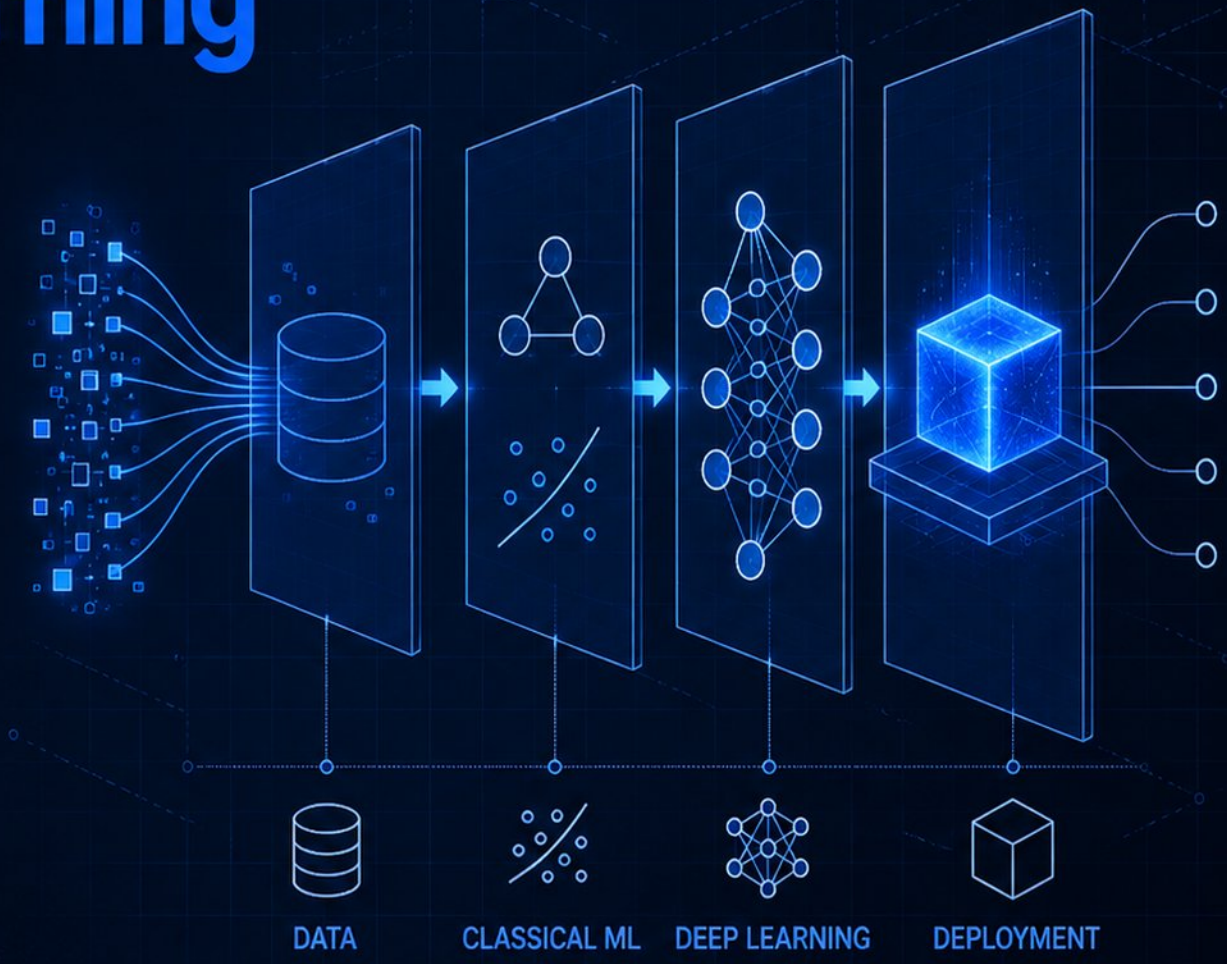
Python Machine Learning

- Python dominates ML in 2026 through ecosystem and network effects
- End-to-end workflow: raw data to monitored production models
- Four ecosystem layers: data, classical ML, deep learning, deployment
- Course equips you to build and deploy production ML systems



The 2026 Python ML Landscape

- Four distinct ecosystem layers: data, classical ML, deep learning, and deployment
- Rust-powered tooling is reshaping Python's performance ceiling
- Scikit-learn remains the standard for tabular and classical ML
- PyTorch leads research; Hugging Face is the gateway to pretrained models

Core Libraries: NumPy, pandas, and Polars

- NumPy arrays: the foundation of numerical computing
- pandas DataFrames: the standard for tabular workflows
- Polars: Rust-powered, 5–20x faster on large data
- DataFrames feed directly into scikit-learn and PyTorch
- 25–33% of new native PyPI packages now use Rust

Scikit-learn:

The Classical ML Workhorse

- Consistent fit, predict, and transform API
- Handles classification, regression, clustering, and preprocessing
- Pipelines and ColumnTransformer prevent data leakage and ensure reproducibility
- Beats deep learning on cost, speed, and interpretability

Deep Learning: PyTorch, TensorFlow, and Keras

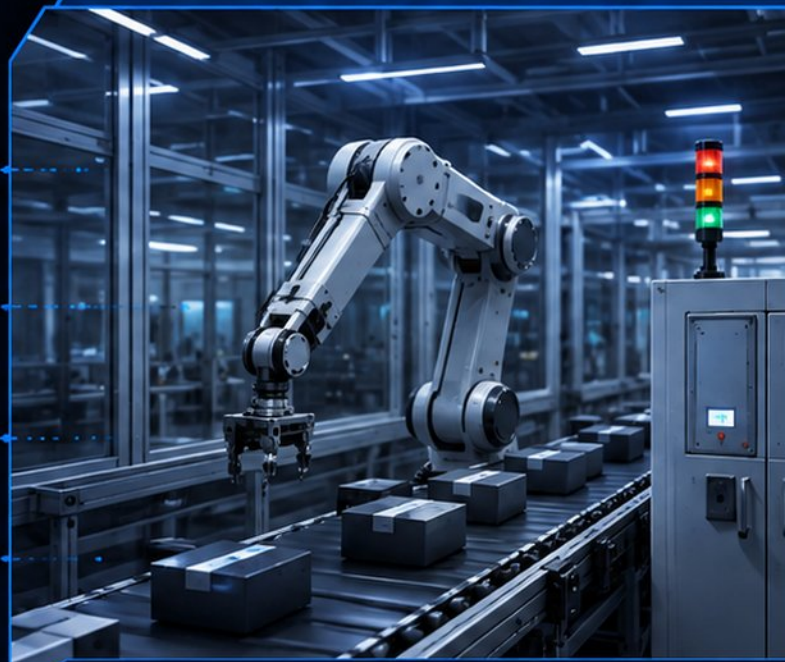
PYTORCH
RESEARCH & CUSTOM ARCHITECTURES

Flexible. Experimental.
Built for discovery.

LAYERED STACK ARCHITECTURE

- APPLICATION LAYER
- MODEL LAYER
- FRAMEWORK LAYER
- RUNTIME LAYER
- INFRASTRUCTURE LAYER

TENSORFLOW + KERAS
PRODUCTION & EDGE DEPLOYMENT STRENGTH



Reliable. Scalable.
Built for production.

- **PyTorch:** default for research and custom architectures
- **TensorFlow + Keras:** production and edge deployment strength
- **Keras 3.0:** write once, run on PyTorch, TensorFlow, or JAX backends
- **Match framework to task:** research, deployment, and team expertise

Defining Success Before Modeling

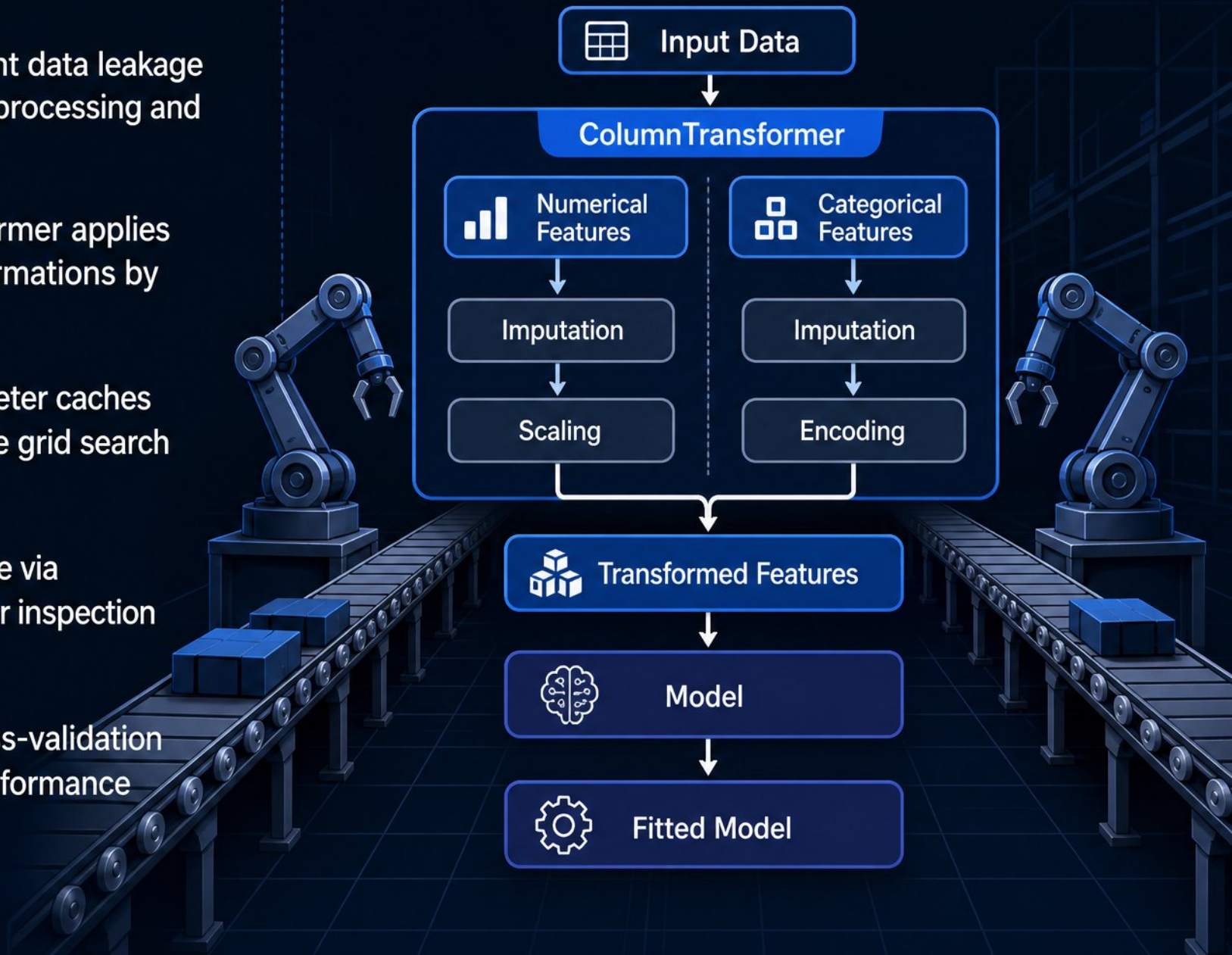
- Align metrics with business goals before writing code
- Classification: precision, recall, F1; Regression: MAE, RMSE
- Baselines set the minimum viable performance bar
- Accuracy often misleads; choose metrics that reflect real costs
- Avoid leakage-driven optimism in metric selection

Splitting Data Without Leakage

- Data leakage: test information reaches the model, inflating performance
- Always split data before any preprocessing steps
- Fit transformers only on training data; apply to both sets
- Use TimeSeriesSplit for chronological data to prevent lookahead
- Stratified and group-based splits handle imbalanced or grouped data

Pipelines: Reproducible Preprocessing and Modeling

- Pipelines prevent data leakage by chaining preprocessing and models
- ColumnTransformer applies distinct transformations by feature type
- memory parameter caches fits to accelerate grid search tuning
- Steps accessible via named_steps for inspection and reuse
- Pipelines + cross-validation yield honest performance estimates



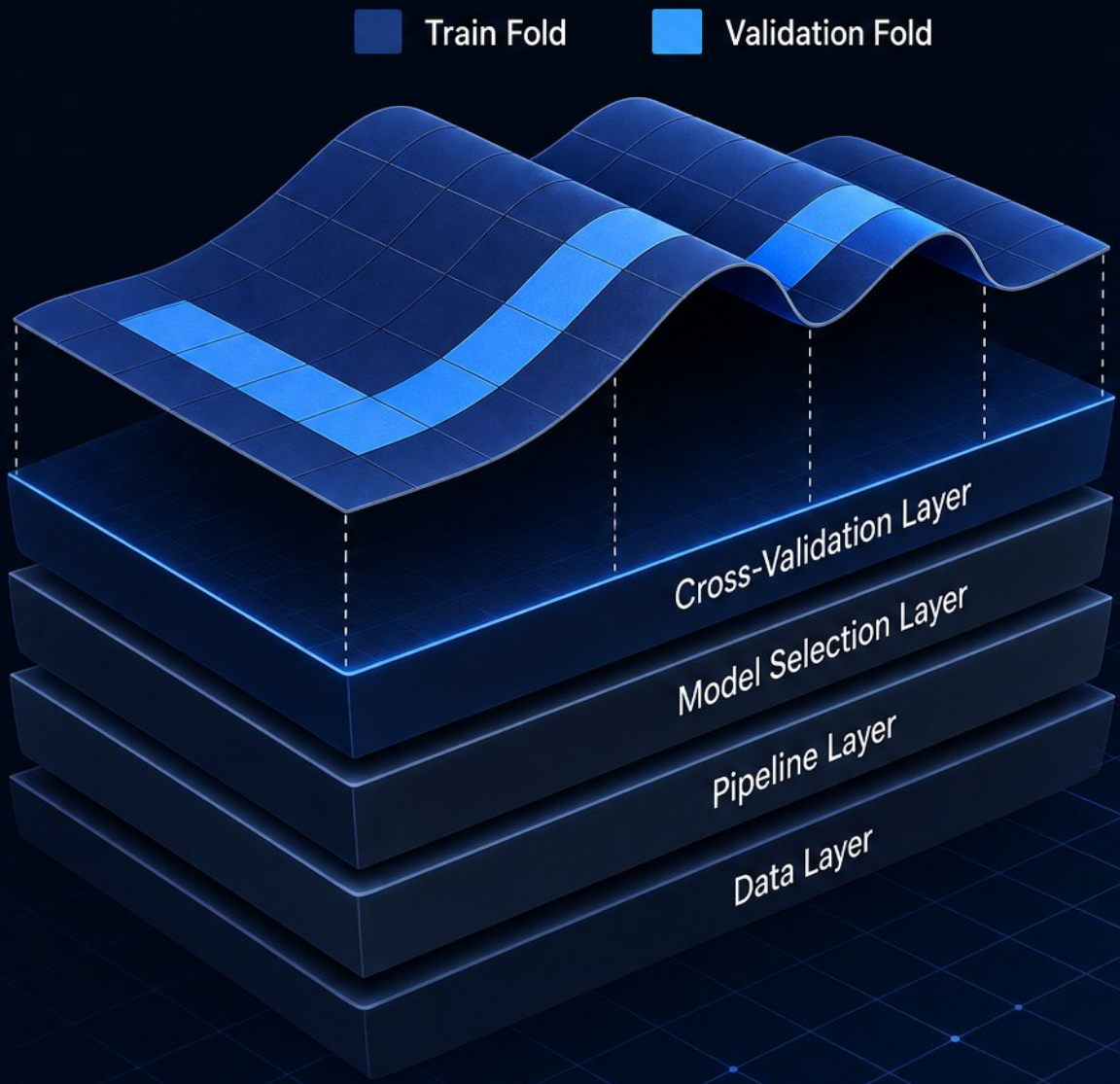
Cross-Validation and Model Selection

- K-fold CV splits data into multiple train/test folds, averaging performance

- Cross-validation score functions integrate directly with scikit-learn Pipelines

- Nested CV evaluates tuning pipelines honestly, preventing selection bias

- Use KFold, GroupKFold, or TimeSeriesSplit for specialized data structures

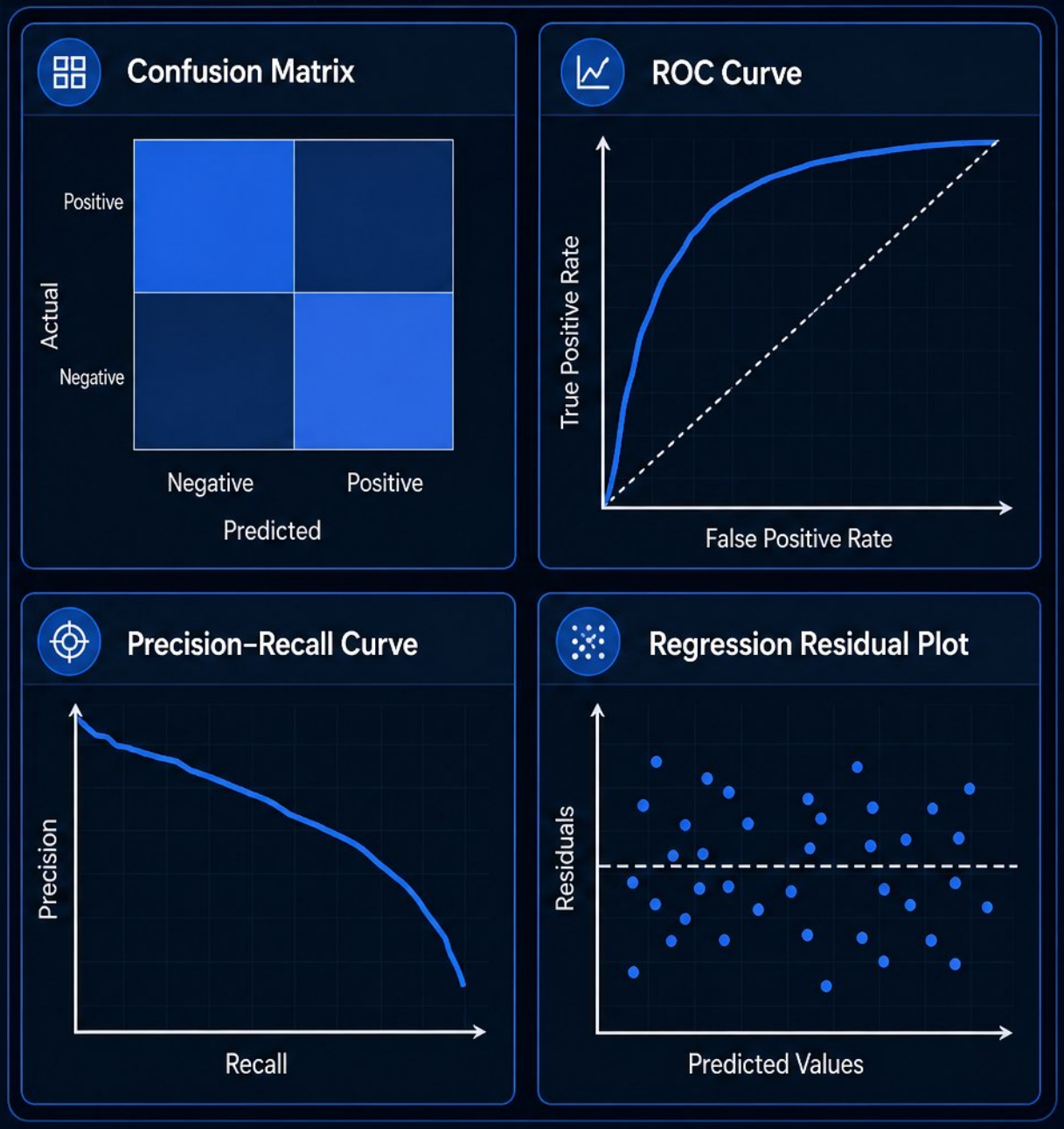


Hyperparameter Tuning with Grid and Random Search

- GridSearchCV exhaustively evaluates all combinations in a defined grid
- RandomizedSearchCV samples candidates efficiently for broad parameter spaces
- Use dunder syntax (e.g., model__C) to tune preprocessing and model parameters together
- Pipelines prevent data leakage during cross-validation and tuning
- Alternatives like Optuna offer efficient Bayesian-based search methods

Evaluation Beyond Accuracy

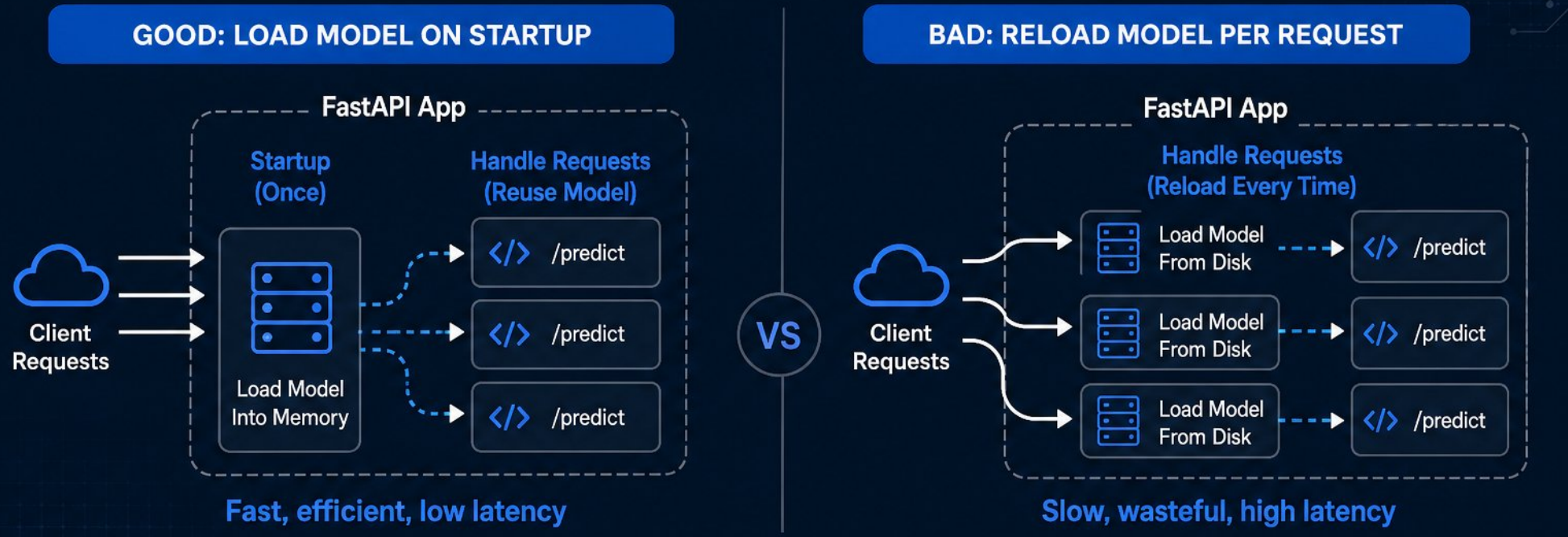
- Accuracy misleads in imbalanced datasets
- Precision, recall, F1, ROC-AUC for classification
- MAE, MSE, RMSE, R-squared for regression
- Confusion matrices and learning curves for diagnosis



Serializing and Versioning Models

- Save models with joblib or torch.save, storing full pipeline
- Embed sidecar metadata: training date, metrics, feature order
- Graduate to MLflow model registry for scalable versioning
- Never load serialized files from untrusted sources

Serving Models with FastAPI

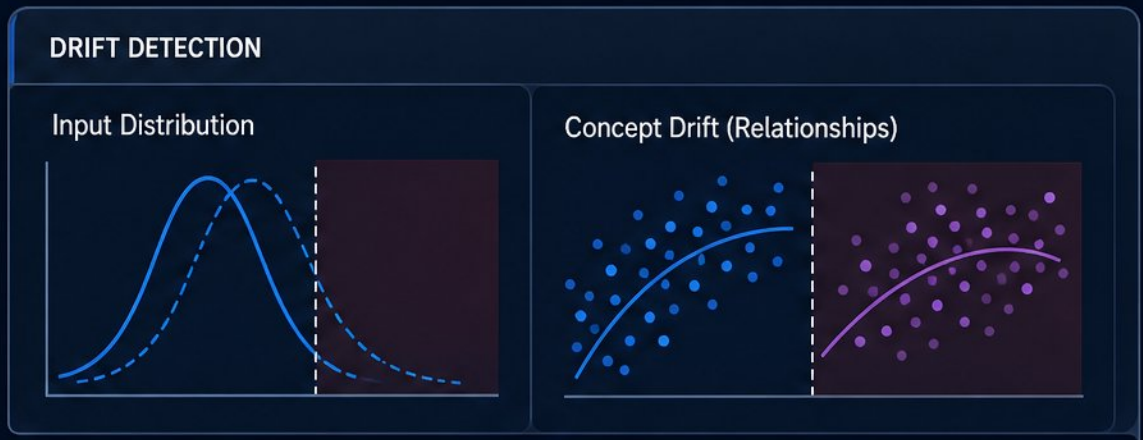


- FastAPI: lightweight, fast, and ideal for inference services
- Load models once at startup, not per request—prevents latency
- Pydantic automatically validates JSON and returns 422 errors
- Docker ensures reproducible deployments across environments
- Patterns: Cloud Run autoscaling, Kubernetes rolling updates, serverless

Monitoring and Maintaining Production Models

- Data drift: input distribution shifts; concept drift: relationships change
- Log predictions and inputs; track latency and error rates via Prometheus
- Health checks, load testing, and rolling deployment for zero downtime
- Establish baseline metrics; retrain on new data when drift is detected

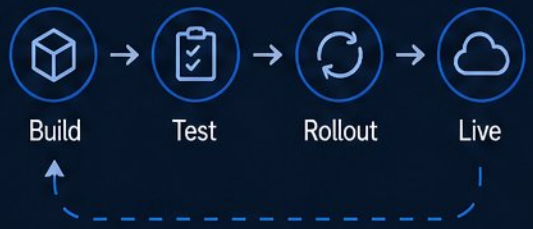
DRIFT DETECTION



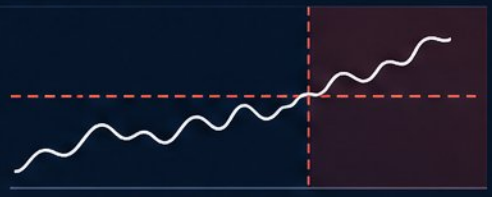
HEALTH CHECKS

SYSTEM MONITORING

DEPLOYMENT PIPELINE



RETRAIN TRIGGER



Retrain when drift threshold is crossed

PersonWise.

PersonWise • AI digital-human course platform

Scan the code or click anywhere to watch this course live, with voice Q&A

personwise.ai/t/1a5d452d/public