Real-time Financial Fraud Detection with Modern Python

How to build production-grade fraud detection systems that make sub-second assessments on financial transactions while adversaries constantly adapt.

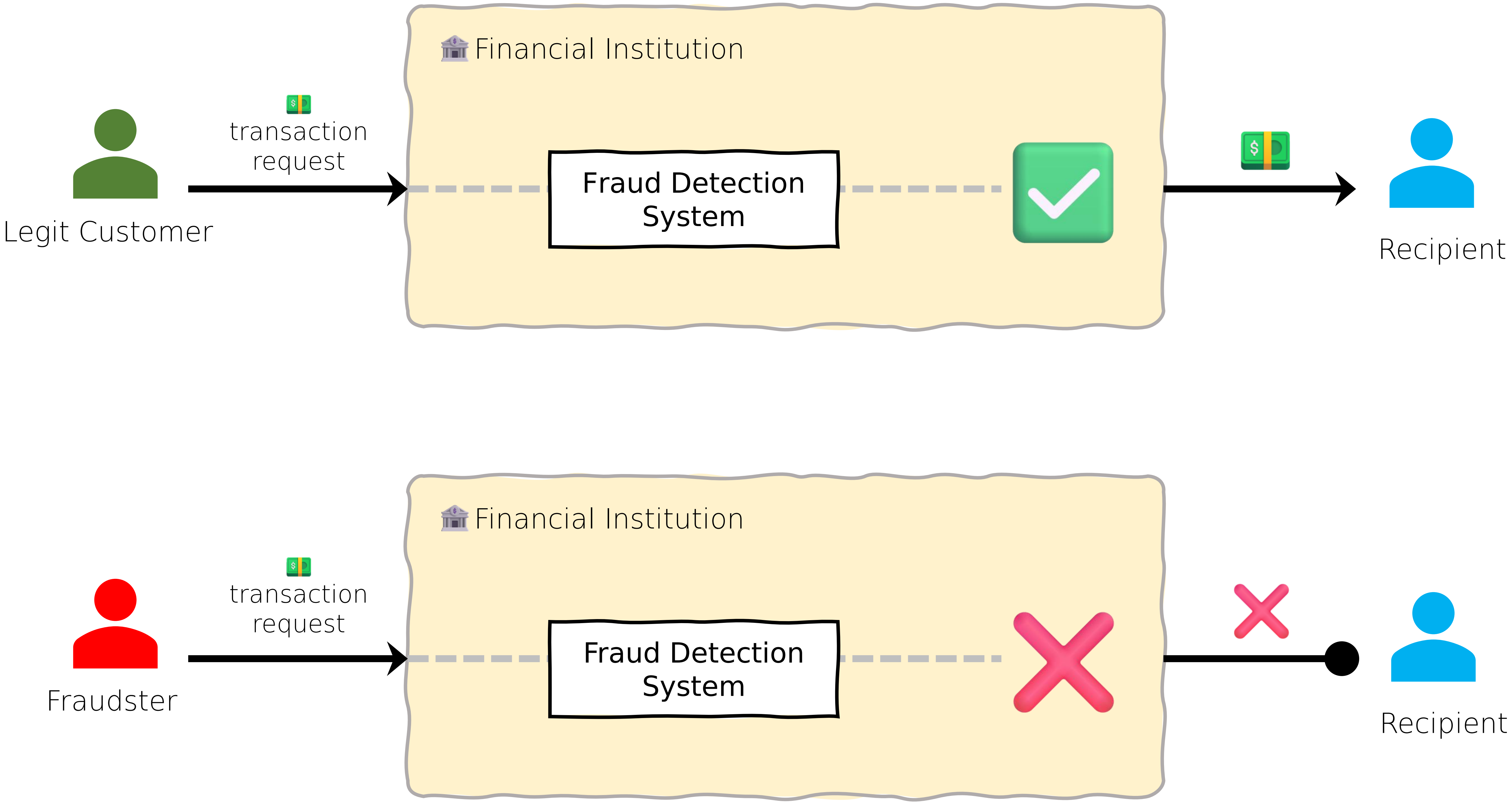
César Soto Valero

Data Scientist / ML Engineer

In This Session

1. Problem framing
2. Decision-first metrics
3. Time-aware validation
4. Latency-aware modeling
5. Shipping & operations

1. Problem Framing



Financial Institution

Legit Customer

💵 transaction request

Fraud Detection System

✓

💵

Recipient

Financial Institution

Fraudster

💵 transaction request

Fraud Detection System

✗

✗

Recipient

The Stakes Are Very High!

💸 Real Money, Realtime

Every transaction requires a decision in milliseconds. Delays mean lost revenue or unhappy customers.

🎯 Fraud Evolves Daily

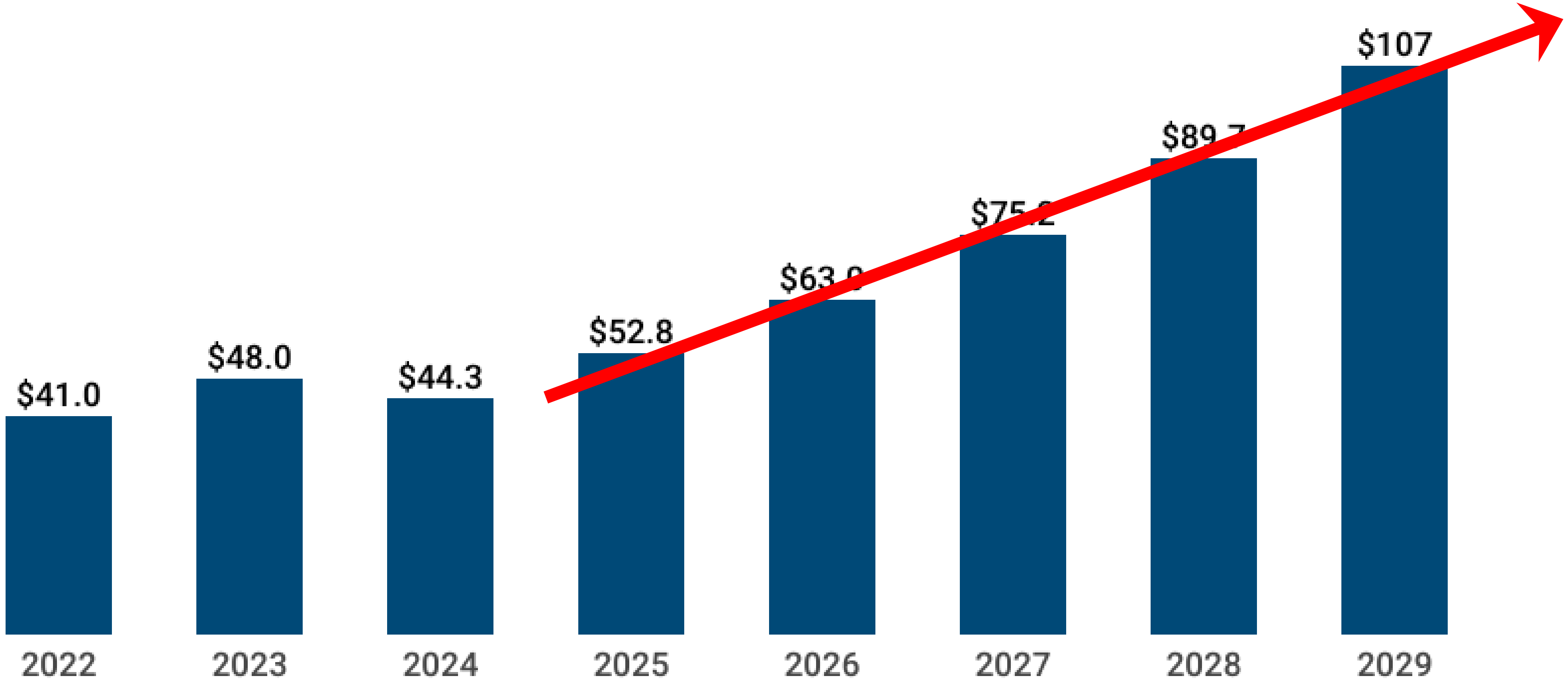
Rule-based systems can't keep the pace with adversaries who iterate and adapt like well-funded startups.

😤 Trust Is Fragile

False positives block legitimate users, eroding trust faster than fraud costs money.

Global Losses to Online Payment Fraud

(in billions)



Source: Statista and Merchant Risk Council

Years 2025 onward are projections

Why This Is Hard

Extreme Class Imbalance

Fraud represents 0.1–2% of transactions in most systems. Finding needles in haystacks is the core technical challenge. Traditional accuracy metrics become meaningless.

Delayed Ground Truth

Labels arrive days or weeks after decisions. Chargebacks and investigations take time. Training on outdated patterns while fraud tactics shift creates constant model drift.

Active Adversaries

Fraudsters deliberately probe your thresholds and test detection boundaries. They reverse-engineer your models through systematic experimentation, forcing constant adaptation.

2. Decision-First Metrics

		GROUND TRUTH	
		POSITIVE (fraudulent txt)	NEGATIVE (genuine txt)
PREDICTION	POSITIVE (fraudulent txt)	True Positive (TP) Reality: Fraud Prediction: Fraud Result: Good	False Positive (FP) Reality: Not Fraud Prediction: Fraud Result: Unnecessary costs and operational inefficiency
	NEGATIVE (genuine txt)	False Negative (FN) Reality: Fraud Prediction: Not Fraud Result: Non-compliance and induced costs for the bank and society	True Negative (TN) Reality: Not Fraud Prediction: Not Fraud Result: Good

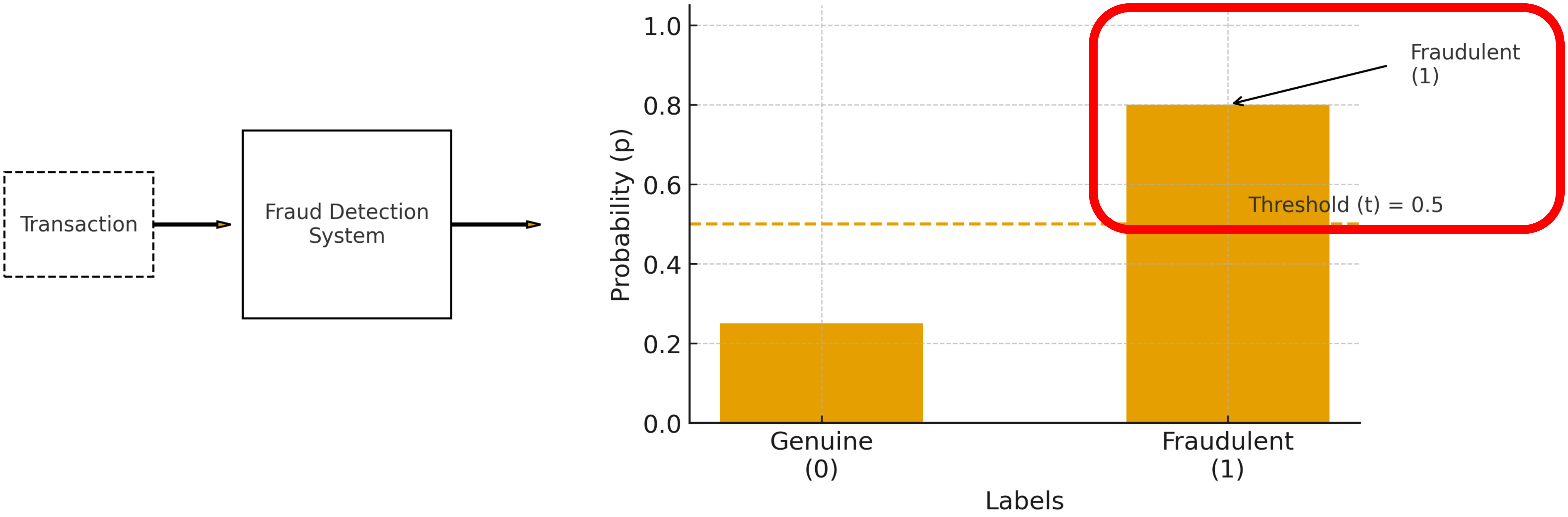
		GROUND TRUTH	
		POSITIVE (fraudulent txt)	NEGATIVE (genuine txt)
PREDICTION	POSITIVE (fraudulent txt)	True Positive (TP) Reality: Fraud Prediction: Fraud Result: Good	False Positive (FP) Reality: Not Fraud Prediction: Fraud Result: Unnecessary costs and operational inefficiency
	NEGATIVE (genuine txt)	False Negative (FN) Reality: Fraud Prediction: Not Fraud Result: Non-compliance and induced costs for the bank and society	True Negative (TN) Reality: Not Fraud Prediction: Not Fraud Result: Good

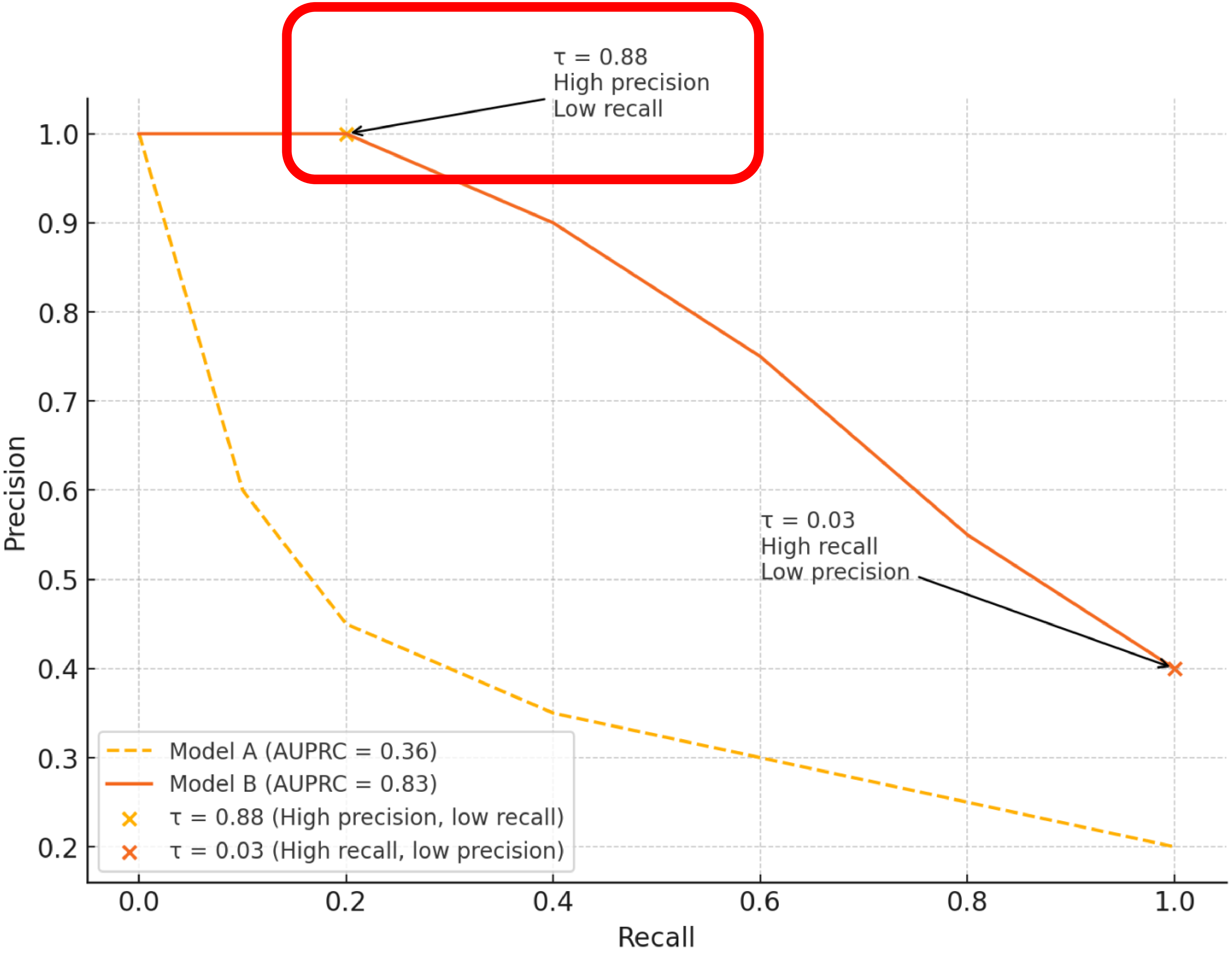
		GROUND TRUTH	
		POSITIVE (fraudulent txt)	NEGATIVE (genuine txt)
PREDICTION	POSITIVE (fraudulent txt)	True Positive (TP) Reality: Fraud Prediction: Fraud Result: Good	False Positive (FP) Reality: Not Fraud Prediction: Fraud Result: Unnecessary costs and operational inefficiency
	NEGATIVE (genuine txt)	False Negative (FN) Reality: Fraud Prediction: Not Fraud Result: Non-compliance and induced costs for the bank and society	True Negative (TN) Reality: Not Fraud Prediction: Not Fraud Result: Good

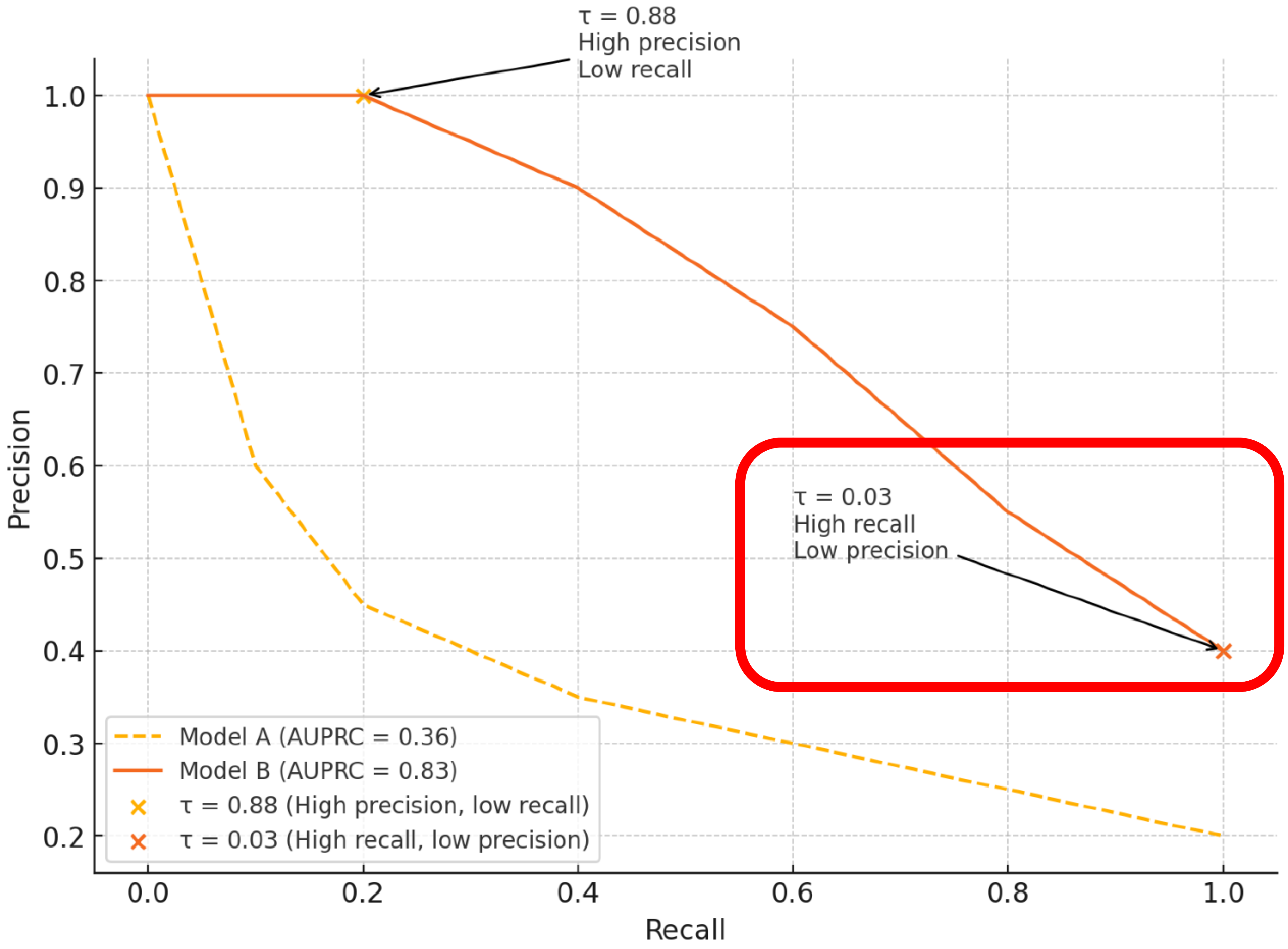
$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\boxed{\text{Recall} = \frac{TP}{TP + FN}}$$

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$$


Transaction
Fraud Detection System
1.0
0.8
0.6
0.4
0.2
0.0
Probability (p)
Fraudulent (1)
Threshold (t) = 0.5
Genuine (0)
Fraudulent (1)
Labels


τ = 0.88
High precision
Low recall
τ = 0.03
High recall
Low precision
Precision
Recall
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.0
0.2
0.4
0.6
0.8
1.0
Model A (AUPRC = 0.36)
Model B (AUPRC = 0.83)
τ = 0.88 (High precision, low recall)
τ = 0.03 (High recall, low precision)


τ = 0.88
High precision
Low recall
τ = 0.03
High recall
Low precision
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
Precision
0.0
0.2
0.4
0.6
0.8
1.0
Recall
Model A (AUPRC = 0.36)
Model B (AUPRC = 0.83)
τ = 0.88 (High precision, low recall)
τ = 0.03 (High recall, low precision)

scikit learn

```python
from sklearn.metrics import precision_score, recall_score, precision_recall_curve

# y_true:  0 = genuine, 1 = fraud
# y_pred:  model's predicted labels after thresholding probabilities
precision = precision_score(y_true, y_pred, pos_label=1)
recall    = recall_score(y_true, y_pred, pos_label=1)

precision, recall, thresholds = precision_recall_curve(y_true, p_fraud, pos_label=1)
```

Extreme Class Imbalance

> A model that labels everything as legitimate achieves 99.9% accuracy while catching zero fraud. Accuracy is a dangerous illusion when classes are imbalanced.

Precision and Recall Tell the Real Story

Precision measures how many alerts are actually fraud. Recall captures what percentage of fraud you catch. Both matter intensely for business outcomes.

Thresholds Drive Business Impact

Moving a decision threshold changes precision-recall balance and directly affects revenue, review costs, and customer experience. The threshold is a business lever, not just a technical parameter.

3. Time-Aware Validation

Delayed Ground Truth

Delayed Ground Truth

Temporal Train/Val/Test Splits

Never randomize timestamps. Train on past data, validate on near-future, test on held-out future. Respect time's arrow.

1

Delayed Ground Truth

1

Temporal Train/Val/Test Splits

Never randomize timestamps. Train on past data, validate on near-future, test on held-out future. Respect time's arrow.

2

Rolling Windows with Safety Gaps

Use sliding time windows that mimic production deployment. Add gaps between train and validation to simulate label delay.

Delayed Ground Truth

1

Temporal Train/Val/Test Splits

Never randomize timestamps. Train on past data, validate on near-future, test on held-out future. Respect time's arrow.

2

Rolling Windows with Safety Gaps

Use sliding time windows that mimic production deployment. Add gaps between train and validation to simulate label delay.

3

Strictly Prevent Leakage Across Time

Future information must never leak into training. One timestamp error invalidates all results and creates false confidence.

Delayed Ground Truth

1

Temporal Train/Val/Test Splits

Never randomize timestamps. Train on past data, validate on near-future, test on held-out future. Respect time's arrow.

2

Rolling Windows with Safety Gaps

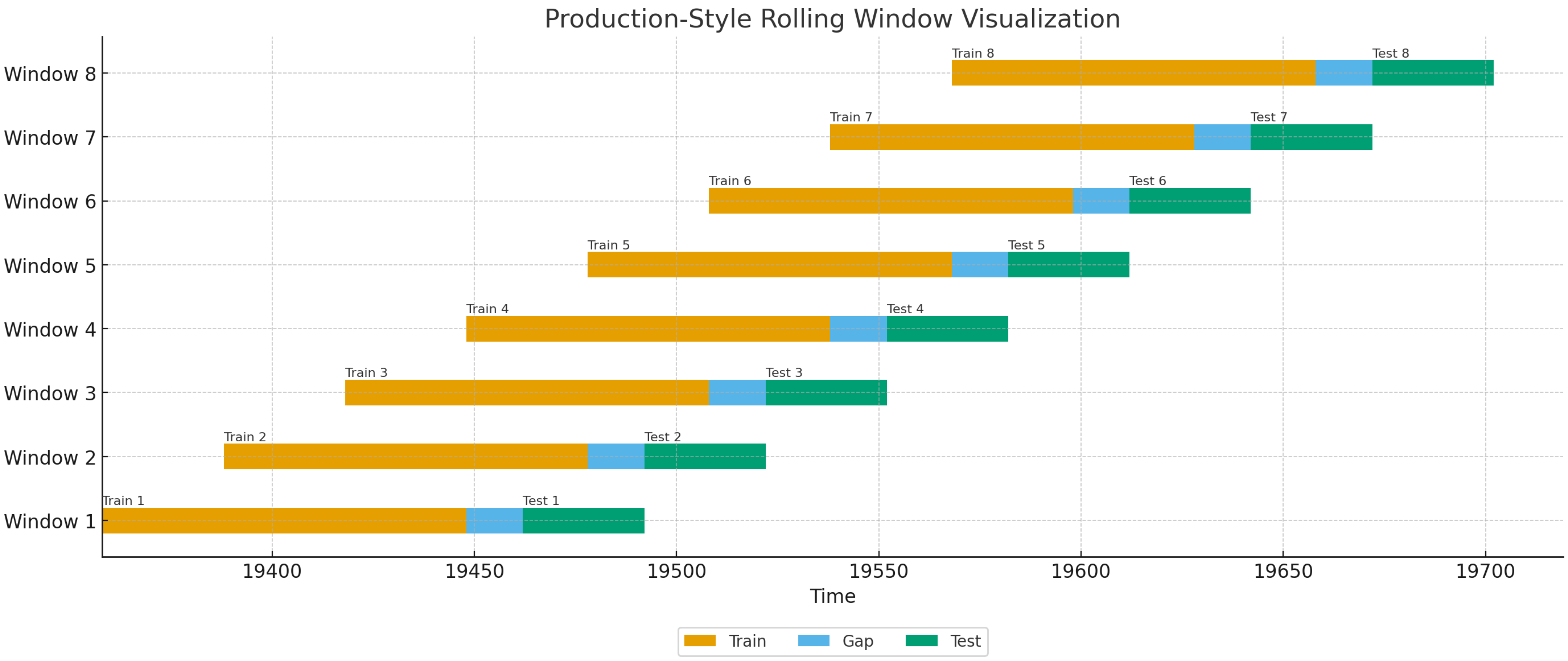
Use sliding time windows that mimic production deployment. Add gaps between train and validation to simulate label delay.

3

Strictly Prevent Leakage Across Time

Future information must never leak into training. One timestamp error invalidates all results and creates false confidence.

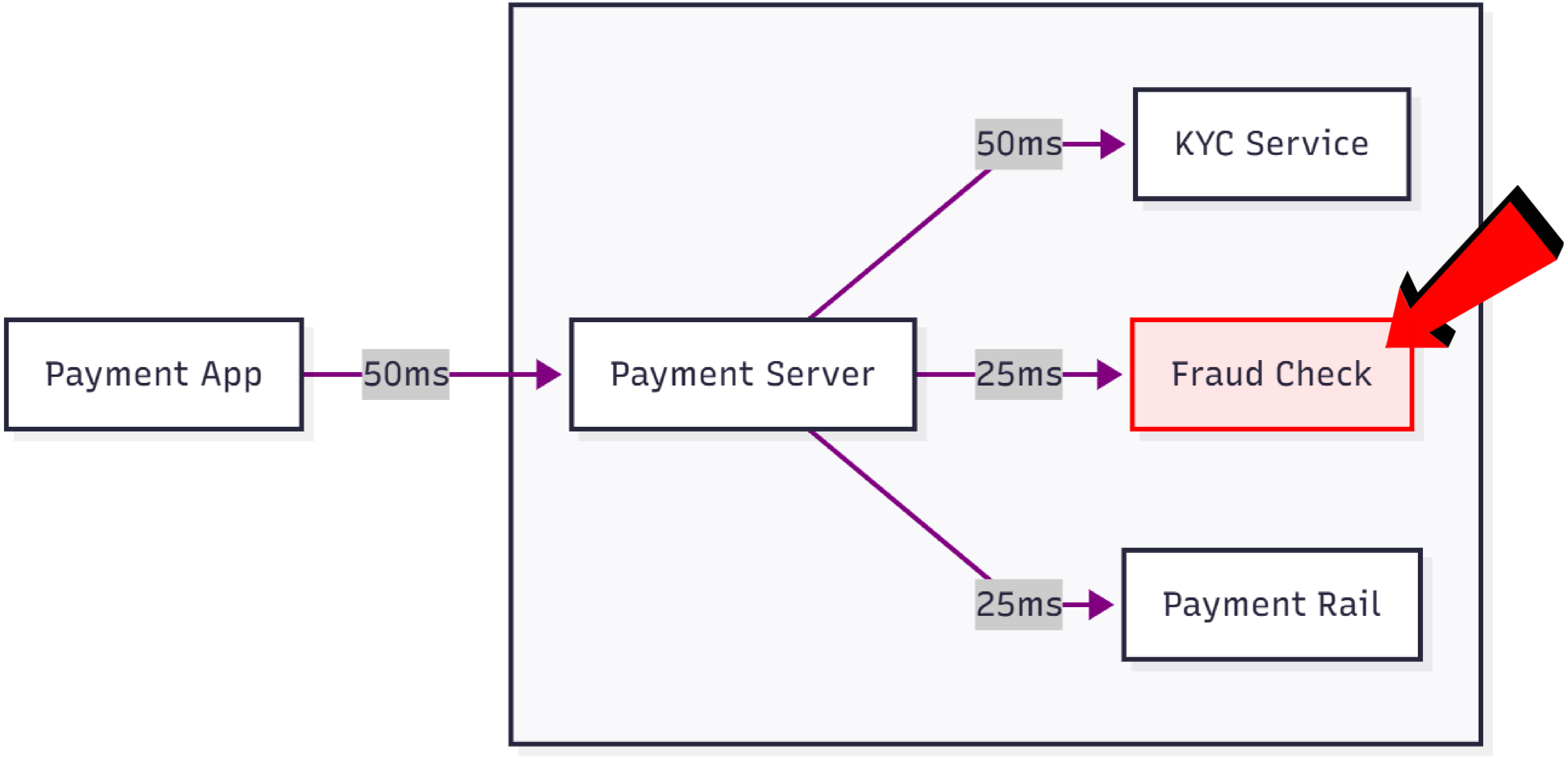
Remember: Random splits give you optimistic metrics that collapse in production. Time-based validation predicts real performance.



Production-Style Rolling Window Visualization

4. Latency-Aware Modeling

Latency


Payment App
50ms
Payment Server
50ms
KYC Service
25ms
Fraud Check
25ms
Payment Rail

Latency Budgets: Rules vs ML

<1ms

Rule-Based Systems

Blazingly fast, deterministic, explainable. Perfect for known patterns and hard constraints like blocklists or velocity checks.

10-200ms

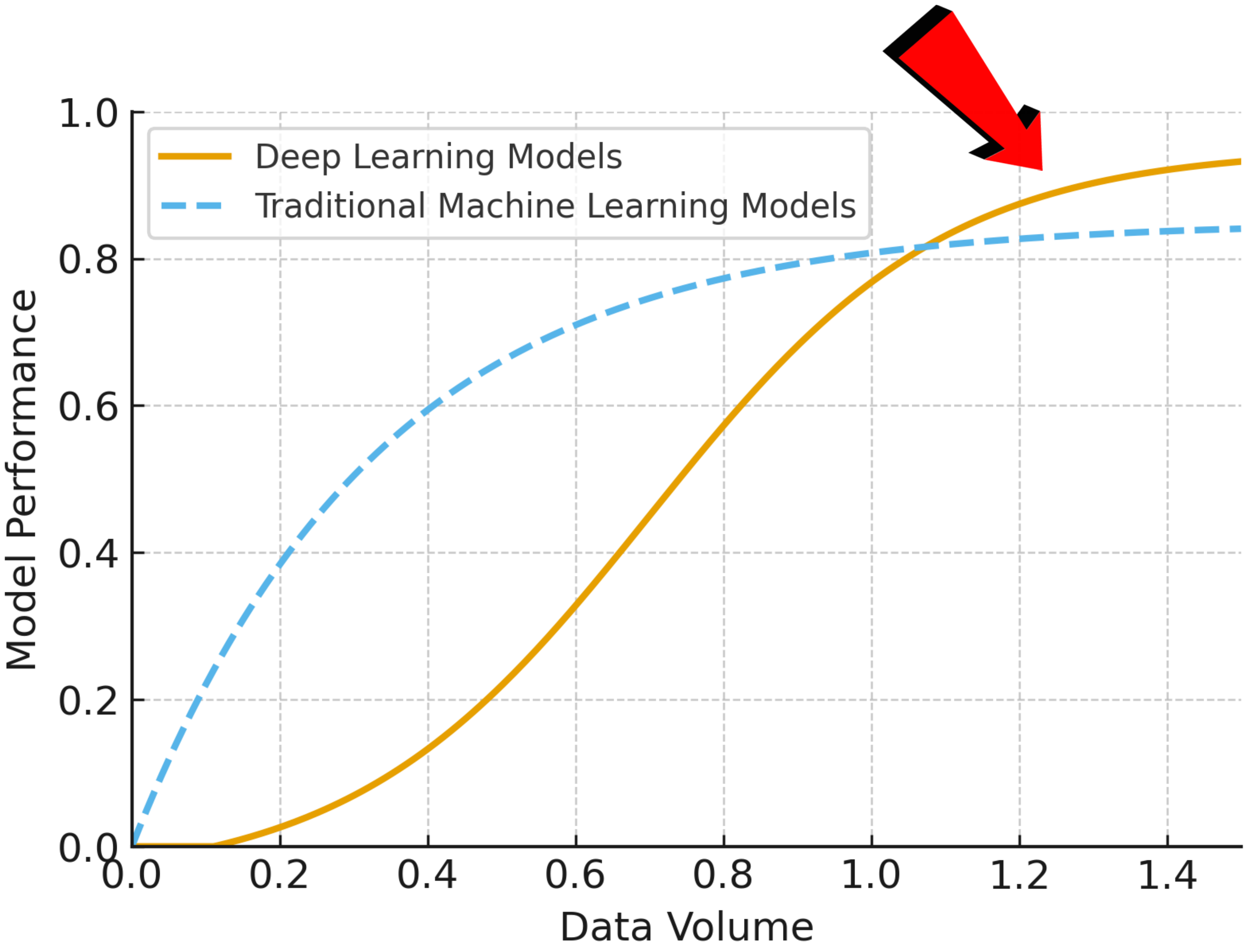
ML Model Inference

Gradient-boosted trees typically run in tens of milliseconds. Neural networks can reach 100–200ms depending on complexity and infrastructure.

~

Hybrid Architecture

Blend rules for instant failsafes and ML for incremental lift. Rules catch obvious fraud at <1ms, models score the nuanced middle at acceptable latency.

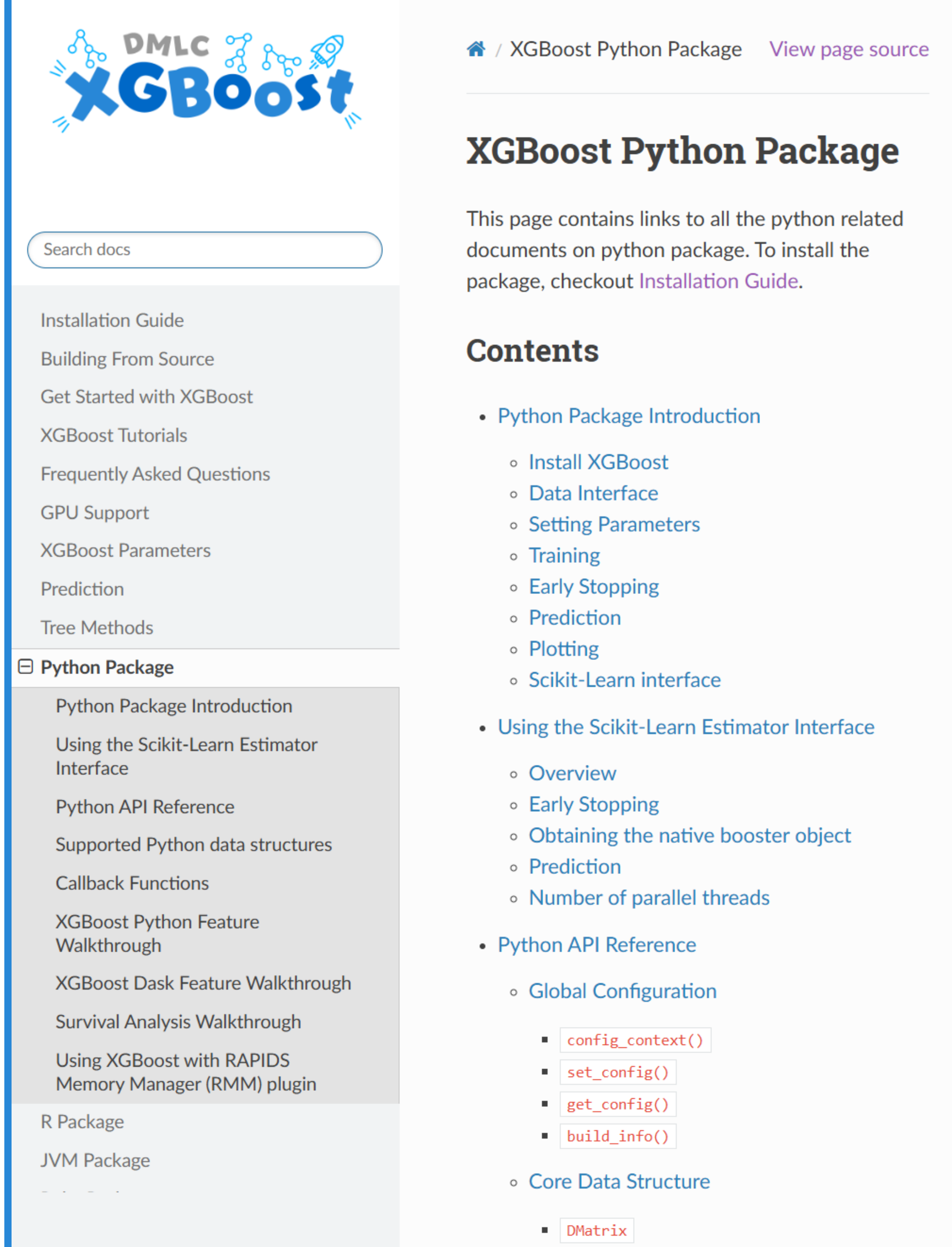

Deep Learning Models
Traditional Machine Learning Models
Model Performance
Data Volume
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4

Start Strong With XGBoost

Why XGBoost Shines on Tabular Data

- Handles heterogeneous features naturally without complex preprocessing
- Fast inference fits realtime latency budgets with sub-100ms predictions
- Interpretable feature importances and tree structures support debugging and stakeholder trust
- Robust to missing values and outliers common in production fraud data

Pro tip: Start here before exploring deep learning. XGBoost delivers 80% of the value with 20% of the complexity.

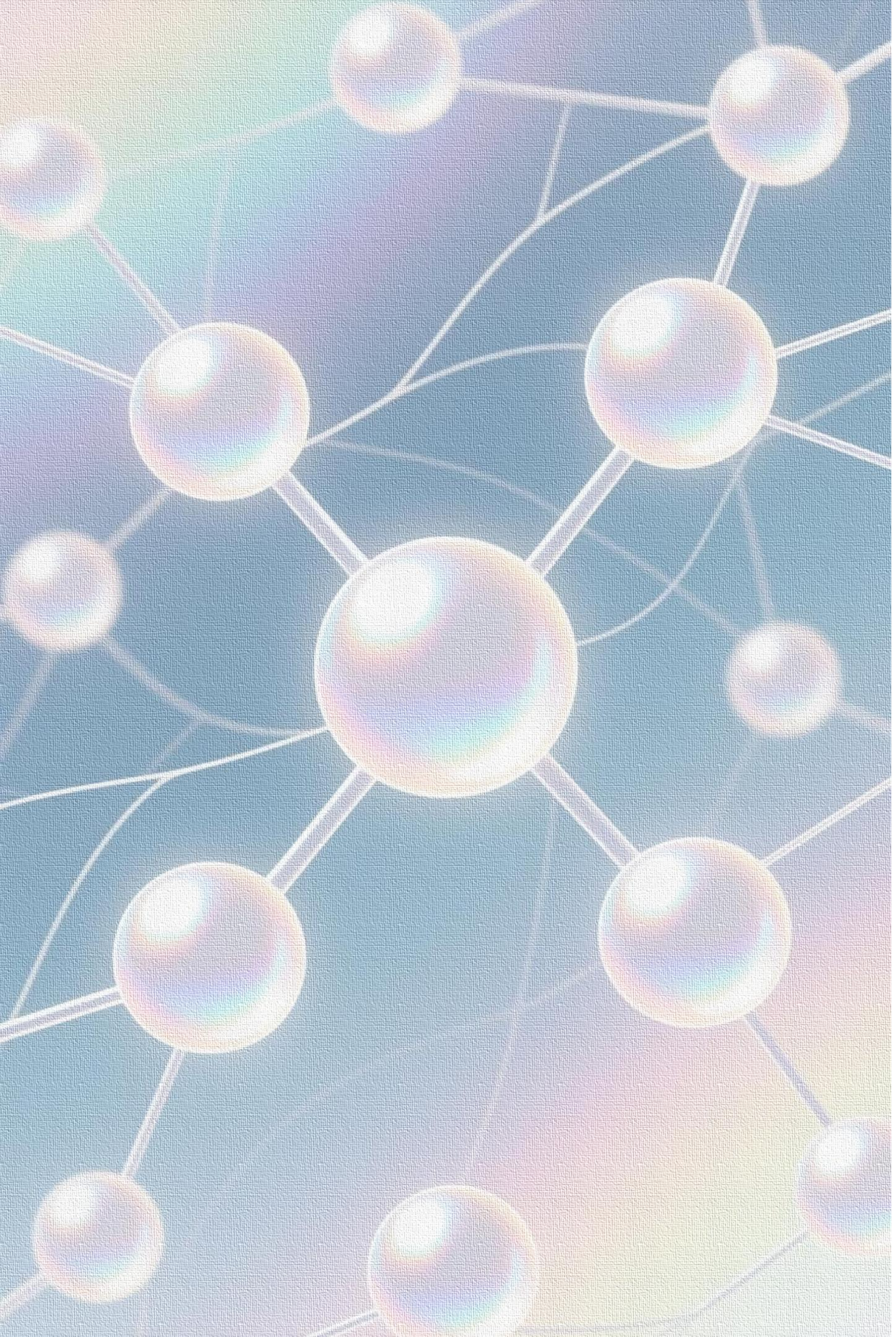


Installation Guide
Building From Source
Get Started with XGBoost
XGBoost Tutorials
Frequently Asked Questions
GPU Support
XGBoost Parameters
Prediction
Tree Methods
⊟ Python Package
Python Package Introduction
Using the Scikit-Learn Estimator Interface
Python API Reference
Supported Python data structures
Callback Functions
XGBoost Python Feature Walkthrough
XGBoost Dask Feature Walkthrough
Survival Analysis Walkthrough
Using XGBoost with RAPIDS Memory Manager (RMM) plugin
R Package
JVM Package

XGBoost Python Package

This page contains links to all the python related documents on python package. To install the package, checkout Installation Guide.

Contents



When to Add Deep Learning

Deep learning shines when other models hit a ceiling. If the feature space is sparse and high-cardinality, then tabular architectures like TabNet or FT-Transformer can capture complex patterns that gradient boosting misses.

Sparse, high-cardinality features benefit

Embeddings compress categorical explosions into dense representations, enabling models to learn nuanced entity relationships.

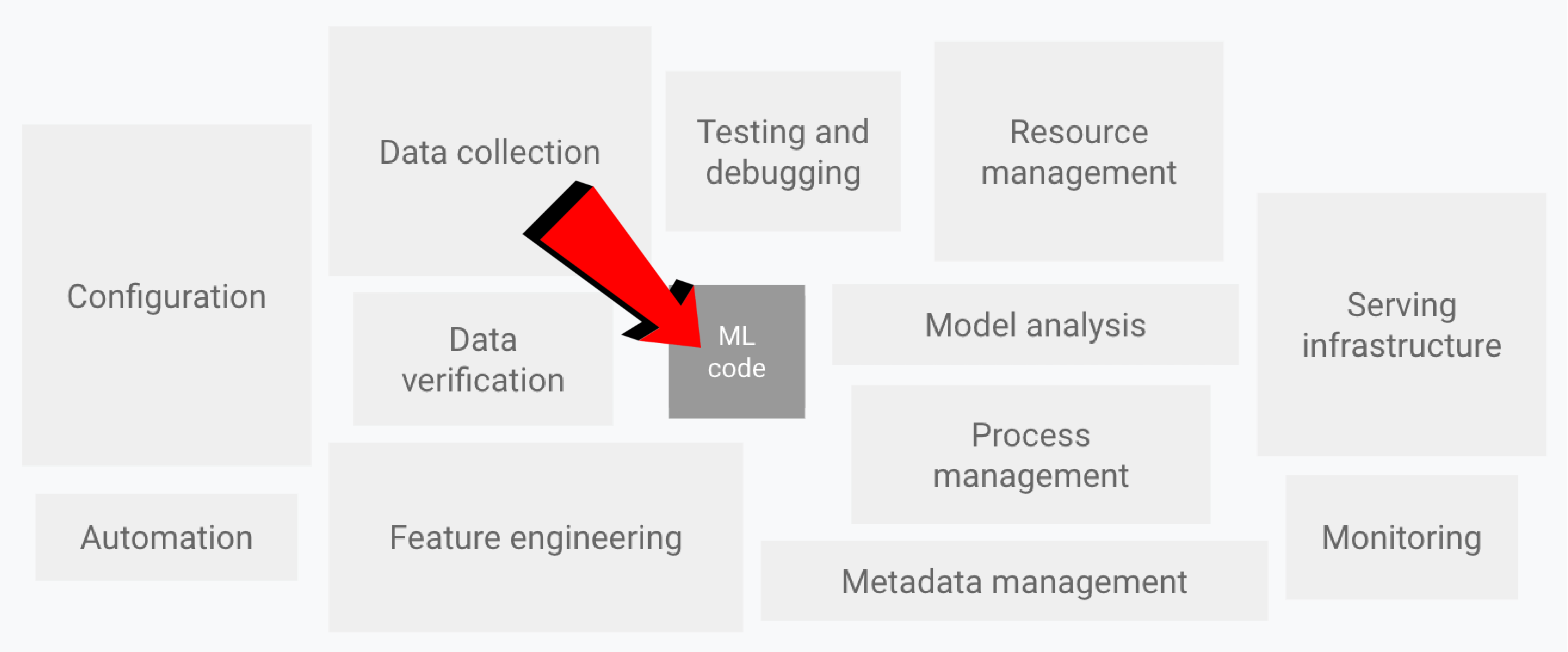
Learn interactions automatically

Deep architectures discover feature crosses without manual engineering, reducing technical debt and iteration cycles.

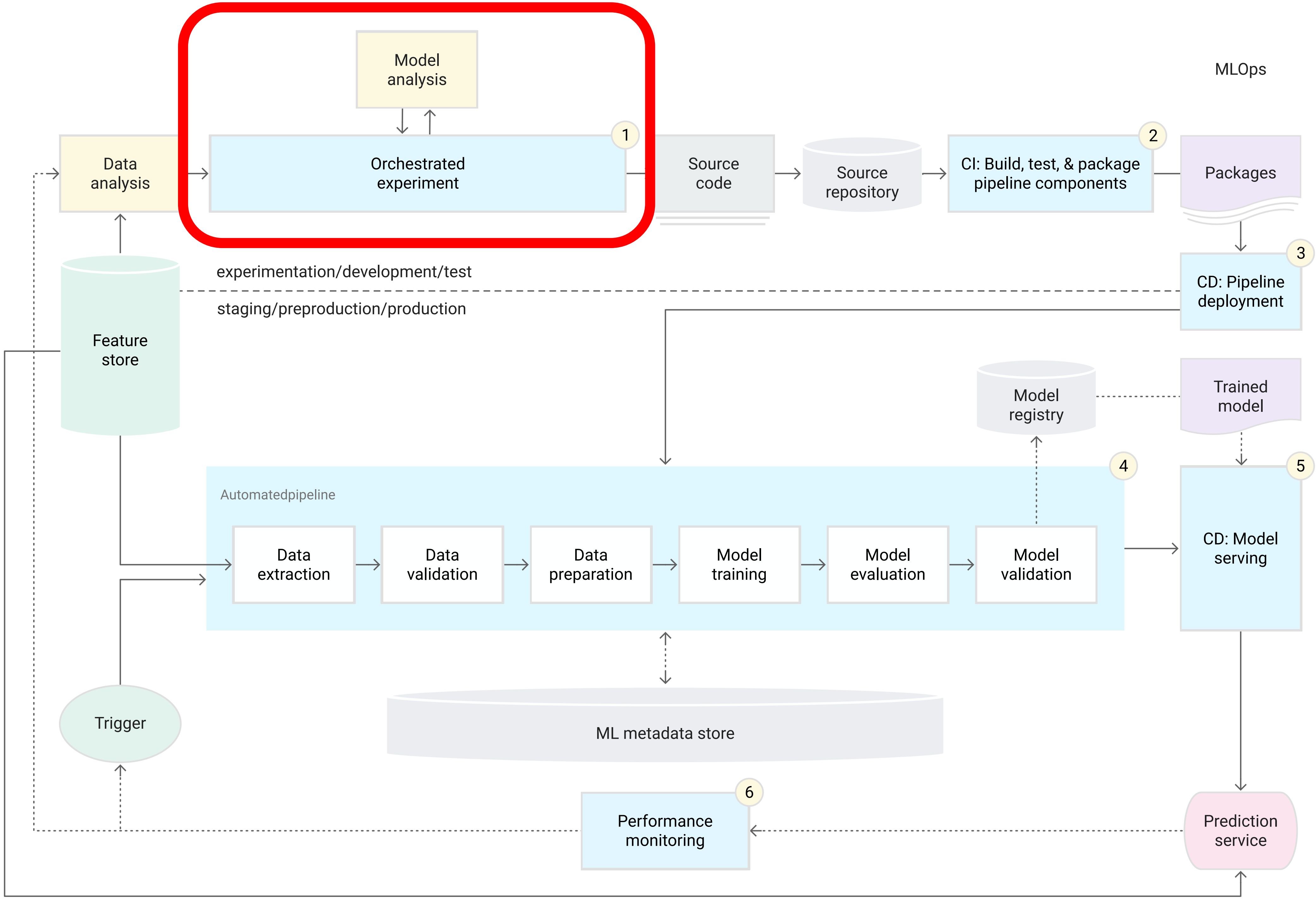
Validate gains against latency budget

Measure P99 inference time in production. If your scoring SLA is 50ms, confirm DL delivers ROI before scaling.

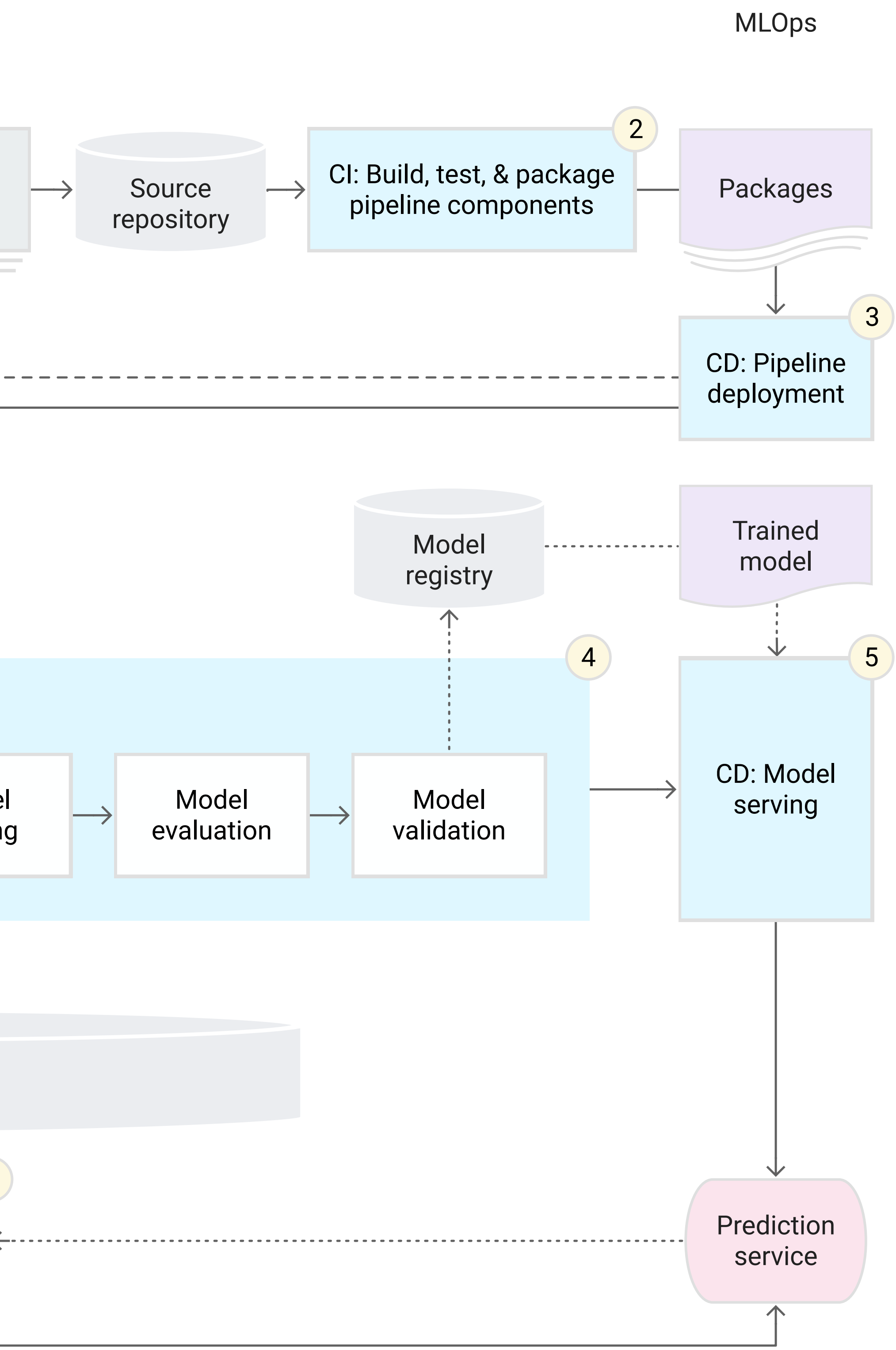
5. Shipping & Operations



Source: https://docs.cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning



Source: https://docs.cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning



From Notebook to Service

Production fraud detection demands more than a trained model. The gap between research code and a resilient service is wide.

1

Small, testable scoring core

Isolate inference logic into a pure function: features in, prediction out. Unit test edge cases (e.g., null fields, out-of-vocabulary tokens, schema drift).

2

Package model plus preprocessors

Bundle your serialized model with feature transformers, scalers, and encoders. Version everything together to prevent train-serve skew.

3

Serve behind a FastAPI endpoint

Wrap your scoring function in a lightweight REST API. FastAPI provides async support, automatic docs, and validation, which is critical for uptime and debugging.

Serving Patterns That Scale

No single serving architecture fits every fraud use case. Match your pattern to transaction velocity, latency requirements, and operational complexity. Hybrid approaches often win in production.

Batch scoring for nightly sweeps

Score historical transactions offline. Perfect for investigating past fraud rings or recomputing risk for dormant accounts (no real-time pressure).

Online scoring for checkout flows

Synchronous prediction at transaction time. Requires sub-100ms P99 latency. Cache aggressively, use feature stores, and pre-warm models.

TorchServe or TFServing when needed

Leverage framework-native serving for GPU inference, model versioning, and A/B testing.

Patterns That Survive Reality

> Fraud detection in production is a systems problem, not just a modeling problem. The best ML model fails if it can't ship, scale, or adapt.

Decide for cost, not vanity scores

Optimize for business impact (false positive costs, analyst capacity, customer friction). Precision-recall tradeoffs are product decisions.

Validate in time, always

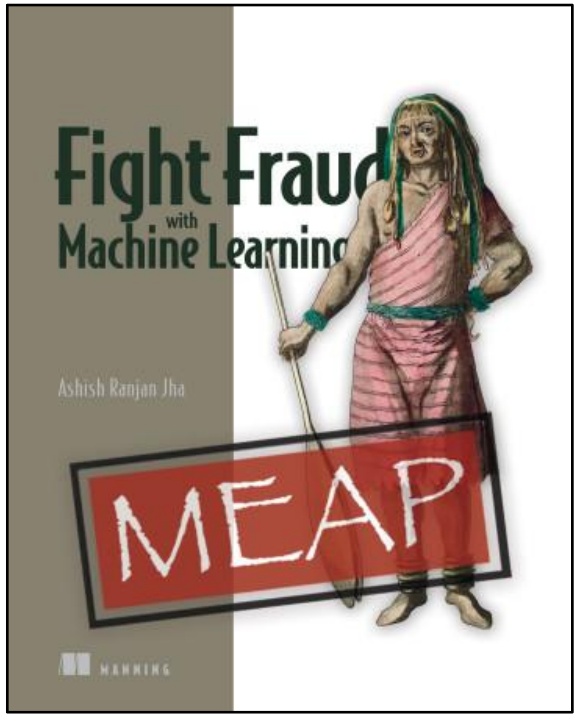
Temporal cross-validation prevents overfitting to old fraud patterns. Adversaries evolve and your evaluation strategy must reflect that reality.

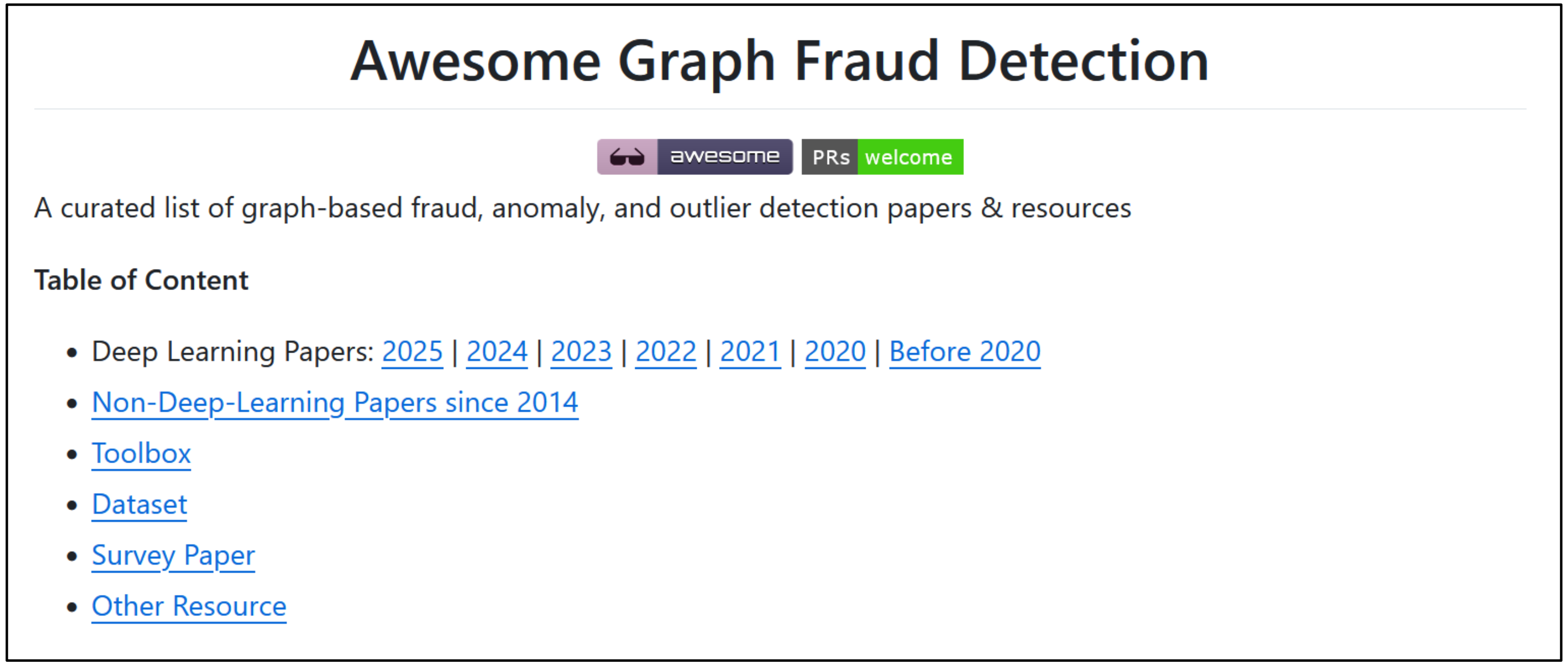
Ship with guardrails, monitor relentlessly

Use a hybrid approach, add complexity to the model only if necessary, and stick to MLOps good practices. Observability is not optional.

Recommended Resources

1. “Fight Fraud with Machine Learning”, MEAP book by Manning
2. https://github.com/safe-graph/graph-fraud-detection-papers/





Awesome Graph Fraud Detection

A curated list of graph-based fraud, anomaly, and outlier detection papers & resources

Table of Content

- Deep Learning Papers: 2025 | 2024 | 2023 | 2022 | 2021 | 2020 | Before 2020
- Non-Deep-Learning Papers since 2014
- Toolbox
- Dataset
- Survey Paper
- Other Resource

Q&A