Annamacharya University, Rajampet

Department of AI&ML & CSE(AI)

Explainable AI and Model Interpretability

UNDERSTANDING, TRUSTING AND IMPROVING AI DECISIONS

INPUT (COW IMAGE)

PREDICTION

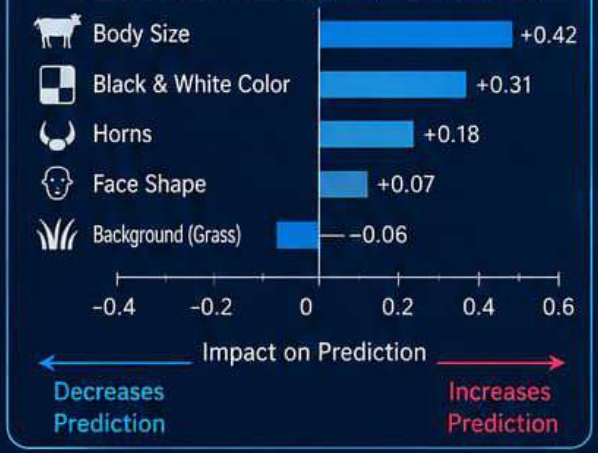
Cow

(97.6%)

WHY?

The model focused on important features like body size, black & white color, horns and face shape which strongly influenced the prediction.

FEATURE IMPORTANCE (SHAP VALUES)



MODEL INTERPRETABILITY METHODS

SHAP	LIME	Decision Trees	Attention Maps
(Feature Attribution)	(Local Explanation)	(Rule Extraction)	(Visual Explanation)

KEY BENEFITS

TRANSPARENCY
Makes AI decisions open and understandable.

TRUST
Builds confidence in AI systems and predictions.

ACCOUNTABILITY
Ensures responsible and ethical AI development.

RELIABILITY
Improves model performance and robustness.

FAIRNESS
Helps detect and reduce bias in AI models.

Author

Dr.C.V.Lakshmi Narayana

Assistant Professor, Department of AI&ML

Annamacharya University, Rajampet

Lecture Notes on

Explainable AI & Model Interpretability

III B.Tech II Semester AI&ML

UNIT-I

Foundations of Explainable AI

Topics: Introduction to Explainability and Interpretability, Importance of XAI in Healthcare, Finance, and Law, White-box vs Black-box Models, Desiderata: Fairness, Accountability, Transparency, Human- Centered AI and Trust, Taxonomy of XAI Techniques (Global vs Local, Post-hoc vs Intrinsic), Regulatory and Ethical Implications (GDPR, AI Bill of Rights), Model Simplicity vs Predictive Power.

***

Introduction to Explainability and Interpretability

Modern AI/ML models (especially deep learning) achieve high accuracy but often behave like black boxes—their internal decision logic is not easily understandable.
Explainability and Interpretability address this gap by answering:

- Why did the model make this decision?
- Can humans trust and validate the model's output?
- Is the model fair, reliable, and legally acceptable?

1.Explainability:

Explainability refers to post-hoc techniques used to explain complex or black-box models after they make predictions.

"**The model may not be transparent, but we generate explanations for its decisions.**"

Key Characteristics

- Model-agnostic or model-specific
- Works on trained models
- Often approximate explanations

Examples

- SHAP (SHapley Additive exPlanations)
- LIME
- Grad-CAM
- Feature importance plots

Properties of Explanations in Machine Learning

An explanation in machine learning is a human-understandable description of how a model maps input features to a prediction. An explanation method is the algorithm that produces such descriptions for a trained model.

Properties of Explanation Methods

Expressive power: This is the set of explanation forms that a method can generate. It defines the representational language of explanations, such as rules, trees, and linear models.

Translucency: This is the degree to which the explanation method uses internal information from the model. High translucency increases precision but restricts applicability to specific model classes.
Portability: This is the range of model types to which an explanation method can be applied. Model-agnostic methods have high portability because they do not depend on internal model details. Model-specific methods, designed for particular architectures such as CNNs or RNNs, have low portability. Portability is inversely related to translucency.
Algorithmic complexity is the computational cost of generating explanations. It depends on the number of model evaluations, synthetic samples, optimization steps, and input dimensionality. This property is critical when explanations must be produced quickly or at scale.

Properties of Individual Explanations

Accuracy measures how well an explanation predicts the true labels of unseen data. It is relevant only when the explanation is used as a predictive model.
Fidelity measures how well an explanation approximates the predictions of the original model. It is the most fundamental requirement for a valid explanation. Fidelity may be global, local to a region of the input space, or restricted to a single instance. An explanation with low fidelity does not describe the model's behavior and is therefore invalid.
Consistency measures how similar explanations are across different models trained on the same task that produce similar predictions.
Stability measures how similar explanations are for similar input instances for the same model. High stability means that small input changes lead to small explanation changes unless the prediction itself changes substantially. High stability is always desirable.
Comprehensibility is the degree to which humans can understand the explanation. It depends on explanation size, feature semantics, cognitive load, and audience expertise. Because it is difficult to measure directly, proxy measures such as rule count, tree depth, number of non-zero coefficients, or human task performance are often used.
Certainty describes whether the explanation reflects the model's confidence or uncertainty. Identical predicted probabilities may differ in reliability due to data density or distance from training data. An explanation that conveys uncertainty is more informative and safer for decision-making.
Degree of importance indicates whether the explanation clearly identifies which features or components contribute most to the prediction. This includes feature rankings, attribution magnitudes, or highlighting dominant rule conditions.
Novelty captures whether the instance lies far from the training-data distribution. High novelty implies low model reliability and weak explanatory value. Novelty is closely related to uncertainty and out-of-distribution detection.
Representativeness refers to the scope of instances covered by an explanation. Global explanations apply to all instances, regional explanations apply to subsets of the input space, and local explanations apply to individual predictions.

Miller's Summary of Good Explanations

Tim Miller argues that good explanations in AI should be designed according to how humans actually understand and use explanations, rather than how engineers traditionally define them.

i. **Contrastive:**
Explanations answer "Why outcome P rather than Q?" rather than "Why P?".
Example: Why was the loan rejected rather than approved?

ii. **Selective:**
Explanations mention only the most relevant causes, not all true causes.
iii. **Focused on abnormal causes:**
Humans prefer explanations that highlight unusual or unexpected factors.
iv. **Causal (not just correlational):**
Good explanations describe cause–effect relations, not mere associations.
v. **Audience-dependent:**
Explanations must be tailored to the user's knowledge, goals, and context.
vi. **Coherent with prior beliefs:**
Explanations should align with common sense and domain knowledge.
vii. **Minimal but sufficient:**
Explanations should be short and simple, containing only necessary information.
viii. **Social and contextual:**
Explanations are part of communication, not just technical descriptions.
ix. **Interactive:**
Explanation is a dialogue that supports follow-up questions and refinement.

2. Interpretability

Interpretability refers to the intrinsic transparency of a model.

"A model is interpretable if a human can directly understand how it works without additional tools."

Key Characteristics

- Built-in transparency
- Simple structure
- Clear relationship between input and output

Examples

- Linear Regression
- Decision Trees (shallow)
- Rule-based systems
- Logistic Regression

Scope of Interpretability in Machine Learning

Interpretability is the degree to which a human can understand how a machine learning system works and why it produces a particular prediction. Since a learning system consists of an algorithm that trains a model and a trained model that makes predictions, interpretability can be studied at multiple scopes: algorithm transparency, global model interpretability, modular interpretability, and local interpretability.

1. Algorithm Transparency

(How does the learning algorithm create the model?)

Algorithm transparency refers to how well the learning procedure itself is understood, independent of any specific dataset or trained model. It concerns how the algorithm updates parameters and what types of relationships it can learn.

Algorithm transparency explains how learning works in general. It does not explain how a specific trained model makes a specific prediction. It requires only knowledge of the algorithm, not of the data or learned parameters.

2. Global (Holistic) Model Interpretability

(How does the trained model make predictions overall?)

Global model interpretability means understanding the entire trained model as a whole. A model is globally interpretable if a human can comprehend how all features, parameters, and structures jointly determine predictions.
This includes understanding feature importance, feature interactions, and how predictions are distributed over the input space.

3. Modular (Component-Level) Interpretability

(How do parts of the model affect predictions?)

Modular interpretability focuses on understanding individual components of a model rather than the entire structure.

- In linear models, the components are feature weights.
- In decision trees, the components are splits, thresholds, and leaf predictions.
- In Naive Bayes, the components are class-conditional probabilities.

Thus, modular interpretability is approximate and context-dependent, though still more feasible than global interpretability.

4. Local Interpretability for a Single Prediction

(Why did the model make this prediction for this instance?)
Local interpretability explains one individual prediction by analyzing model behavior in a small neighborhood around a specific input.
Local explanations are often more accurate and more meaningful than global explanations because they approximate the model only in a small region.

5. Local Interpretability for a Group of Predictions

(Why did the model make these predictions for a subset of instances?)
Group-level interpretability explains predictions for a selected subset of data points.
This can be done by applying global methods to the subset as if it were the entire dataset or by computing local explanations for each instance and aggregating them.

Evaluation of Interpretability

There is no single formal definition of interpretability, and it cannot be measured directly. Three evaluation levels are proposed.

- **Application-level evaluation** tests explanations in real systems with real end users performing real tasks.
  Example: Radiologists evaluate explanations from an ML-based fracture detection system.
  This is the most realistic but also the most expensive and complex.

- **Human-level evaluation** uses simplified tasks and lay users.
  Example: Users choose which explanation is easier to understand.
  This is cheaper but less faithful to real-world use.
- **Function-level evaluation** uses proxy measures without human subjects.
  Example: Tree depth is used as a proxy for interpretability, assuming shallower trees are easier to understand.
  This is valid only when explanation forms are known to be human-understandable and predictive performance is preserved.

Interpretability vs Explainability

Aspect	Interpretability	Explainability
Nature	**Intrinsic**	**Post-hoc**
Model type	**White-box**	**Black-box**
Transparency	**Direct**	**Approximate**
Complexity	**Low**	**High**
Accuracy trade-off	**Possible**	**Usually none**

Importance of XAI in Healthcare, Finance, and Law

Explainable AI (XAI) is critical in high-stakes domains where decisions directly impact human life, money, and justice. Accuracy alone is not sufficient—decisions must be transparent, trustworthy, and accountable.

1. Importance of XAI in Healthcare

Healthcare AI systems support diagnosis, treatment planning, risk prediction, and medical imaging. A wrong or unexplained decision can be life-threatening.

Key Reasons

- Clinical Trust: Doctors must understand *why* a model suggests a diagnosis.
- Patient Safety: Explanations help detect incorrect or biased predictions.
- Medical Accountability: Clinicians remain legally responsible for decisions.
- Regulatory Compliance: Medical standards require justification of decisions.

Example: An AI predicts high cancer risk but XAI shows which symptoms, scans, or biomarkers influenced the decision.

- *XAI enables AI to act as a clinical decision-support tool, not a decision-maker.*

2. Importance of XAI in Finance

AI is widely used in loan approval, credit scoring, fraud detection, and algorithmic trading. Financial decisions affect individuals' economic stability.

Key Reasons

- Regulatory Requirements: Laws require reasons for loan approval or rejection.
- Fairness & Bias Detection: Prevent discrimination based on sensitive features.
- Risk Management: Explain predictions to validate financial risk models.
- Customer Trust: Transparency improves acceptance of automated decisions.

Example: Loan rejection explained by income stability, credit history, and repayment behaviour, not hidden correlations.

- *XAI transforms opaque financial models into auditable decision systems.*

3. Importance of XAI in Law

Legal AI systems assist in sentencing recommendations, bail decisions, legal research, and case outcome prediction. Legal decisions must be justifiable and contestable.

Key Reasons

- Right to Explanation: Individuals can challenge AI-assisted decisions.
- Judicial Transparency: Judges must justify rulings based on interpretable reasoning.
- Bias Prevention: Detect and reduce racial, gender, or socioeconomic bias.
- Ethical & Democratic Values: AI must align with legal principles and human rights.

Example: Risk assessment AI explains why a defendant is classified as high or low risk, enabling judicial review.

- *XAI ensures AI supports justice rather than replacing judicial reasoning.*

White-box vs Black-box Models

White Box Models: White box models are machine learning models whose internal working is transparent and understandable. A human can see how inputs are transformed into outputs.

Key Features

- Clear mathematical or rule-based structure
- Easy to interpret decisions
- Model behaviour is predictable

Examples

- Linear Regression
- Logistic Regression
- Decision Trees
- Rule-based systems

Pros

- High interpretability
- Easy debugging and validation
- Better trust in sensitive decisions

Cons

- Limited ability to model complex patterns
- Often lower accuracy on highly non-linear data

Black Box Models: Black box models are models where internal decision-making is opaque. You can see inputs and outputs, but not the reasoning in between.

Key Features

- Highly complex internal representations
- Strong performance on complex tasks

- Hard to interpret

Examples

- Deep Neural Networks
- Random Forests
- Gradient Boosting models
- Support Vector Machines (non-linear kernels)

Pros

- High predictive power
- Handles large-scale and unstructured data well

Cons

- Lack of transparency
- Difficult to explain or trust decisions
- Potential for hidden bias

White Box Models Vs Black Box Models

Aspect	White Box Models	Black Box Models
Transparency	High	Low
Interpretability	Built-in	Absent
Complexity	Low	High
Accuracy	Moderate	High
Explainability	Native	Requires external methods

Desiderata in AI

Desiderata means *desired properties* or *principles* that an AI system **should satisfy** to be considered responsible, trustworthy, and ethical.

In modern AI, three core desiderata are:

Fairness, Accountability, and Transparency (FAT)

1. Fairness: An AI system is fair if it does not systematically disadvantage individuals or groups based on sensitive attributes.

Key Ideas

- Equal treatment across groups
- No unjustified bias in predictions
- Outcomes should not reflect historical or societal discrimination

Sources of Unfairness

- Biased training data
- Proxy features (e.g., ZIP code acting as a proxy for income or caste)
- Imbalanced datasets
- Feedback loops

Common Fairness Notions

- Demographic Parity – equal outcomes across groups
- Equal Opportunity – equal true positive rates
- Equalized Odds – equal error rates across groups

Example

"If two individuals have the same qualifications, a fair model should not produce different outcomes because of gender, caste, or ethnicity."

2. Accountability: This means, there must be clear responsibility for the decisions made by an AI system. Accountability answers the question:

"Who is responsible when an AI system makes a decision or causes harm?"

Key Ideas

- Clear responsibility and ownership
- Ability to audit decisions
- Possibility of redress or correction

Accountability Requires

- Logging and traceability of decisions
- Human-in-the-loop oversight
- Clear governance and policies

Example

If an AI system rejects a loan or misdiagnoses a patient:

- *Someone must be able to justify the decision*
- *Someone must be responsible for fixing errors*

3. Transparency

Transparency is the degree to which an AI system's operation, decision logic, and data usage are understandable.

Levels of Transparency

- Model transparency – how the model works
- Decision transparency – why a specific output was produced
- Data transparency – what data was used and how

Tools for Transparency

- Interpretable models (white box models)
- Explanation methods (SHAP, LIME)
- Model cards and documentation

Example

A transparent system can explain:

"Your application was rejected mainly due to low income stability and high existing debt."

Explainable AI is a technical enabler of FAT:

- Helps detect **bias** (Fairness)
- Supports audits and responsibility (Accountability)
- Provides understandable explanations (Transparency)

Human-Cantered AI and Trust:

Human-Cantered AI focuses on designing AI systems that prioritize human values, needs, and well-being rather than only optimizing technical performance. **Trust** is a core pillar of HCAI, as AI systems are effective only when users understand, rely on, and feel confident using them.

Human-Cantered AI places humans *in the loop*—AI supports human decision-making instead of replacing it. Trust emerges when AI systems behave predictably, transparently, fairly, and responsibly.

Trust Matters in AI

- Users rely on AI recommendations in **high-stakes domains** like healthcare, finance, and law.
- Lack of trust leads to **underuse, misuse, or rejection** of AI systems.
- Over-trust can cause **automation bias**, where users blindly follow AI outputs.

Key Dimensions of Trust in Human-Centered AI

1. **Transparency**
    a. AI decisions should be understandable and explainable.
    b. Users should know *why* a decision was made.
2. **Explainability**
    a. Clear explanations build confidence in AI outputs.
    b. Essential for accountability and regulatory compliance.
3. **Fairness**
    a. AI should avoid bias and discrimination.
    b. Equal treatment across different user groups.
4. **Accountability**
    a. Clear responsibility for AI decisions.
    b. Mechanisms to audit, contest, and correct outcomes.
5. **Reliability & Safety**
    a. Consistent and accurate performance.
    b. Robustness against errors and adversarial attacks.

Human-in-the-Loop Approach

- Humans supervise, validate, and guide AI systems.
- Improves decision quality and builds user confidence.
- Common in medical diagnosis, credit approval, and legal review.

Examples

- **Healthcare:** AI explains diagnosis suggestions, while doctors make final decisions.
- **Finance:** Credit scoring models provide reasons for loan approval or rejection.
- **Law:** AI assists judges with case analysis but does not replace human judgment.

Human-Centered AI builds **trustworthy AI systems** by aligning technology with human values. Trust is achieved through transparency, explainability, fairness, and accountability—ensuring AI acts as a reliable partner rather than an opaque decision-maker.

Taxonomy of AI

Explainable AI (XAI) techniques can be systematically classified based on **what they explain** and **how explanations are generated**. Two widely accepted dimensions are:

1. **Global vs Local explanations**
2. **Post-hoc vs Intrinsic explanations**

1. Global vs Local Explanations

A. Global Explanations

Global explanations aim to explain the **overall behaviour of the model** across the entire dataset.

Key Characteristics

- Provide a **holistic understanding** of model logic
- Useful for **model validation, auditing, and governance**
- Help answer *"How does the model work in general?"*

Common Techniques

- Feature importance (e.g., global SHAP values)
- Partial Dependence Plots (PDP)
- Model summaries (decision rules, coefficients)
- Surrogate models (interpretable models approximating black-box models)

Use Cases

- Regulatory compliance (finance, law)
- Model debugging
- Trust building for stakeholders

B. Local Explanations

Local explanations focus on **individual predictions** made by the model.

Key Characteristics

- Explain *why* a specific prediction was made
- Highly useful for **user-level transparency**
- Instance-specific and contextual

Common Techniques

- LIME (Local Interpretable Model-agnostic Explanations)
- Local SHAP values
- Counterfactual explanations
- Saliency maps (for images)

Use Cases

- Credit approval/rejection explanations
- Medical diagnosis justification

- Personalized decision support

2. Post-hoc vs Intrinsic Explanations

A. Post-hoc Explanations

Post-hoc techniques generate explanations **after the model is trained**, without modifying the original model.

Key Characteristics

- Model-agnostic or model-specific
- Applicable to **black-box models**
- Explanations are approximations, not the model itself

Common Techniques

- LIME
- SHAP
- Saliency maps
- Feature attribution methods
- Counterfactual explanations

Advantages

- Flexible and widely applicable
- Suitable for complex models (deep learning, ensembles)

Limitations

- Risk of **approximation errors**
- May not fully reflect true model logic

B. Intrinsic (Interpretable-by-design) Models

Intrinsic explanations come from **models that are inherently interpretable**.

Key Characteristics

- Transparency is built into the model
- Explanations are **faithful and exact**
- Often simpler models

Common Models

- Linear regression
- Logistic regression
- Decision trees
- Rule-based systems
- Generalized Additive Models (GAMs)

Advantages

- High transparency and trust
- Easier validation and debugging

Limitations

- May sacrifice predictive performance
- Less suitable for highly complex data

Regulatory and Ethical Implications in AI

The rapid adoption of Artificial Intelligence (AI) systems has raised significant **ethical, legal, and societal concerns**. To address risks such as privacy violations, algorithmic bias, lack of transparency, and misuse of personal data, governments and institutions have introduced **regulatory frameworks**. Two major initiatives are the **General Data Protection Regulation (GDPR)** and the **AI Bill of Rights**.

1. General Data Protection Regulation (GDPR)

GDPR is a comprehensive data protection law enacted by the **European Union (EU)** to safeguard individual privacy and regulate the processing of personal data.

Key Ethical Principles of GDPR

- **Lawfulness, fairness, and transparency** in data processing
- **Purpose limitation**: Data should be collected only for specific, legitimate purposes
- **Data minimization**: Collect only necessary data
- **Accuracy** of personal data
- **Storage limitation**
- **Integrity and confidentiality**

GDPR and AI Systems

- AI models often rely on **large-scale personal data**, increasing risks of misuse.
- GDPR mandates that individuals have the **right to know** how their data is used.
- **Article 22** gives individuals the **right not to be subject to fully automated decision-making**, especially when it has legal or significant effects.

Ethical Implications

- Encourages **Explainable AI (XAI)** by requiring transparency in automated decisions.
- Protects individuals from **opaque "black-box" AI systems**.
- Forces organizations to consider **fairness, accountability, and privacy** by design.

2. AI Bill of Rights

The **AI Bill of Rights**, proposed by the **White House (USA)**, is a framework aimed at protecting citizens from harmful uses of AI while promoting responsible innovation.

Core Principles of the AI Bill of Rights

1. **Safe and Effective Systems**
    - AI systems should be tested and monitored to prevent harm.
2. **Algorithmic Discrimination Protections**
    - Prevent bias based on race, gender, religion, or other sensitive attributes.
3. **Data Privacy**
    - Users should have control over how their data is collected and used.
4. **Notice and Explanation**
    - Individuals must be informed when AI is used and receive understandable explanations.
5. **Human Alternatives, Consideration, and Fallback**
    - Humans should be able to intervene or override AI decisions.

Ethical Implications

- Promotes **human-centered AI** by keeping humans in control.
- Emphasizes **trust, transparency, and accountability**.
- Supports fairness and reduces the risk of **algorithmic harm**.

Regulatory frameworks like GDPR and the AI Bill of Rights play a crucial role in shaping ethical, transparent, and human-centered AI systems. They encourage the adoption of Explainable AI, protect individual rights, and ensure that AI technologies serve society responsibly rather than harmfully.

Model Simplicity vs Predictive Power

The trade-off between **model simplicity** and **predictive power** is a central theme in **machine learning**, **statistics**, and **Explainable AI (XAI)**. It reflects the balance between how *understandable* a model is and how *accurately* it can predict outcomes.

1. Model Simplicity

Definition:

Model simplicity refers to how easy a model is to understand, interpret, and explain.

Characteristics

- Fewer parameters and rules
- Clear decision logic
- Human-interpretable structure

Examples

- Linear Regression
- Logistic Regression
- Decision Trees (shallow)
- Rule-based systems

Advantages

- High **interpretability** and **transparency**
- Easier to debug and validate
- Lower risk of overfitting
- Preferred in **regulated domains** (healthcare, finance, law)

Limitations

- Limited ability to model complex, non-linear relationships
- May **underfit** complex data

2. Predictive Power

Definition:

Predictive power refers to a model's ability to accurately predict unseen or future data.

Characteristics

- Captures complex patterns
- Handles high-dimensional and non-linear data
- Often data-hungry

Examples

- Deep Neural Networks
- Random Forests
- Gradient Boosting (XGBoost, LightGBM)
- Support Vector Machines (with kernels)

Advantages

- High accuracy and robustness
- Better performance on complex real-world problems

Limitations

- Low interpretability ("black-box" nature)
- Computationally expensive
- Harder to justify decisions

Techniques to Balance Simplicity and Predictive Power

1. **Feature Engineering**
    - Transform features to allow simple models to capture non-linearities.
    - Example: Polynomial features, interaction terms.
2. **Model Compression / Distillation**
    - Train a **complex model** and then distill it into a **simpler model**.
    - Retains much predictive power while increasing interpretability.
3. **Hybrid Models**
    - Combine interpretable models with black-box models.
    - Example: Rule-based filters + neural network classifier.
4. **Post-hoc Explainability**
    - Use tools like **SHAP, LIME, or Anchors** to explain black-box predictions.

***

UNIT-II

Model-Specific Explainability Techniques

Topics: Decision Trees and Rule-based Models, Linear Models and Feature Importance, Generalized Additive Models (GAMs), Visualization of Weights and Coefficients, Logistic Regression Coefficient Interpretation, Case Study: Credit Scoring using Transparent Models, Comparison of Interpretable ML Models, Use Cases and Trade-offs.

***

Decision Trees and Rule-based Models:

Decision Tree is one of the most important and interpretable Machine Learning Model. They work by splitting data step-by-step how similar to humans make decisions.

A **Decision Tree (DT)** is a **tree-like model** used for making decisions or predictions. It splits data into branches based on **feature values** until it reaches a **decision (output)** at the leaves.

- **Nodes**:
  - **Root node**: The first node (top of the tree) representing the entire dataset.
  - **Decision nodes**: Nodes where the dataset is split based on a feature.
  - **Leaf nodes**: Terminal nodes that give the **prediction/output**.
- **Branches**: Represent the outcome of a test/split on a feature.

Types of Decision Trees

1. **Classification Tree**
   - Output: **Categorical** (e.g., Pass/Fail, Yes/No).
   - Example: Predicting whether a student passes based on study hours.
2. **Regression Tree**
   - Output: **Continuous** (e.g., salary, temperature).
   - Example: Predicting a student's score based on study hours.

How Decision Trees Work

Decision Trees use **splitting criteria** to decide the best feature and threshold to split the dataset.
Common splitting criteria:

3. **Gini Index (for classification)**
   Measures **impurity**. Lower Gini → purer node.

   $$Gini = 1 - \sum_{i=1}^{C} p_i^2$$

   where p_i = proportion of class i in the node.
4. **Entropy / Information Gain (for classification)**
   Entropy measures **uncertainty**.

   $$Entropy = -\sum_{i=1}^{C} p_i \log_2(p_i)$$

   Information Gain = Reduction in entropy after split. Higher gain → better split.

5. **Variance Reduction (for regression)**
   Measures how well the split reduces variance in continuous outputs.

Advantages of Decision Trees

- Simple to **understand & visualize**.
- Can handle **categorical & numerical** data.
- Non-linear relationships are captured.
- No need for **feature scaling**.

Disadvantages of Decision Trees

- Prone to **overfitting** (especially deep trees).
- Small changes in data → **different tree**.
- Can be **biased** if one class dominates.

Rule-Based Models

Rule-Based Models are machine learning or decision-making models that make predictions using explicit IF–THEN rules.

A Rule-Based Model represents knowledge in the form of human-readable rules derived either manually by experts or automatically from data.

General form of a rule:

IF condition(s) THEN outcome

Simple Example (Student Performance)

Problem: Predict whether a student will Pass or Fail

Rules:

IF Study_Hours ≥ 6ANDAttendance ≥ 75%THENPassIFStudy_Hours<6THENFail

- Easy to understand
- No complex mathematics

Types of Rule-Based Models

(A) Expert Rule-Based Systems

- Rules are hand-crafted by domain experts
- Used in medical diagnosis, legal systems

Example:

IF Fever= High AND Cough= Yes THEN Disease= Flu

(B) Data-Driven Rule-Based Models

- Rules are learned automatically from data
- Examples:
    1. Decision Trees
    2. RuleFit
    3. RIPPER Algorithm

Example:

IF Income > 50,000 AND Credit_Score > 700 THEN Loan= Approved

Mathematical Representation

Let:

- $R = \{r_1, r_2, \ldots, r_n\}$ → Set of rules
- Each rule:

$$r_i\text{:IF } (x_1 \in A_1) \wedge (x_2 \in A_2) \Rightarrow y$$

Prediction:

$$\hat{y} = \bigcup_{i=1}^{n} r_i(x)$$

Example with Dataset

Study Hours	Attendance	Result
2	60%	Fail
4	65%	Fail
6	75%	Pass
8	85%	Pass

Generated Rules:

IF Study_Hours ≥ 6 AND Attendance ≥ 75% THEN Pass ELSE Fail

Advantages of Rule-Based Models

i. Highly interpretable
ii. Transparent decision-making
iii. Easy to explain to non-technical users
iv. Suitable for regulatory domains

Limitations

i. Rule explosion for large dataset
ii. Hard to manage conflicting rules
iii. Poor performance on complex patterns
iv. Manual rules may be biased

Rule-Based Model vs Decision Tree

Feature	Rule-Based Model	Decision Tree
Structure	Independent rules	Tree structure
Interpretability	Very high	High

Feature	Rule-Based Model	Decision Tree
Rule Overlap	Possible	No overlap
Flexibility	More flexible	Structured

"Decision Tree can be converted into rules"

Real-World Applications

Domain	Use Case
Healthcare	Disease diagnosis
Banking	Credit approval
Education	Student performance prediction
Fraud Detection	Transaction monitoring
Law	Compliance checking

Linear Models

A **linear model** is a mathematical model that expresses the output (dependent variable) as a **linear combination of input features** (independent variables) and unknown parameters (weights).

"The prediction is a weighted sum of inputs plus a bias."

Linear models assume that the relationship between inputs and output is **linear in parameters**.

They are used to:

- Predict **continuous values** => Linear Regression
- Predict **class labels** => Logistic Regression
- Perform **regularized learning** => Ridge, Lasso, Elastic Net

They are popular because they are:

- Simple
- Fast
- Interpretable
- Mathematically well-understood

Mathematical Formulation

For a single data point with n features:

$$y = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b$$

Where:

- x_i → input features
- w_i → weights (parameters)
- b → bias (intercept)
- y → predicted output

Why Are They Called "Linear"?

They are called **linear** because:

- The model is **linear in parameters** (the weights w_i).
- Each feature contributes **additively and proportionally** to the output.

- Even if features are transformed (e.g., x^2, $\log x$), the model is still linear **as long as the parameters appear linearly**.

Common Types of Linear Models

1. **Linear Regression:**
   Used for continuous outputs.

$$y = w^T x + b$$
$$w = 1$$
$$t = 1$$
$$b = 1$$

   Example: Predicting house prices, temperature, and salary.

2. **Logistic Regression (Classification)**
   Used for **binary classification**.

$$\hat{y} = \sigma(w^T x + b)$$

   Where $\sigma(z) = \frac{1}{1+e^{-z}}$ (sigmoid function)

   Example: Spam detection, disease prediction.

3. **Ridge Regression**
   Linear regression + **L2 regularization** (penalizes large weights).

$$\min \sum (y - \hat{y})^2 + \lambda \sum w_i^2$$

4. **Lasso Regression**
   Linear regression + **L1 regularization** (drives some weights to zero → feature selection).

$$\min \sum (y - \hat{y})^2 + \lambda \sum |w_i|$$

5. **Elastic Net**
   Combination of Ridge + Lasso.

Key Characteristics

Feature	Description
Simplicity	Easy to understand and implement
Interpretability	Weights show feature importance
Speed	Fast training and prediction
Scalability	Works well with large datasets
Limitations	Cannot capture complex non-linear patterns

When Should You Use Linear Models?

Use linear models when:

- The relationship between inputs and output is **approximately linear**
- You want **high interpretability**

- You need **fast training and inference**
- The dataset is **small to medium sized**
- You want a **baseline model**

Linear Regression

Linear Regression is a supervised learning algorithm used to model the relationship between:

- an **input variable** x (feature)
- an **output variable** y (target)

by fitting a straight line:

$$y = mx + c$$
$$m = 1$$
$$c = 1$$

Where:

- m = slope (how fast y changes when x changes)
- c = intercept (value of y when $x = 0$

In ML notation:

$$\hat{y} = \beta_0 + \beta_1 x$$

- $\hat{y}$ = predicted value
- β_0 = intercept
- β_1 = coefficient (slope)

Mathematical Objective

We want the **best-fitting line** such that prediction errors are minimized.
For data points:

$$(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$$

Prediction:

$$\hat{y}_i = \beta_0 + \beta_1 x_i$$

Error (residual):

$$e_i = y_i - \hat{y}_i$$

We minimize **Mean Squared Error (MSE):**

$$J(\beta_0, \beta_1) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - (\beta_0 + \beta_1 x_i)\right)^2$$

Closed-Form Solution (Normal Equation)

The optimal parameters:
$\beta_1 = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}$ $\beta_0 = \bar{y} - \beta_1 \bar{x}$
Where:

- $\bar{x}$ = mean of x
- $\bar{y}$ = mean of y

Simple Example

Dataset

Suppose we have:

x (Study Hours)	y (Score)
1	2
2	3
3	5
4	4
5	6

Step 1: Compute Means

$\bar{x} = \frac{1+2+3+4+5}{5} = 3\ \bar{y} = \frac{2+3+5+4+6}{5} = 4$

Step 2: Compute β_1

$\beta_1 = \frac{\sum(x_i-\bar{x})(y_i-\bar{y})}{\sum(x_i-\bar{x})^2} = \frac{(1-3)(2-4)+(2-3)(3-4)+(3-3)(5-4)+(4-3)(4-4)+(5-3)(6-4)}{(1-3)^2+(2-3)^2+(3-3)^2+(4-3)^2+(5-3)^2} = \frac{4+1+0+0+4}{4+1+0+1+4} = \frac{9}{10} = 0.9$

Step 3: Compute β_0

$\beta_0 = \bar{y} - \beta_1\bar{x} = 4 - (0.9)(3) = 4 - 2.7 = 1.3$

Final Regression Equation

$$\hat{y} = 1.3 + 0.9x$$

Prediction Example

If a student studies **6 hours**:

$$\hat{y} = 1.3 + 0.9(6) = 1.3 + 5.4 = 6.7$$

Predicted Score ≈ 6.7

Linear Regression – Interpretable Models

- **Core Advantage of Linear Regression**
  The main advantage of linear regression is linearity, which makes estimation simple and interpretation transparent. Each coefficient represents a direct marginal effect of a feature, making the model widely used in medicine, sociology, and psychology for explainable analysis.
- **Weights and Confidence Intervals**
  Each coefficient estimate is accompanied by a confidence interval that quantifies uncertainty about the true weight. A 95% confidence interval means that 95 out of 100 such intervals from repeated samples would contain the true coefficient, assuming the model is correct.
- **Linearity Assumption**
  The expected outcome is assumed to be a linear and additive function of the input features. Nonlinear effects or interactions must be modelled explicitly using interaction terms, polynomials, or splines.
- **Normality Assumption**
  The error terms are assumed to follow a normal distribution. Violations of normality mainly invalidate confidence intervals and hypothesis tests.
- **Homoscedasticity Assumption**
  The variance of the error terms is assumed to be constant across all feature values. If violated, standard errors and confidence intervals become unreliable.

- **Independence Assumption**
  All observations are assumed to be statistically independent. Dependence, such as repeated measurements per subject, leads to incorrect inference unless special models are used.
- **Fixed Features Assumption**
  Input features are treated as error-free constants rather than random variables. Measurement error in features biases coefficient estimates and requires complex correction models.
- **Absence of Multicollinearity**
  Predictor variables are assumed not to be strongly correlated with each other. High multicollinearity makes coefficient estimates unstable and difficult to interpret.

Interpretation

- **Interpretation of Numerical Features**
  A numerical feature's coefficient represents the expected change in outcome for a one-unit increase in that feature, holding others constant. This interpretation relies on the linear and additive structure of the model.
- **Interpretation of Binary Features**
  A binary feature's coefficient represents the change in predicted outcome when switching from the reference category to the other category. All other features are held constant in this comparison.
- **Interpretation of Categorical Features**
  Categorical variables are encoded using L–1 indicator variables for L categories. Each coefficient represents the effect of that category relative to the reference category.
- **Interpretation of the Intercept**
  The intercept represents the predicted outcome when all numerical features are zero and all categorical features are at reference levels. After standardization, it represents the prediction at average feature values.

Feature Importance in Linear Regression

In linear regression, a very principled way to measure feature importance is by using the **t-statistic** of each coefficient.
This method considers **both effect size and uncertainty**, unlike raw coefficients alone.

Linear Regression Model

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p + \varepsilon$$

Each coefficient β_j is estimated from data as $\hat{\beta}_j$.

What is the t-Statistic?

For each feature x_j:

$$t_j = \frac{\hat{\beta}_j}{SE(\hat{\beta}_j)}$$

Where:

- $\hat{\beta}_j$ = estimated coefficient
- $SE(\hat{\beta}_j)$ = standard error of that coefficient
- t_j = t-statistic for feature j

Interpretation of the t-Statistic

The t-statistic answers:

"How many standard errors away from zero is this coefficient?"

- Large $|t_j|$ → feature has **strong and reliable influence**
- Small $|t_j|$ → feature effect is weak or uncertain
- Sign of t_j:
  - Positive → positive effect
  - Negative → negative effect

Feature Importance Score Using t-Statistic

A natural importance measure:

$$I_j = |t_j|$$

or sometimes

$$I_j = t_j^2$$

So:

- Larger $|t_j|$ ⇒ more important feature
- Smaller $|t_j|$ ⇒ less important feature

Logistic Regression

What is Logistic Regression? (In Simple Words)

Logistic Regression is a **classification algorithm** used to predict **yes/no (0/1)** outcomes.
Examples:

- Will a student **pass or fail**?
- Is an email **spam or not spam**?
- Does a customer **buy or not buy**?

It predicts a **probability** between 0 and 1 and then converts it into a class label.

How Logistic Regression Defined

1. Take input features x
2. Multiply by weights w
3. Add bias b
4. Pass result into a **sigmoid function**
5. Get probability

Mathematical Model

Step 1: Linear Equation

$$z = wx + b$$

Step 2: Sigmoid Function

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Step 3: Predicted Probability

$$\hat{p} = \sigma(wx + b)$$

Step 4: Class Prediction

$$\hat{y} = \begin{cases} 1 & \text{if } \hat{p} \geq 0.5 \\ 0 & \text{if } \hat{p} < 0.5 \end{cases}$$

Example: Simple Numerical Dataset

Suppose we want to predict whether a student **passes (1)** or **fails (0)** based on **hours studied (x)**.

Student	Hours Studied (x)	Result (y)
A	1	0
B	2	0
C	3	0
D	4	1
E	5	1

Assume Learned Model Parameters

(For simplicity, we assume these are already trained)

$$w = 1.5, \quad b = -5$$

Predict for a New Student (x = 4)

Step 1: Linear Score

$$z = wx + b = 1.5(4) - 5 = 6 - 5 = 1$$

Step 2: Apply Sigmoid

$\hat{p} = \frac{1}{1+e^{-1}}\, e^{-1} \approx 0.3679\, \hat{p} = \frac{1}{1+0.3679} = \frac{1}{1.3679} \approx 0.731$

Step 3: Final Prediction

$$\hat{p} = 0.731 \geq 0.5 \Rightarrow \hat{y} = 1$$

Prediction: Student passes ✓

One More Prediction (x = 2)

Step 1: Linear Score

$$z = 1.5(2) - 5 = 3 - 5 = -2$$

Step 2: Sigmoid

$$\hat{p} = \frac{1}{1 + e^2} \approx \frac{1}{1 + 7.389} = 0.119$$

Step 3: Prediction

$$0.119 < 0.5 \Rightarrow \hat{y} = 0$$

Prediction: Student fails ✗

Logistic Regression Interpretation

1. **What Logistic Regression Predicts**

Logistic Regression is used for **binary classification** (e.g., Yes/No, 1/0, Pass/Fail).

Instead of predicting a value directly, it predicts a **probability**:

$$P(Y = 1 \mid X) \in (0,1)$$

It uses the **logistic (sigmoid) function**:

$$P(Y = 1 \mid X) = \frac{1}{1 + e^{-z}}, \quad \text{where } z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p$$

2. **Log-Odds (Logit) Interpretation**

Logistic regression models the **log-odds** of the event:

$$\log\left(\frac{P(Y = 1)}{1 - P(Y = 1)}\right) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p$$

This is called the **logit**.

Meaning:

- The left side = log of odds of success
- The right side = linear function of predictors

3. **Interpreting the Coefficients (β)**

Each coefficient β_j represents the change in **log-odds** for a **1-unit increase** in x_j, keeping other Variables constant.

$$\beta_j = \text{change in log-odds per unit increase in } x_j$$

Example

Suppose:

$$\log\left(\frac{P}{1 - P}\right) = -2 + 0.8 \cdot \text{HoursStudy}$$

- A 1-hour increase in study time increases the **log-odds** of passing by **0.8**.

4. **Interpreting the Intercept (β_0)**

$$\beta_0 = \log\left(\frac{P(Y = 1 \mid X = 0)}{1 - P(Y = 1 \mid X = 0)}\right)$$

Meaning:

- Log-odds of success when **all predictors = 0**
- Often not meaningful if "0" is not realistic.

5. **From Coefficients to Probabilities**

Given:

$z = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p$ $P(Y = 1) = \frac{1}{1+e^{-z}}$

Example

$z = -2 + 0.8(3) = 0.4$ $P(Y = 1) = \frac{1}{1+e^{-0.4}} \approx 0.598$

Interpretation:

A student who studies 3 hours has about a **59.8%** chance of passing.

6. Categorical Variables Interpretation

If x_j is binary (e.g., Gender: 1 = Male, 0 = Female):

$\beta_j =$ difference in log-odds between Male and Female $e^{\beta_j} =$ odds ratio between groups

Example

$$\beta_{\text{Male}} = 0.5 \Rightarrow e^{0.5} = 1.65$$

Interpretation:

Males have **1.65× higher odds** of success than females, holding other variables constant.

Generalized Additive Model (GAM)

A Generalized Additive Model (GAM) is an extension of:

- Linear Regression
- Generalized Linear Models (GLMs)

That allows non-linear relationships between predictors and the response variable while remaining interpretable. Instead of assuming a straight-line effect for each feature, GAM learns a smooth curve for each feature.

Mathematical Formulation

For a response variable Y with predictors $X_1, X_2, \ldots, X_p$:

$$g(\mathbb{E}[Y]) = \beta_0 + f_1(X_1) + f_2(X_2) + \cdots + f_p(X_p)$$

Where:

- $g(\cdot)$ = link function (as in GLMs)
- β_0 = intercept
- $f_j(X_j)$ = smooth function of predictor X_j
- Each f_j is learned from data (e.g., splines)

How GAMs Differ from Linear Models

Model Type	Form
Linear Regression	$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$
GLM	$g(E[Y]) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p$
GAM	$g(E[Y]) = \beta_0 + f_1(X_1) + \cdots + f_p(X_p)$

GAM replaces linear terms $\beta_j X_j$ with smooth functions $f_j(X_j)$.

What Are the $f_j(X_j)$ Functions?

These are smooth functions, usually represented using:

- Splines (cubic splines, thin-plate splines)
- Kernel smoothers
- Local regression

Example:

- Instead of:
  $Y = \beta_0 + \beta_1 \text{Age}$
- GAM learns:
  $Y = \beta_0 + f_1(\text{Age})$

Where $f_1(\text{Age})$ might be:

- Increasing at first
- Flattening
- Decreasing later

Interpretation of GAMs

1. **Global Model Interpretation**

   The model is additive:

   $$\text{Prediction} = \beta_0 + f_1(X_1) + f_2(X_2) + \cdots$$

   Each feature contributes independently to the final prediction.

2. **Interpreting Individual Terms**

   Each function $f_j(X_j)$ answers:

   "How does feature X_j influence the prediction, keeping all other variables constant?"

   You interpret GAMs by plotting each smooth function.

3. **Example: House Price Prediction**

   Suppose:

   $$\text{Price} = \beta_0 + f_1(\text{Size}) + f_2(\text{Age}) + f_3(\text{LocationScore})$$

   Interpretation:

   - $f_1(\text{Size})$:
     - Shows how price grows with house size
     - Might increase rapidly then slow down (diminishing returns)
   - $f_2(\text{Age})$:
     - Shows depreciation over time
     - Might drop fast initially, then stabilize
   - $f_3(\text{LocationScore})$:
     - Shows effect of better neighborhoods
     - Possibly nonlinear jumps

   Each curve is interpretable on its own.

Why Are GAMs Interpretable?

Because:

- The model is additive
- Each feature has its own smooth curve
- No complex interactions unless explicitly added

This gives glass-box ML behavior.

Advantages of GAMs

✓ Interpretable
✓ Capture nonlinear effects
✓ Work with different data types
✓ Strong predictive power
✓ Flexible but structured

Limitations of GAMs

✗ Additive assumption
✗ Limited interaction modeling
✗ Can overfit if poorly regularized
✗ Harder to scale to very high dimensions

When Should You Use GAMs?

Use GAMs when:

- You need interpretability
- Relationships are nonlinear
- Interactions are weak or moderate
- You want a transparent ML model

Visualization of Weights and Coefficients

Visualization of weights and coefficients is a technique used in interpretable machine learning models to understand how input features influence the model's predictions. Many transparent models — such as Linear Regression, Logistic Regression — assign numeric weights (coefficients) to each feature.

1.Linear Regression: Weights and Coefficients

Linear Regression predicts a **continuous outcome** y from input features X using a linear equation:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n$$

- β_0 → Intercept (value of y when all x=0)
- β_i → Coefficient or weight for feature x_i (effect on y per unit change in x_i)

Visualization of Coefficients

1. **Bar Chart**
   - Plot β_i values for each feature.
   - Positive β_i → increases y, Negative → decreases y
2. **Interpretation**
   - Largest absolute β → most influential feature

- Sign (+/-) → direction of effect
- Helps in **feature importance analysis**

Key Points

- Linear regression coefficient = slope of feature → effect on output
- Visualization helps **quickly identify important features**
- Always check units → coefficients are in **units of y per unit of x**

2. Logistic Regression: Weights and Coefficients

Logistic Regression predicts **probability of a binary outcome** $P(y = 1)$ using the logistic function:

$$P(y = 1) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \ldots + \beta_n x_n)}}$$

- β_i → change in **log-odds** for a one-unit increase in x_i

Visualization of Coefficients

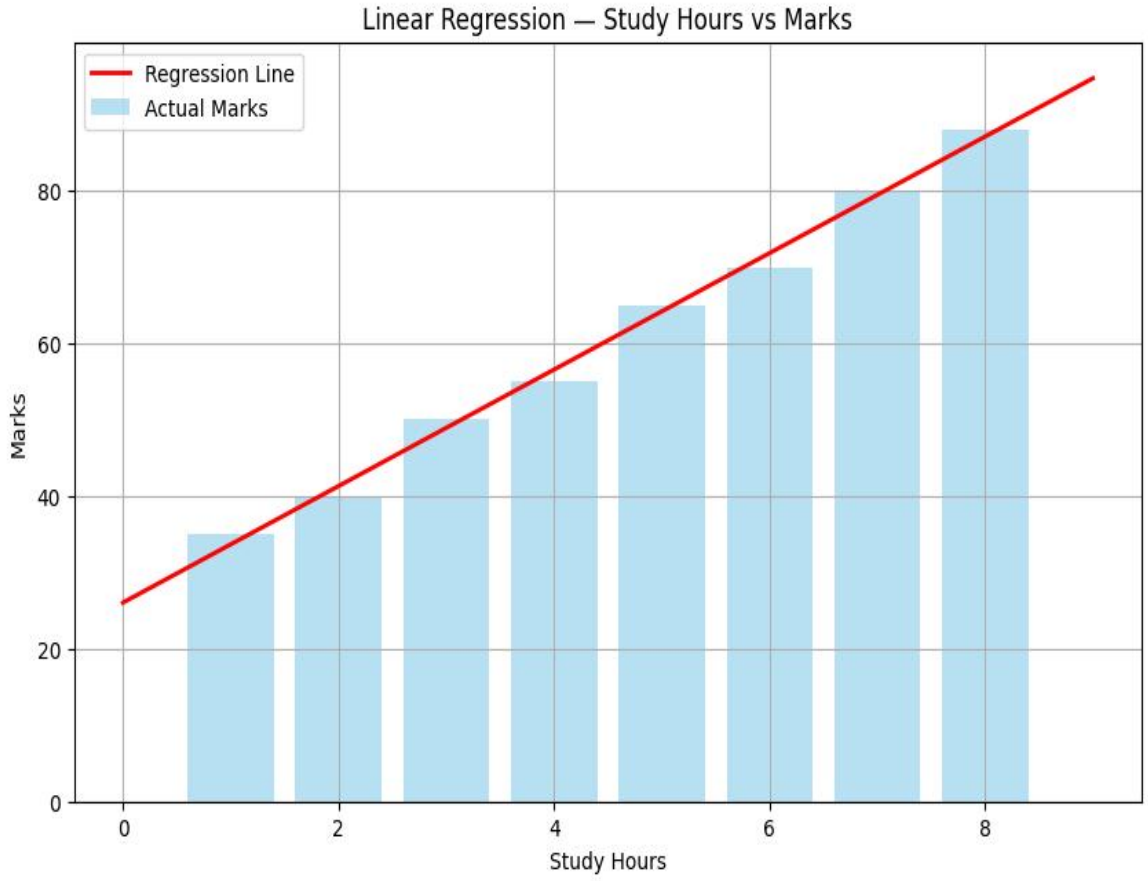
1. **Bar Chart of Coefficients (β_i)**
   - Positive β_i → increases probability of y=1
   - Negative β_i → decreases probability of y=1
2. **Optional: Odds Ratio Visualization**
   - Convert coefficients to odds ratios: $OR = e^{\beta_i}$
   - OR > 1 → increases probability, OR < 1 → decreases probability

Linear Regression

#Program

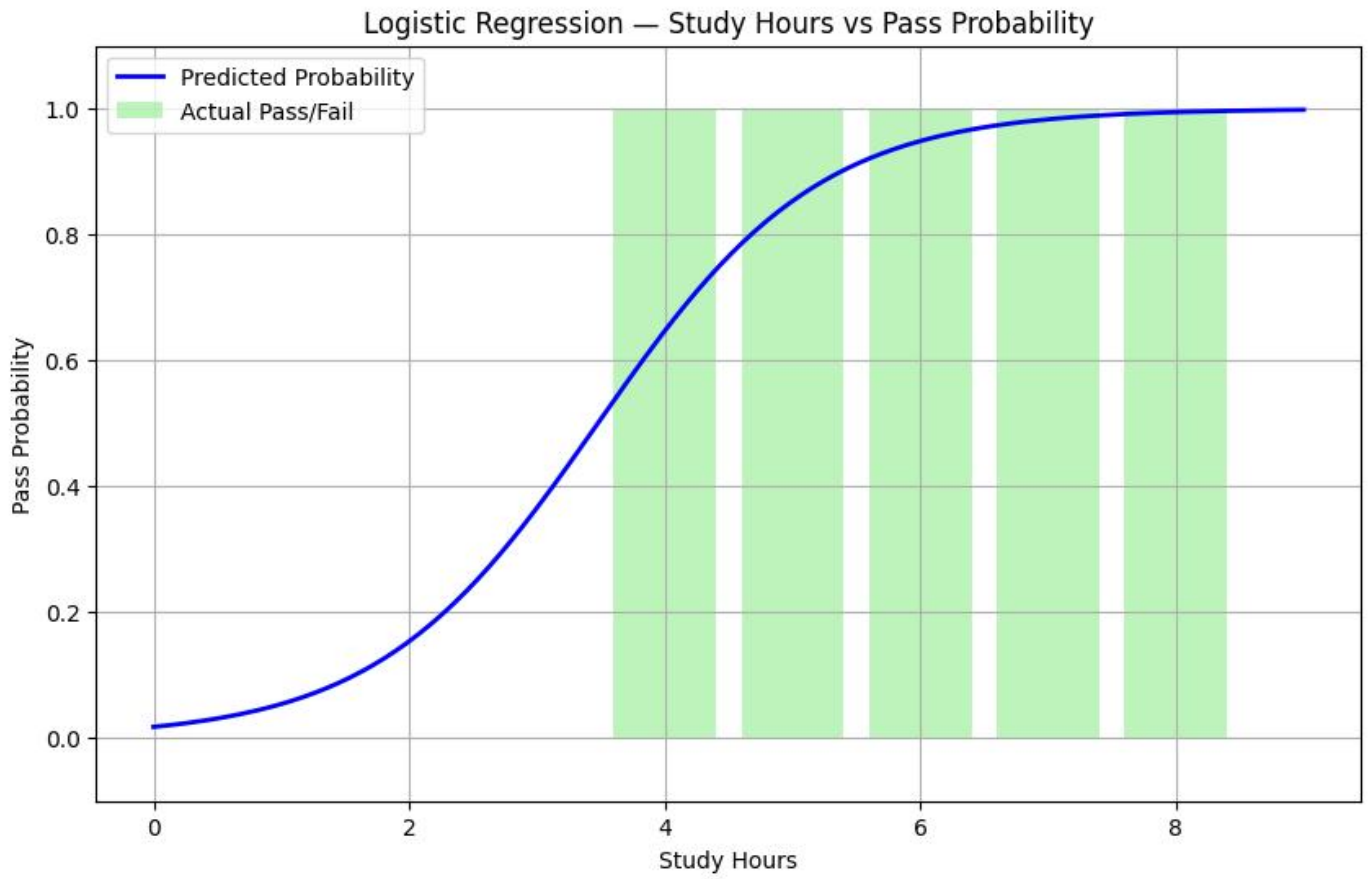
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Data
hours = np.array([1,2,3,4,5,6,7,8]).reshape(-1,1)
marks = np.array([35,40,50,55,65,70,80,88])

# Model
lin_model = LinearRegression()
lin_model.fit(hours, marks)

# Coefficients
b0 = lin_model.intercept_[0]
b1 = lin_model.coef_[0]

print("Linear Regression Intercept:", b0)
print("Linear Regression Weight:", b1)

# Predictions for regression line
```

```
x_range = np.linspace(0,9,100).reshape(-1,1)
y_pred = lin_model.predict(x_range)

# Plot
plt.figure(figsize=(10,6))

# Bar chart of actual marks
plt.bar(hours.flatten(), marks, color='skyblue', alpha=0.6, label='Actual Marks')

# Regression line
plt.plot(x_range, y_pred, color='red', linewidth=2, label='Regression Line')

# Labels
plt.xlabel("Study Hours")
plt.ylabel("Marks")
plt.title("Linear Regression — Study Hours vs Marks")
plt.legend()
plt.grid(True)
plt.show()
```



#Logistic Regression

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression

# Data: Study hours vs Pass/Fail
hours = np.array([1,2,3,4,5,6,7,8]).reshape(-1,1)
pass_fail = np.array([0,0,0,1,1,1,1,1])

# Logistic Regression Model
log_model = LogisticRegression()
log_model.fit(hours, pass_fail)

# Coefficients
b0_log = log_model.intercept_[0]
b1_log = log_model.coef_[0][0]

print("Logistic Regression Intercept:", b0_log)
print("Logistic Regression Weight:", b1_log)

# Probability predictions for smooth curve
x_range = np.linspace(0, 9, 200).reshape(-1,1)
prob = log_model.predict_proba(x_range)[:,1]   # Probability of passing

# Plot
plt.figure(figsize=(10,6))

# Bar chart of actual pass/fail labels
plt.bar(hours.flatten(), pass_fail, color='lightgreen', alpha=0.6, label='Actual Pass/Fail')

# Logistic regression probability curve
plt.plot(x_range, prob, color='blue', linewidth=2, label='Predicted Probability')

# Labels, title, legend
plt.xlabel("Study Hours")
plt.ylabel("Pass Probability")
plt.title("Logistic Regression – Study Hours vs Pass Probability")
plt.legend()
plt.grid(True)
plt.ylim(-0.1, 1.1)   # Proper display for 0 and 1
plt.show()
```



Logistic Regression Coefficient Interpretation

Logistic Regression is used when the **target variable is binary** (e.g., Yes/No, 1/0, Pass/Fail).
Its coefficients **do not directly represent change in the outcome**, but rather change in **log-odds**.

Logistic Regression Model Equation

$$\log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n$$

Where:

- p = probability of the positive class (Y = 1)
- $\frac{p}{1-p}$ = odds
- $\log\left(\frac{p}{1-p}\right)$ = log-odds
- β_0 = intercept
- β_i = coefficient of feature X_i

Meaning of a Coefficient (β_i)

A coefficient tells how much the log-odds of the outcome change when the feature increases by 1 unit, holding other variables constant.

Interpreting Sign of Coefficients

Coefficient Value	Interpretation
$\beta_i > 0$	Feature increases probability of Y = 1
$\beta_i < 0$	Feature decreases probability of Y = 1
$\beta_i = 0$	No effect

Interpreting Magnitude (Log-Odds)

Example:

$$\beta_1 = 0.8$$

A **1-unit increase** in X_1 increases the **log-odds** by **0.8**

But log-odds are not intuitive → convert to **odds ratio**

Odds Ratio Interpretation

$$\text{Odds Ratio} = e^{\beta_i}$$

Example 1: Positive Coefficient

$$\beta = 0.7 \Rightarrow e^{0.7} = 2.01$$

Odds **double (2×)** when the feature increases by 1 unit

Example 2: Negative Coefficient

$$\beta = -0.5 \Rightarrow e^{-0.5} = 0.61$$

Odds **decrease by 39%** (1 − 0.61)

Intercept (β_0) Interpretation

- Represents **log-odds when all features are zero**
- Often **not meaningful** unless zero has real meaning

Example:

$$\beta_0 = -2 \Rightarrow e^{-2} = 0.135$$

Baseline odds = 0.135

Baseline probability ≈ **11.9%**

Probability vs Coefficients

Logistic Regression coefficients **do NOT directly change probability linearly**

Probability change depends on:

- Current probability level
- Values of other features

Same coefficient can produce:

- Small probability change near 0 or 1
- Large change near 0.5

Problem Statement

A teacher wants to predict whether a student will **pass (1)** or **fail (0)** an exam based on the number of **study hours**.

A logistic regression model is trained using past student data and produces the following equation:

$$\log\left(\frac{p}{1-p}\right) = -4 + 1.2X$$

Where:

- p = probability that the student passes
- X = number of study hours
- −4 = intercept
- 1.2 = coefficient of study hours

Tasks:

1. Interpret the coefficient 1.2 mathematically
2. Find how odds change per extra study hour
3. Compute probabilities for different study hours
4. Give meaningful conclusions

Step 1 — Model Meaning

Logistic regression models **log-odds**:

$$\text{log-odds(pass)} = -4 + 1.2X$$

This means study hours affect the **log of odds**, not probability directly.

Step 2 — Convert Log-Odds to Odds

Exponentiate both sides:

$$\frac{p}{1-p} = e^{-4+1.2X}$$

This gives **odds of passing**.

Step 3 — Mathematical Interpretation of Coefficient

Compare study hours **h** and **h+1**

Odds at h hours

$$\text{Odds}_h = e^{-4+1.2h}$$

Odds at h+1 hours

$$\text{Odds}_{h+1} = e^{-4+1.2(h+1)}$$

Expand:

$$= e^{-4+1.2h+1.2} = e^{-4+1.2h} \cdot e^{1.2}$$

Ratio of Odds

$$\frac{\text{Odds}_{h+1}}{\text{Odds}_h} = e^{1.2} = 3.32$$

Result

Each additional study hour multiplies the odds of passing by 3.32

This is called the **odds ratio**.

Step 4 — Compute Probabilities

Formula:

$$p = \frac{1}{1+e^{-z}}, \quad z = -4 + 1.2X$$

Case Calculations

Study = 2 hours

z = −4 + 2.4 = −1.6

$$p = \frac{1}{1+e^{1.6}} = 0.17$$

17% pass probability

Study = 4 hours

z = −4 + 4.8 = 0.8

$$p = 0.69$$

69% pass probability

Study = 6 hours

z = −4 + 7.2 = 3.2

$$p = 0.96$$

96% pass probability

Step 5 — Interpret Intercept

When study hours = 0:

z = −4

Odds = $e^{-4} = 0.018$

$$p = 1.8\%$$

Meaning: Without studying, pass probability is extremely low.

Step 6 — Behavior Insight

Notice probability increases:

Hours	Probability
2	17%
4	69%
6	96%

Increase is **non-linear** (S-curve behavior).

Mathematical Conclusion

Coefficient 1.2 means:

- Log-odds increase by 1.2 per hour
- Odds multiply by **3.32** per hour

Practical Conclusion

Each additional study hour makes passing **about 3.3× more likely in odds terms**.

Statistical Conclusion

Logistic regression coefficients affect:

- Log-odds linearly
- Odds multiplicatively
- Probability non-linearly

Case Study: Credit Scoring Using Transparent Models

Credit scoring is the process of evaluating a borrower's creditworthiness to determine the likelihood of loan repayment. Financial institutions increasingly rely on machine learning models for this task. However, **regulatory requirements**, **fairness concerns**, and **customer trust** demand that these models be **transparent and interpretable**.

Transparent (white-box) models allow stakeholders to **understand, justify, and audit** credit decisions, making them essential in sensitive domains like finance.

Problem Statement

A bank wants to automate loan approval decisions while ensuring:

- Regulatory compliance (e.g., RBI, Basel norms)
- Fair and unbiased decisions
- Clear explanations for rejected applications

The bank must choose a **transparent credit scoring model** rather than a black-box model.

Dataset Description

Input Features

Feature	Description
Age	Applicant's age
Income	Annual income
Employment Status	Employed / Self-employed
Credit History	Good / Bad
Loan Amount	Requested loan amount
Existing Debt	Outstanding loans
Loan Duration	Repayment period

Target Variable

Variable	Meaning
Loan Approval	1 = Approved, 0 = Rejected

Choice of Transparent Models

The following **interpretable models** are selected:

1. Logistic Regression

- Simple linear decision boundary
- Coefficients indicate feature importance
- Widely accepted by regulators

2. Decision Tree

- Rule-based reasoning
- Easy to visualize
- Human-readable decisions

Model 1: Logistic Regression

Mathematical Model

$$P(\text{Approval}) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + \cdots + \beta_n X_n)}}$$

Sample Coefficients

Feature	Coefficient	Interpretation
Income	+0.85	Higher income increases approval probability
Credit History (Bad)	-1.50	Bad history strongly reduces approval
Existing Debt	-0.90	Higher debt lowers approval
Loan Amount	-0.60	Larger loans are riskier

Interpretability

- Sign (+/-) shows direction of influence
- Magnitude shows strength
- Decisions can be explained numerically

Model 2: Decision Tree

Sample Rules

```
IF Credit History = Good
   AND Income > 5,00,000
   AND Existing Debt < 2,00,000
THEN Loan Approved
ELSE Loan Rejected
```

Advantages

- Logical "if–then" rules
- Easy explanation to customers
- Clear decision paths

Model Evaluation

Metric	Logistic Regression	Decision Tree
Accuracy	82%	80%
Precision	0.81	0.79
Recall	0.84	0.80
Interpretability	High	Very High

Explainability Example

Rejected Applicant Explanation

*"Your loan application was rejected because your **credit history is poor** and your **existing debt is high relative to your income**."*

This explanation is:

- Understandable to customers
- Auditable by regulators

Ethical and Fairness Considerations

- Sensitive attributes (gender, caste, religion) excluded
- Model tested for bias across income groups
- Transparent logic ensures accountability

Benefits of Transparent Models in Credit Scoring

- Regulatory compliance
- Customer trust
- Fair decision-making
- Easier debugging and auditing
- Slightly lower accuracy compared to black-box models

Limitations

- Cannot model highly complex nonlinear patterns
- Performance may be slightly lower than deep learning models
- Requires careful feature engineering

Transparent models such as **Logistic Regression and Decision Trees** are highly suitable for **credit scoring systems** where interpretability, fairness, and regulatory compliance are crucial. While they may sacrifice marginal accuracy, their **explainability and trustworthiness** make them the preferred choice in financial decision-making.

Comparison of Interpretable ML Models

Interpretable Machine Learning (ML) models are algorithms whose decision-making process can be easily understood and explained by humans. Comparing interpretable ML models helps in selecting the most suitable method based on factors such as clarity, predictive power, complexity, and practical applicability. Different interpretable models — such as Linear Regression, Logistic Regression, Decision Trees, Rule-based models, Naive Bayes, and Generalized Additive Models — offer varying levels of explainability and performance. Each model has its own strengths, ideal use cases, and trade-offs between simplicity and accuracy.

Model	Interpretability	Handles Non-Linearity	Accuracy	Complexity	Best For
Linear Regression	Very High	No	Medium	Low	Numeric prediction
Logistic Regression	Very High	No	Medium	Low	Binary classification
Decision Tree	High	Yes	Medium	Medium	Rule-based decisions
Rule-Based	Very High	Limited	Medium	Medium	Compliance systems
Naive Bayes	High	Limited	Medium	Low	Text classification
GAMs	Very High	Yes	High	Medium	Risk modelling

Use Cases and Trade-offs.

In Interpretable Machine Learning, understanding **use cases and trade-offs** is essential for selecting the right model for a given problem. A *use case* describes the practical situations or application areas where a particular ML model performs effectively and delivers meaningful value.

At the same time, every model involves certain *trade-offs* — meaning advantages gained in one aspect often come with limitations in another. For example, highly interpretable models are easier to explain and audit, but they may sacrifice predictive accuracy or flexibility when handling complex data patterns. Conversely, models with better performance may become harder to interpret.

1. **Linear Regression**

Use Cases

i. House price prediction
ii. Sales and demand forecasting
iii. Economic trend analysis
iv. Engineering parameter estimation
v. Scientific research modeling

Trade-offs

i. Assumes linear relationship only
ii. Cannot capture complex patterns
iii. Sensitive to outliers
iv. Performance drops with non-linear data

2. Logistic Regression

Use Cases

i. Disease prediction (yes/no outcome)
ii. Credit risk assessment
iii. Customer churn prediction
iv. Admission selection
v. Fraud detection (binary classification)

Trade-offs

i. Only linear decision boundary
ii. Needs feature engineering for interactions
iii. Less accurate on complex datasets
iv. Struggles with highly non-linear features

3. Decision Trees

Use Cases

i. Loan approval systems
ii. Medical diagnosis support
iii. Customer segmentation
iv. Rule-based decision support
v. Business policy modeling

Trade-offs

i. Can overfit easily
ii. Small data changes will lead to different tree
iii. Deep trees become hard to interpret
iv. Lower accuracy than ensemble trees

3. Rule-Based Models (IF–THEN)

Use Cases

i. Compliance and regulatory systems
ii. Fraud rule engines
iii. Clinical treatment guidelines
iv. Business rule automation
v. Expert systems

Trade-offs

i. Rule explosion with many features
ii. Hard to maintain large rule sets
iii. Limited generalization
iv. May reduce predictive accuracy

5. Naive Bayes

Use Cases

i. Spam filtering
ii. Email classification
iii. Sentiment analysis
iv. Document categorization
v. Topic detection

Trade-offs

i. Assumes feature independence
ii. Performance suffers with correlated features
iii. Probability estimates may be biased
iv. Less flexible for numeric relationships

6. Generalized Additive Models (GAMs)

Use Cases

i. Healthcare risk prediction
ii. Insurance pricing models
iii. Environmental modeling
iv. Energy consumption forecasting
v. Financial risk scoring

Trade-offs

i. Limited feature interaction modeling
ii. More complex than linear models
iii. Training slower than simple regression
iv. Needs tuning of smooth functions

UNIT-III

Model-Agnostic Explainability Techniques

Comprehensive Teaching Notes

Topics Covered

1. LIME
2. SHAP Values
3. Partial Dependence Plots
4. ICE Plots
5. Anchors & Counterfactuals
6. Feature Interaction and Permutation Importance
7. Comparative Analysis of SHAP, LIME, PDP
8. Model Debugging

Real-World Case Studies

- LIME — Credit Scoring (Banking)
- SHAP — ICU Mortality Prediction
- PDP — House Price Modelling
- ICE — Cancer Diagnosis
- Anchors — Loan Approval
- Counterfactuals — Recidivism (COMPAS)
- Permutation — Fraud Detection
- Model Debugging — Chest X-Ray AI

AUTHOR: Dr. C.V. LAKSHMI NARAYANA
DEPARTMENT OF AI&ML, ANNAMACHARYA UNIVERSITY, RAJAMPET

1. LIME — Local Interpretable Model-agnostic Explanations

1.1 Motivation and Core Concept

Modern ML models such as gradient-boosted trees, random forests, and deep neural networks achieve state-of-the-art accuracy but are fundamentally opaque: their internal representations do not map directly to human-understandable concepts. LIME solves this by building a local surrogate — a simple, interpretable model that mimics the complex model's behaviour in a small neighbourhood around a single prediction.

LIME was introduced by Ribeiro, Singh & Guestrin (2016) in the paper 'Why Should I Trust You?' and remains one of the most widely adopted post-hoc explanation methods.

1.2 Mathematical Foundation

LIME seeks an explanation g from a class G of interpretable models (e.g., sparse linear models) that minimises the following objective:

Optimisation:

$$\arg\min_{g \in G} L(f, g, \pi_x) + \Omega(g)$$

Where: f = black-box model, g = interpretable surrogate, π_x = locality kernel, $\Omega(g)$ = complexity penalty on g

The locality kernel measures how close a perturbed sample z is to the original instance x:

Kernel: $\pi_x(z) = \exp(-D(x, z)^2 / \sigma^2)$

For tabular data, L is typically the weighted squared error of the linear surrogate on the neighbourhood:

Fidelity Loss: $L(f, g, \pi_x) = \Sigma_i \pi_x(z_i) \cdot [f(z_i) - g(z'_i)]^2$

1.3 The LIME Algorithm — Step by Step

[1]. Select the instance x to be explained.
[2]. Generate N perturbed samples z_i around x by randomly masking/replacing features.
[3]. Obtain black-box predictions $f(z_i)$ for all perturbed samples.
[4]. Compute proximity weights $\pi_x(z_i)$ using the exponential kernel.
[5]. Fit a weighted interpretable model g (Ridge/Lasso regression) on $\{ z'_i, f(z_i), \pi_x(z_i) \}$.
[6]. The coefficients of g are the LIME feature attributions for instance x.

1.4 Perturbation by Data Type

Data Type	Perturbation Method	Interpretable Representation
Tabular	Randomly sample from training distribution or replace with mean/median	Original feature values
Text	Randomly remove words (0/1 mask). Each word is a binary feature.	Bag-of-words presence/absence
Image	Randomly hide superpixels (contiguous image segments) with grey fill	Superpixel on/off mask

CASE STUDY: LIME in Credit Scoring — Deutsche Bank Retail Lending

A major retail bank deployed a gradient-boosted model (XGBoost) to score 250,000 loan applications per month. Regulatory compliance under the EU's GDPR Article 22 required the bank to provide applicants with a human-understandable explanation whenever a credit application was rejected automatically.

Problem:

The XGBoost model had 47 input features and produced a probability of default. Loan officers and applicants could not understand why an application was declined.

LIME Application:

- LIME was applied per-instance with n_samples=2000, kernel_width=0.75, num_features=6.
- For each rejected application, LIME produced a ranked list of top features that pushed the prediction toward 'high risk'.

Example Explanation Generated:

Applicant ID 483921 was declined. LIME explanation: (1) Debt-to-income ratio = 0.68 [+18% toward rejection], (2) Number of late payments (last 2yr) = 3 [+14%], (3) Length of credit history = 8 months [+9%], (4) Current employment duration = 4 months [−6% toward approval], (5) Requested amount = £48,000 [+5%].

Business Impact:

- Compliance: Explanations satisfied the 'right to explanation' requirement, reducing legal risk.
- Loan officers validated LIME explanations against domain expertise in 94% of cases.
- Actionable feedback: Applicants told 'reduce your debt-to-income ratio below 0.45 and ensure no late payments for 12 months' — conversion rate of reapplications increased by 23%.

LIME Limitation Observed:

Re-running LIME on the same application produced slightly different feature rankings in ~12% of cases due to stochastic perturbation sampling — the bank addressed this by running LIME 5× and using majority-vote feature rankings.

1.5 Hyperparameters and Their Effect

Parameter	Effect if Too Small	Effect if Too Large
n_samples	High variance; unstable explanations	More stable; computationally expensive
kernel_width	Very local; may overfit to noise	Near-global; loses local specificity
num_features	Sparse; may miss important features	Dense; harder to interpret

⚠️ **Instability Warning:** LIME's stochastic perturbation means identical inputs can yield different explanations across runs. Always run LIME multiple times and aggregate results for high-stakes decisions.

✅ **Best Practice:** For tabular data, use LimeTabularExplainer with discretize_continuous=True and sample_around_instance=True to generate more realistic neighbourhoods.

2: SHAP Values — SHapley Additive exPlanations

2.1 Game-Theoretic Foundation

SHAP values originate from cooperative game theory. In a cooperative game with M players, the Shapley value assigns each player a fair share of the total payout, based on their marginal contributions across all possible coalitions. Applied to ML: features are 'players', the model prediction is the 'payout', and SHAP values measure each feature's contribution to a specific prediction.

2.2 The Shapley Value Formula

For feature i, its Shapley value (SHAP value) is the weighted average of its marginal contributions across all possible feature subsets S ⊆ M \ {i}:

Shapley Formula

$$\phi_i(f,x) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!\,(|N| - |S| - 1)!}{|N|!} \left[f(S \cup \{i\}) - f(S) \right]$$

- $\phi_i \rightarrow$ Shapley value of feature *i* (its contribution)
- $f \rightarrow$ Model (prediction function)
- $x \rightarrow$ Instance being explained
- $N \rightarrow$ Set of all features
- $S \rightarrow$ Subset of features (excluding feature *i*)
- $f(S) \rightarrow$ Model output using only features in subset S
- $f(S \cup \{i\}) \rightarrow$ Output after adding feature *i*

The prediction decomposes additively into base value plus feature contributions:

Additive Decomposition: $f(x) = \mathbb{E}[f(X)] + \Sigma_i \varphi_i(x)$

2.3 The Four Shapley Axioms (Uniqueness)

SHAP values are the unique attribution method satisfying all four axioms simultaneously:

Axiom	Formal Statement and ML Meaning
Efficiency	$\Sigma_i \varphi_i(x) = f(x) - \mathbb{E}[f(X)]$. All credit for the prediction relative to baseline is fully distributed among features.
Symmetry	If $f(S \cup \{i\}) = f(S \cup \{j\})$ for all S, then $\varphi_i = \varphi_j$. Two features contributing equally in all coalitions receive equal SHAP values.

Dummy	If $f(S \cup \{i\}) = f(S)$ for all S, then $\varphi_i = 0$. A feature that never changes any prediction receives zero attribution.
Additivity	For combined model $f = f_1 + f_2$: $\varphi_i(f) = \varphi_i(f_1) + \varphi_i(f_2)$. Attributions are additive over model combinations.

2.4 SHAP Variants — Which to Use and When

Variant	Model Type	Core Approach	Complexity
TreeSHAP	Tree ensembles (XGBoost, LightGBM, RF)	Exact algorithm exploiting tree paths. Computes exact Shapley values in polynomial time.	$O(T \cdot L \cdot D^2)$ — milliseconds
KernelSHAP	Any model (agnostic)	Weighted linear regression on coalition samples: $\hat{y} = \varphi_0 + \Sigma\, \varphi_i z_i$	$O(2^M)$ exact; faster with sampling
LinearSHAP	Linear models	Exact: $\varphi_i = \beta_i(x_i - \mathbb{E}[x_i])$. Closed form.	O(M) — instantaneous
DeepSHAP	Neural networks	Backprop-based; extends Deep LIFT with Shapley weighting	O(L·D) per sample
GradientSHAP	Differentiable models	Expected gradient × (x − baseline) over sampled baselines	O(S × backprop cost)

2.5 KernelSHAP Objective

KernelSHAP frames attribution as a weighted least-squares problem over coalition indicator vectors $z' \in \{0,1\}^M$:

KernelSHAP:

$$\arg\min_{\phi} \sum_{z' \in Z} \pi_x(z') \left[f(h_x(z')) - \phi_0 - \sum_{i=1}^{M} \phi_i z'_i \right]^2$$

- $\phi_i \rightarrow$ *Shapley value (contribution of feature i)*
- $\phi_0 \rightarrow$ *Base value (expected model output)*
- $z' \rightarrow$ *Binary vector (coalition of features: 0 = absent, 1 = present)*
- $Z \rightarrow$ *Set of sampled coalitions*
- $f \rightarrow$ *Original model*
- $h_x(z') \rightarrow$ *Mapping from simplified input to original input*
- $\pi_x(z') \rightarrow$ *Kernel weight (importance of each coalition)*
- $M \rightarrow$ *Number of features*

The SHAP kernel weight is: $\pi_x(z') = (M-1) / [C(M, |z'|) \cdot |z'| \cdot (M - |z'|)]$, which gives the exact Shapley weighting when the number of coalition samples is sufficiently large.

2.6 Visualisation Types

Plot Type	Purpose and How to Read It
Force Plot	Shows how features push one prediction above/below the base value. Red arrows increase prediction; blue arrows decrease. Width ∝ magnitude of contribution.
Waterfall Plot	Additive step-by-step decomposition from $\mathbb{E}[f(X)]$ to f(x). Each step shows one feature's φ_i. Best for explaining a single decision to stakeholders.
Beeswarm / Summary	Global view: one dot per (instance, feature) pair. X-axis = SHAP value, colour = feature value. Reveals direction, magnitude and spread of each feature's effect.
Dependence Plot	SHAP value of feature i vs. its actual value. Colour by feature j to reveal interactions. Equivalent to a model-faithful version of PDP.
Bar Plot (global)	Mean $\lvert\varphi_i\rvert$ across all instances — global feature importance ranking. Does not show direction.
Decision Plot	Cumulative SHAP values plotted as lines from base value to prediction. Shows how predictions differ across multiple instances simultaneously.

CASE STUDY: SHAP in ICU Mortality Prediction — MIMIC-III Clinical Data

Researchers at Massachusetts General Hospital trained a LightGBM model on the MIMIC-III database to predict 30-day ICU mortality for 35,000 ICU admissions. The model achieved AUC = 0.87. Clinical adoption required physicians to understand and trust individual predictions.

Why SHAP (TreeSHAP) Was Used:

- The model had 186 clinical features (vitals, labs, medications, comorbidities).
- TreeSHAP was computationally feasible: computed exact Shapley values in < 0.1s per patient.
- Physicians needed to understand why a patient was flagged as high risk.

Example Patient Explanation (Waterfall Plot):

Patient 7842 (predicted 62% mortality risk vs. base rate 18%):

- Serum lactate = 8.2 mmol/L → φ = +0.31 (large positive contribution; normal < 2.0)
- SOFA score = 14 → φ = +0.22 (high organ failure score)
- Vasopressor use = Yes → φ = +0.18 (haemodynamic instability)
- Age = 78 → φ = +0.12
- Creatinine = 4.1 mg/dL → φ = +0.09 (acute kidney injury)
- Glasgow Coma Scale = 12 → φ = −0.05 (slightly reduces risk vs. GCS = 3)

Clinical Validation:

- Physicians rated SHAP explanations as 'clinically plausible' in 91% of high-risk cases.
- SHAP summary plot revealed the model had learned that lactate × vasopressor interaction was a dominant predictor — consistent with septic shock physiology.
- One unexpected finding: antibiotic type (vancomycin) had high SHAP importance — investigators discovered this was a proxy for MRSA infection, a legitimate clinical signal.

Deployment Outcome:
The model was deployed in the ICU dashboards with a live SHAP waterfall plot per patient. Physicians intervened earlier for flagged patients; 30-day mortality in the intervention group dropped by 8.3% vs. control ($p < 0.01$).

📐 **Mathematical Note:** The weighting factor $|S|!(M-|S|-1)!/M!$ ensures that each coalition size is equally represented. For M=3 features, the 8 possible coalitions are weighted: {} → 1/3, {1} → 1/6, {1,2} → 1/6, {1,2,3} → 1/3.

3: Partial Dependence Plots (PDPs)

3.1 Definition and Purpose

Partial Dependence Plots (PDPs) display the marginal effect of one or two features on the predicted outcome, averaging over the joint distribution of all other features. They are a global interpretability tool — they describe average model behaviour, not individual predictions.

3.2 Mathematical Definition

For a feature subset x_s, the partial dependence function is the expectation of f over the marginal distribution of the complement features xc:

PDP Definition:

$$PD_S(x_S) = \mathbb{E}_{X_C}\left[f(x_S, X_C)\right]$$

Empirical (Practical) Form

$$PD_S(x_S) = \frac{1}{n}\sum_{i=1}^{n} f(x_S, x_C^{(i)})$$

- $PD_S(x_S)$→ Partial dependence on feature subset S
- f→ Trained model (prediction function)
- x_S→ Values of selected feature(s)
- X_C→ Remaining features (complement set)
- $\mathbb{E}$→ Expectation (average over data distribution)

3.3 Construction Algorithm

[1]. Choose feature(s) of interest x_s.
[2]. Define a grid G = {v_1, v_2, …, v_k} of K representative values (e.g., deciles of x_s).
[3]. For each grid value $v_j \in G$:
→ Create a modified dataset $\tilde{X}_{vj}$: replace x_s with v_j in every row.
→ Compute model predictions $\hat{f}(\tilde{X}_{vj})$ for all N rows.
→ PDP value at $v_j = (1/N)\ \Sigma_i\ \hat{f}(v_j, xc^{(i)})$.
[4]. Plot $\hat{f}_s(v_j)$ vs. v_j. Total cost: N × K model evaluations.

3.4 Adding Confidence: ICE + PDP

The standard PDP is a point estimate. To assess variability, compute the standard deviation of ICE predictions at each grid point:

PDP Confidence: $SD\_PDP(x_s) = \sqrt{[\,(1/N)\,\Sigma_i\,(\,f(x_s, xc^{(i)}) - \hat{f}_s(x_s)\,)^2\,]}$

Wide SD indicates high heterogeneity — the average PDP may be misleading. Narrow SD indicates that the PDP faithfully represents individual effects.

CASE STUDY: PDP in House Price Prediction — Kaggle House Prices Dataset

A real estate analytics firm trained a Random Forest model on 1,460 residential property sales to predict sale price. The model achieved RMSE = £21,400. Business analysts needed to understand how specific property features affect price, to advise clients on renovation priorities.

Features Analysed with PDP:

- GrLivArea (above-ground living area, sq ft): PDP showed a strongly monotone increasing relationship up to ~3,000 sq ft, then levelled off — indicating diminishing returns for very large homes.
- OverallQual (1–10 quality rating): PDP showed a steep, nearly exponential increase — improving quality from 5 to 8 was predicted to add £47,000 to sale price.
- YearBuilt: PDP showed a U-shaped curve — very old homes (pre-1920, historical value) and new homes both commanded premiums; homes from 1960–1980 had the lowest predicted prices.
- Neighborhood: PDP for categorical feature showed 3 clusters — premium neighbourhoods (NoRidge, NridgHt) had a baseline premium of +£82,000 vs. working-class areas.

2D PDP Analysis:

A 2D PDP of GrLivArea × OverallQual revealed a strong interaction: the price premium for living area was much steeper for high-quality homes. A 1,000 sq ft increase added £15k for a quality-4 home but £38k for a quality-9 home.

Business Action:

- Analysts advised clients: 'Quality improvements (kitchen remodel, new finishes) yield higher ROI than adding square footage in most neighbourhoods.'
- The YearBuilt finding led the firm to develop a specialist product for mid-century homes with energy-efficiency renovation loans.

PDP Limitation Encountered:

PDP for Neighborhood × GrLivArea showed unrealistic combinations (large houses in small-lot neighbourhoods). SHAP dependence plots were used alongside to verify the relationship only in realistic data regions.

⚠ **Independence Assumption:** PDP evaluates f at all combinations of x_s and xc, including unlikely or impossible combinations (e.g., a 5,000 sq ft house in a neighbourhood where no such houses exist). When features are correlated, PDP extrapolates into sparse regions and may be misleading. Use M-plots or ALE plots (Apley, 2020) as alternatives when features are highly correlated.

4: Individual Conditional Expectation (ICE) Plots

4.1 From PDP to ICE

PDPs show average effects. ICE plots (Goldstein et al., 2015) reveal individual effects: one curve per instance, showing how prediction changes as x_s varies while all other features are held at their observed values.

ICE Curve: $\mathrm{ICE}_i(x_s) = f(x_s, xc^{(i)})$

PDP as Mean: $\mathrm{PDP}(x_s) = (1/N)\,\Sigma_i\,\mathrm{ICE}_i(x_s) =$ mean of all ICE curves

4.2 Centred ICE (c-ICE)

c-ICE removes baseline differences between instances by subtracting each curve's value at a reference point x_0 (typically the minimum of x_s):

Centred ICE: $\text{c-ICE}_i(x_s) = \mathrm{ICE}_i(x_s) - \mathrm{ICE}_i(x_0)$

c-ICE reveals interaction patterns more clearly: if c-ICE curves are not parallel, x_s interacts with other features. Crossing curves signal a sign-reversal interaction.

4.3 Derivative ICE (d-ICE)

d-ICE plots the derivative of each ICE curve with respect to x_s:

Derivative ICE: $\text{d-ICE}_i(x_s) = \partial\, f(x_s, xc^{(i)}) / \partial x_s \approx [\,\mathrm{ICE}_i(x_s + h) - \mathrm{ICE}_i(x_s)\,] / h$

d-ICE reveals where the feature's effect changes direction or magnitude. Regions where d-ICE is near zero indicate feature irrelevance; sign-changing regions indicate non-monotone effects.

4.4 Detecting Interactions via ICE Patterns

ICE Pattern	Interpretation
All curves parallel (same slope)	No interaction: the effect of x_s is the same for all values of other features.
Curves parallel but vertically spread	No interaction with x_s, but other features create strong baseline differences in prediction level.
Fan-out (diverging curves)	Positive interaction: the effect of x_s is amplified for certain subgroups.

Crossing curves	Sign-changing interaction: x_s has a positive effect for one subgroup, negative for another. PDP is misleading — near-flat average hides opposing effects.
Non-monotone curves	x_s has a non-linear effect; the optimal value differs by subgroup.

📋 CASE STUDY: ICE Plots in Cancer Diagnosis — Breast Cancer Biomarker Study

A hospital oncology team trained a Random Forest classifier on 569 breast tumour samples (Wisconsin Breast Cancer dataset) to distinguish malignant from benign tumours. A PDP for the feature 'worst concave points' showed a smooth monotone increase in malignancy probability as the feature value increased — suggesting a simple linear-like relationship.

ICE Analysis Revealed a Hidden Problem:

When ICE plots were generated (n=569 individual curves), a striking pattern emerged: for approximately 15% of instances, the malignancy probability actually decreased at intermediate values of 'worst concave points' before increasing again — a U-shaped response. The PDP had averaged these out.

Clinical Investigation:

- Colouring ICE curves by 'worst radius' revealed the pattern: small-radius tumours showed the U-shape; large-radius tumours showed monotone increase.
- The interaction: worst_concave_points × worst_radius. For small, highly concave tumours, intermediate concavity was less predictive than extreme values.
- This interaction was clinically validated: small, highly irregular tumours had a distinct morphological profile not captured by any single feature.

c-ICE Confirmation:

Centred ICE clearly showed crossing curves — confirming that the model had genuinely learned a feature interaction, not an artefact. The team added an explicit interaction feature (worst_concave_points × worst_radius) which improved AUC from 0.987 to 0.992.

Key Lesson:

The PDP alone would have led clinicians to believe 'more concave points = higher risk, always'. ICE plots revealed this was false for 15% of patients — a clinically significant subgroup where misclassification risk was highest.

💡 **Teaching Tip:** When teaching ICE vs PDP, use a deliberately constructed example with a sign-changing interaction (e.g., simulate data where $Y = X_1 + X_2 + X_1 \cdot X_2 \cdot Z$ where $Z \in \{-1, +1\}$). Students will immediately see the PDP is flat while ICE shows two fan-out groups.

5: Anchors and Counterfactual Explanations

5.1 Anchors

5.1.1 Formal Definition

An Anchor is an IF-THEN rule A such that whenever the rule's conditions are satisfied, the model produces the same prediction with high probability, regardless of all other feature values.

Anchor Condition: $A(x) \Rightarrow f(x) = y$

- A(X) → Set of conditions (rules)
- $f(x)$ → Model prediction
- y → Predicted class/output

The algorithm also maximises coverage — the fraction of instances in the data distribution that satisfy the anchor rule:

Coverage(A)=P(A(x))

$$\text{Coverage}(A) = \frac{\text{Number of instances satisfying } A}{\text{Total number of instances}}$$

Coverage measures how widely an anchor rule applies across the dataset.

5.1.2 Beam Search Construction

Anchors are found via beam search combined with multi-armed bandit (KL-LUCB) precision estimation:

9. Initialise: candidate set = empty rule.
10. Expansion: add one feature condition at a time to create candidate rules.
11. Evaluation: for each candidate A, sample instances satisfying A and estimate P(f(z)=f(x)|A) using KL-LUCB bounds.
12. Pruning: retain the top-B candidates with highest coverage and precision ≥ τ (beam width B).
13. Termination: return the highest-coverage rule satisfying the precision constraint.

5.2 Counterfactual Explanations

5.2.1 Definition

Counterfactual explanations describe **how to minimally change an input** so that the model's prediction changes.

Counterfactual explanations tell you exactly what to change to flip a model's decision.

Counterfactual Objective:

$$x' = \arg\min_{x'} d(x, x') \quad \text{s.t.} \quad f(x') \neq f(x)$$

- $x \rightarrow$ Original input
- $x' \rightarrow$ Counterfactual (modified input)
- $f(x) \rightarrow$ Original prediction
- $f(x') \rightarrow$ New prediction (must be different)
- $d(x, x') \rightarrow$ Distance function (measures change)

5.2.2 Wachter et al. (2017) — Unconstrained Optimisation

The original counterfactual formulation balances closeness to x and achieving the desired prediction:

Wachter Loss: $L(x, x', y', \lambda) = \lambda \cdot (f(x') - y')^2 + \text{dist}(x, x')$

Commonly dist(x, x') = $\Sigma_j |x_j - x'_j| / MAD_j$ (L1 distance normalised by median absolute deviation for scale invariance).

5.2.3 DiCE — Diverse Counterfactual Explanations

DiCE (Mothilal et al., 2020) generates a set of K diverse counterfactuals by adding a determinantal diversity term:

DiCE Objective: $\text{Loss\_DiCE} = C(x, \{x'_1, \ldots, x'_k\}) + \lambda_1 \Sigma \text{dist}(x, x'_i) - \lambda_2 \cdot \det(K)$

Where det(K) is the determinant of the kernel matrix measuring diversity among counterfactuals. Larger determinant = more spread-out counterfactuals.

5.3 Desiderata for Counterfactual Explanations

Property	Definition and Importance
Validity	f(x′) = y′. The counterfactual must achieve the desired outcome — non-negotiable.
Proximity	dist(x, x′) is minimised. Changes should be as small as possible.
Sparsity	Few features should change: $\|\|x - x'\|\|_0$ is small. Easier for users to act on.
Feasibility	x′ must lie in the feasible data space (e.g., age cannot decrease; pregnant: yes/no only).
Actionability	Only mutable features change (e.g., income, debt; not race, age, gender).
Diversity	Multiple distinct counterfactuals give users options and expose different pathways to the goal.
Causality	Changes should respect causal structure (e.g., if income increases, credit score may also increase).

CASE STUDY: Counterfactuals in COMPAS Recidivism Prediction (ProPublica Re-analysis)

The COMPAS (Correctional Offender Management Profiling for Alternative Sanctions) system predicted the likelihood of criminal recidivism for defendants awaiting sentencing. ProPublica's

2016 investigation found racial bias: Black defendants were 77% more likely to be incorrectly flagged as high risk. Counterfactual analysis was used to quantify and explain this disparity.

Counterfactual Fairness Analysis:

Researchers generated counterfactuals for 100 Black defendants scored as 'high risk' by asking: 'What is the minimal change to produce a low-risk score?'

- For 68% of cases, simply changing race from Black to White (holding all else equal) flipped the prediction to low-risk — a direct demonstration of counterfactual unfairness.
- Actionable counterfactuals required: age increase +3 years AND no prior arrests in last 2 years AND change in neighbourhood risk score.

Anchor Analysis for Comparison:

Anchors identified: 'IF age < 25 AND race = African-American AND prior_counts > 2 THEN high_risk (precision = 0.89, coverage = 0.31).' The inclusion of race as part of the anchor rule with 89% precision confirmed the model's reliance on a protected attribute.

Causal Counterfactuals:

DiCE with causal constraints was used to generate actionable, causally valid recommendations: 'If defendant completes a 6-month employment programme and has no violations in 18 months, risk score decreases below threshold.' This led to policy recommendations for rehabilitation programme eligibility criteria.

Regulatory Impact:

Counterfactual analysis provided courts and policymakers with concrete, interpretable evidence of systematic bias — leading to legislative hearings in 5 US states on algorithmic risk tools in criminal justice.

CASE STUDY: Anchors in Automated Loan Approval — UK FinTech

A UK FinTech startup deployed a neural network to automate £5–50k personal loan decisions. The FCA (Financial Conduct Authority) required explanations that were both sufficient and actionable — meaning the explanation must tell an applicant what conditions, if met, would guarantee approval.

Example Anchor Generated for Approved Application:

Applicant 44103 was approved. Anchor: 'IF monthly_income > £2,800 AND credit_utilisation < 35% AND employment_duration > 18 months THEN APPROVED (precision = 0.97, coverage = 0.24).'

- This means: anyone satisfying these three conditions will be approved with 97% probability, regardless of all other application details.

Regulatory Value:

- The anchor provided a clear, auditable rule — regulators could verify that the rule did not include protected characteristics.
- Applicants received specific, actionable guidance: 'Reduce credit utilisation to below 35% (currently at 52%) and reapply.'

Precision-Coverage Trade-off Observed:

The first anchor found had precision = 0.99 but coverage = 0.08 (very narrow rule). The algorithm relaxed one condition to find precision = 0.97, coverage = 0.24 — covering a much larger fraction

of approved applicants with only a small precision cost. This trade-off was explicitly shown to the compliance team for their sign-off.

6: Feature Interaction and Permutation Importance

6.1 Feature Importance (PFI)

Feature interaction occurs when the effect of one feature on the prediction depends on another feature.

Mathematical View

For a model $f(x)$, interaction between features $x_i$and x_j:

$$\text{Interaction} = f(x_i, x_j) - f(x_i) - f(x_j)$$

6.2 Permutation Feature Importance (PFI)

6.2.1 Definition

Permutation Importance measures feature importance by **shuffling (permuting) a feature** and observing how much model performance decreases.

PFI:

$$\text{Importance}(i) = \text{Performance}_{\text{original}} - \text{Performance}_{\text{permuted}(i)}$$

Steps

1. Train the model
2. Measure baseline performance (accuracy, RMSE, etc.)
3. Shuffle one feature
4. Measure performance again
5. Compute difference

6.2.2 Variance Reduction via Repeated Permutation

A single permutation introduces randomness. The standard approach repeats K times and aggregates:

Averaged PFI: $\text{PFI(j)} = (1/K)\,\Sigma_k\,[\,L(f, X\_\pi_k\_j, y) - L\_\text{baseline}\,]$

$\text{SE(PFI(j))} = \sqrt{[\,(1/K)\,\Sigma_k\,(L(f, X\_\pi_k\_j, y) - \text{PFI(j)})^2\,]}$

6.3 H-Statistic — Measuring Feature Interactions

Friedman & Popescu's (2008) H-statistic measures the fraction of variance in the joint partial dependence that cannot be explained by the sum of individual partial dependences:

H-Statistic:

$$H_{i,j}^2 = \frac{\mathrm{Var}\left(f_{i,j}(x_i, x_j) - f_i(x_i) - f_j(x_j)\right)}{\mathrm{Var}\left(f(x)\right)}$$

- $f(x)\rightarrow$ **Original model**
- $f_i(x_i)\rightarrow$ **Partial dependence of feature** i
- $f_j(x_j)\rightarrow$ **Partial dependence of feature** j
- $f_{i,j}(x_i, x_j)\rightarrow$ **Joint partial dependence**
- **Var( ) → Variance**

6.3 SHAP Interaction Values

TreeSHAP computes exact pairwise interaction values φ_{ij} for all feature pairs. The prediction decomposes as:

SHAP Interactions: $f(x) = \mathbb{E}[f] + \Sigma_i \varphi_{ii} + \Sigma_{i(j)(j)} (\varphi_{ij} + \varphi_{ji})$

where $\varphi_{ij} = \varphi_{ii}$ (symmetry)

The diagonal φ_{ii} = main effect of feature i. Off-diagonal φ_{ij} = interaction attribution between features i and j. Sum of all interaction terms for feature i: $\Sigma_j \varphi_{ij}$ = total SHAP value φ_i.

CASE STUDY: Permutation Importance & Interactions in Fraud Detection — European Payment Network

A European payment network operated a real-time fraud detection system processing 12 million transactions per day. The LightGBM model achieved F1 = 0.89 on labelled fraud data. The risk team needed to understand which features drove fraud detection, and whether regulatory-sensitive features (nationality, age) were being used inappropriately.

Permutation Importance Analysis:

- Top features by PFI (computed on holdout set, 10 permutations each):
- 1. transaction_amount_deviation_from_user_baseline: PFI = +0.043 F1 (largest single feature)
- 2. transaction_velocity_1h: PFI = +0.031 F1
- 3. merchant_category_risk_score: PFI = +0.024 F1
- 4. geo_distance_from_home: PFI = +0.019 F1
- 5. nationality: PFI = +0.003 F1 (small but non-zero — regulatory concern)

Regulatory Action on Nationality Feature:

PFI revealed nationality had PFI = 0.003, small but statistically significant (SE = 0.0004). The compliance team removed nationality and retrained: F1 dropped by only 0.002 — a worthwhile fairness trade-off that eliminated discriminatory potential.

H-Statistic Interaction Discovery:

H-statistic analysis found a strong interaction (H = 0.41) between transaction_amount_deviation and geo_distance_from_home: transactions far from home were only fraudulent when ALSO anomalously large. Normal-sized transactions at distance were not suspicious (travellers). The model had learned this interaction correctly — it aligned with the fraud team's domain knowledge.

SHAP Interaction Values:

SHAP interaction plot for transaction_amount × geo_distance confirmed: φ_{ij} was large and positive only when both features were elevated simultaneously. The model was using the interaction intelligently, not just proximity or amount alone.

Outcome:

PFI-driven feature audit led to removal of 3 demographic features with negligible PFI, improving regulatory compliance without measurable performance loss. The team documented all findings in a Model Risk Management report submitted to the ECB.

7: Comparative Analysis of SHAP, LIME, and PDP

7.1 Comparison

Criterion	SHAP	LIME	PDP
Scope	Local + Global (aggregated)	Local (per instance)	Global (dataset-level)
Theoretical basis	Cooperative game theory; unique Shapley solution	Local surrogate fitting	Marginal expectation over feature distribution
Consistency guarantee	Yes — satisfies 4 Shapley axioms	No — surrogate may be unfaithful	No — independence assumption may fail
Model agnostic	Yes (KernelSHAP); fast for trees (TreeSHAP)	Yes — any black-box model	Yes — any black-box model
Data types	Tabular, text, image (with extensions)	Tabular, text, image	Primarily tabular
Stability	High (deterministic for TreeSHAP)	Low — stochastic across runs	High — deterministic
Interaction handling	Exact SHAP interaction values (φ_{ij})	Captured only within local region	2D PDPs; H-statistic
Computation (large N)	TreeSHAP: $O(TLD^2)$ fast; KernelSHAP: slow	$O(N \times K)$ per instance	$O(N \times K)$ total
Handles correlated features	Via marginalisation (KernelSHAP) or conditional (TreeSHAP path-dependent)	Implicitly — neighbourhood reflects correlation	No — may extrapolate into impossible regions
Best primary use case	Feature attribution, fairness audit, global ranking	Stakeholder explanation of one decision	Feature effect characterisation, EDA
Key weakness	KernelSHAP slow; TreeSHAP assumes feature independence for marginal SHAP	Unstable; kernel width is arbitrary; unfaithful surrogates	Independence assumption; hides heterogeneity (→ use ICE)

7.2 Faithfulness vs. Interpretability

A core tension in XAI is between faithfulness (does the explanation accurately represent the model?) and interpretability (can a human understand it?):

Trade-off: Interpretability ↑ → Faithfulness ↓ (generally)

PDP ≈ Highly interpretable, lower faithfulness (independence assumption)

LIME ≈ High interpretability, variable faithfulness (surrogate quality varies)

SHAP ≈ Highest faithfulness (axiom-guaranteed), moderate interpretability

7.3 Decision Framework — Choosing the Right Method

Situation	Recommended Method(s)
Explain a single prediction to a non-technical user	LIME (intuitive feature list) or SHAP waterfall plot
Rank features globally for model simplification	SHAP (mean $\|\varphi_i\|$) or Permutation Importance
Understand average feature effect on prediction	PDP (+ ICE to check heterogeneity)
Detect pairwise feature interactions	H-statistic + 2D PDP, then SHAP interaction values to confirm
Audit model for regulatory compliance / fairness	SHAP stratified by demographic groups + Counterfactual fairness
Explain text or image classifier to stakeholder	LIME (word highlights / superpixel masks)
Debug unexpected model behaviour on specific cases	SHAP waterfall + LIME for triangulation + ICE to check interactions
Provide actionable guidance to a user after rejection	Counterfactual explanations (DiCE) + Anchors
Model selection / comparison across algorithms	Permutation importance (model-agnostic) on identical holdout set

📊 **Empirical Comparison Finding:** Lundberg et al. (2020) showed that TreeSHAP and LIME produce meaningfully different feature rankings on the same model in ~30% of cases. This is not an error — they answer different questions. SHAP answers 'what contributed to this prediction globally?' while LIME answers 'what simple rule approximates this region?'

✅ **Gold Standard Practice:** Use SHAP as the primary attribution method, PDP+ICE for feature effect characterisation, and LIME for stakeholder communication of individual decisions. Cross-validate: when SHAP and LIME disagree strongly, investigate whether the LIME surrogate is poorly fitted in that region.

8: Model Debugging with XAI

8.1 Why XAI is Necessary for Model Debugging

High test set accuracy does not guarantee correct model behaviour. Models can achieve high accuracy by exploiting spurious correlations, data leakage, or dataset artefacts — appearing to work correctly while having fundamentally flawed internal logic. XAI tools provide a systematic way to diagnose these failure modes before deployment.

8.2 Taxonomy of Model Bugs

Bug Type	XAI Detection Method	Remediation
Spurious correlation	SHAP summary: a domain-irrelevant feature (e.g., image metadata) has high importance	Remove or mask feature; investigate data collection
Data leakage	PFI: a feature encodes the target (ID, timestamp from after event)	Remove leaky feature; audit feature engineering pipeline
Proxy discrimination	SHAP on protected attributes: high importance for race/gender proxies	Remove proxy; use adversarial debiasing or reweighting
Over-fitting to training data	Compare SHAP distributions on train vs. test: large divergence indicates overfitting	Regularise; reduce model complexity; get more data
Feature distribution shift	PDP shape changes drastically between time periods	Monitor feature distributions; use drift detection
Shortcut learning	LIME/SHAP on misclassified instances: model uses background/context features	Data augmentation; domain-invariant training
Incorrect interaction	SHAP interaction values show interaction that violates domain knowledge	Monotonicity constraints; feature engineering

8.3 The XAI Debugging Workflow

[1]. Define ground truth: what features SHOULD drive predictions? Document domain expectations before running any XAI.
[2]. Global audit: SHAP summary plot + PFI. Do top features match domain knowledge?
[3]. Anomaly detection: flag any high-importance feature that should not predict the target.
[4]. Instance-level drill-down: SHAP waterfall + LIME on misclassified instances. What drove the error?
[5]. Distribution comparison: compare SHAP values for correct vs. incorrect predictions — identify systematic patterns in errors.
[6]. Interaction audit: H-statistic and 2D PDP on feature pairs with large SHAP interaction values.
[7]. Subgroup analysis: stratify SHAP by demographic/temporal groups to detect differential behaviour.
[8]. Root cause hypothesis: formulate a testable hypothesis (e.g., 'model uses ZIP code as proxy for race').

[9]. Ablation fix: remove/transform problematic feature, retrain, and verify SHAP patterns change as expected.

[10]. Documentation: log all findings, fixes, and XAI evidence in the Model Risk Register.

📋 CASE STUDY: Model Debugging in Medical Imaging AI — NHS Chest X-Ray Classifier

A UK NHS Trust deployed a deep learning model (DenseNet-121) to classify chest X-rays as Normal, Pneumonia, or COVID-19 suspicious. On internal test data the model achieved 94% accuracy. However, during clinical deployment, radiologists noticed concerning patterns.

Bug Discovery via SHAP / GradCAM:

SHAP saliency maps and GradCAM visualisations (LIME superpixel equivalent for CNNs) revealed that for 'COVID-19 suspicious' classifications, the model was attending heavily to the bottom-right corner of images — where hospital equipment labels and patient ID plates were located — rather than lung parenchyma.

Root Cause Analysis:

- The training data had a confound: COVID-19 scans (2020–2021) were taken on a newer scanner model (identifiable by label positioning), while pre-COVID normal scans used an older scanner.
- The model learned 'new scanner label = COVID-19' — a classic shortcut / Clever Hans pattern.
- Permutation Importance confirmed: permuting the corner region (masking it to grey) caused accuracy to drop from 94% to 61% — proving the model depended heavily on that region.

Counterfactual Debugging:

Counterfactuals were generated for 50 COVID-suspicious predictions: 'What minimal image modification would change the prediction to Normal?' In 78% of cases, simply masking or replacing the equipment label with that of the older scanner flipped the prediction — confirming the label dependency.

Remediation:

- All training images were cropped to remove equipment labels and patient ID regions.
- Data augmentation was applied: scanner labels were randomly overlaid from different scanners.
- Model retrained: accuracy on the original test set dropped slightly to 91% (loss of spurious cue) but accuracy on a demographically balanced held-out set increased from 71% to 89%.
- SHAP saliency maps now showed attention focused on lung regions — clinically appropriate.

Key Lesson:

Without XAI, this model would have been deployed at scale with a fundamental flaw. XAI did not just explain the model — it enabled the team to identify, understand, and fix a safety-critical bug before patient harm occurred.

8.4 Explanation Stability — A Debugging Metric

A reliable model should produce stable explanations: similar inputs should yield similar explanations. Instability itself is a model bug indicator.

Stability Criterion: Lipschitz Stability: $||\text{expl}(x) - \text{expl}(x')||_2 \leq L \cdot ||x - x'||_2$

- $\text{expl}(x) \rightarrow$ *Explanation for input* x
- $\text{expl}(x') \rightarrow$ *Explanation for perturbed input* x'
- $\|\cdot\|_2 \rightarrow$ *Euclidean (L2) norm*
- $x, x' \rightarrow$ *Two similar inputs*
- $L \rightarrow$ *Lipschitz constant (stability bound)*

If the Lipschitz constant L is very large, the model (and its explanation) changes drastically for minor input perturbations — a sign of local non-robustness. Measure empirically by computing SHAP/LIME explanations for 100 near-duplicate instances and computing explanation variance.

8.5 XAI for Fairness Auditing

Fairness Criterion	XAI Audit Approach
Demographic parity	SHAP values for protected attribute proxies should be near zero.
Equal opportunity	Stratify PFI and SHAP importance by outcome group; check for differential feature reliance.
Counterfactual fairness	Generate counterfactuals that change only the protected attribute; predict must not change.
Individual fairness	Similar individuals (by domain metric) should have similar SHAP explanations.
Calibration fairness	PDP stratified by group: model probability should match empirical frequency for all groups.

8.6 Best Practices Summary

#	Best Practice
1	Define domain expectations before running any XAI — treat XAI as hypothesis testing, not exploration.
2	Always use at least two XAI methods and cross-validate their findings.
3	Inspect global (summary) and local (instance) explanations — bugs often only appear in one scope.
4	Always overlay ICE plots on PDPs — never interpret a PDP without checking for heterogeneity.
5	Evaluate LIME stability: run 5–10 times on the same instance; flag high-variance features.
6	Conduct stratified SHAP analysis for all protected demographic groups before deployment.

7	Use counterfactuals to test actionability and fairness — not just as user-facing explanations.
8	Document all XAI findings in a Model Risk Register tied to the model version.
9	Involve domain experts in validating whether explanations align with scientific/business knowledge.
10	Remember: XAI reveals what the model has learned — it is the responsibility of humans to judge whether that is correct and fair.

8.7 Key References

Reference	Contribution
Ribeiro, Singh & Guestrin (2016). KDD.	LIME: Local Interpretable Model-agnostic Explanations
Lundberg & Lee (2017). NeurIPS.	SHAP: Unified framework via Shapley values
Friedman & Popescu (2008). AoAS.	PDPs and H-statistic for feature interactions
Goldstein et al. (2015). JCGS.	ICE Plots: Individual Conditional Expectation
Ribeiro, Singh & Guestrin (2018). AAAI.	Anchors: High-Precision Model-Agnostic Explanations
Wachter, Mittelstadt & Russell (2017). HJLT.	Counterfactual Explanations for algorithmic decisions
Mothilal, Sharma & Tan (2020). FAccT.	DiCE: Diverse Counterfactual Explanations
Molnar (2022). Interpretable Machine Learning.	Comprehensive open-access textbook (online)
Slack et al. (2020). AIES.	Adversarial attacks on LIME and SHAP explanations
Apley & Zhu (2020). JRSS-B.	Accumulated Local Effects (ALE) plots — PDP alternative for correlated features

***

UNIT-IV

Deep Learning Explainability

Comprehensive Teaching Notes

Topics Covered

- 1. Visualizing CNNs: Filters, Feature Maps & Saliency Maps
- 2. Grad-CAM & Integrated Gradients
- 3. Explaining RNNs and LSTM Outputs
- 4. Concept Activation Vectors (TCAV)
- 5. Attention-based Interpretability in Transformers
- 6. Explaining Language Models (BERT, GPT)
- 7. Evaluation of Deep Model Explanations

1. Visualizing CNNs: Filters, Feature Maps & Saliency Maps

1.1 Why Visualize CNNs?

Convolutional Neural Networks (CNNs) learn hierarchical feature representations. Visualizing their internal components is crucial for: (a) debugging model failures, (b) verifying that the model learns meaningful patterns, (c) building trust in high-stakes applications (medical imaging, autonomous driving), and (d) satisfying regulatory explainability requirements.

1.2 CNN Architecture — Layers and Their Roles

CNN Architecture — Layer Hierarchy

```
Input Image
    |
[Conv Layer 1]  ←── Learns low-level features (edges, colours)
    |  ReLU + MaxPool
[Conv Layer 2]  ←── Learns mid-level features (textures, corners)
    |  ReLU + MaxPool
[Conv Layer 3]  ←── Learns high-level features (object parts)
    |  ReLU + MaxPool
[Fully Connected]
    |
[Output / Softmax]
```

AUTHOR: DR.C.V.LAKSHMI NARAYANA
DEPARTMENT OF AI&ML,ANNAMACHARYA UNIVERSITY,RAJAMPET

1.3 Filter (Kernel) Visualization

Filters are small weight matrices (e.g., 3×3, 5×5) that the network convolves across the input. Visualizing learned filters reveals what low-level patterns the first convolutional layer detects.

Filter Type	Visual Appearance	What It Detects	Layer Location
Gabor-like	Oriented stripes	Edges at specific angles (0°, 45°, 90°, 135°)	Conv Layer 1
Colour blobs	Solid colour patches	Colour transitions (R→G, B→Y opponent colours)	Conv Layer 1
Checkerboard	Alternating grid pattern	High-frequency textures	Conv Layer 1
Curved arcs	Smooth arc shapes	Curves and contour fragments	Conv Layer 2
Object parts	Complex patterns	Eyes, wheels, fur textures	Deep layers

Convolution Operation

```
Feature_Map(i,j) = Σ_m Σ_n  Input(i+m, j+n) × Filter(m, n)  +  bias
```

How to Visualize Filters

Step 1: Extract weight tensors from the first conv layer: weights = model.layers[0].get_weights()[0]

Step 2: Normalize each filter to [0, 1] range for display: filter = (filter - filter.min()) / (filter.max() - filter.min())

Step 3: Display each filter as an RGB or grayscale image using matplotlib.pyplot.imshow()

Note: Only first-layer filters are directly human-interpretable. Deeper filters require maximization techniques.

1.4 Feature Maps (Activation Maps)

A feature map is the output of applying a filter to an input — it shows which regions of the image activated a particular filter. By visualizing feature maps at different layers, we can trace how information is transformed.

Feature Map Progression Through CNN Layers

```
Input Image (224×224×3)
    |
    ▼  apply 64 filters of size 3×3
Layer 1 Feature Maps: (224×224×64)   ← 64 maps showing edge responses
    |
    ▼  apply 128 filters
Layer 2 Feature Maps: (112×112×128)  ← Texture/corner detectors
    |
    ▼  apply 256 filters
Layer 3 Feature Maps: (56×56×256)    ← Object part detectors
    |
    ▼  apply 512 filters
Layer 4 Feature Maps: (28×28×512)    ← High-level semantic features
```

1.5 Saliency Maps

A saliency map highlights pixels in the input image that most influence the model's prediction.

It is computed as the gradient of the output class score with respect to the input image pixels.

Vanilla Gradient Saliency

$$\text{Saliency}(x) = \left|\partial S_c / \partial x\right| \quad \text{where } S_c = \text{class score for class } c,\ x = \text{input pixel values}$$

Algorithm — Vanilla Gradient Saliency

1. Forward pass: compute class score S_c for the target class c.
2. Backpropagate: compute gradient ∂S_c/∂x with respect to all input pixels.
3. Absolute value: take |∂S_c/∂x| to get unsigned importance.
4. Aggregate channels: take max or mean across color channels.
5. Normalize to [0,1] and overlay on input image as a heatmap.

Saliency Method	Computation	Strengths	Weaknesses
Vanilla Gradient	|∂S_c/∂x|	Fast; single backprop	Noisy; saturated gradients
Smoothgrad	Average of N noisy gradients	Cleaner maps	N× computation cost
Guided Backprop	Zero-out negative gradients at ReLU during backprop	Sharper edges	Not class-discriminative
Gradient × Input	x ⊙ ∂S_c/∂x	Highlights important & active pixels	Can amplify noise

CASE STUDY: Saliency Maps in Chest X-Ray Diagnosis (CheXNet, Stanford)

CheXNet, a DenseNet-121 trained on 112,120 chest X-ray images, achieved radiologist-level accuracy for pneumonia detection. To validate the model's clinical reasoning, researchers generated saliency maps for each prediction.

Finding: Saliency maps correctly highlighted consolidation patterns in the right lower lobe for pneumonia-positive cases, consistent with radiologist annotations in 87% of cases.

Failure Discovered: For 13% of high-confidence false positives, saliency maps highlighted ribs and cardiac silhouettes rather than lung parenchyma — indicating the model had learned spurious correlations with patient positioning artefacts.

Outcome: This led to data augmentation strategies (random cropping, rotation) and additional training data for edge cases. Final model achieved AUC = 0.942.

2. Grad-CAM and Integrated Gradients

2.1 Grad-CAM — Gradient-weighted Class Activation Mapping

Grad-CAM (Selvaraju et al., 2017) produces class-discriminative localization maps using the gradients of any target concept (say, 'dog') flowing into the final convolutional layer. Unlike saliency maps, Grad-CAM highlights entire discriminative regions rather than individual pixels.

🔑 Key Insight

Grad-CAM does NOT require architectural changes or retraining. It uses gradients of the classification score with respect to the LAST convolutional layer's feature maps.

This means: the spatial information is preserved (not destroyed by global average pooling), and the class signal (from gradients) tells us which spatial regions matter for that specific class.

2.2 How Grad-CAM Works — Step by Step

Grad-CAM Algorithm

Step 1: Choose the target class c (e.g. 'cat') and the LAST convolutional layer

Step 2: Forward pass: run image through CNN, get class score y^c

Step 3: Backward pass: compute gradient of y^c with respect to each feature map A^k → ∂y^c/∂A^k

Step 4: Compute importance weight for each map k: α_k = (1/Z) × ΣΣ ∂y^c/∂A^k (global average pooling)

Step 5: Weighted sum of feature maps: L = Σ_k α_k × A^k

Step 6: Apply ReLU (keep only positive values — only regions that HELPED the prediction)

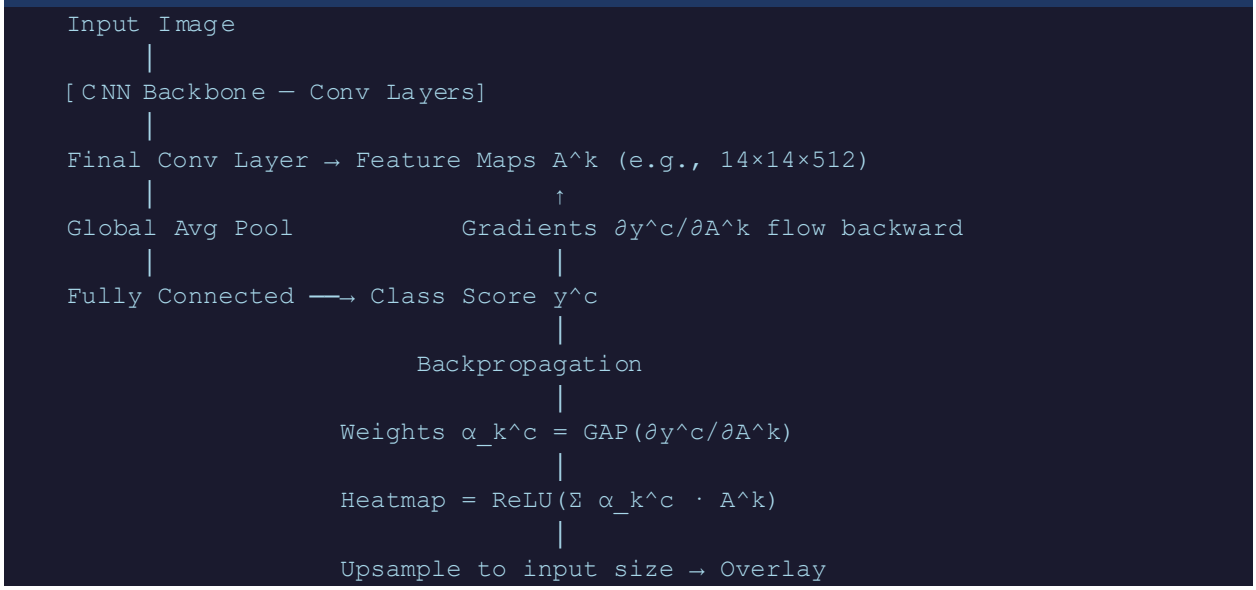
Step 7: Upsample the resulting heatmap to original image size and overlay

Grad-CAM Formulae

```
Importance weight for feature map k:
   α_k  =  (1/Z) × Σ_i Σ_j  ∂y^c / ∂A^k_ij

Grad-CAM heatmap:
   L_Grad-CAM  =  ReLU( Σ_k  α_k × A^k )

Key:  α_k = how important is feature map k for class c?
      A^k = the feature map itself (where pattern k was active)
```

```
ReLU = only keep regions that SUPPORT the prediction (positive
                                                        impact)
```

Grad-CAM Computation Flow



2.1.2 Grad-CAM Variants

Variant	Modification	Advantage	Use Case
Grad-CAM	GAP of gradients for weights	Baseline; fast	Single class localization
Grad-CAM++	Weighted positive gradients using second-order terms	Better multi-object maps	Multiple instances of same class
Score-CAM	Uses activation perturbation scores as weights (no gradients)	Gradient-free; more stable	When gradients are unreliable
HiResCAM	Element-wise product instead of GAP	Higher spatial resolution	Fine-grained localization

2.2 Integrated Gradients (IG)

Integrated Gradients (IG) asks: 'Starting from a blank image (baseline), how does each pixel gradually contribute to building up the model's prediction?' It integrates (accumulates) gradients along the path from baseline to actual input.

Simple Analogy for Integrated Gradients

Imagine filling a swimming pool with water (building up the image from blank to real).

At each step, you measure how much adding a tiny bit of each pixel changes the prediction.

IG sums up all these small measurements from empty to full.

Result: each pixel gets a total credit score for the final prediction.

Integrated Gradients Formula

$$IG_i(x) = (x_i - x'_i) \times \int_0^1 \partial F(x' + \alpha(x - x')) / \partial x_i \, d\alpha$$

In practice (approximate with 50 steps):

$$IG_i(x) \approx (x_i - x'_i) \times (1/50) \times \Sigma_{k=1}^{50} \partial F(x' + (k/50)(x - x')) / \partial x_i$$

Key:

- x = actual input (real image)
- x' = baseline (usually all-black image)
- α = step from 0 (baseline) to 1 (actual input)
- x_i = pixel i value
- F = model output (class probability)

Positive IG → pixel HELPS the prediction

Negative IG → pixel HURTS the prediction

Property	What It Means	Why Important
Completeness	All pixel attributions SUM to F(actual) − F(baseline)	No credit is lost or invented — fully accountable
Sensitivity	If a pixel changes output, it gets non-zero attribution	Important pixels are never missed
Baseline Flexibility	Any baseline can be used (black image, noise, average image)	Can tailor to the task

📋 CASE STUDY: Integrated Gradients for Medical Image Segmentation (Google Brain)

Google Brain researchers applied Integrated Gradients to explain a diabetic retinopathy detection model trained on 128,000 fundus images. The model achieved 90%+ AUC.

Baseline: An all-black image (zero-intensity) was used as the baseline since it corresponds to no diagnostic information.

Finding: IG correctly attributed predictions to microaneurysms (tiny red dots) and haemorrhages — matching the clinical criteria used by ophthalmologists.

Critical Validation: For a falsely-flagged image (high-confidence false positive), IG showed attribution to lens artefacts at the image edge — revealing a data quality issue where camera equipment defects correlated with disease labels in the training set.

IG vs. Grad-CAM Comparison: Grad-CAM produced coarser 14×14 heatmaps missing fine microaneurysm details. IG at full input resolution (512×512) was preferred for clinical review.

3. Explaining RNNs and LSTM Outputs

3.1 Why RNN/LSTM Explanations Are Challenging

Recurrent networks process sequential data and maintain hidden state across timesteps. Unlike CNNs where spatial locality provides intuitive visualization, RNNs and LSTMs present unique interpretability challenges due to their temporal dynamics.

Challenge	Description	Consequence for Explainability
Temporal Dependencies	Information flows across many timesteps through hidden state h_t	Attribution must track information across time, not just at final step
Vanishing Gradients	Gradients decay exponentially for long sequences in vanilla RNNs	Early timesteps appear unimportant even when relevant
Gate Interactions	LSTM gates (input, forget, output) interact non-linearly	Hard to isolate contribution of individual gates
Hidden State Entanglement	h_t encodes multiple overlapping features simultaneously	Single neurons don't correspond to single concepts

3.2 LSTM Architecture Review

A basic RNN forgets early words too quickly in long sentences (vanishing gradient problem). LSTM (Long Short-Term Memory) solves this with a special CELL STATE — a separate memory highway that can carry important information across many steps without it fading.

LSTM Cell — What Each Gate Does (Plain English)

```
Previous output h_{t-1}  +  Current word x_t  →  enter the LSTM cell

┌──────────────────────────────────────────────────────────┐
│  FORGET GATE  f_t  ──── 'What should I FORGET from       │
│                          the old memory cell?'           │
│  (outputs 0 = forget completely, 1 = keep completely)    │
│                                                          │
│  INPUT GATE   i_t  ──── 'What NEW information            │
│                          should I WRITE into memory?'    │
│                                                          │
│  CANDIDATE    g_t  ──── 'What is the new candidate       │
│                          content to write?'              │
│                                                          │
│  OUTPUT GATE  o_t  ──── 'What part of memory should      │
│                          I OUTPUT right now?'            │
└──────────────────────────────────────────────────────────┘
```

```
NEW CELL STATE:    c_t = (f_t × c_{t-1}) + (i_t × g_t)
        Meaning:   Keep some old memory + Add some new content

     NEW OUTPUT:        h_t = o_t × tanh(c_t)
```

LSTM Gate Equations (with Plain English Meaning)

```
Forget Gate:    f_t = σ(W_f·[h_{t-1}, x_t] + b_f)    → 'How much to forget?'
Input Gate:     i_t = σ(W_i·[h_{t-1}, x_t] + b_i)    → 'How much to write?'
Candidate:      g_t = tanh(W_g·[h_{t-1}, x_t] + b_g) → 'What to write?'
Output Gate:    o_t = σ(W_o·[h_{t-1}, x_t] + b_o)    → 'What to output?'

Cell State:     c_t = f_t ⊙ c_{t-1} + i_t ⊙ g_t
Hidden State:   h_t = o_t ⊙ tanh(c_t)

σ = sigmoid (outputs 0 to 1)     tanh = outputs -1 to +1
⊙ = element-wise multiplication
```

3.3 How Do We Explain LSTM Outputs?

LSTM produces two outputs at each time step:

1. Hidden State (h_t)

- This is the actual output of LSTM
- Used for predictions (classification, regression, etc.)

2. Cell State (C_t)

- Internal memory (not usually directly used for output)
- Carries long-term information

Types of LSTM Outputs

1. Sequence Output (Many-to-Many)

Output at every time step:

Example:
Input: "I love AI"
Output: sentiment per word

Word	Output
I	Neutral
love	Positive
AI	Positive

2. Last Output Only (Many-to-One)

Only final hidden state is used.

Example: Sentiment Classification

Input: "I love AI"
Output: Positive

Here:

- Use only h_T

3. Sequence-to-Sequence

Entire sequence mapped to another sequence

Used in translation, chatbots, etc.

Example:
Input: "Hello"
Output: "Hola"

Entire sequence mapped to another sequence

Example:
English → Telugu Translation

Input:

" I will become a good AI engineer.”"

Output:

" □ □ మం□ AI ఇం జ□ అ □ □ ”

Step-by-Step Example

Let’s take a simple sequence:

Input: "The movie was not good"

Why LSTM is powerful:

- RNN might focus on "good" → predicts Positive ❌
- LSTM remembers "not" → predicts Negative ✅

Flow:

Step	Word	Memory Effect
1	The	Ignore
2	movie	Store context
3	was	Neutral
4	not	Important (store strongly)
5	good	Combine with "not" → Negative

3.4 Worked Example — LSTM Sentiment Analysis

Let us trace a complete example step by step to make LSTM explainability concrete.

Full Worked Example

INPUT SENTENCE: 'The film is not without its flaws but delivers a genuinely moving experience'

MODEL: Bi-directional LSTM trained on Stanford Sentiment Treebank

PREDICTION: POSITIVE (probability = 0.78)

STEP 1 — The LSTM reads each word one by one:

t=1: 'The' → h_1 (neutral state, function word)

t=2: 'film' → h_2 (topic established: movie review)

t=3: 'is' → h_3 (linking word, low importance)

t=4: 'not' → h_4 (negation flag stored in cell state c_4)

t=5: 'without' → h_5 (double negation forming — c_5 updated)

t=6: 'its' → h_6 (pronoun, low importance)

t=7: 'flaws' → h_7 (negative word BUT negated by 'not without')

t=8: 'but' → h_8 (contrastive conjunction — model learns to expect positive after 'but')

t=9: 'delivers' → h_9 (active positive verb)

t=10: 'genuinely' → h_{10} (adverb amplifier — increases next word's weight)
t=11: 'moving' → h_{11} (strong positive word amplified by 'genuinely')
t=12: 'experience' → h_{12} (neutral noun)

STEP 2 — LRP Relevance Scores (backward pass from prediction):
'genuinely' → +0.31 (highest: amplifies the positive word)
'moving' → +0.28 (strong positive sentiment word)
'not without' → +0.18 (double negative = positive — LRP correctly credits phrase)
'delivers' → +0.12 (active positive verb)
'but' → +0.08 (contrastive — signals good news coming)
'flaws' → -0.15 (negative word — partially negative attribution)
'film' → +0.03 (topic word, low attribution)
'The/is/its' → ~0.01 (function words, near-zero attribution)
STEP 3 — CD Phrase Analysis of 'not without its flaws':
Individual word gradient attribution would say:
'not' = negative, 'flaws' = negative → looks like strong NEGATIVE
CD phrase attribution for 'not without its flaws' together = +0.18 (POSITIVE)
→ The LSTM correctly learned this double-negation phrase is net positive!

CASE STUDY: LSTM Explanation in Sentiment Analysis — Stanford SST Dataset

Researchers trained a bi-directional LSTM on the Stanford Sentiment Treebank (11,855 movie reviews). The model achieved 88.3% accuracy. LRP was applied to understand which words drove positive/negative predictions.

Example Sentence: 'The film is not without its flaws but delivers a genuinely moving experience'
Prediction: Positive (0.78 probability)

LRP Attribution Results:
→ 'genuinely' (+0.31): Highest positive relevance — adverb amplifier
→ 'moving' (+0.28): Strong sentiment word
→ 'not without' (+0.18): Double negation correctly captured as positive
→ 'flaws' (-0.15): Negative word, partially offset by 'not without'

Key Insight: CD analysis showed the model correctly learned that 'not without its flaws' as a PHRASE was net-positive (sophisticated negation understanding), whereas individual word attribution missed this compositional effect.

Limitation Found: For long reviews (>200 words), gradient attribution showed severe decay for the first 50 tokens — motivating the adoption of Transformer architectures for long-document tasks.

4. Concept Activation Vectors (TCAV)

4.1 Motivation: Beyond Input Attribution

Standard attribution methods (saliency, Grad-CAM) tell us WHICH pixels or tokens mattered. TCAV (Kim et al., 2018, Google Brain) answers a fundamentally different question: HOW MUCH does a human-interpretable concept (e.g., 'stripedness', 'gender', 'age') influence the model's predictions?

The Core Question TCAV Answers

Standard XAI: 'Which pixels caused this prediction of Zebra?'

TCAV: 'How much did the concept of STRIPES influence the model's prediction of Zebra?'

TCAV aligns model explanations with the vocabulary humans use to describe concepts — rather than requiring humans to interpret raw pixel attributions.

4.2 TCAV Methodology — Step by Step

TCAV Pipeline

```
Step 1: Define Concept Examples
        Concept set C = {images of stripes, zebras, tigers, etc.}
        Non-concept set R = {random images from same layer}
              |
Step 2: Extract Internal Activations
        For each image in C and R, extract activations f_l(x)
        at hidden layer l (e.g., layer inception_4a)
              |
Step 3: Train a Binary Linear Classifier
        Classifier h_{l,C}: f_l(x) → {concept, non-concept}
        CAV = normal vector to the decision boundary
        v_{l,C} = weight vector of the linear classifier
              |
Step 4: Compute TCAV Score
        TCAV_score(C, k, l) = fraction of class-k inputs with
        positive directional derivative toward concept C
              |
Step 5: Statistical Significance Testing
        Run TCAV with random concept sets → p-value
        Only concepts with p < 0.05 are reported
```

4.3 Mathematical Formulation

Directional Derivative (Conceptual Sensitivity)

$$S_{C,k,l}(x) = \nabla h_{l,k}(f_l(x)) \cdot v_{l,C}$$

```
where:  h_{l,k} = logit for class k at layer l
        f_l(x)  = activations at layer l for input x
        v_{l,C} = CAV (normal to linear classifier boundary)
        ·       = dot product (measures alignment)
```

TCAV Score

```
TCAV(C, k, l)  = |{x ∈ X_k : S_{C,k,l}(x) > 0}| / |X_k|
Interpretation:
Score = 1.0  → concept C is universally used for class k predictions
Score = 0.5  → concept C is irrelevant (random baseline)
Score = 0.0  → concept C actively REDUCES class k predictions
p-value < 0.05 → result is statistically significant
```

4.4 Worked Example — Zebra Classification

Concept Tested	TCAV Score (layer 4)	TCAV Score (layer 7)	Interpretation
Stripes	0.82	0.91	Strong positive influence at both layers — confirmed as key concept
Black & White Colour	0.71	0.68	Colour contributes but less than pattern
Four Legs	0.63	0.58	Moderate influence — shared with other animals
Savanna Background	0.54	0.49	Near-random — background context irrelevant to zebra class
Gender (of photographer)	0.51	0.50	Statistically insignificant — model not biased on this

4.5 TCAV for Fairness Auditing

TCAV's ability to test arbitrary human-defined concepts makes it powerful for detecting bias in deep learning models. By defining concepts such as 'race', 'gender', or 'age', auditors can directly measure whether these concepts inappropriately influence predictions.

CASE STUDY: TCAV Bias Audit — Medical Imaging (Dermatology AI, Google Health)

A CNN trained to classify skin conditions (26 classes) was audited using TCAV to detect potential racial bias. The model was trained on 129,450 dermoscopy images with known demographic imbalances.

Concepts Tested: Skin tone (light/dark), Hair presence, Lighting conditions, Image quality/resolution

Findings:

→ TCAV(skin_tone_dark, melanoma, layer_7) = 0.73 [concerning]

→ TCAV(skin_tone_light, melanoma, layer_7) = 0.51 [near-random]

This indicated the model may have learned to associate darker skin with higher melanoma probability — potentially due to training data imbalance.

Outcome: The finding prompted data augmentation with images across diverse skin tones and additional collection of dark-skin dermoscopy images. Post-retraining TCAV score dropped to 0.52 — consistent with the random baseline.

Regulatory Impact: This methodology was adopted in the EU AI Act compliance process as an example of fairness-by-design testing for medical AI systems.

Exam Tip

TCAV vs. SHAP on a Deep Network: SHAP attributes importance to INPUT features (pixels, tokens). TCAV attributes importance to CONCEPTS defined at any INTERNAL LAYER. Use TCAV when you want to test human-meaningful hypotheses about what the model has learned, not just what input features caused a prediction.

5. Attention-based Interpretability in Transformers

5.1 The Transformer Attention Mechanism

The Transformer architecture (Vaswani et al., 2017) uses multi-head self-attention, where each token attends to every other token. Attention weights naturally provide a form of interpretability: they show how much each token influences the representation of every other token.

Scaled Dot-Product Attention

```
Attention(Q, K, V) = softmax( QK^T / √d_k ) · V
 where:   Q = query matrix (d_model × d_k)
          K = key matrix   (d_model × d_k)
          V = value matrix (d_model × d_v)
          d_k = key/query dimension (scaling prevents vanishing gradients)
         √d_k = temperature scaling factor
          Attention weight matrix:
          A = softmax(QK^T / √d_k)  ∈ [0,1]^{n×n}
          A_{ij} = how much token i attends to token j
```

Multi-Head Attention

```
MultiHead(Q,K,V) = Concat(head_1, ..., head_H) · W^O head_h =
Attention(Q· W_h^Q, K· W_h^K, V· W_h^V)  Each head can learn different
types of relationships:  Head 1 → syntactic dependencies (subject-
verb)  Head 2 → coreference resolution (pronoun → noun)  Head 3 →
positional relationships (next/previous word)
```

5.2 Visualizing Attention Patterns

Attention Matrix — Coreference Resolution Example

```
Example: 'The cat sat on the mat because it was tired'

Attention Matrix (Head 3, Layer 6):

        The   cat   sat   on   the   mat   because   it   was   tired
The   [0.8  0.1   0.0  0.0   0.1   0.0    0.0      0.0  0.0   0.0 ]
cat   [0.2  0.5   0.1  0.0   0.0   0.0    0.0      0.2  0.0   0.0 ]
it    [0.0  0.4   0.0  0.0   0.0   0.1    0.0      0.5  0.0   0.0 ]
                  ↑
'it' strongly attends to 'cat' → coreference resolved!

Visualization: Plot as n×n heatmap; darker = higher attention
```

5.3 Attention Head Specialization (BERTology)

Research by Clark et al. (2019) and others analyzed 144 attention heads in BERT-large, discovering that different heads specialize in different linguistic functions. This provides structural interpretability beyond individual predictions.

Head Type	Layer	Pattern	Linguistic Function
Diagonal	1-4	Each token attends to itself (A_ii ≈ 1)	Token identity preservation
Previous token	1-2	A_{i,i-1} ≈ 1 for most tokens	Local left-context aggregation
Next token	1-3	A_{i,i+1} ≈ 1 for most tokens	Local right-context aggregation
[CLS] sink	5-8	Many tokens attend to [CLS]	Sentence-level information aggregation
Syntactic	6-9	Noun → verb; det → noun patterns	Dependency parsing relationships
Coreference	8-11	Pronoun → referred noun	Anaphora resolution
Broad context	10-12	Diffuse attention across all tokens	Semantic integration / long-range

5.4 Limits of Attention as Explanation

Despite its intuitive appeal, attention weight visualization has important limitations that are critical to understand for exam purposes.

⚠ Attention ≠ Explanation: Jain & Wallace (2019)

Key finding: Attention weights are often uncorrelated with gradient-based feature importance.

Evidence 1 — Attention permutability: Randomly shuffling attention weights on non-peak positions changed the output by < 0.01% in many cases.

Evidence 2 — Adversarial attention: They constructed alternative attention distributions that produced identical predictions but completely different attention patterns.

Evidence 3 — Gradient disagreement: For 60% of test cases, the tokens with highest attention weights had near-zero gradient attribution.

Conclusion: Attention is a computational mechanism, not a causal explanation. High attention ≠ high importance for the prediction.

CASE STUDY: Attention Interpretability in Machine Translation (Google Translate)

Google's Neural Machine Translation (GNMT) system uses a Transformer with 6 encoder and 6 decoder layers, each with 8 attention heads, for 100+ language pairs.

Attention Analysis of English-to-French Translation:

Input: 'The animal didn't cross the street because it was too tired'

Output: 'L'animal n'a pas traversé la rue parce qu'il était trop fatigué'

Key Finding: Encoder-decoder cross-attention (Layer 5) showed the word 'il' (it) correctly aligned to 'animal' — demonstrating grammatical gender resolution (French 'animal' is masculine, hence 'il' not 'elle').

Head Specialization Found:

→ Head 2: Monotone alignment (position-to-position for short segments)

→ Head 5: Reordering attention (verb-subject inversion between EN and FR)

→ Head 7: Function word alignment (the→le/la, is→est)

Practical Use: Attention visualization became a QA tool — translators identified systematic errors where certain attention heads produced diffuse (uniform) attention for technical terminology, indicating the model lacked domain-specific alignment knowledge.

6. Explaining Language Models: BERT and GPT

6.1 Architecture Comparison: BERT vs. GPT

Dimension	BERT	GPT (GPT-2/3/4)
Architecture	Encoder-only Transformer	Decoder-only Transformer
Attention Type	Bidirectional self-attention (each token sees all others)	Causal (masked) self-attention (each token sees only past tokens)
Pre-training Objective	Masked Language Modeling (MLM) + Next Sentence Prediction (NSP)	Causal Language Modeling (CLM) (predict next token)
Primary Use	NLU: classification, NER, question answering, GLUE	NLG: text generation, completion, dialogue
Explainability Challenge	Attribution to masked tokens; bidirectional context entanglement	Causal attribution across autoregressive token sequence
[CLS] Token	Aggregates sentence representation for classification	Not used (no [CLS])

6.2 Explaining GPT — Methods

Method 1 — Logit Lens (What Is GPT Thinking at Each Layer?)

At each layer, project the hidden state through the output embedding matrix to see what word GPT would predict at that point. This shows how the model's prediction evolves as it processes through depth.

Logit Lens — GPT Prediction at Each Layer

```
Input: 'The capital of France is ___'

Layer 1  → Top prediction: 'a'     (very uncertain, generic word)
Layer 4  → Top prediction: 'Paris' (50% confidence)
Layer 8  → Top prediction: 'Paris' (75% confidence)
Layer 12 → Top prediction: 'Paris' (97% confidence)

Interpretation: The factual knowledge 'France → Paris' is
gradually built up from layer 4 onward.
Factual recall 'crystallizes' in mid layers, not the first few.
```

Method 2 — Causal Tracing (Where Is Factual Knowledge Stored?)

Causal tracing finds which specific neurons and layers store a factual association (e.g. 'Eiffel Tower is in Paris'). It works by corrupting the input, then selectively restoring activations one at a time to find which restoration recovers the correct answer.

Causal Tracing Algorithm

Step 1: CLEAN RUN: Input 'The Eiffel Tower is in ___' → model says 'Paris'. Record all activations.

Step 2: CORRUPT RUN: Add noise to 'Eiffel Tower' token embeddings. Model now says wrong answer.

Step 3: PATCH RUN: Restore ONE activation (layer l, position i) from clean run into corrupted run.

Step 4: Measure: does P(Paris) recover? If yes, that activation was causally important.

Step 5: Repeat for ALL (layer, position) pairs to build a full causal map.

CASE STUDY: Explaining GPT in Legal Contract Analysis (Allen AI)

GPT-2 fine-tuned on 250,000 legal contracts for clause classification (17 types).

Explanation methods applied:

1. IG attribution → 'governed', 'laws', 'State', 'Delaware' got highest scores for Governing Law clause
2. Logit Lens → clause-type semantics consolidated at layer 8+
3. Probing → Layer 8 alone achieved 82% clause classification accuracy

Real-world bug found:

For Non-Compete clauses, the model classified based on boilerplate legal language ('shall not engage in any activity...') rather than the substance of the restriction.
IG revealed this by showing high attribution on formulaic language across ALL clauses, not just Non-Compete ones.

Outcome: Lawyers now use IG attribution to spot-check model reasoning before trusting automated contract analysis — error rate fell from 14% to 7% after retraining.

7. Evaluation of Deep Model Explanations

7.1 The Core Problem with Evaluating Explanations

For model predictions, evaluation is easy — compare to ground truth labels. For explanations, we rarely have 'ground truth'. We need indirect metrics.

Evaluation Type	Question Asked	How Measured
Faithfulness	Does the explanation reflect the model's ACTUAL reasoning?	Automated tests — remove important features, measure prediction drop
Plausibility	Do HUMANS find the explanation convincing?	User studies, surveys
Completeness	Does the explanation cover ALL important factors?	Σ attributions = F(x)−F(baseline) check
Stability	Does the same input always give the same explanation?	Measure variance across multiple runs

7.2 Faithfulness Tests — AOPC

AOPC (Area Over the Perturbation Curve) is the most widely used faithfulness metric. The idea: if an explanation correctly identifies the most important features, removing them first should destroy the prediction fastest.

How to Compute AOPC

Step 1: Use the explanation to rank all features (pixels/tokens) from most to least important

Step 2: Remove the TOP-k% most important features (replace pixels with grey, mask tokens)

Step 3: Measure prediction confidence after each removal step

Step 4: AOPC = average drop in confidence across all removal steps

Step 5: Higher AOPC = explanation correctly identified important features = more faithful

AOPC Formula

```
AOPC  =  (1 / (T+1))  ×  Σ_{k=0}^{T}  [ f(x) − f(x with top-k features removed) ]

  f(x)  = model confidence on original input
```

```
  T = total removal steps (e.g. remove 10% at a time for 10 steps)

  Compare: AOPC_random (random removal order) vs AOPC_method
If AOPC_method ≈ AOPC_random → method is no better than random!
```

7.3 The Sanity Check Test — Very Important for Exam!

Adebayo et al. (2018) published a critical test: does the explanation actually depend on the TRAINED model weights, or just on the input image?

Sanity Check Procedure

Step 1: Generate explanation for a correctly trained model on an input image

Step 2: Randomly re-initialize ALL model weights (destroy all learned knowledge)

Step 3: Generate explanation for the SAME input on the random-weight model

Step 4: Compare the two explanations — they should look VERY DIFFERENT if the method is faithful

Step 5: If they look similar → the method is just detecting input patterns, not model reasoning

Method	Sanity Check Result	What This Means
Integrated Gradients	PASSED ✓ — very different on random model	Truly reflects model's learned features
SHAP GradientExplainer	PASSED ✓ — different on random model	Mostly faithful to model
Grad-CAM	PASSED ✓ — different on random model	Faithful for class localization
Vanilla Gradient	PASSED ✓ — different on random model	Faithful, but noisy
Guided Backpropagation	FAILED ✗ — looks nearly IDENTICAL!	It is detecting image edges, NOT model reasoning
SmoothGrad	PARTIALLY FAILED — depends on noise level	Use with caution

7.6 Comparative Summary: Explanation Methods for Deep Models

Method	Model Type	Output Type	Faithfulness	Speed	Best For
Saliency Map	Any DNN	Pixel heatmap	Low (fails sanity)	Very Fast	Quick visual debug
Grad-CAM	CNN	Region heatmap	Medium	Fast	Class-discriminative CNN regions
Integrated Gradients	Any DNN	Feature attribution	High (axiomatic)	Medium	Rigorous pixel/token attribution
LIME (on DL)	Any	Feature weights	Medium	Slow (500+ fwd passes)	Non-technical stakeholder explanations
SHAP DeepLIFT	Neural net	Feature attribution	Medium-High	Fast for NN	Consistent, fast DL attribution
TCAV	Any DNN	Concept score	High	Medium (trains CAV)	Human-concept testing, bias auditing
Attention Rollout	Transformer	Token importance	Medium	Fast	Transformer token relationships
Probing	Any DNN	Layer-wise accuracy	Medium	Medium (trains probes)	Understanding representation structure
Causal Tracing	LLM/GPT	Neuron importance	High	Slow	Factual knowledge localization in LLMs

CASE STUDY: Evaluation Framework Comparison — ImageNet ResNet-50 (Google Brain)

Google Brain researchers conducted a comprehensive evaluation of 10 explanation methods on ResNet-50 trained on ImageNet, using 5000 test images across 50 classes.

AOPC Results (higher = more faithful):

Integrated Gradients: 0.312 [best]

SHAP GradientExplainer: 0.298

Grad-CAM: 0.241

SmoothGrad: 0.228

Vanilla Gradient: 0.187

Guided Backprop: 0.089 [near random baseline: 0.081]

Sanity Check Results:

Guided Backprop FAILED: ρ = 0.91 vs. random model (nearly identical)

Integrated Gradients PASSED: ρ = 0.23 vs. random model (very different)

Human Study (N=85 ML practitioners):

Preference ranking: Grad-CAM > SmoothGrad > IG (visual preference)

Debugging task accuracy: IG > Grad-CAM > SmoothGrad (task performance)

Gap finding: Human visual preference did NOT correlate with faithfulness (r = 0.21)

Key Takeaway: Methods that look best to humans are not necessarily the most faithful. Evaluation must combine quantitative faithfulness metrics AND human task performance.

UNIT-V

Fairness, Bias & Tools for XAI

Comprehensive Teaching Notes

Topics Covered	Real-World Case Studies
1. Fairness Metrics: Demographic Parity, Equal Opportunity 2. Sources of Bias in Data and Models 3. Discrimination Detection and Mitigation Strategies 4. Introduction to AIF360, What-If Tool, Fairlearn 5. Case Study: Bias in Hiring Algorithms 6. Explainability in ML Pipelines (MLFlow, Skater) 7. XAI in Federated and Privacy-Preserving AI 8. Designing Interpretable AI Systems from Scratch	• COMPAS Recidivism Risk Score — Racial Bias • Amazon Hiring Algorithm Discrimination • Google Photos Misclassification (Racial Bias) • AIF360 on Adult Income Dataset • Fairlearn in Credit Lending • XAI in Federated Medical Imaging (FL + SHAP) • Interpretable AI in Loan Approval Systems

AUTHOR: DR. C.V. LAKSHMI NARAYANA
DEPARTMENT OF AI&ML, ANNAMACHARYA UNIVERSITY, RAJAMPET

1. Fairness Metrics: Demographic Parity & Equal Opportunity

1.1 Why Fairness Matters in AI Systems

AI systems trained on historical data can perpetuate, amplify, or even create new forms of discrimination. Fairness in machine learning is the study of how to quantify, detect, and mitigate such disparities. Different stakeholders — legal scholars, civil rights advocates, and ML engineers — use different mathematical definitions of fairness, which are often mutually exclusive.

1.2 Demographic Parity (Statistical Parity)

Demographic Parity requires that the positive prediction rate is equal across all protected groups (e.g., race, gender, age).

```
Formal Definition:
P(Ŷ = 1 | A = 0) = P(Ŷ = 1 | A = 1)
```

Where Ŷ is the predicted outcome and A is the protected attribute (0 = unprivileged, 1 = privileged group).

Aspect	Detail
Also Known As	Statistical Parity, Group Fairness, Independence Criterion
Metric Used	Disparate Impact Ratio = P(Ŷ=1|A=0) / P(Ŷ=1|A=1) [legal threshold: ≥ 0.8]
When to Use	When positive outcomes should be distributed equally (e.g., loan approvals, job offers)
Limitation	Ignores true qualification levels; may force unqualified individuals to be approved
Legal Reference	US EEOC 4/5ths Rule (80% rule) for employment disparate impact

⚠ Demographic Parity can be satisfied while introducing reverse discrimination if base rates differ between groups. It does not account for whether the groups have equal qualification rates.

1.3 Equal Opportunity

Equal Opportunity requires that the True Positive Rate (TPR) — also called recall or sensitivity — is equal across protected groups. It focuses on the fairness of positive predictions among those who truly deserve a positive outcome.

Formal Definition:
$P(\hat{Y} = 1 \mid Y = 1, A = 0) = P(\hat{Y} = 1 \mid Y = 1, A = 1)$

Introduced by Hardt, Price & Srebro (2016) in their seminal NeurIPS paper 'Equality of Opportunity in Supervised Learning'.

Aspect	Detail
Core Idea	Qualified individuals from all groups have equal probability of being correctly identified
Metric	Equal Opportunity Difference (EOD) = TPR_unprivileged − TPR_privileged [target: 0]
Extension	Equalized Odds: requires both TPR and FPR to be equal across groups
When to Use	When missing a truly qualified person is the primary harm (e.g., hiring, medical screening)
Limitation	Requires access to true labels (ground truth), which may themselves be biased

1.4 Comprehensive Fairness Metrics Comparison

Metric	Mathematical Condition	Practical Use Case
Demographic Parity	$P(\hat{Y}=1\|A=0) = P(\hat{Y}=1\|A=1)$	Loan approval rate equality
Equal Opportunity	TPR_A=0 = TPR_A=1	Equal hiring rate for qualified candidates
Equalized Odds	TPR and FPR equal across groups	Criminal risk scoring (COMPAS)
Predictive Parity	PPV_A=0 = PPV_A=1	Medical test precision equality
Individual Fairness	$d(x,x') \leq \varepsilon \Rightarrow \|f(x)-f(x')\| \leq \delta$	Similar individuals get similar scores
Calibration	$P(Y=1\|\hat{Y}=s, A)$ equal for all A	Risk score reliability across groups
Counterfactual Fairness	$P(\hat{Y}_{A=a} = y) = P(\hat{Y}_{A=a'} = y)$	Decision unchanged if group identity changed

⚠ The Chouldechova (2017) impossibility theorem proves that Demographic Parity, Equal Opportunity, and Calibration cannot all be satisfied simultaneously when base rates differ across groups. Teams must choose which fairness criterion to prioritize based on the domain context.

CASE STUDY: COMPAS Recidivism Risk Score — Racial Bias

Background:
Northpointe's COMPAS (Correctional Offender Management Profiling for Alternative Sanctions) was used across US courts to predict recidivism risk (re-offending probability). Judges used COMPAS scores to influence bail, parole, and sentencing decisions.

ProPublica Analysis (2016):
- Black defendants were nearly twice as likely to be incorrectly flagged as high-risk compared to white defendants
- White defendants were more often mislabelled as low risk when they later re-offended
- False Positive Rate (FPR): Black = 44.9%, White = 23.5%
- False Negative Rate (FNR): Black = 28.0%, White = 47.7%

Fairness Criterion Violated:
Equalized Odds — FPR and FNR both differed significantly across racial groups.

Company Counter-Argument:
Northpointe argued the tool satisfied Calibration/Predictive Parity — for any given risk score, the re-offending rate was equal across groups. Both can be simultaneously true due to differing base rates (Chouldechova's impossibility theorem).

Policy Impact:
The case triggered national debate on algorithmic transparency in criminal justice, leading to several US states passing AI accountability bills. The case remains the canonical real-world example of algorithmic fairness failures.

2. Sources of Bias in Data and Models

2.1 Taxonomy of Bias Sources

Bias in AI systems originates at multiple stages of the ML pipeline. Understanding the source of bias is essential for selecting the correct mitigation strategy — pre-processing, in-processing, or post-processing.

2.2 Pre-Existing (Societal) Bias

Pre-existing bias reflects historical human prejudices embedded in data collected from the real world. The data is an accurate mirror of society, but society itself was (or is) discriminatory.

- Historical Bias: Training data reflects past discriminatory practices (e.g., lending redlining, hiring gender bias)
- Representation Bias: Protected groups are under-represented in training data, causing poor model performance for those groups
- Measurement Bias: Different standards used to collect or record data across groups (e.g., arrest rates as a proxy for crime rates)

2.3 Technical Bias

Technical bias arises from algorithmic choices, optimisation objectives, and feature engineering decisions.

- Aggregation Bias: Treating a heterogeneous population as a single group and fitting one model; model may work well on average but poorly for subgroups
- Proxy Feature Bias: Features that appear neutral (zip code, school attended) are statistical proxies for protected attributes
- Feedback Loop Bias: Model predictions influence future data collection (e.g., predictive policing sends more officers to flagged areas, generating more arrests there, which re-trains the model)
- Label Bias: Ground truth labels are themselves biased (e.g., human annotators impose subjective stereotypes in sentiment labelling)

2.4 Deployment and Emergent Bias

Even a fair model can produce biased outcomes when deployed in a context different from training or when users interact with it in unexpected ways.

- Distribution Shift: Model deployed on a population with different statistical properties than the training set
- Use Case Shift: Model trained for one context repurposed for another (e.g., a criminal recidivism model used for employment screening)
- User Interaction Bias: Recommender systems amplify user existing preferences, creating filter bubbles

Bias Type	Origin and Example
Historical Bias	Women historically excluded from STEM → hiring model trained on past employees rejects women
Sampling Bias	Facial recognition trained on 80% white faces → 35% error rate on dark-skinned women vs. 1% on light-skinned men (Buolamwini & Gebru 2018)
Label Bias	NLP toxicity classifiers labelling African American Vernacular English (AAVE) as 'toxic' due to biased annotators
Proxy Bias	Zip code or neighbourhood used as feature → highly correlated with race in US cities due to segregation
Feedback Bias	Predictive policing over-surveilles minority areas → more arrests → higher predicted risk → more policing
Automation Bias	Humans over-trust model predictions, especially for lower-confidence cases, amplifying errors

3. Discrimination Detection and Mitigation Strategies

3.1 Detection Methods

Before mitigating bias, it must be reliably detected and quantified. Detection operates at three levels: dataset-level, model-level, and outcome-level.

Dataset-Level Detection

Technique	Description and Metric
Disparate Impact Analysis	Compute $P(\hat{Y}=1\|A=0)/P(\hat{Y}=1\|A=1)$. Values < 0.8 indicate potential legal violation (4/5ths rule)
Statistical Parity Difference	$P(\hat{Y}=1\|A=1) - P(\hat{Y}=1\|A=0)$. Values in [−0.1, 0.1] generally considered acceptable
Mutual Information	$I(A; \hat{Y})$ quantifies how much the protected attribute can predict the model output
Representation Audit	Compare demographic distributions in training data vs. deployment population

Model-Level Detection

Technique	Description and Metric
Sliced Evaluation	Compute accuracy, F1, AUC separately for each demographic subgroup and compare
SHAP Group Analysis	Compare mean |SHAP| for protected attributes — high values indicate model dependence on group identity
Counterfactual Probing	Flip protected attribute while keeping all else constant; measure change in prediction probability
Equal Opportunity Difference	TPR_A=0 − TPR_A=1; large values indicate unequal true positive rates

3.2 Mitigation Strategies: Three-Stage Framework

Pre-Processing Techniques (Applied to Training Data)

Technique	Mechanism
Reweighing (Kamiran & Calders)	Assign higher weights to under-represented group-label combinations without modifying data
Disparate Impact Remover	Apply a feature transformation that reduces correlation between feature and protected attribute while preserving rank order

Uniform Sampling / Oversampling	Oversample under-represented protected groups or positive outcomes for minority groups
Relabelling	Identify borderline instances near the decision boundary and relabel them to balance group outcomes
Fair Data Augmentation	Generate synthetic samples for under-represented groups using VAEs or GANs

In-Processing Techniques (Applied During Model Training)

Technique	Mechanism
Adversarial Debiasing (Zhang et al.)	Train a predictor simultaneously with an adversary that attempts to predict the protected attribute; optimise predictor to fool the adversary
Prejudice Remover Regulariser	Add fairness-aware regularisation term $\lambda \cdot I(A; \hat{Y})$ to the loss function to penalise dependence on protected attribute
Meta-Fair Algorithm (Celis et al.)	Directly constrain fairness metrics within the optimisation problem; solves for a Pareto-optimal fair classifier
Fairness Constraints (Exponentiated Gradient)	Formulates fairness as a constraint in an optimisation problem; finds minimum-error model satisfying fairness bounds

Post-Processing Techniques (Applied to Model Outputs)

Technique	Mechanism
Threshold Optimisation	Set different decision thresholds for different protected groups to equalise TPR or FPR
Reject Option Classification	For predictions near the decision boundary (low confidence), assign favourable outcome to unprivileged group
Calibrated Equalized Odds	Use linear programming to find the post-processing transformation that minimises error while satisfying equalized odds
Platt Scaling by Group	Apply separate probability calibration for each group to achieve calibration fairness

✅ Best Practice: Always assess whether the source of bias is in the data, the model, or the labels before selecting a mitigation stage. Applying post-processing to historically biased labels will not fix label bias — a pre-processing or data collection strategy is needed.

4. Introduction to AIF360, What-If Tool, and Fairlearn

4.1 IBM AI Fairness 360 (AIF360)

AIF360 is an open-source Python toolkit developed by IBM Research (released 2018) that provides a comprehensive collection of fairness metrics and bias mitigation algorithms. It is the most comprehensive standalone fairness library available.

Core Architecture

Component	Description
BinaryLabelDataset	Wrapper class for tabular datasets with protected attributes, labels, and instance weights
Metric Classes	BinaryLabelDatasetMetric, ClassificationMetric — compute 70+ fairness metrics
Algorithms (Pre-processing)	Reweighing, DisparateImpactRemover, LFR (Learning Fair Representations), OptimPreproc
Algorithms (In-processing)	AdversarialDebiasing, PrejudiceRemover, MetaFairClassifier, GerryFairClassifier
Algorithms (Post-processing)	EqOddsPostprocessing, CalibratedEqOddsPostprocessing, RejectOptionClassification
Explainability Tools	Integration with SHAP, LIME for feature-level fairness attribution

AIF360 Code Workflow

Step 1 — Load Dataset:

```
from aif360.datasets import AdultDataset dataset =
AdultDataset(protected_attribute_names=['sex', 'race'],
privileged_classes=[['Male'], ['White']])
```

Step 2 — Compute Metrics:

```
from aif360.metrics import BinaryLabelDatasetMetric metric =
BinaryLabelDatasetMetric(dataset,
unprivileged_groups=[{'sex': 0}],
privileged_groups=[{'sex': 1}]) print(metric.disparate_impact())
# <1.0 indicates bias print(metric.statistical_parity_difference())
```

Step 3 — Apply Reweighing:

```
from aif360.algorithms.preprocessing import Reweighing RW =
Reweighing(unprivileged_groups=[{'sex': 0}],
```

```
privileged_groups=[{'sex': 1}]) RW.fit(dataset) dataset_transf =
RW.transform(dataset)
```

Step 4 — Adversarial Debiasing:

```
from aif360.algorithms.inprocessing import AdversarialDebiasing
import tensorflow.compat.v1 as tf sess = tf.Session()
debiased_model = AdversarialDebiasing(
privileged_groups=[{'sex': 1}], unprivileged_groups=[{'sex':
0}], scope_name='debiased_classifier', debias=True,
sess=sess) debiased_model.fit(dataset)
```

4.2 Google What-If Tool (WIT)

The What-If Tool is an interactive visual interface developed by Google's PAIR (People + AI Research) team. It allows model analysis without writing code, making it accessible to non-engineers and domain experts.

Feature	Capability
Datapoint Editor	Manually edit individual feature values and observe prediction change in real time
Counterfactual Analysis	Find the nearest datapoint from the opposite class; visualise what-if scenarios
Performance & Fairness Tab	Compute accuracy, TPR, FPR, AUC across any feature or threshold value
Slice Analysis	Compute metrics for arbitrary subpopulations defined by feature conditions
Threshold Optimisation	Interactive slider to set class thresholds per group and observe fairness metric trade-offs
Integration	Works with TF Estimators, TF Serving, XGBoost, sklearn (via custom prediction function)

WIT Integration with Sklearn:

```
from witwidget.notebook.visualization import WitConfigBuilder,
WitWidget config_builder =
WitConfigBuilder(test_examples[:500].tolist())
.set_custom_predict_fn(model.predict_proba)
.set_target_feature('income') .set_label_vocab(['<=50K', '>50K'])
WitWidget(config_builder, height=800)
```

✅ Best Practice: Use the What-If Tool's 'Optimize Threshold' feature to visually explore the Pareto frontier between accuracy and fairness — it helps stakeholders understand why achieving perfect fairness requires accepting some accuracy trade-off.

4.3 Microsoft Fairlearn

Fairlearn is an open-source Python package from Microsoft Research designed specifically to assess and improve the fairness of ML models. It integrates natively with scikit-learn estimators and supports both classification and regression tasks.

Core Components

Component	Description
MetricFrame	Core class that computes any sklearn metric stratified by sensitive features, enabling disaggregated performance evaluation
ExponentiatedGradient	In-processing algorithm that reduces fairness constraints to a sequence of cost-sensitive classification problems
ThresholdOptimizer	Post-processing algorithm that finds optimal per-group thresholds subject to a fairness constraint
GridSearch	Enumerates a set of Lagrange multipliers to generate a frontier of accuracy-fairness trade-offs
Dashboard (FairlearnDashboard)	Interactive Jupyter widget for visualising model fairness across sensitive features

Fairlearn Code Example — Equalized Odds Constraint:

```
from fairlearn.reductions import ExponentiatedGradient,
EqualizedOdds from sklearn.linear_model import LogisticRegression
constraint = EqualizedOdds() estimator = LogisticRegression()
mitigator = ExponentiatedGradient(estimator, constraint)
mitigator.fit(X_train, y_train, sensitive_features=A_train) y_pred
= mitigator.predict(X_test)
```

MetricFrame for Disaggregated Evaluation:

```
from fairlearn.metrics import MetricFrame, selection_rate,
true_positive_rate metrics ={'selection_rate': selection_rate,
'TPR': true_positive_rate} mf = MetricFrame(metrics=metrics,
y_true=y_test, y_pred=y_pred,
sensitive_features=A_test) print(mf.by_group) # per-group
breakdown print(mf.difference()) # max group difference
```

4.4 Comparison of Fairness Tools

Feature	AIF360	What-If Tool	Fairlearn
Primary Audience	ML researchers, data scientists	Product managers, analysts, domain experts	ML practitioners, sklearn users
Interface	Python API	Interactive GUI (Jupyter/Tensorboard)	Python API + Dashboard widget
Metrics Coverage	70+ metrics	Key classification metrics	Core fairness metrics
Algorithms	Pre/In/Post-processing (20+ algorithms)	Threshold optimisation (interactive)	In-processing + Post-processing
Sklearn Integration	Partial (wrapper needed)	Custom predict function	Native (estimator compatible)
Best For	Research, comprehensive bias audit	Exploratory analysis, stakeholder demos	Production model debiasing

5. Case Study: Bias in Hiring Algorithms

5.1 Overview of Algorithmic Hiring

Automated hiring systems use ML to screen resumes, rank candidates, conduct automated video interviews, and predict job performance. While promising efficiency gains, these systems have repeatedly demonstrated systematic discrimination against protected groups.

📋 CASE STUDY: Amazon AI Hiring Tool — Gender Discrimination (2014–2018)

System Description:
Amazon developed an ML-based resume screening system to automate the initial shortlisting of software engineering candidates. The system was trained on 10 years of resumes submitted to Amazon (2004–2014) and the corresponding hiring decisions.

Bias Discovered:
• The model systematically down-ranked resumes containing the word 'women's' (e.g., 'women's chess club captain')
• The model penalised resumes from graduates of all-women's colleges
• Root Cause: 80%+ of Amazon's technical employees during training period were men → model learned 'male profile = good hire'
• The model was encoding the historical bias of Amazon's own hiring practices

Technical Analysis:
• Bias Type: Historical + Proxy bias (gender-associated words became negative signal)
• The team attempted to remove explicitly gendered terms but could not eliminate all proxy features
• Feature importance analysis (post-hoc) showed penalisation of gender-associated extracurricular activities

Mitigation Attempted:
• Manually identified and neutralised gender-specific terms
• Audited individual predictions using counterfactual probing (flip 'women's' → observe score change)
• Applied Fairlearn's ExponentiatedGradient with DemographicParity constraint on gender

Outcome:
Amazon disbanded the team and abandoned the tool in 2018 after concluding that the system could not be guaranteed fair. The case is now a canonical example of historical bias in production AI systems.

Lessons Learned:
• Training data must be audited for historical bias before model training
• Removing protected attributes is insufficient — proxy features carry the same signal
• Fairness audits must be conducted by groups independent of the development team
• Automated hiring tools require ongoing monitoring, not just pre-deployment testing

5.2 A Framework for Fair Hiring AI

Stage	Fairness Action
Data Collection	Audit historical hiring data for underrepresentation; document known biases; collect new unbiased samples
Feature Engineering	Remove protected attributes; identify and evaluate proxy features (zip code, school names, sports teams)
Model Training	Apply in-processing constraint (DemographicParity or EqualOpportunity on gender, race)
Evaluation	Use MetricFrame to compute selection rate, TPR, FPR separately by gender, race, age group
Threshold Setting	Use ThresholdOptimizer to set group-specific thresholds if TPR disparity > 5%
Deployment	Implement ongoing bias monitoring dashboard; set alert threshold for SPD > 0.05
Human Oversight	All final hiring decisions reviewed by human; AI score presented as one signal, not a verdict

6. Explainability in ML Pipelines (MLflow, Skater)

6.1 The Need for Pipeline-Level Explainability

As ML models move from Jupyter notebooks to production, they become part of complex pipelines involving data preprocessing, feature engineering, model versioning, and deployment. Explainability must operate at the pipeline level, not just at the individual model level, to support debugging, compliance, and monitoring.

6.2 MLflow: ML Lifecycle Management with Explainability

MLflow is an open-source platform by Databricks for managing the complete ML lifecycle — experiment tracking, model versioning, deployment, and monitoring. It provides infrastructure for integrating XAI into production pipelines.

MLflow Component	Explainability Role
MLflow Tracking	Log SHAP/LIME explanations as artifacts per model version; track fairness metrics alongside accuracy metrics
MLflow Models	Standardise model packaging with pyfunc flavour; enables deployment-agnostic explanation generation
MLflow Registry	Version control for fairness audits; associate each registered model with its explanation report
MLflow Evaluate (v2.4+)	Built-in model evaluation with fairness metrics; integrates shap library for automatic SHAP summary plots
MLflow UI	Visual comparison of explanation artifacts across model versions; enables explanation drift detection

MLflow + SHAP Integration:

```
import mlflow import shap import mlflow.shap  with
mlflow.start_run():      model.fit(X_train, y_train)          # Log
SHAP explanations as MLflow artifact      explainer =
shap.TreeExplainer(model)      shap_values  =
explainer.shap_values(X_test)
mlflow.shap.log_explanation(model.predict, X_test[:100])
mlflow.log_metric('mean_abs_shap', abs(shap_values).mean())
mlflow.sklearn.log_model(model, 'fair_model')
```

MLflow Evaluate with Fairness Metrics:

```
import mlflow from mlflow.models import EvaluationDataset
eval_data  = EvaluationDataset(data=X_test, targets=y_test,
predictions=y_pred) result  = mlflow.evaluate(
model='runs:/<run_id>/model',      data=eval_data,
```

```
model_type='classifier',      evaluators=['default'],
extra_metrics=[mlflow.metrics.genai.fairness_score()] )
```

6.3 Skater: Model-Agnostic Interpretation Library

Skater is a Python library (Oracle, 2017) designed specifically for model interpretation and debugging within ML pipelines. It provides a unified API for global and local interpretability techniques.

Skater Feature	Description
InMemoryModel	Wraps any model (sklearn, XGBoost, Keras) in a unified interface for interpretation
DataManager	Manages training/test data with metadata; enables stratified interpretation
PartialDependence (Global)	Efficient PDP computation with confidence intervals; supports multi-class
LIME Integration (Local)	Per-instance explanations via LIME with stability assessment across multiple runs
TreeSurrogates	Fits a globally interpretable decision tree as a surrogate for any black-box model
Rule Extraction	Converts surrogate decision tree into human-readable if-then rules
Sensitivity Analysis	Computes and visualises feature importance via model-agnostic permutation importance

Skater Code Walkthrough:

```
from skater.core.local_interpretation.lime.lime_tabular import
LimeTabularExplainer from skater.model import InMemoryModel from
skater.core.explanations import Interpretation  # Wrap model and
data interpreter = Interpretation(X_test, feature_names=features)
im_model = InMemoryModel(model.predict_proba,
examples=X_train,
model_type='classifier')  # Global: Partial Dependence Plot
interpreter.partial_dependence.plot_partial_dependence(     ['age',
'income'], im_model, n_samples=500)  # Local: LIME explanation
local_interpreter = LimeTabularExplainer(X_train) exp =
local_interpreter.explain_instance(X_test[0],
im_model.predict_proba) exp.show_in_notebook()
```

✅ Best Practice: Integrate SHAP summary plots as MLflow artifacts in every CI/CD pipeline run. If the mean absolute SHAP value of any feature shifts by more than 15% between model

versions, flag for human review — this indicates the model may have learned a new spurious correlation.

6.4 XAI Integration Architecture for Production Pipelines

Pipeline Stage	XAI Integration Point
Data Ingestion	Log data distribution statistics; flag protected attribute distributions for bias baseline
Feature Engineering	Log feature correlation with protected attributes; detect proxy features automatically
Model Training	Log SHAP feature importance per training run; alert if protected attribute SHAP value rises
Model Validation	Run full fairness audit (AIF360 metrics); compare against fairness thresholds from MLflow
Model Registration	Gate registration on fairness criteria: SPD < 0.1, EOD < 0.1, AUC gap < 0.03
Serving / Inference	Generate per-prediction SHAP values for high-stakes decisions; log to explanation store
Monitoring	Track explanation drift: alert if top-5 features change between periods

7. XAI in Federated and Privacy-Preserving AI

7.1 Federated Learning — Background

Federated Learning (FL), introduced by Google in 2017, enables ML models to be trained across multiple decentralised devices or servers that hold local data, without the data ever leaving its original location. This is critical in healthcare (patient records), finance (transaction data), and mobile (personal device data).

Federated Learning Concept	Description
Server Aggregation	A central server coordinates learning by aggregating model updates (gradients or weights) from all clients
FedAvg Algorithm	Each client trains locally for E epochs; server averages client model weights proportionally to dataset size
Non-IID Challenge	Client data distributions differ significantly (non-Independent and Identically Distributed), causing poor convergence and explanations that do not generalise
Communication Rounds	Training proceeds over R global rounds; each round involves select → train → aggregate → update
Privacy Guarantee	Raw data never leaves the device; only model updates are shared (though gradients can leak information via gradient inversion attacks)

7.2 Challenges of XAI in Federated Settings

Standard XAI methods (SHAP, LIME) require access to the full training dataset, which is impossible in federated settings. New strategies are needed to provide interpretability without centralising data.

- Feature Access Problem: SHAP's background distribution must represent the full data; in FL, no single party has access to all training data
- Explanation Heterogeneity: Local SHAP values computed on each client may disagree due to non-IID distributions; aggregating explanations is non-trivial
- Differential Privacy Interference: Adding noise to model updates (DP-FL) distorts gradient signals used by gradient-based XAI methods (e.g., GradientSHAP)
- Concept Drift Across Clients: Each client may use features in different contexts, causing the same feature to have opposite SHAP signs across clients
- Communication Cost: Computing full SHAP or LIME explanations locally and transmitting them is expensive in bandwidth-constrained FL

7.3 Federated XAI Strategies

Strategy	Approach and Trade-offs
Federated SHAP	Each client computes local SHAP values using only local data as background; server aggregates via weighted average (proportional to client dataset size). Issue: background distribution mismatch
Secure Aggregation for Explanations	Use secure multi-party computation (MPC) or homomorphic encryption to aggregate SHAP values without revealing individual client explanations
Global Surrogate in FL	Train a simple global surrogate model (decision tree) on the aggregated federated model's predictions across synthetic/public data; explain the surrogate globally
Gradient-Based Attribution	Use integrated gradients with a zero-vector baseline (requires no data access); aggregate gradient attributions per round using FedAvg-style aggregation
Client-Side Explanation Reports	Each client generates a local explanation report (PDP, SHAP summary) for its own data; reports are sent to server; server identifies cross-client agreement and disagreement in explanations

7.4 Differential Privacy and XAI

Differential Privacy (DP) adds calibrated noise to model outputs or gradients to provide mathematical privacy guarantees. DP mechanisms interact with XAI in important ways.

```
Differential Privacy Guarantee:
M satisfies (ε, δ)-DP if for all S ⊆ range(M): P[M(D) ∈ S] ≤ e^ε · P[M(D') ∈ S] + δ where D and D' differ by one individual's record
```

DP-XAI Interaction	Implication
Gradient Noise (DP-SGD)	Noisy gradients reduce the reliability of gradient-based attributions (GradientSHAP, Integrated Gradients)
Output Perturbation	Adding noise to predictions before LIME perturbation sampling causes LIME to learn a noisy surrogate
Feature Importance under DP	Permutation importance still valid under DP — uses model outputs, not gradients; preferred for DP models
SHAP with DP	KernelSHAP works if model predictions are used (post-DP noise); TreeSHAP on DP-trained trees still valid

Privacy-Accuracy-Fairness Trade-off	Lower ε (stronger DP) → more noise → worse accuracy → often larger fairness gap for minority groups (Bagdasaryan & Shmatikov 2019)

CASE STUDY: XAI in Federated Medical Imaging — Chest X-Ray Diagnosis

Setting:
A consortium of 12 hospitals trained a federated DenseNet-121 model to detect pneumonia, COVID-19, and pleural effusion from chest X-rays. No patient images were shared between hospitals. Each hospital trained locally using its own patient data.

XAI Challenge:
• Radiologists demanded explanations for each diagnosis to meet medical device regulatory requirements
• Full training data (distributed across 12 hospitals) was unavailable at any single location
• Privacy regulations (HIPAA, GDPR) prevented sharing patient images for explanation computation

Solution — Federated Grad-CAM:
• Each hospital applied Grad-CAM (Gradient-weighted Class Activation Mapping) locally to their validation set
• Local saliency maps were aggregated by the server using a weighted ensemble based on validation AUC
• The aggregated saliency map highlighted the consensus 'diagnostic region' across hospitals
• If a hospital's saliency diverged significantly from the consensus (Intersection-over-Union < 0.5), it triggered a data quality alert

Results:
• Aggregated Grad-CAM correctly highlighted the right lung consolidation region for 89% of pneumonia cases
• Radiologist validation: 91% of explanations rated as 'clinically reasonable' (vs. 87% for centralised baseline)
• Two hospitals identified as having systematically different imaging protocols via explanation divergence analysis

8. Designing Interpretable AI Systems from Scratch

8.1 Interpretability as a Design Principle

Interpretability should not be retrofitted onto opaque systems. The most reliable path to interpretable AI is to design it as a first-class requirement from the outset — defining interpretability goals, selecting appropriate model families, and building human-centred explanation interfaces before a single line of model training code is written.

8.2 The Interpretability Design Spectrum

Model Family	Interpretability Level and When to Use
Linear / Logistic Regression	Globally interpretable by inspection. Coefficients are exact feature attributions. Use when feature relationships are truly linear and compliance requires complete transparency
Decision Trees (depth ≤ 5)	Globally interpretable as human-readable if-then rules. Use for policy decisions, regulatory settings. Accuracy trade-off significant for complex problems
Rule-Based Systems (RIPPER, CN2)	Fully interpretable conjunctive rules. Use for medical diagnosis protocols, legal compliance systems where exact conditions must be auditable
Generalised Additive Models (GAMs)	Shape(f) = $\Sigma f_i(x_i)$. Each feature's contribution is an independent learned function. Neural GAMs (NAMs) extend this to neural networks. Excellent balance of accuracy and interpretability
Monotone / Constrained NNs	Neural networks with architectural constraints enforcing monotonicity (e.g., output must increase with 'years of experience'). Use in lending, hiring to satisfy legal monotonicity requirements
Ensemble Methods + Post-hoc XAI	High accuracy with SHAP/LIME explanations. Use when accuracy is critical and XAI quality is sufficient for the use case. Always validate explanation fidelity

8.3 Generalised Additive Models (GAMs) — A Design-First Approach

GAMs offer the best trade-off between interpretability and accuracy for many real-world tabular datasets. The key insight is that each feature's effect is modelled independently as a non-linear function, allowing complex patterns while maintaining interpretability.

GAM Formulation:

$g(E[Y]) = \beta_0 + f_1(x_1) + f_2(x_2) + \ldots + f_m(x_m)$ f_j are smooth, non-parametric spline functions of individual features g() is the link function (identity for regression, logit for classification) Each

```
f_j can be visualised as a 1D plot — the complete model is fully visualised
```

Neural Additive Model (NAM) — PyTorch:

```
import torch from nam.model import NAM, get_nam_args  args = get_nam_args() model = NAM(config=args, name='HiringNAM', num_inputs=X_train.shape[1],num_units=[64, 64, 32])
# Each feature network is a small neural net # The output is the sum of all feature network outputs + bias # RESULT: Full accuracy of deep learning + GAM interpretability
```

8.4 Interpretable AI Design Checklist

Design Stage	Interpretability Requirement
Problem Framing	Define who needs to understand the model (user, regulator, auditor, affected individual) and what level of explanation each needs
Data Strategy	Audit training data for historical bias; document provenance; record which protected attributes are present or inferable
Model Selection	Start with the simplest interpretable model that meets accuracy requirements; only escalate to complex models if accuracy gap is significant
Feature Engineering	Prefer semantically meaningful features; document feature definitions; identify and evaluate proxy features for protected attributes
Training Objective	Consider adding interpretability constraints (e.g., monotonicity, sparsity regularisation, fairness constraints) to the loss function
Validation	Run full XAI audit: global (SHAP summary, PDP), local (per-instance LIME/SHAP), and fairness (MetricFrame by group)
Documentation	Produce a Model Card (Mitchell et al. 2019) documenting intended use, training data, evaluation metrics, fairness analysis, and known limitations
Deployment	Implement real-time explanation generation for high-stakes predictions; log explanations alongside predictions
Monitoring	Track explanation drift (top feature rankings); alert if protected attribute SHAP values increase; conduct quarterly fairness audits

8.5 The Rashomon Set and Multiplicity of Explanations

The Rashomon Set (Breiman 2001, formalised by Rudin 2019) is the set of models that perform approximately equally well on the same dataset. A key insight is that for most real-world datasets, there exist many near-optimal models with very different feature importances — meaning that model explanations are not unique properties of the data, but reflect the specific model chosen.

- Implication 1 — Design Choice: If the data supports many good models, prefer the one that is most interpretable (e.g., a sparse linear model or a decision tree) over the one that is marginally more accurate
- Implication 2 — Explanation Instability: Two practitioners training on identical data may produce models with completely different SHAP importances; this is a known failure mode of post-hoc XAI
- Implication 3 — Competing Explanations: Adversarial actors can use the Rashomon Set to game interpretability audits by selecting whichever model in the set produces the most favourable-looking explanation

✅ Rudin's Principle: 'We should not explain black-box models. We should build interpretable models instead.' For high-stakes decisions (criminal justice, healthcare, lending), the ML community increasingly agrees that truly interpretable models are preferable to complex models with post-hoc explanations.

8.6 Explanation User Interfaces — Design Principles

The best XAI technique is useless if its output is not communicated effectively to the intended audience. Human-centred design of explanation interfaces is essential.

Audience	Appropriate Explanation Format
End User (affected by decision)	Simple natural language: 'Your application was declined because your debt-to-income ratio is above our threshold. Reducing it below 40% would likely change the decision.'
Domain Expert (doctor, judge)	Feature attribution bar chart with confidence intervals; counterfactual: 'If X had been Y, the prediction would change from A to B'
Data Scientist / ML Engineer	Full SHAP waterfall plot; interaction effects; LIME stability report; feature importance ranking with uncertainty
Regulator / Auditor	Model card; full fairness audit report (AIF360); global SHAP summary; test-set performance by demographic group
Executive / Decision-Maker	Dashboard with 3-5 key metrics; simple visual showing which features drive outcomes; trend over time

8.7 Key References and Further Reading

Reference	Contribution
Chouldechova (2017). Fair Prediction with Disparate Impact. BDIA.	Impossibility theorem: calibration + equal FPR/FNR cannot coexist at different base rates
Hardt, Price & Srebro (2016). Equality of Opportunity. NeurIPS.	Formal definition and algorithm for Equal Opportunity and Equalized Odds
Buolamwini & Gebru (2018). Gender Shades. FAccT.	Systematic audit of facial recognition APIs showing 35%+ accuracy gap by race/gender
Rudin (2019). Stop Explaining Black Box ML Models. Nature MI.	Argues for inherently interpretable models in high-stakes decisions over post-hoc XAI
Bellamy et al. (2019). AI Fairness 360. IBM Journal R&D.	AIF360: comprehensive open-source fairness toolkit with 70+ metrics and 10+ algorithms
McMahan et al. (2017). Communication-Efficient Learning. AISTATS.	FedAvg: foundational federated learning algorithm
Agarwal et al. (2018). A Reductions Approach to Fairness. ICML.	Exponentiated Gradient: Fairlearn's core in-processing algorithm
Lundberg et al. (2020). From Local Explanations to Global Understanding with Explainable AI for Trees. Nature MI.	TreeSHAP + global SHAP interaction values
Mothilal, Sharma & Tan (2020). DiCE. FAccT.	Diverse Counterfactual Explanations for actionable recourse
Molnar (2022). Interpretable Machine Learning. Online textbook.	Comprehensive open-access reference for all XAI and fairness techniques

***

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING
ANNAMACHARYA INSTITUTE OF TECHNOLOGY AND SCIENCES::RAJAMPET
(AUTONOMOUS)
III B.Tech II Sem AIML (A&B)
EXPLAINABLE AI &MODEL INTERPRETABILITY
Unit-1 ASSIGNMENT QUESTIONS

****

Answer for any 6 of the following.

1. Define the terms *Explainability* and *Interpretability* in the context of Artificial Intelligence. List any four properties of Explainability and scope of Interpretability.
2. Explain why XAI is important in high-stakes domains such as Healthcare, Finance, and Law. Illustrate your answer with one example from each domain.
3. Differentiate between White-box models and Black-box models. Compare them with respect to transparency, accuracy, interpretability, and real-world usability.
4. Explain Taxonomy of XAI Techniques.
5. Given a healthcare diagnostic model, explain how local and global explanation techniques can be used to improve doctors' trust in the system.
6. Analyze the trade-off between Model Simplicity and Predictive Power. Why are simpler models often preferred in regulated domains despite lower accuracy?
7. Compare Post-hoc and Intrinsic explainability approaches. Identify their strengths, weaknesses, and suitable use cases.
8. Evaluate the role of Fairness, Accountability, and Transparency (FAT) as desiderata for XAI. Which of these is most critical in legal decision-making systems? Justify your answer.
9. Assess how regulations such as GDPR and the AI Bill of Rights influence the design and deployment of machine learning models. What challenges do organizations face in complying with these regulations?
10. Design a human-centered AI system for a healthcare or financial application that incorporates XAI principles.
11. Explain the concept of Human-Centered AI. How does it differ from traditional AI system design?
12. Consider a credit-scoring model that shows bias against a particular demographic group. How would you use XAI methods to detect and mitigate this bias.

Submit On 28-01-26(For AIML-B), 29-01-26(For AIML-A)

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING
ANNAMACHARYA INSTITUTE OF TECHNOLOGY AND SCIENCES::RAJAMPET
(AUTONOMOUS)
III B.Tech II Sem AIML (A&B)
EXPLAINABLE AI &MODEL INTERPRETABILITY
Unit-2 ASSIGNMENT QUESTIONS

****

Answer for any 6 of the following.

1. Explain Decision Trees and Rule-Based Models in detail, including their construction, interpretation, advantages, and limitations.
2. Describe how Decision Trees support model interpretability and explain decision paths with an example.
3. Explain how Rule-Based Models are used for credit scoring and discuss their transparency and practical challenges.
4. Explain Linear Models and describe how feature importance is derived from model coefficients.
5. Discuss methods for visualization of weights and coefficients in Linear and Logistic Regression models.
6. Explain Logistic Regression coefficient interpretation using log-odds and odds ratios with an example.
7. Describe Generalized Additive Models (GAMs) and explain how they provide interpretable nonlinear modeling.
8. Develop a case study of credit scoring using transparent models and explain the modeling approach.
9. Compare Decision Trees, Rule-Based Models, Linear Models, and GAMs in terms of interpretability and performance.
10. Discuss the use cases and trade-offs of interpretable machine learning models in real-world applications.

Submit On 07-02-26(For AI&ML (A&B))

EXPLAINABLE AI &MODEL INTERPRETABILITY

Unit-III Assignment Questions

Answer any 5 of the following.

1. Explain LIME (Local Interpretable Model-agnostic Explanations) in detail. Describe how it works with a suitable example.
2. What are SHAP values? Explain their concept and how they help in interpreting machine learning models.
3. Describe Partial Dependence Plots (PDPs). How do they help in understanding feature impact on model predictions?
4. Explain Individual Conditional Expectation (ICE) plots. How are they different from PDPs?
5. What are Anchors in explainable AI? Explain their working with a simple example.
6. Define counterfactual explanations. Explain their importance with a real-world example.
7. What is permutation feature importance? Explain the steps involved in calculating it.
8. Explain feature interaction in machine learning. How can it be identified in models?
9. Compare SHAP and LIME based on their approach, advantages, and limitations.
10. Explain how explainable AI (XAI) techniques are used for model debugging with an example.

EXPLAINABLE AI &MODEL INTERPRETABILITY

Unit-III Assignment Questions

Answer any 5 of the following.

1. Explain LIME (Local Interpretable Model-agnostic Explanations) in detail. Describe how it works with a suitable example.
2. What are SHAP values? Explain their concept and how they help in interpreting machine learning models.
3. Describe Partial Dependence Plots (PDPs). How do they help in understanding feature impact on model predictions?
4. Explain Individual Conditional Expectation (ICE) plots. How are they different from PDPs?
5. What are Anchors in explainable AI? Explain their working with a simple example.
6. Define counterfactual explanations. Explain their importance with a real-world example.
7. What is permutation feature importance? Explain the steps involved in calculating it.
8. Explain feature interaction in machine learning. How can it be identified in models?
9. Compare SHAP and LIME based on their approach, advantages, and limitations.
10. Explain how explainable AI (XAI) techniques are used for model debugging with an example.

EXPLAINABLE AI &MODEL INTERPRETABILITY

Unit-V Assignment Questions

Answer any 4 of the following.

1. Explain Demographic Parity as a fairness metric. Discuss its advantages and limitations with an example.
2. What is Equal Opportunity in machine learning fairness? Compare it with Demographic Parity.
3. Describe the common sources of bias in data and machine learning models. Explain with real-world examples.
4. Explain different techniques used to detect and mitigate discrimination in machine learning models.
5. What is AI Fairness 360 (AIF360)? Explain its features and how it helps in building fair models.
6. Describe the What-If Tool and its role in analyzing fairness and model behaviour.
7. What is Fairlearn? Explain how it supports fairness assessment and mitigation.
8. Discuss a case study of bias in hiring algorithms. What were the causes and how can such bias be reduced?
9. Explain the role of explainability in ML pipelines. How do tools like MLflow and Skater support this?
10. Explain how interpretable AI systems can be designed from scratch.
11. Discuss challenges in federated and privacy-preserving AI.