provenance	Terms of Art
Human and Other Sensitive Attributes	Reflections of Data

At Google, we provide a data card template that captures 15 themes besides a summary.
These are the themes we frequently look for when making decisions, many of which are not traditionally captured in technical dataset documentation.



Each of the themes is a section on the data card template like shown in the screenshot.
You can find an example of a full data card template in the resources.



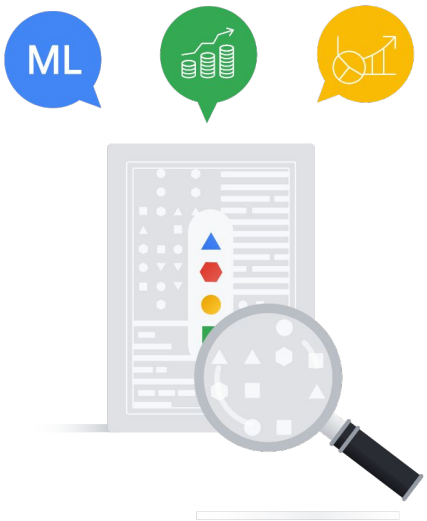
The data card playbook is a toolkit for transparency in AI dataset documentation.

Google provides a data card playbook to help AI/ML practitioners achieve and maintain proactive data transparency.
The playbook contains four modules designed with participatory activities that define long-term transparency for datasets and in contexts. The transparency patterns capture practical ways to create data cards that are people-centric, purposeful and actionable.

reference

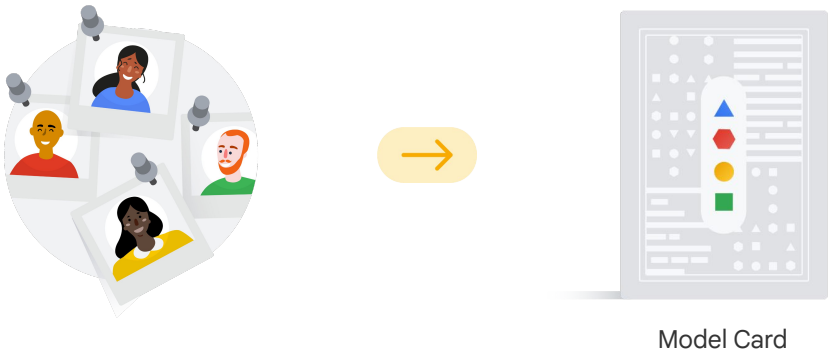
Model Cards

Model cards are short documents accompanying trained machine learning models that provide benchmarked evaluation in a variety of conditions. They disclose the context in which models are intended to be used, details of the performance evaluation procedures, and other relevant information.



Now that we've learned about the data card, let's look at another toolkit that provides transparency, the model card.
Model cards explain what the model is supposed to be used for, how its performance was tested, and other important details.
Model cards offer benchmarked assessments for a range of conditions, including diverse cultural, demographic, or phenotypic groups, as well as intersectional groups pertinent to the model's intended application domains. They also reveal the context for which the model is designed, elaborate on the performance evaluation procedures, and provide additional relevant information.

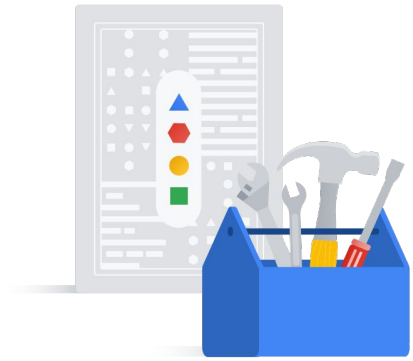
Model Card development



Model developers are primarily responsible for creating a model card. However, anyone involved in idea generation, development, or testing should also be involved in the process, since the requisite knowledge is often distributed across the team.

Model Card Toolkit

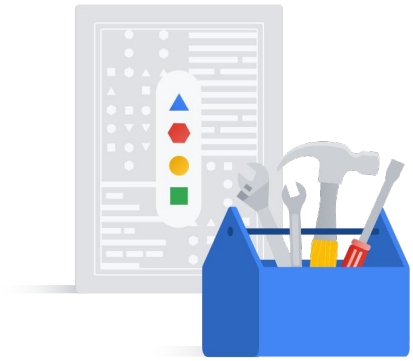
- Streamlines and automates Model Card generation for model transparency.



The Model Card Toolkit (MCT) library streamlines and automates generation of model cards.

Model Card Toolkit

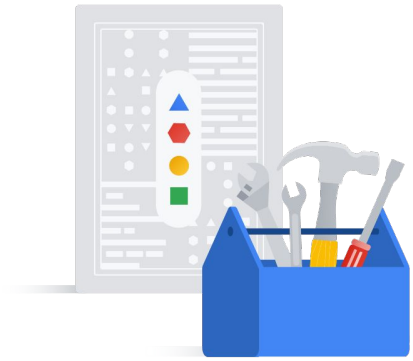
- Streamlines and automates Model Card generation for model transparency.
- Customize model card structure using Jinja templates or pre-made options.



Jinja templates form the underlying structure of a Model Card document.

Model Card Toolkit

- Streamlines and automates Model Card generation for model transparency.
- Customize model card structure using Jinja templates or pre-made options.
- Populate fields automatically with TFX or manually via Python API.



The Model Card Toolkit provides a few ready-made templates, but you have the freedom to modify these templates or even create your own.
For users of TensorFlow Extended (TFX), the MCT can automatically fill these fields using ML Metadata (MLMD). Additionally, you can manually populate model card fields through a Python API.

```
import model_card_toolkit

# Initialize the Model Card Toolkit with a
path to store generate assets
model_card_output_path = ...
mct = model_card_toolkit.ModelCardToolkit(
    model_card_output_path
)

# Initialize the
model_card_toolkit.ModelCard, which can be
freely populated
model_card = mct.scaffold_assets()
model_card.model_details.name = 'My Model'
model_card.(...)

# Write the model card data to a JSON file
mct.update_model_card_json(model_card)

# Return the model card document as an
HTML page
html = mct.export_format()
```

Here’s an example.

1

```
import model_card_toolkit

# Initialize the Model Card Toolkit with a
path to store generate assets
model_card_output_path = ...
mct = model_card_toolkit.ModelCardToolkit(
    model_card_output_path
)
```

```
# Initialize the
model_card_toolkit.ModelCard, which can be
freely populated
model_card = mct.scaffold_assets()
model_card.model_details.name = 'My Model'
model_card.(...)

# Write the model card data to a JSON file
mct.update_model_card_json(model_card)

# Return the model card document as an
HTML page
html = mct.export_format()
```

To get started, import the model_card_toolkit module in Python, then initialize the model card toolkit with a path for storing generated assets.

```
import model_card_toolkit

# Initialize the Model Card Toolkit with a
path to store generate assets
model_card_output_path = ...
mct = model_card_toolkit.ModelCardToolkit(
    model_card_output_path
)
```

2

```
# Initialize the
model_card_toolkit.ModelCard, which can be
freely populated
model_card = mct.scaffold_assets()
model_card.model_details.name = 'My Model'
model_card.(...)
```

```
# Write the model card data to a JSON file
mct.update_model_card_json(model_card)

# Return the model card document as an
HTML page
html = mct.export_format()
```

Next, initialize the model_card_toolkit.ModelCard with a path for storing generated assets, then initialize the _model_card_toolkit.ModelCard, which can be flexibly customized.

```
import model_card_toolkit

# Initialize the Model Card Toolkit with a
path to store generate assets
model_card_output_path = ...
mct = model_card_toolkit.ModelCardToolkit(
    model_card_output_path
)

# Initialize the
model_card_toolkit.ModelCard, which can be
freely populated
model_card = mct.scaffold_assets()
model_card.model_details.name = 'My Model'
model_card.(...)
```

3

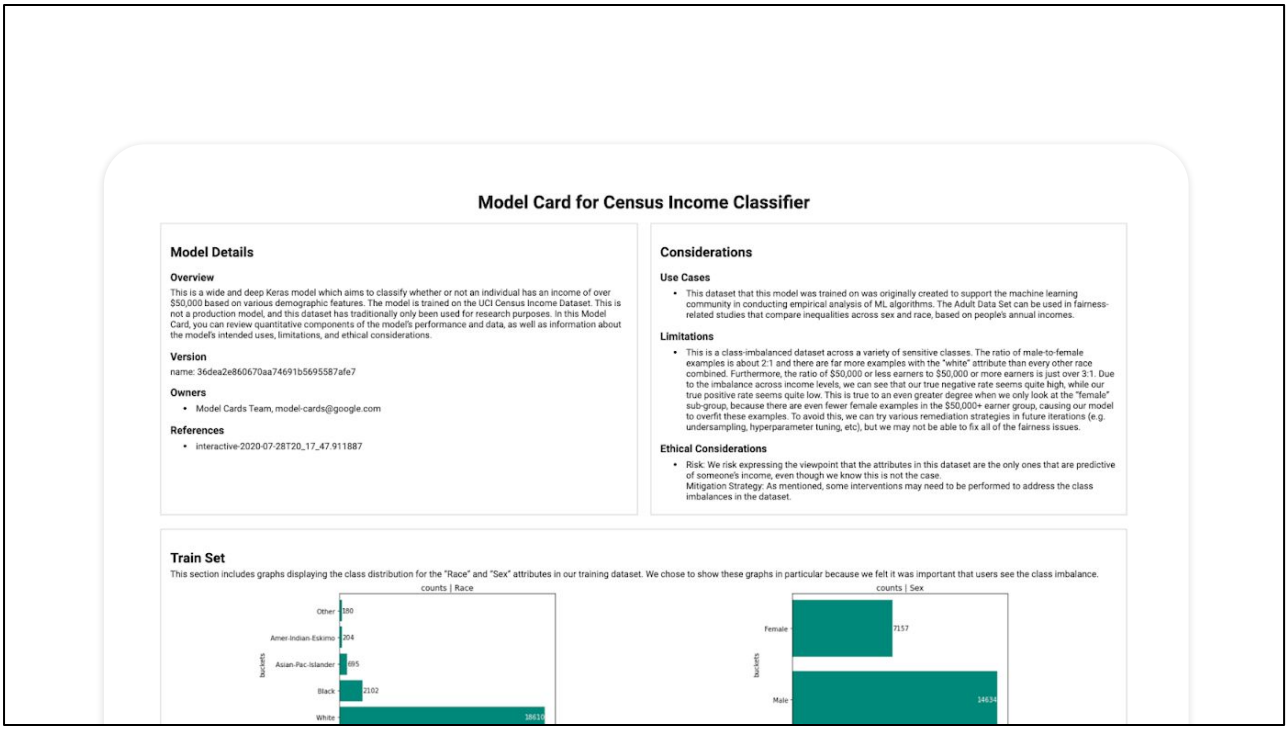
```
# Write the model card data to a JSON file
mct.update_model_card_json(model_card)

# Return the model card document as an
HTML page
html = mct.export_format()
```

Finally, we write the model card data to a JSON file and generate the model card document as an HTML page.

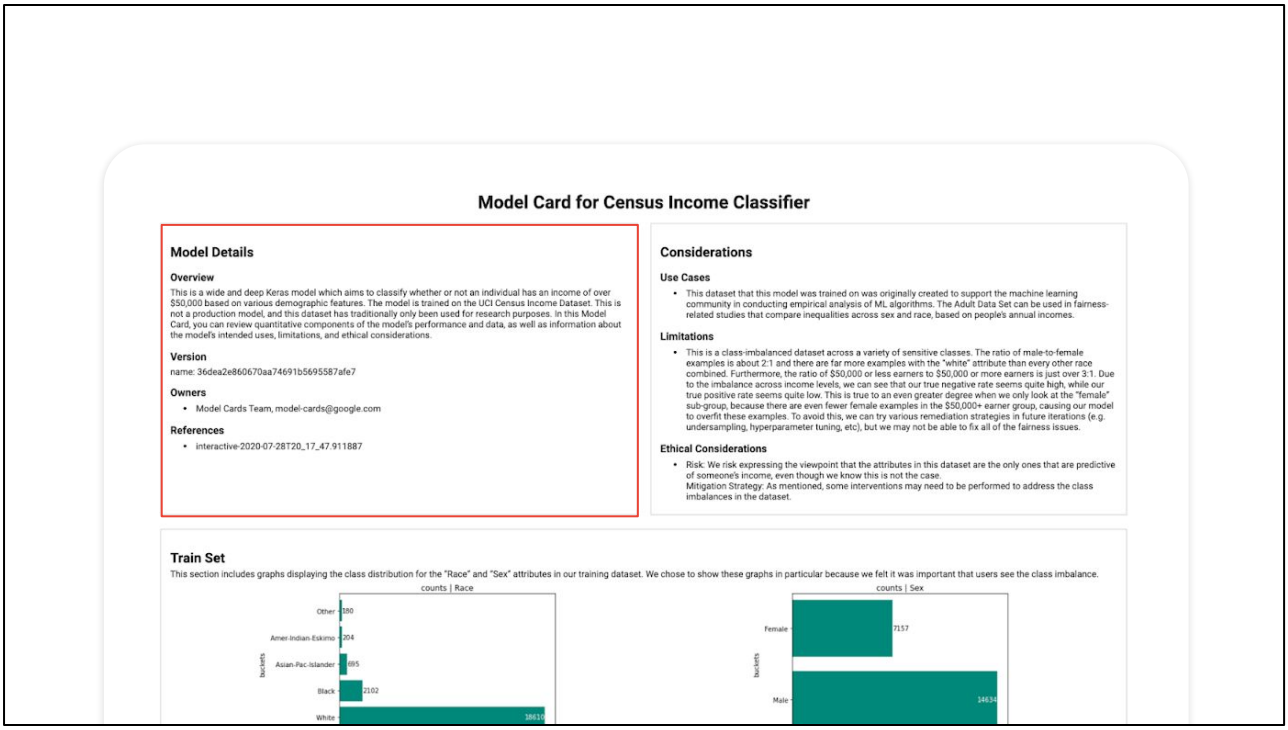
```
pip install tfx-addons[model_card_generator]
```

If you're using TensorFlow Extended (TFX), you can integrate model card generation into your TFX pipeline by using the ModelCardGenerator component. However, as the ModelCardGenerator component is being migrated to the TFX Addons library, you'll need to install the tfx-addons package beforehand.

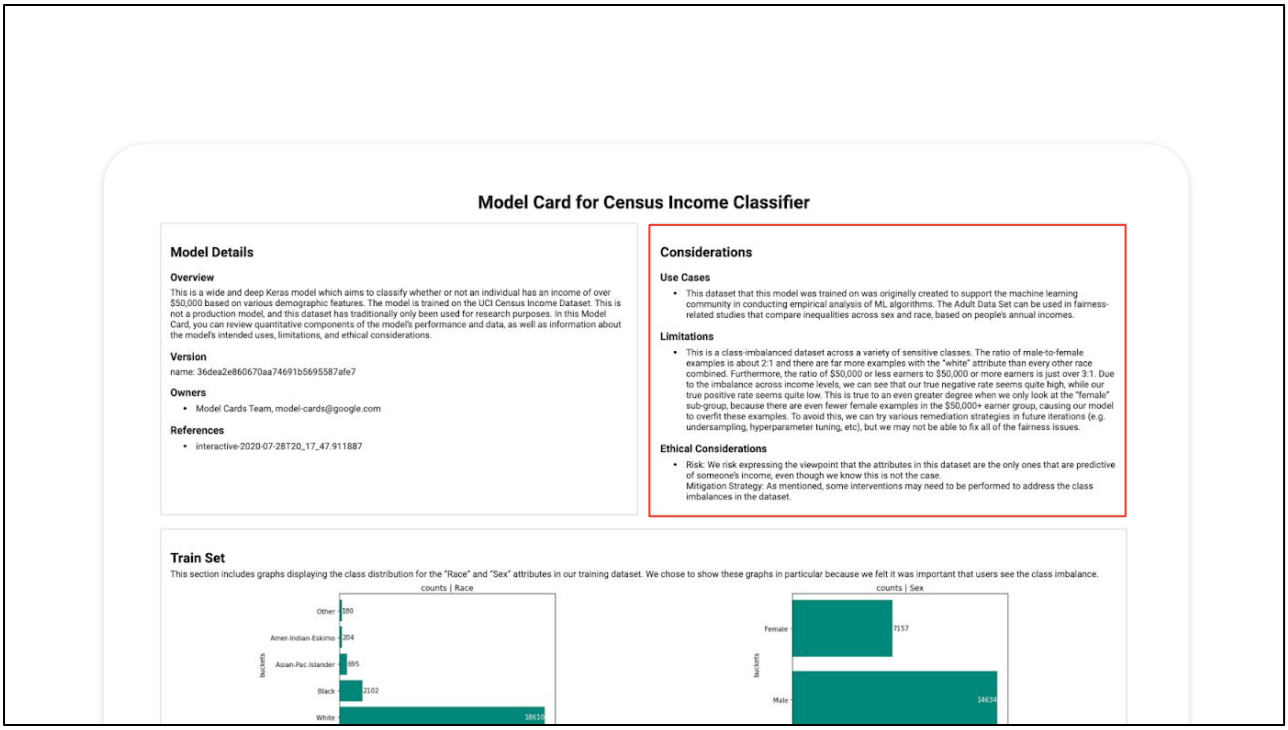
resource



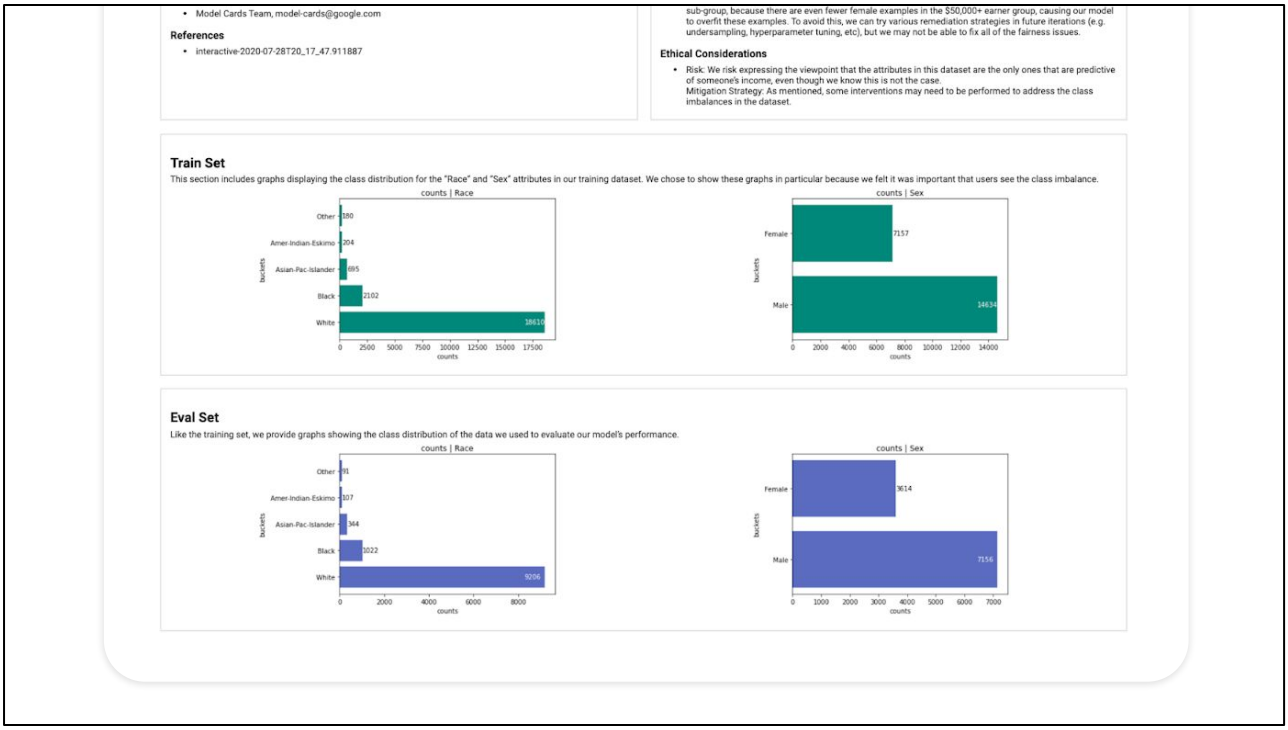
The model card for the Census Income Classifier serves as a prime example.



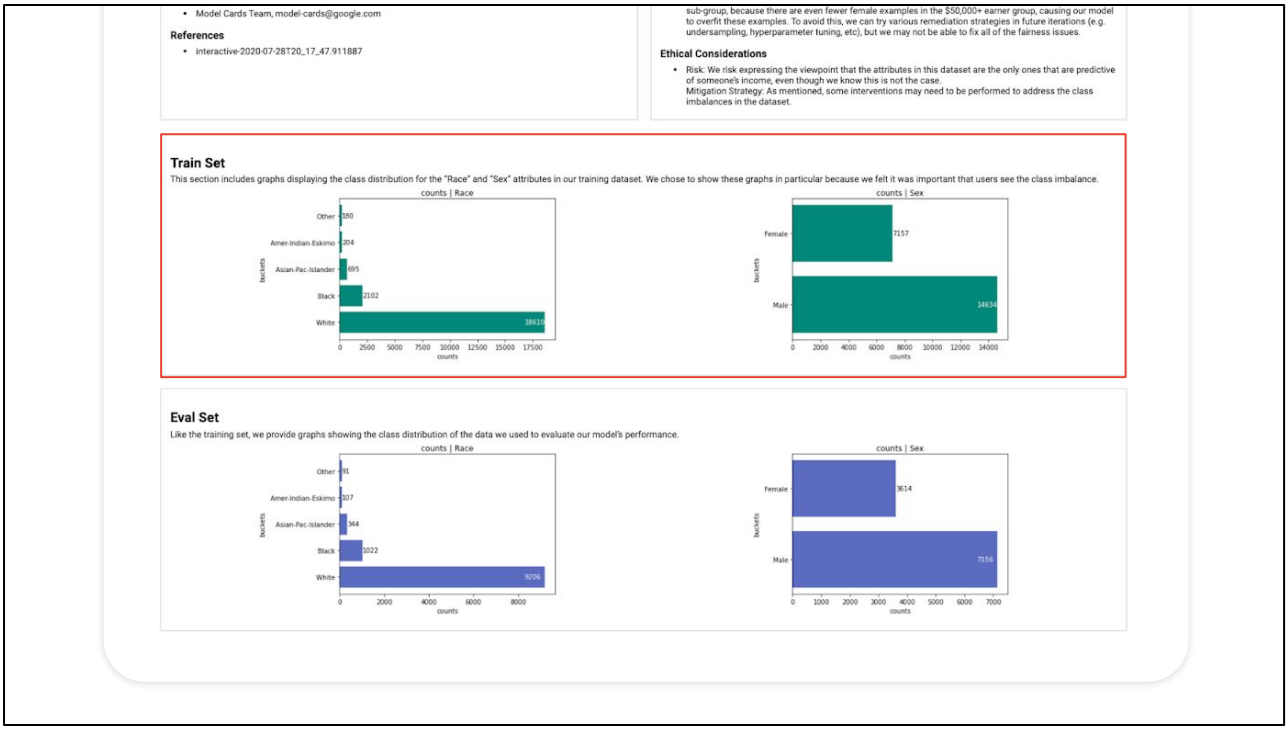
In the model details section, you'll find an overall description of the model, including its purpose, the dataset it was trained on, and other relevant information. Additionally, you can access information about the model's version, owners, and references.



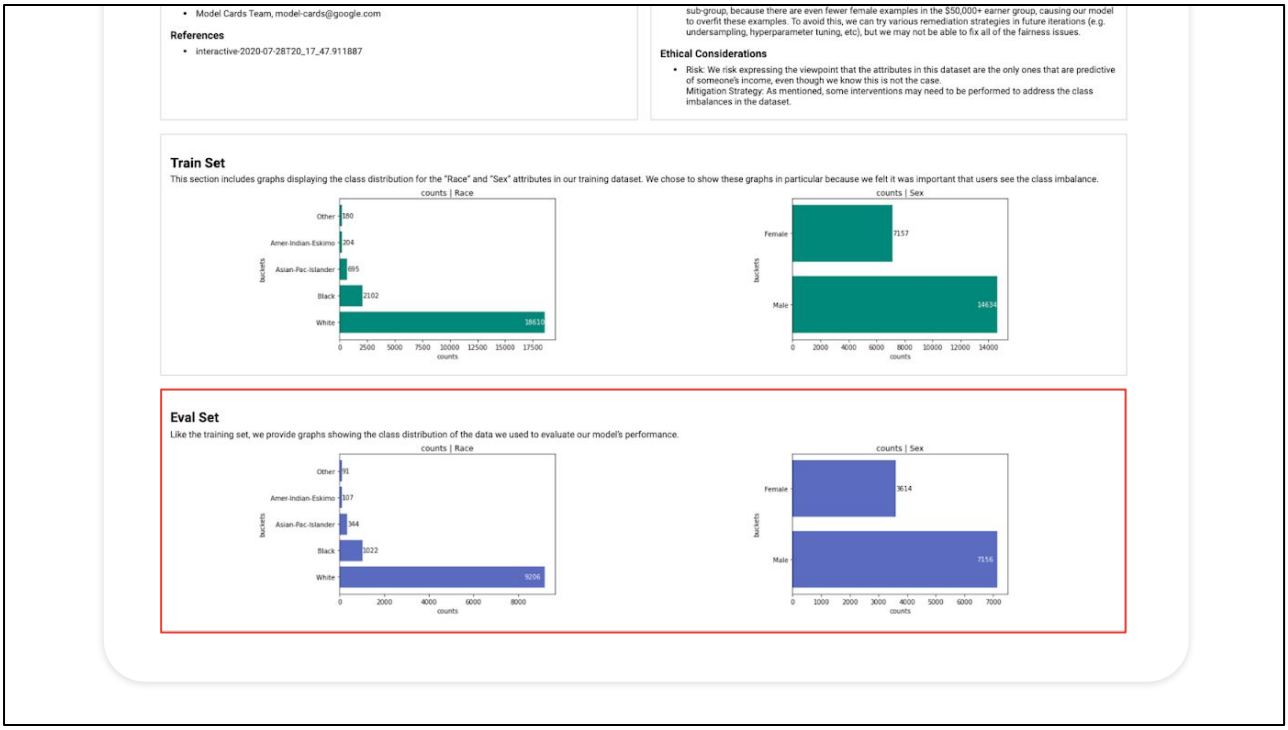
The Consideration section delves into the model's use cases, limitations, and ethical considerations.



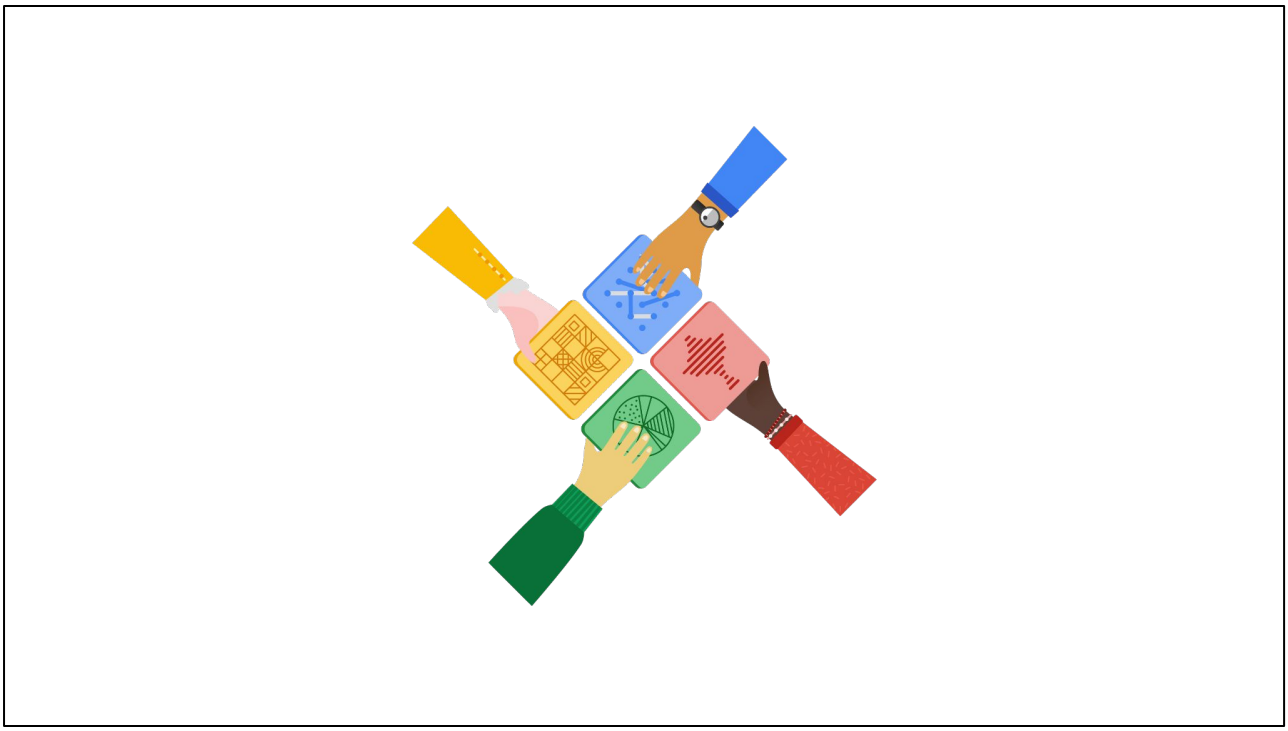
The lower portion of the model card presents graphs for the



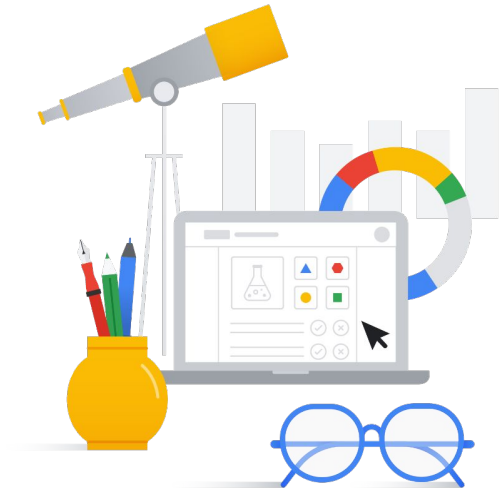
Train Set and



Eval Set, which provide insights into the main training data elements and performance evaluation metrics.



Transparency in AI is a fundamental principle that should be accessible to all. This is why model cards are not intended to be a proprietary Google product, but a shared, evolving framework that draws from diverse perspectives and contributions.

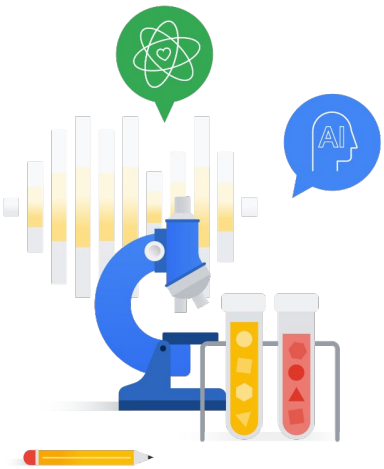


Hands-on lab:

Explaining an Image Classification Model with Vertex Explainable AI

Let’s perform an exercise in a hands-on lab. This lab shows you how to train a classification model on image data and deploy it to Vertex AI to serve predictions with explanations (or feature attributions).

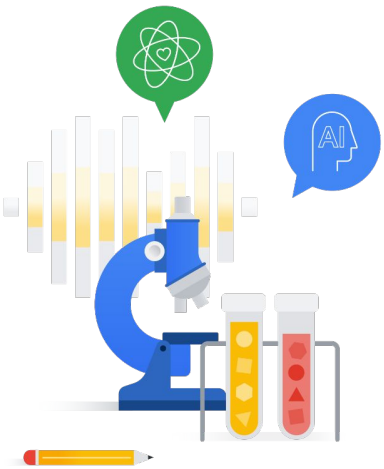
In this Lab, you will:



In this lab, you will:

In this Lab, you will:

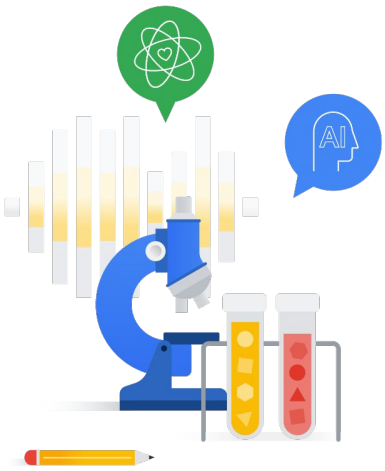
- Explore the dataset.



- Explore the dataset.

In this Lab, you will:

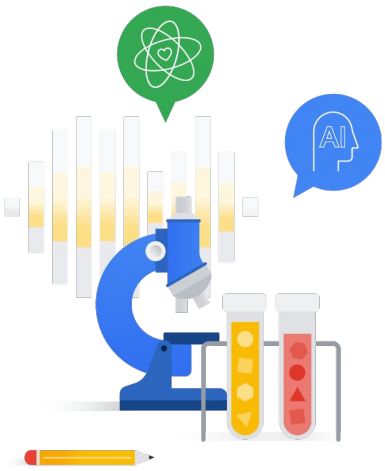
- Explore the dataset.
- Build and train a custom image classification model with Vertex AI.



- Build and train a custom image classification model with Vertex AI.

In this Lab, you will:

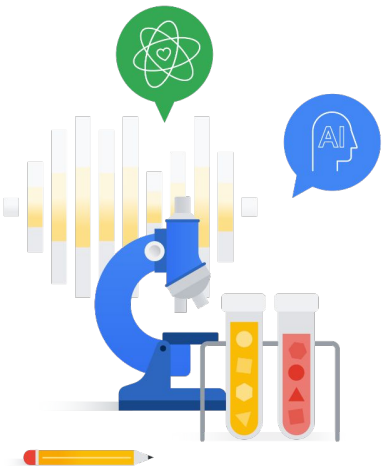
- Explore the dataset.
- Build and train a custom image classification model with Vertex AI.
- Deploy the model to an endpoint.



- Deploy the model to an endpoint.

In this Lab, you will:

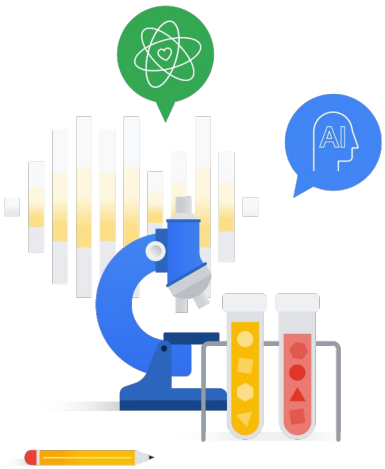
- Explore the dataset.
- Build and train a custom image classification model with Vertex AI.
- Deploy the model to an endpoint.
- Serve predictions with explanations.



- Serve predictions with explanations.

In this Lab, you will:

- Explore the dataset.
- Build and train a custom image classification model with Vertex AI.
- Deploy the model to an endpoint.
- Serve predictions with explanations.
- Visualize feature attributions from Integrated Gradients.



- Visualize feature attributions from Integrated Gradients.