Lecture 19:

Model Explainability - XAI Fundamentals and Traditional Methods

Ho-min Park

homin.park@ghent.ac.kr

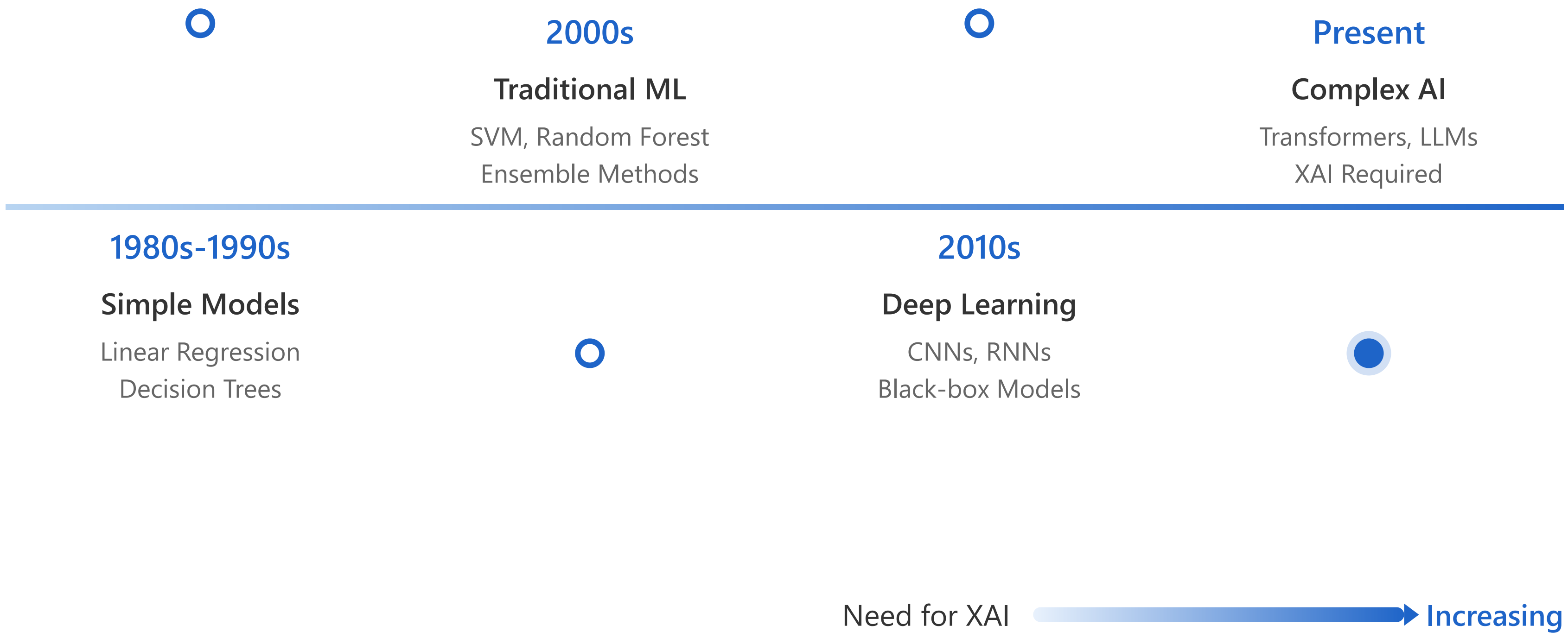
powersimmani@gmail.com

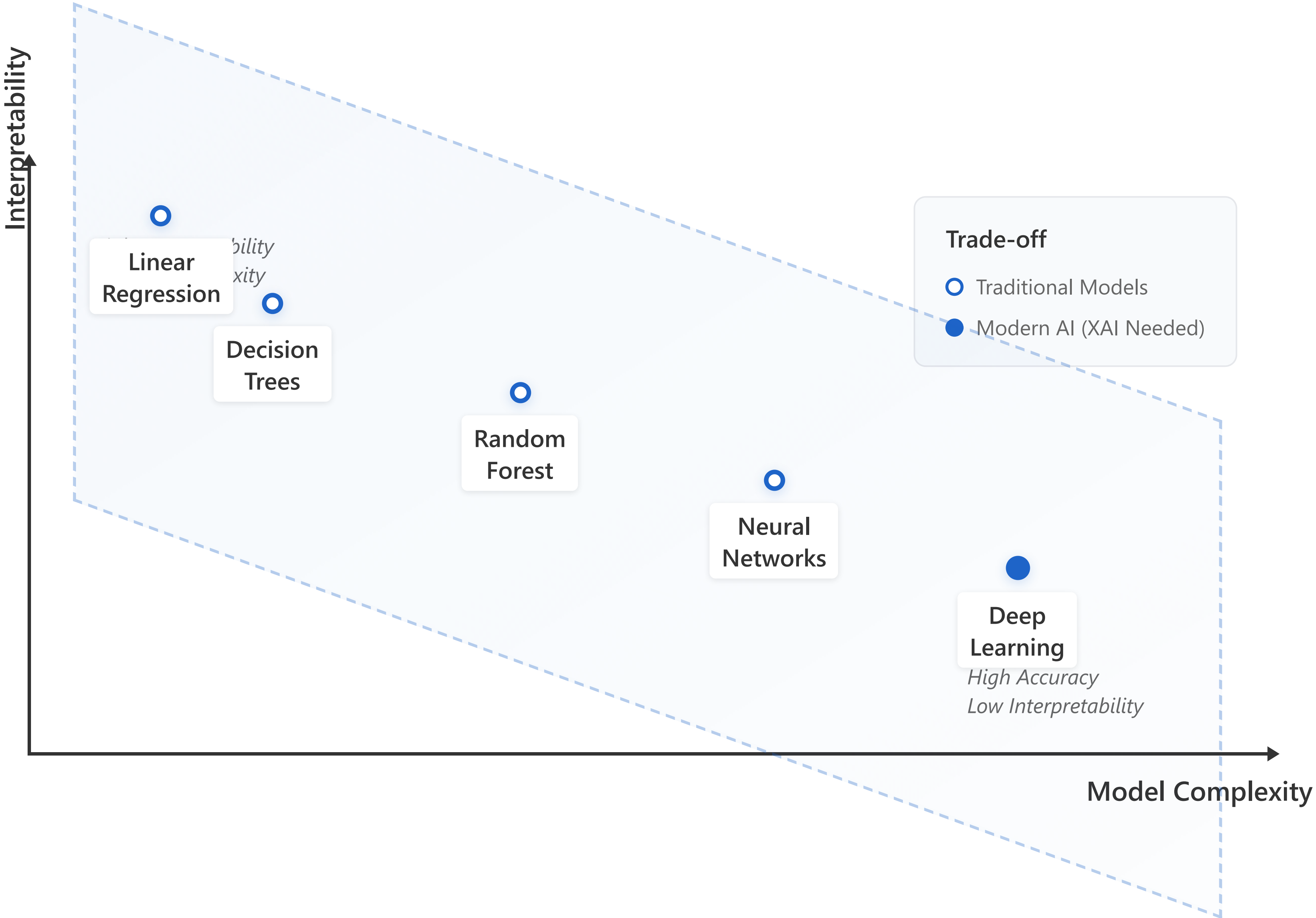
Lecture Contents

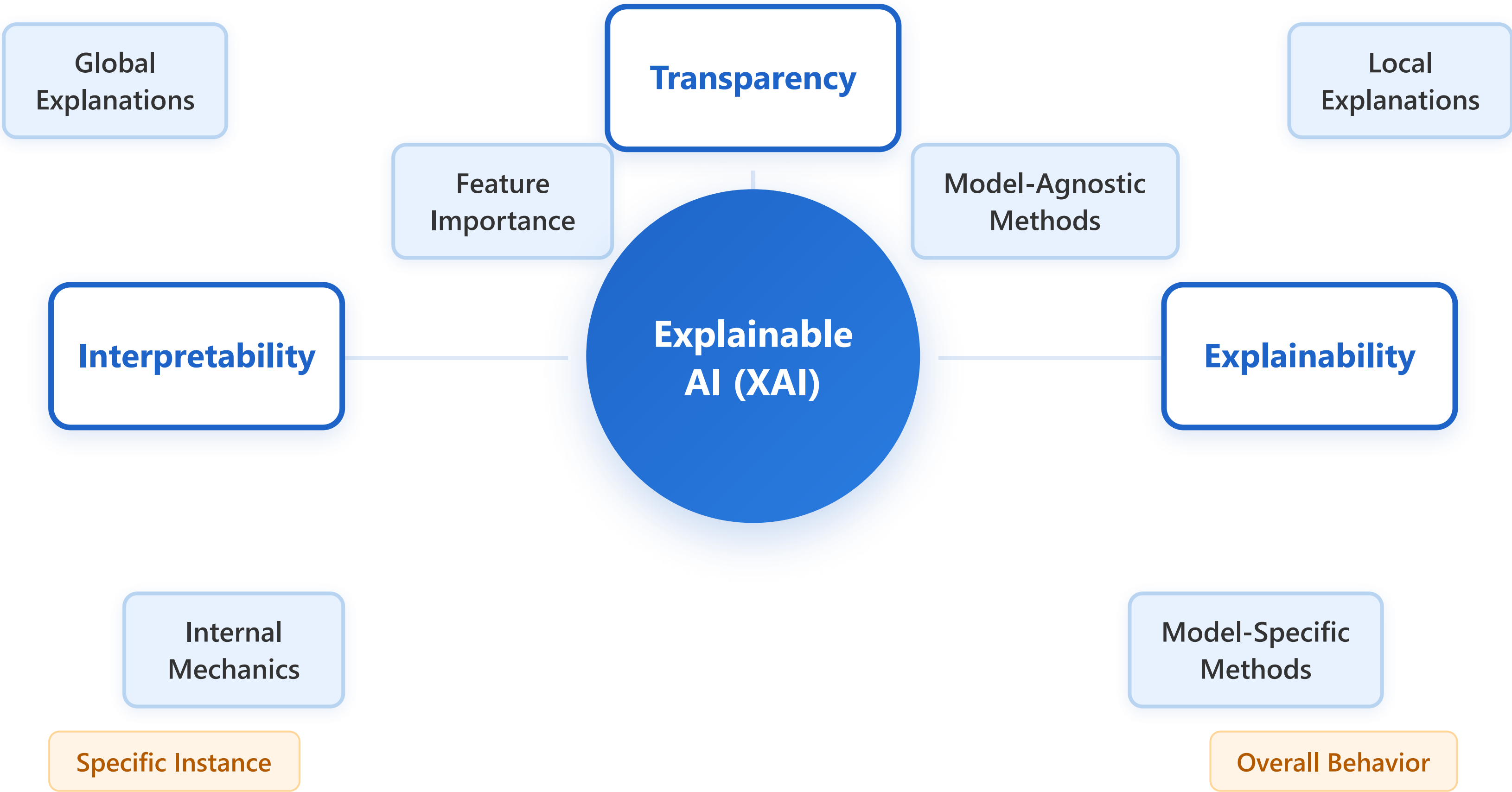
Part 1/4:

Introduction to XAI and Its Importance

1. Course Introduction - The Era of Explainable AI
2. Model Complexity vs Interpretability Trade-off
3. Core Concepts and Terminology in XAI
4. Why Explainability is Necessary
5. XAI Classification Framework
6. XAI Applications by Industry
7. XAI Evaluation Criteria


2000s
Traditional ML
SVM, Random Forest
Ensemble Methods
Present
Complex AI
Transformers, LLMs
XAI Required
1980s-1990s
Simple Models
Linear Regression
Decision Trees
2010s
Deep Learning
CNNs, RNNs
Black-box Models
Need for XAI
Increasing


Interpretability
Model Complexity
Linear Regression
Decision Trees
Random Forest
Neural Networks
Deep Learning
High Accuracy
Low Interpretability
Trade-off
Traditional Models
Modern AI (XAI Needed)



Global Explanations
Transparency
Local Explanations
Feature Importance
Model-Agnostic Methods
Interpretability
Explainable AI (XAI)
Explainability
Internal Mechanics
Model-Specific Methods
Specific Instance
Overall Behavior

Why Explainability is Necessary

Trust
Users need to understand AI decisions before accepting them

Accountability
Identifying responsibility when AI makes errors

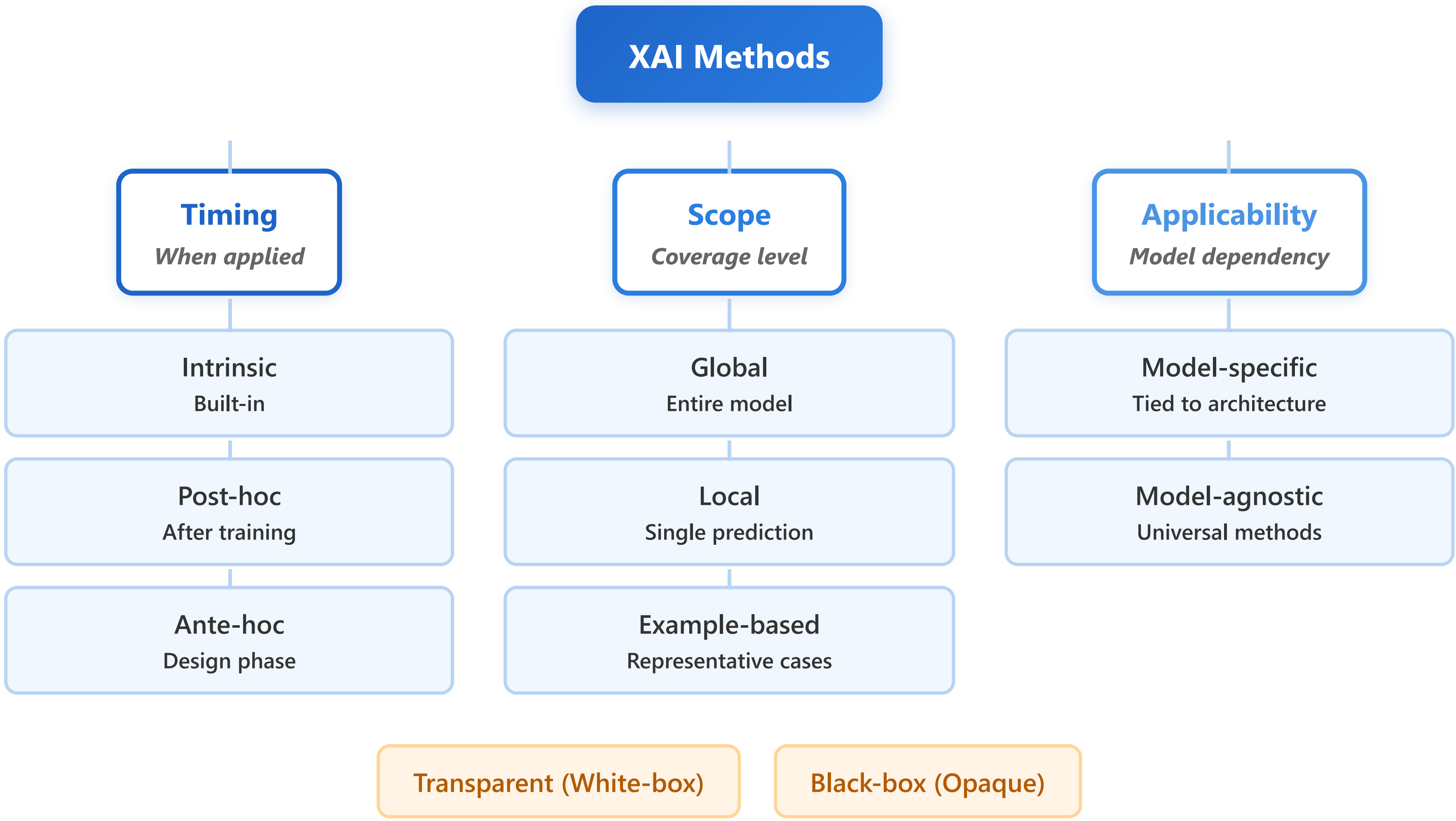
Debugging
Finding and fixing model biases or data issues

Regulatory Compliance
Legal requirements for automated decision systems

Fairness
Detecting and mitigating discrimination in AI systems

Scientific Discovery
Learning new insights from model patterns

Safety
Ensuring AI behaves as intended in critical applications



XAI Methods
Timing
When applied
Scope
Coverage level
Applicability
Model dependency
Intrinsic
Built-in
Post-hoc
After training
Ante-hoc
Design phase
Global
Entire model
Local
Single prediction
Example-based
Representative cases
Model-specific
Tied to architecture
Model-agnostic
Universal methods
Transparent (White-box)
Black-box (Opaque)

XAI Applications by Industry

Healthcare

Diagnosis predictions, treatment recommendations

Finance

Credit scoring, fraud detection reasoning

Autonomous Vehicles

Decision transparency for safety certification

Criminal Justice

Risk assessment for parole decisions

Human Resources

Fair hiring practices, bias detection

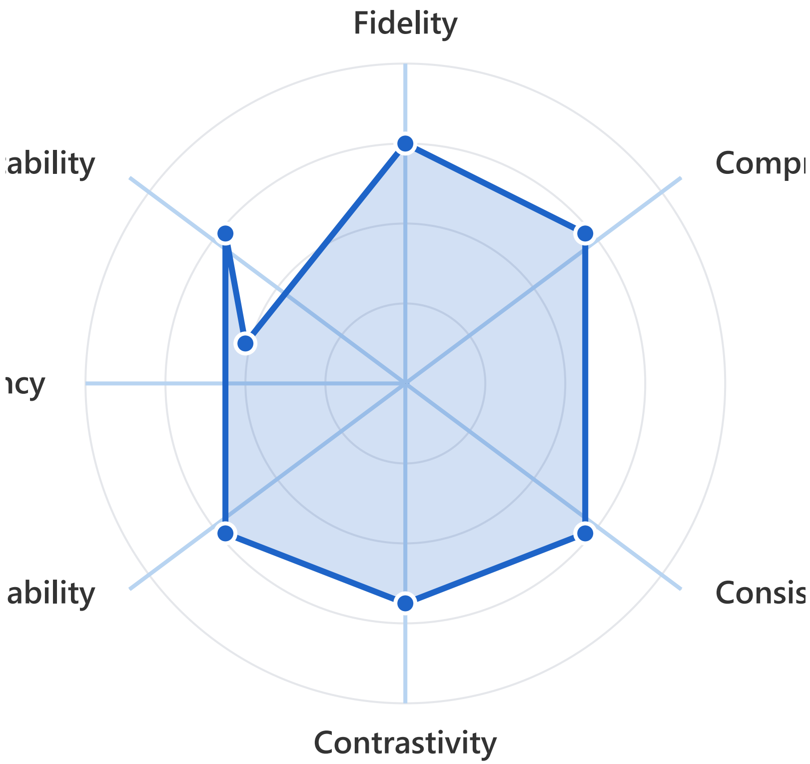
E-commerce

Recommendation system transparency

Manufacturing

Predictive maintenance, quality

XAI Evaluation Dimensions



1 Fidelity
How accurately explanations reflect actual model behavior

2 Comprehensibility
Human ability to understand explanations

3 Consistency
Similar instances receive similar explanations

4 Contrastivity
Explaining why this prediction vs alternatives

5 Actionability
Ability to use explanations for decisions

6 Stability
Robustness to small input perturbations

7 Efficiency
Computational cost of generating explanations

Part 2/4:

Intrinsically Interpretable Models

8. Interpreting Linear Models
9. Transparency of Decision Trees
10. Generalized Additive Models (GAM)
11. Rule-Based Models
12. Monotonic Constraint Models
13. Sparse Linear Models
14. Hands-on: Building Interpretable Models with scikit-learn

Linear Model Interpretation

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

β_0	β_i	x_i
Intercept	Coefficients	Features

Coefficient Magnitude

Indicates feature importance

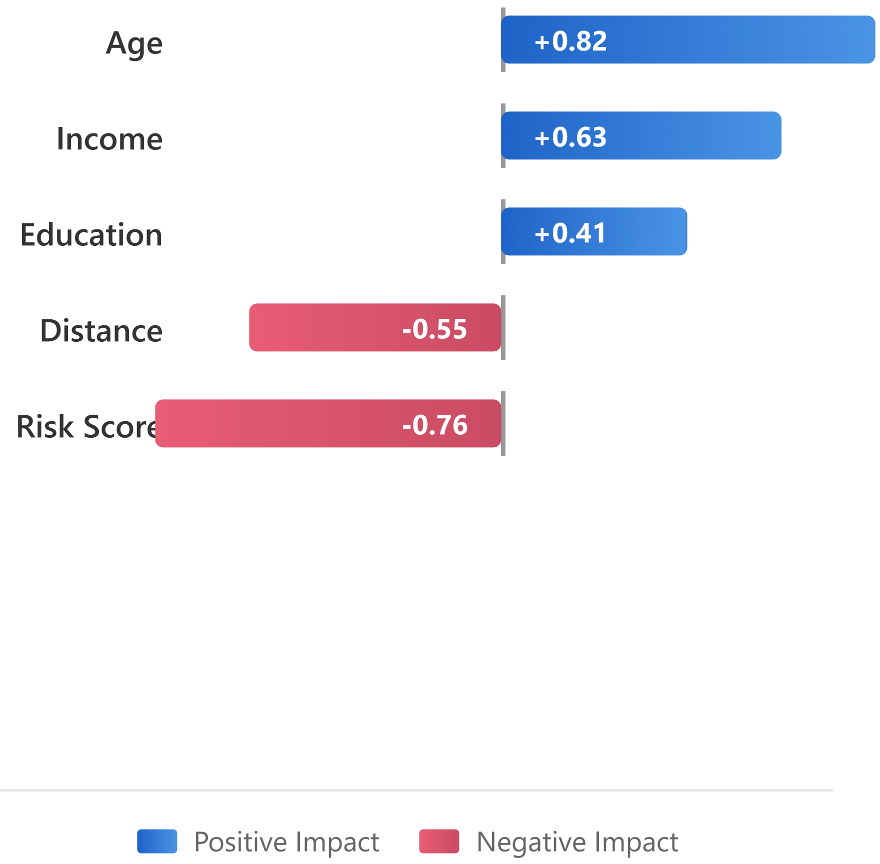
Sign Direction

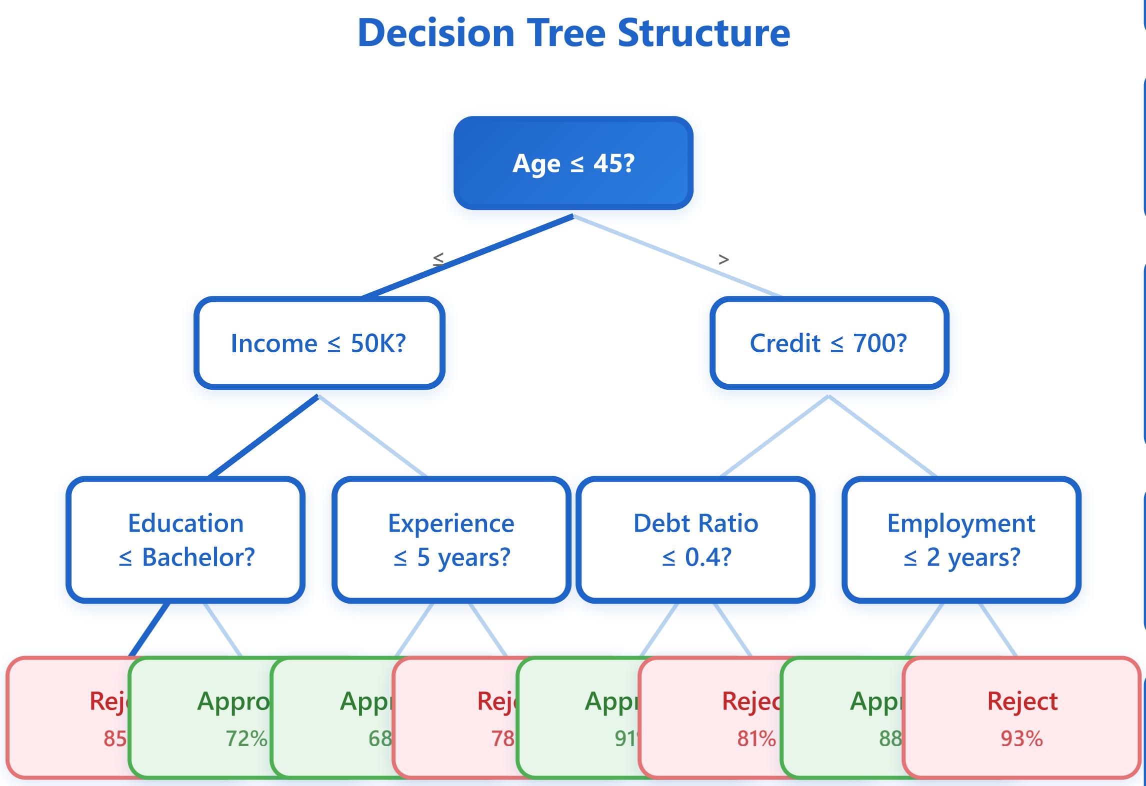
Shows positive/negative relationship

Regularization

L1 (Lasso) for selection, L2 (Ridge) for stability

Feature Coefficients





If-Then Rules

Binary splits create human-readable decision paths

Complete Path

Full decision logic visible from root to leaf

Feature Importance

Determined by split frequency and information gain

Natural Interactions

Hierarchical splits handle feature interactions

Trade-offs

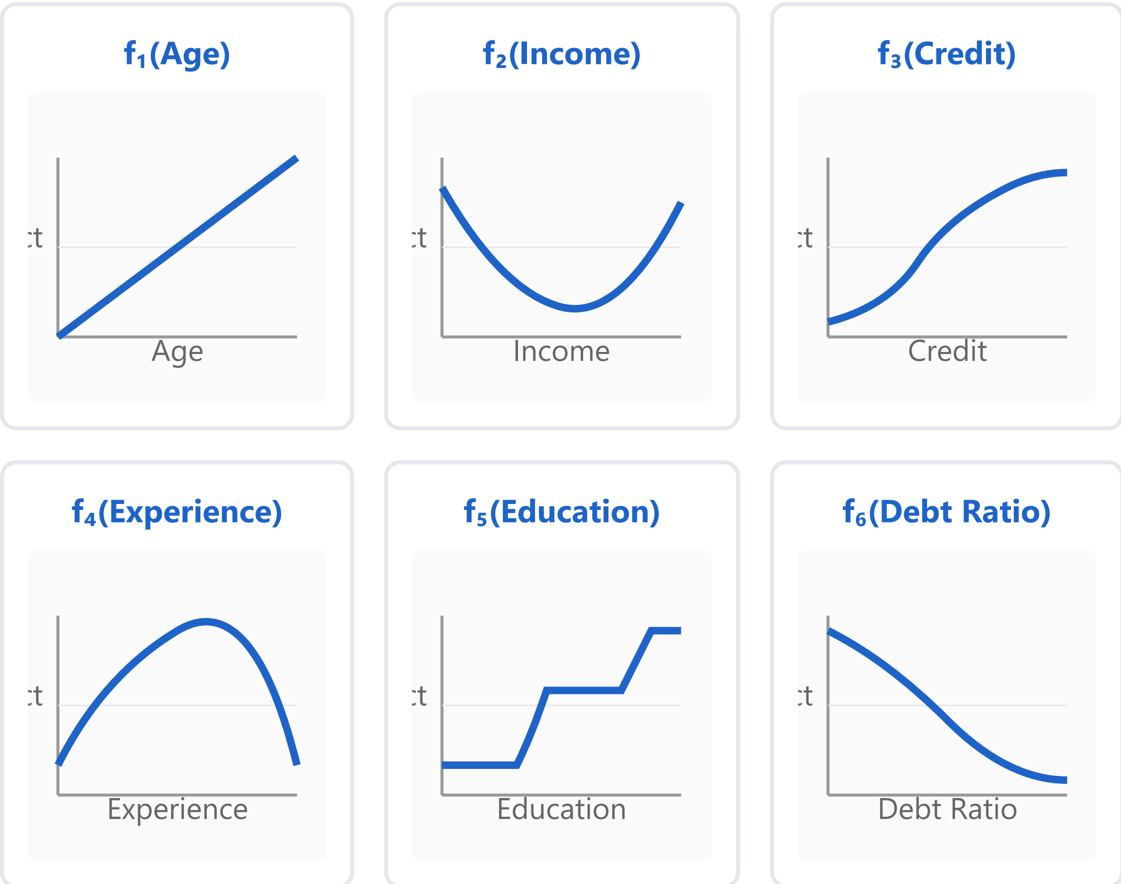
Deep trees lose interpretability; pruning balances accuracy

Decision Node
Approve Outcome
Reject Outcome

Generalized Additive Models (GAM)

$$y = \beta_0 + f_1(x_1) + f_2(x_2) + f_3(x_3) + \ldots$$

where f_i are smooth shape functions



Key Features

Independent
Each feature contributes separately

Non-linear
Capture complex patterns via smooth functions

Interpretable
Visualize individual feature effects

Separable
Effects remain isolated and analyzable

Balanced
Between linear and black-box models

Rule-Based Decision System

1 RULE

IF `Age > 30` **AND** `Income > 50K`

THEN Decision = **APPROVE**

Coverage: 85% Confidence: 92%

2 RULE

IF `Credit Score > 700` **AND** `Debt Ratio < 0.3`

THEN Decision = **APPROVE**

Coverage: 72% Confidence: 88%

3 RULE

IF `Age < 25` **AND** `Employment < 1 year`

Key Concepts

IF-THEN Structure

Rules extracted from data patterns with logical conditions

Coverage

Number of instances satisfying rule conditions

Confidence

Prediction accuracy for instances matching rule

Interpretability

Each prediction explained by triggering rules

Trade-offs

Balancing concise rule sets with high coverage and accuracy

Metrics Guide

THEN Decision = **REJECT**

Coverage: % of data covered

Confidence: % prediction accuracy

Monotonic Constraint Models

Enforcing Logical Relationships in Predictions

⚠ Non-Monotonic Model

❌ Counter-intuitive: Higher score

✓ Monotonic Model

✓ Logical: Higher credit score

Key Features

Logical Constraints
Enforces domain knowledge relationships

Trust Building
Predictions follow expected behavior patterns

Implementation
XGBoost, LightGBM, monotonic NNs

Trade-offs
May slightly reduce accuracy for interpretability

Critical For:
- Regulated industries
- Financial decisions

sometimes decreases approval

consistently increases approval

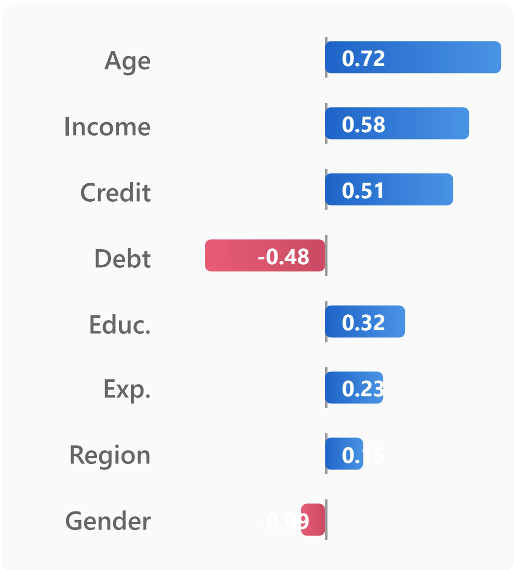
• Healthcare applications
• High-stakes predictions

Sparse Linear Models

L1 Regularization for Feature Selection

Dense Model

Standard Linear Regression

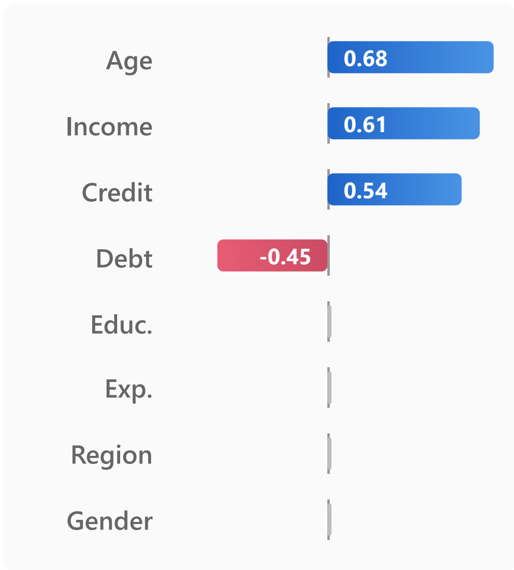


Non-zero
8

Features
8/8

Sparse Model

Lasso (L1 Regularization)



Non-zero
4

Features
4/8

Benefits

Auto Selection
Only relevant features retained

Interpretability
Reduced feature count

Implementation
Easier to deploy and debug

High-Dim
Valuable when p >> n

Regularization

L1 (Lasso): Sparsity

L2 (Ridge): Stability

Elastic Net: Balanced

Building Interpretable Models with scikit-learn

Hands-on Workflow & Code Examples

Implementation Steps

1 **Import & Load Data**
Load dataset and dependencies

↓

2 **Preprocessing**
Feature scaling, categorical encoding

↓

3 **Train Models**
Linear, Tree-based classifiers

↓

Code Examples

📊 Linear Model

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
model.fit(X_train, y_train)

# Access coefficients
coef = model.coef_
print(f"Coefficients: {coef}")
```

4 **Extract Insights**
Coefficients, feature importance

5 **Visualize & Compare**
Accuracy vs interpretability

🌲 Decision Tree

```
from sklearn.tree import DecisionTreeClassifier

tree = DecisionTreeClassifier(max_depth=3)
tree.fit(X_train, y_train)

# Feature importance
importance = tree.feature_importances_
```

🔍 Inspection Tools

```
from sklearn.inspection import PartialDependenceDisplay

# Visualize partial dependence
PartialDependenceDisplay.from_estimator(
    model, X, features=[0, 1]
)
```

Key sklearn Modules

`linear_model`	`tree`
`inspection`	`preprocessing`

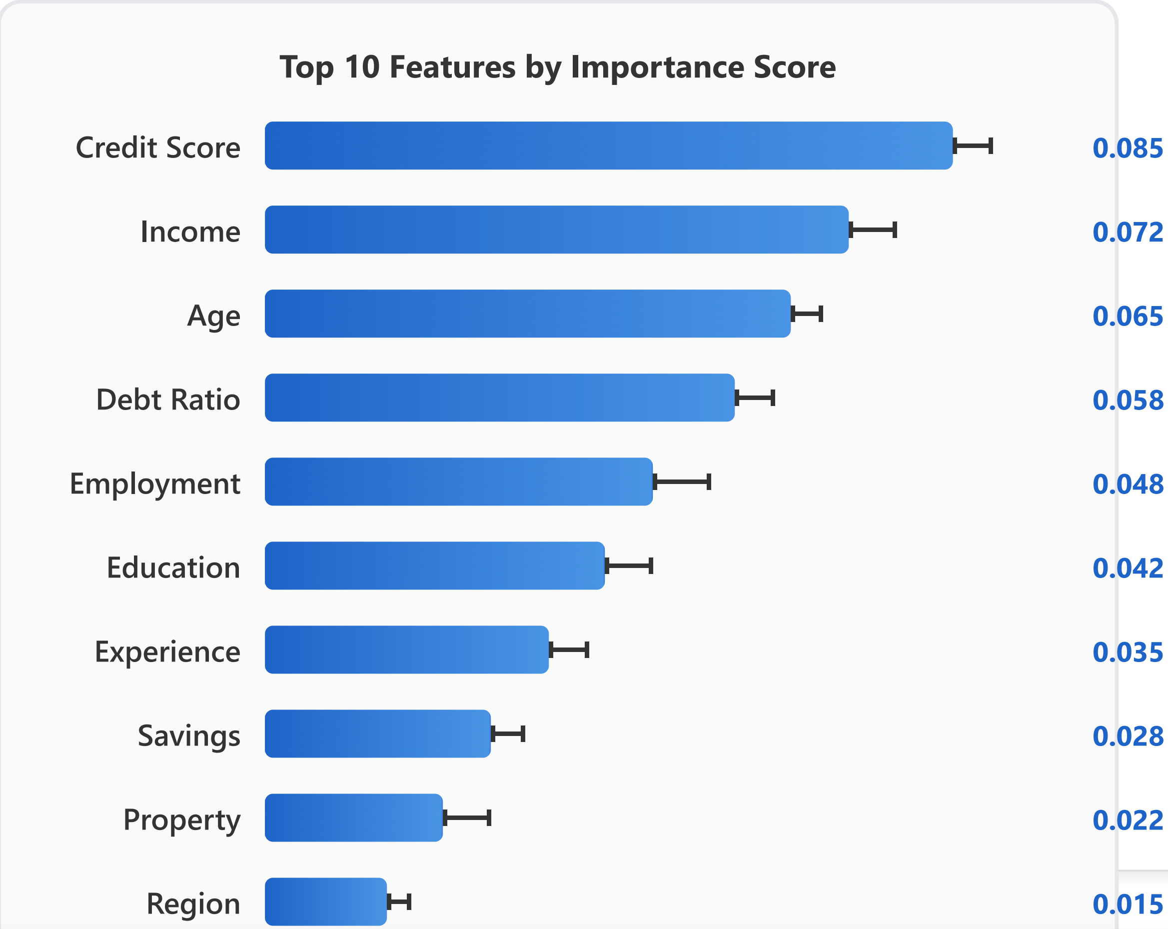
Part 3/4:

Feature Importance Methodologies

15. Permutation Importance
16. Drop-Column Importance
17. Partial Dependence Plots (PDP)
18. Individual Conditional Expectation (ICE)
19. Accumulated Local Effects (ALE)
20. Feature Interaction Analysis

Permutation Importance

Feature Importance via Random Shuffling



How It Works

Process

1. Measure baseline performance
2. Shuffle feature values randomly
3. Re-evaluate model performance
4. Importance = Performance drop

Model-Agnostic

Works with any trained model

Interactions

Captures feature contributions including interactions

Implementation

sklearn.inspection.permutation_importance()

✓ Advantages

Simple, reliable, no assumptions

⚠ Limitations

Correlated features may deflate

Permutation Importance: Detailed Algorithm (Example)

1 Calculate Baseline Score

Evaluate the trained model on validation dataset to establish baseline performance metric

2 Select Feature to Test

Choose one feature column from the dataset to permute

```
baseline_score = model.score(X_val, y_val) → baseline_score = 0.850 (R² score)
```

```
feature_to_test = 'Credit Score' # Testing most important feature first
```

3 Randomly Shuffle Feature

Randomly permute the values of selected feature, breaking its relationship with target

```
X_permuted = X_val.copy() X_permuted['Credit Score'] = shuffle([750, 680, 820, ...]) → [820, 750, 680, ...] # Randomly shuffled
```

4 Re-evaluate Performance

Calculate new performance score with shuffled feature values

```
permuted_score = model.score(X_permuted, y_val) → permuted_score = 0.765 (R² dropped!)
```

5 Calculate Importance

Feature importance = Performance drop caused by shuffling

```
importance = baseline_score - permuted_score importance = 0.850 - 0.765 = 0.085 ✓
```

6 Repeat for Stability

Repeat shuffling multiple times and average for stable estimate

```
importances = [0.085, 0.082, 0.088, 0.084, 0.086] mean_importance = 0.085 ± 0.002
```

7 Iterate All Features

Repeat process for every feature to get complete ranking

```
Credit Score: 0.085 Income: 0.072 Age: 0.065 Debt Ratio: 0.058 ...
```

8 Rank & Visualize

Sort features by importance and create visualization with error bars

```
sorted_features = sort_by_importance() → See bar chart above! 📊
```

Drop-Column Importance

Feature Removal Impact Analysis

Importance Score Comparison

Feature	Permutation	Drop-Column	Δ Diff
Credit Score	0.085	0.092	+0.007
Income	0.072	0.078	+0.006
Age	0.065	0.048	-0.017
Debt Ratio	0.058	0.061	+0.003
Employment	0.048	0.032	-0.016
Education	0.042	0.044	+0.002
Experience	0.035	0.036	+0.001
Savings	0.028	0.028	0.000

Method

Process

1. Train with all features
2. Remove one feature
3. Retrain model
4. Measure performance drop

Accuracy
More accurate than permutation

Cost
Requires n retrainings for n features

Correlations
Handles feature correlations better

Use Case
Final feature selection decisions

Key Insight

Fast	Accurate

Highlighted rows show redundant features

📊 Calculation Example

BASELINE All Features Model

Credit Score | Income | Age | Debt Ratio

Accuracy **85.0%**

→

STEP 2 Remove "Age" Feature

Credit Score | Income | ~~Age~~ ✕ | Debt Ratio

New Accuracy **80.2%**

→

RESULT Importance Score

Drop-Column Importance = 85.0% - 80.2%

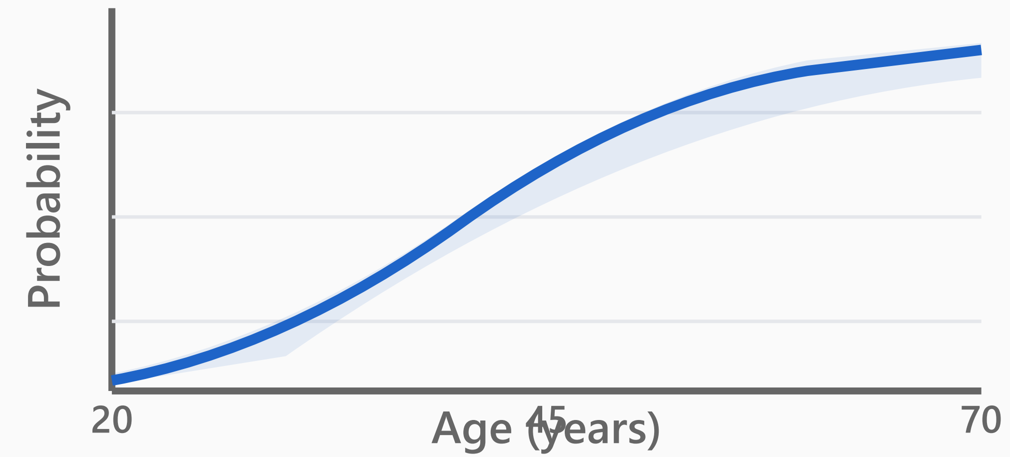
Age Importance: **4.8%**

💡 **Interpretation:** Removing "Age" causes a 4.8% drop in accuracy, indicating it's a moderately important feature. Compare this across all features to identify which ones are truly essential for your model. Features showing lower importance in Drop-Column than Permutation (like Age: -0.017) might be redundant when other correlated features are present.

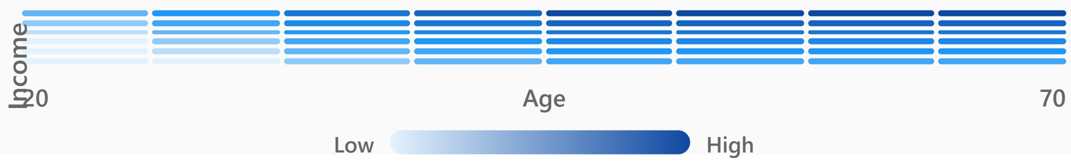
Partial Dependence Plots (PDP)

Marginal Effect Visualization

1D PDP: Age vs Loan Approval Probability



2D PDP: Age × Income Interaction



Key Concepts

Marginal Effect
Average prediction as feature varies

1D PDP
Single feature effect (line plot)

2D PDP
Two-feature interactions (heatmap)

Implementation
sklearn.inspection.PartialDependenceDisplay

⚠ Assumption
Features are independent - misleading if violated

Reveals

- Non-linear relationships
- Thresholds & transitions
- Average behavior patterns

How PDP is Calculated

Step-by-Step Mathematical Process

📐 PDP Formula

$$PDP_{xs}(x_s) = E_{X_c}[f(x_s, X_c)] = (1/n)\ \Sigma_{i=1}^{n} f(x_s, x_c^{(i)})$$

x_s: Target feature value (fixed) | **X_c**: All other features (vary across data) | **f**: Model prediction function | **n**: Number of data points

1

Original Data

Age	Inc	ŷ
25	40K	0.3
35	60K	0.7
45	80K	0.9
55	50K	0.6

Start with training dataset containing all features and their predictions

2

Fix Target Feature

Age	Inc	ŷ
30	40K	?
30	60K	?
30	80K	?
30	50K	?

Replace target feature (Age) with specific value (e.g., 30) for ALL samples

3

Predict & Average

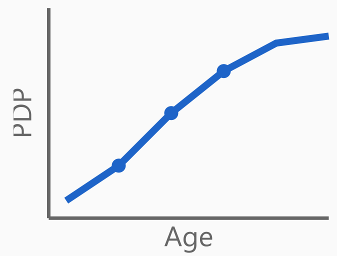
f(30, 40K) = 0.45
f(30, 60K) = 0.65
f(30, 80K) = 0.75
f(30, 50K) = 0.55

PDP(Age=30) = (0.45+0.65+0.75+0.55)/4 = 0.60

Get predictions for all modified samples, then compute their average

4

Repeat & Plot



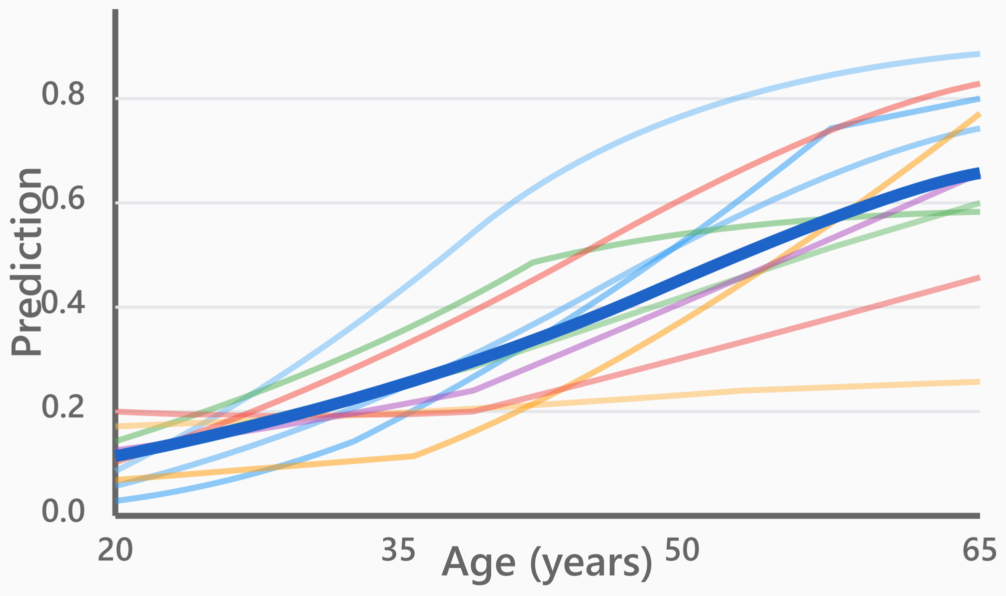
Repeat steps 2-3 for different Age values to create the complete PDP curve

💡 **Key Insight**

PDP marginalizes out other features by averaging predictions across their distribution. This shows the **average effect** of the target feature, assuming feature independence.

Individual Conditional Expectation (ICE)

Instance-Level Feature Effects



Key Concepts

ICE vs PDP

ICE
Individual instance trajectories

PDP
Average of all ICE curves

Heterogeneity

Reveals variation in feature effects across instances

Subgroups

Identifies different feature-outcome relationships

Interactions

Detects interactions PDP may hide

Non-parallel Lines

Shows unexpected patterns & heterogeneity

Centered ICE

Subtract baseline for better comparison

What to Look For

- Parallel lines = homogeneous effect
- Diverging lines = heterogeneous effect

Individual ICE Curves — PDP (Average)

- Clusters = distinct subgroups

How ICE Works: Step-by-Step Process

1 Select Instance

Choose a single data point from your dataset

x_1 = (age=30, income=50k, ...)

→

2 Vary Feature

Change target feature across its range, keep others fixed

age: 20, 25, 30, ..., 65
income=50k (fixed)

→

3 Get Predictions

Run model for each feature value

$\hat{y}(20)$, $\hat{y}(25)$, $\hat{y}(30)$, ..., $\hat{y}(65)$

→

4 Plot & Repeat

Draw curve for this instance, repeat for all instances

n curves for n instances
Average = PDP

Accumulated Local Effects (ALE)

Unbiased Feature Effects with Correlated Features

⚠ **PDP (Biased)**

✓ **ALE (Unbiased)**

Key Features

ALE Advantages

- ✓ Unbiased with correlated features
- ✓ Local neighborhoods only
- ✓ Computationally efficient
- ✓ No marginalization needed

Local Effects

Accumulated across feature range

Reliability

More trustworthy for realistic datasets

Implementation

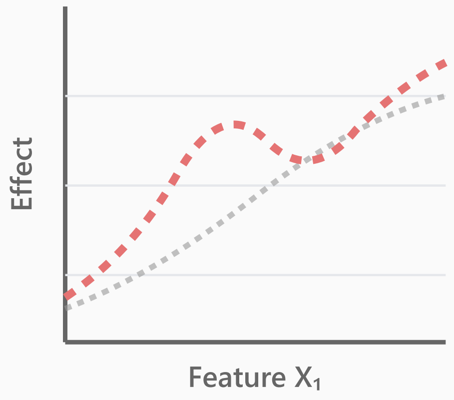
ALEPython package

Quick Comparison

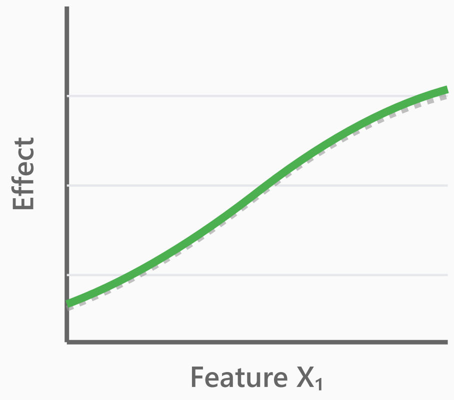
	PDP	ALE
Correlated	Biased	Unbiased
Speed	Slow	Fast

- PDP (Biased)
- ALE (Unbiased)
- True Effect

How



ntervals. E
ally exists.



Differences

edictions when the feature value
nterval. This represents the "local

```
}) - f(z_j, X_{-j})
ts in each interval
```

ys i

cts

Average the local effects of all data points within each interval. This cancels out the effects of other features.

```
ALE_j = (1/n_j) Σ [f(z_{j+1}, X_i) - f(z_j, X_i)]
n_j: number of data points in interval j
```

Starting from the first interval, accumulate the local effects of each interval. This is what "Accumulated" in Accumulated Local Effects means.

```
ALE(z) = Σ(j=1 to k) ALE_j
Cumulative from left to right
```

5 Center the Plot

Center the entire ALE plot so the average is 0. This allows us to see the relative effect of the feature rather than its absolute effect.

```
ALE_centered(z) = ALE(z) - E[ALE(z)]
Interpretation: effect relative to average
```

6 Visualize & Interpret

The final ALE plot shows the pure effect on model predictions at each feature value. An upward slope indicates positive effect, while a downward slope indicates negative effect.

💡 Key Takeaway

ALE operates only within **local regions where actual data exists**, so it doesn't suffer from the bias caused by unrealistic data combinations like PDP does. Even when features are correlated, ALE measures pure effects while maintaining the natural distribution of other features by considering only small changes within each interval.

Feature Interaction Analysis

Pairwise Interaction Strength Matrix

H-Statistic Interaction Matrix

	Age	Income	Credit	Debt	Employ	Educ	Savings	Region
Age	0	0.65	0.23	0.18	0.31	0.12	0.15	0.08
Income	0.65	0	0.58	0.72	0.41	0.26	0.49	0.16
Credit	0.23	0.58	0	0.52	0.34	0.19	0.28	0.11
Debt	0.18	0.72	0.52	0	0.27	0.21	0.38	0.09
Employ	0.31	0.41	0.34	0.27	0	0.29	0.17	0.13
Educ	0.12	0.26	0.19	0.21	0.29	0	0.16	0.10
Savings	0.15	0.49	0.28	0.38	0.17	0.16	0	0.07
Region	0.08	0.16	0.11	0.09	0.13	0.10	0.07	0

Weak — Strong

Methods

Interaction Detection

- **H-statistic:** Quantify strength
- **2D PDP:** Visual effects
- **SHAP:** Pairwise values
- **PD-based:** Joint vs separate

Non-additive

Features work together synergistically

Understanding

Model behavior & debugging

Engineering

Guide interaction term creation

Complexity

$O(n^2)$ - quadratic with features

Interpretation Guide

- 0.0-0.3: Weak interaction
- 0.3-0.6: Moderate interaction
- 0.6-1.0: Strong interaction

Part 4/4:

Model-Agnostic Methods

21. Surrogate Models

22. Introduction to LIME

23. LIME Advanced Topics

24. Anchor Explanations

25. Practical Guidelines and Best Practices

Surrogate Models

Approximating Black-Box with Interpretable Models

Surrogate Model Process

BLACK BOX

Complex Model

Neural Network, Ensemble, etc.

Generate predictions

Collect Predictions on Data

Use complex model to predict all instances

Train surrogate

WHITE BOX

Surrogate Model

Decision Tree, Linear Model, Rules

Evaluate &

Key Concepts

Types

Global Surrogate

Mimic entire model behavior

Local Surrogate

Approximate specific region

Fidelity

How well surrogate matches original

Advantages

Leverage simple model interpretability

Limitations

May not capture full complexity

Use Cases

Compression, explanation, debugging

Common Surrogates

Interpret Surrogate

Extract insights and explanations

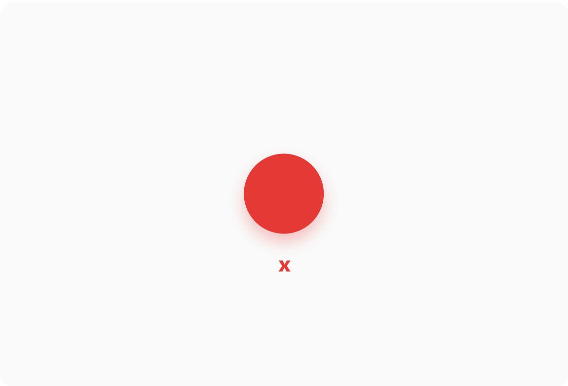
- Decision Trees
- Linear Models
- Rule Sets
- Generalized Additive Models

Introduction to LIME

Local Interpretable Model-agnostic Explanations

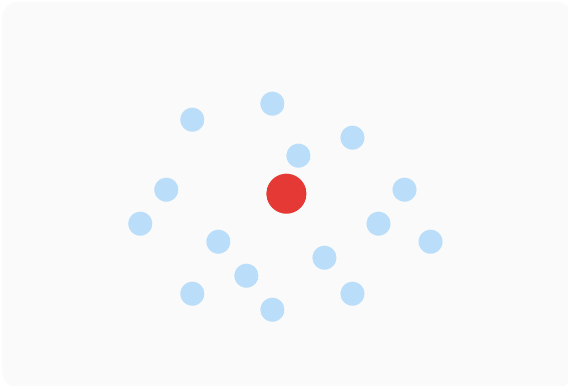
LIME Process (4 Steps)

1 Original Instance



Instance to explain

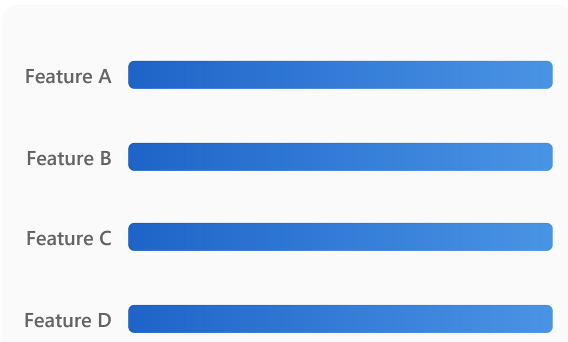
2 Perturbed Samples



Generate neighbors

3 Local Linear Model

4 Feature Importance



Key Ideas

Core Concepts

- **Local:** Explains one prediction
- **Proximity:** Weight by distance
- **Linear:** Interpretable locally
- **Agnostic:** Any model type

Model-Agnostic

Works with any ML model

Perturbation

Create samples around instance

Weighting

Closer samples matter more

Interpretability

Feature importance shows factors

Python Implementation

Fit weighted model

Extract explanation

lime package:

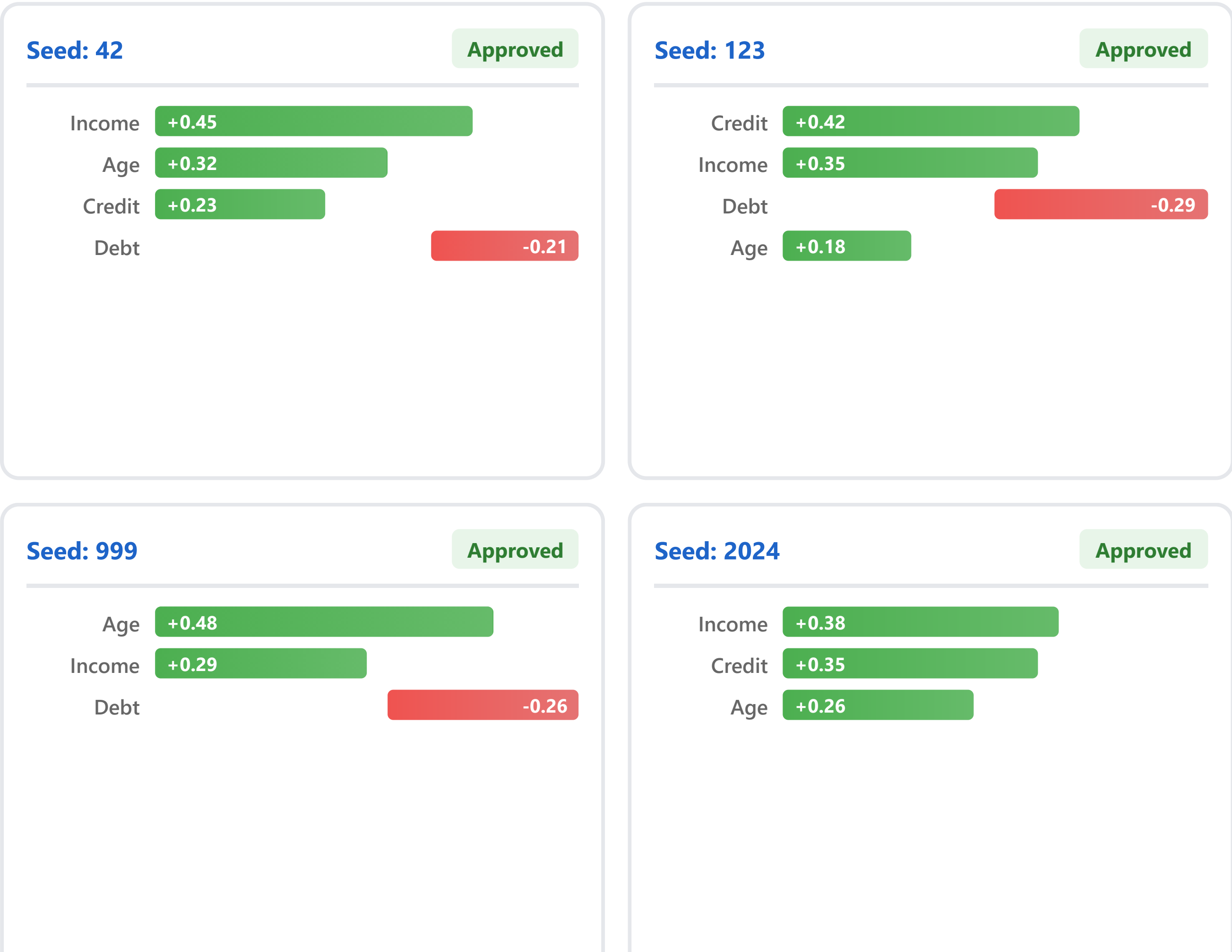
- lime.lime_tabular
- lime.lime_text
- lime.lime_image

LIME Advanced Topics

Stability, Hyperparameters, and Limitations

Stability Issue Demonstration

⚠ Same instance, different random seeds → different explanations



Key Topics

Challenges

- ⚠ Stability issues
- ⚠ Inconsistent explanations
- ⚠ Sampling sensitivity

Hyperparameters
Kernel width, sample size

Feature Selection
Subset for simplicity

Categorical Vars
Special perturbation strategies

SP-LIME
Representative instances

Text & Image
Segment-based perturbations

Best Practices

- ✓ Run multiple times

Credit +0.20

Debt -0.23

✓ Compare with other methods

Anchor Explanations

Sufficient Conditions with High Precision

⚓ **ANCHOR RULE EXAMPLE**

IF Age > 30 **AND** Income > 50K **THEN** **Approve**

PRECISION

0.95

Accuracy when rule applies

COVERAGE

0.30

Fraction of instances covered

Covered Region Visualization

Key Features

Advantages

- ✓ More stable than LIME
- ✓ Human-readable rules
- ✓ High confidence regions
- ✓ Less perturbation-sensitive

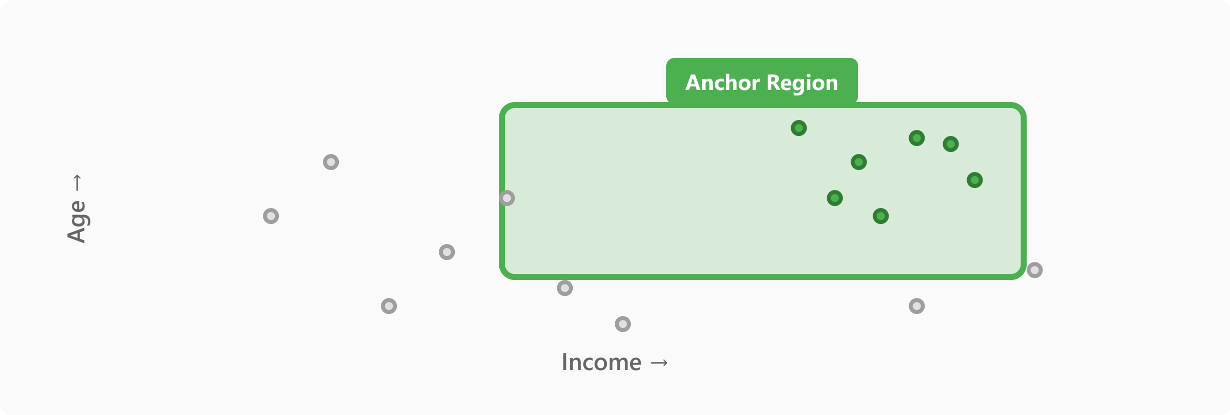
Sufficient Conditions
Rules that guarantee prediction

Precision Metric
Rule accuracy on samples

Coverage Metric
Fraction where rule applies

Beam Search
Finds optimal anchor rules

Implementation

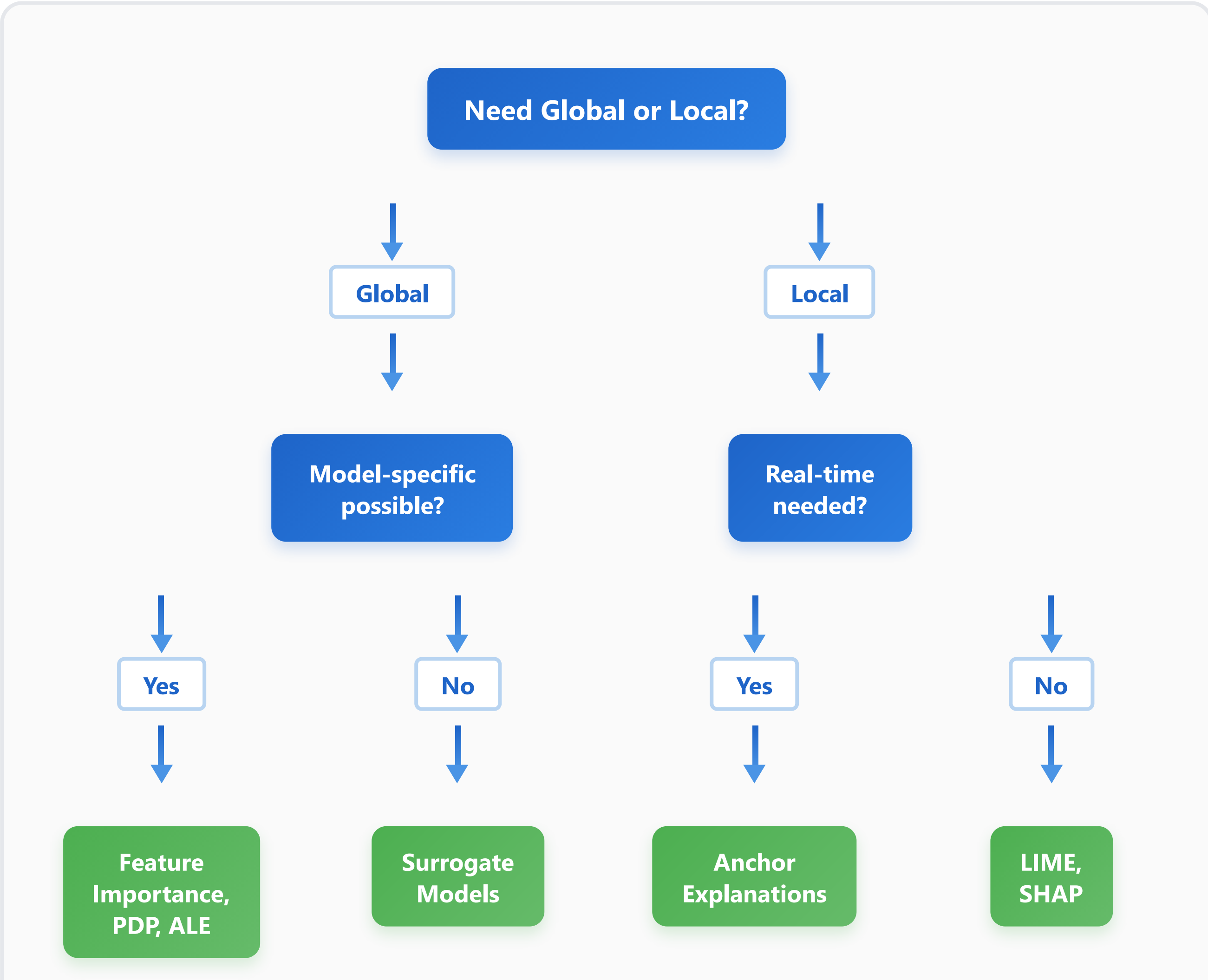


Python: anchor package (integrated with Alibi library)

Practical Guidelines and Best Practices

Selecting the Right XAI Method

XAI Method Selection Decision Tree



Best Practices

Key Guidelines

- Use multiple XAI methods
- Validate with domain experts
- Consider computational budget
- Document limitations
- Match stakeholder needs

Validation
Match domain expertise

Budget
Real-time vs batch processing

Documentation
Record assumptions & limits

Monitoring
Track explanation drift

⚠ Watch Out For

! Adversarial examples
! Data distribution shifts
! Over-reliance on single method

Thank you

Ho-min Park

homin.park@ghent.ac.kr

powersimmani@gmail.com