Term Project Proposal

Towards Intelligent Financial Fraud Detection: Exploring Modern Techniques for Existing Threats

Professor: Chantri Polprasert

Presented by:
Team Duo
st124997 - Suryansh Srivastava
st125457 - Ulugbek Shernazarov

Overview

ML Pipeline

```
# Modeling
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)
X_train.shape, X_test.shape
```

```
((5090096, 6), (1272524, 6))
```

Train test split

SMOTE

Pipeline design

BaseLine Model

ML Pipeline

```
categorical_transformer = Pipeline(steps=[
    ('encoder', OneHotEncoder()),
])

# No need for imputation - no missing values
numerical_transformer = Pipeline(steps=[
    ('scaler', StandardScaler())
])

preprocessor = ColumnTransformer(transformers=[
    ('num', numerical_transformer, num_columns),
    ('cat', categorical_transformer, cat_columns),
])

X_train_preprocessed = preprocessor.fit_transform(X_train)
X_test_preprocessed = preprocessor.transform(X_test)
```

Train test split

Pipeline design

SMOTE

BaseLine Model

ML Pipeline

```python
smote = SMOTE(random_state=42)
X_train_preprocessed_resampled, y_train_resampled = smote.fit_resample(X_train_preprocessed, y_train)

print(f"Before smote class distribution: {collections.Counter(y_train)}\nAfter: {collections.Counter(y_train_resampled)}")
```

```
Before smote class distribution: Counter({0: 5083526, 1: 6570})
After: Counter({0: 5083526, 1: 5083526})
```

Train test split

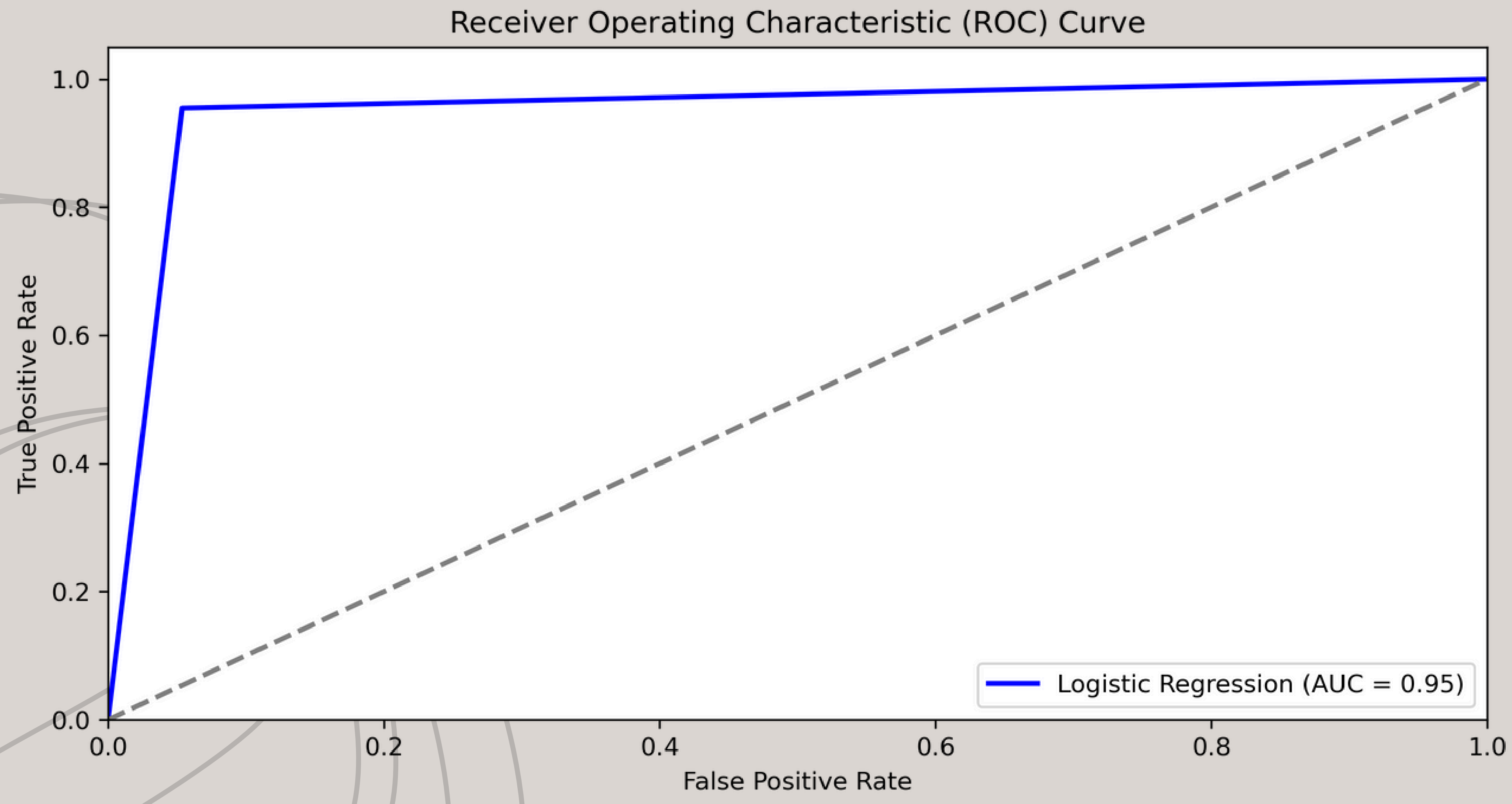
SMOTE

Pipeline design

BaseLine Model

ML Pipeline

```
model = LogisticRegression(class_weight='balanced', random_state=42)
model.fit(X_train_preprocessed_resampled, y_train_resampled)

y_pred = model.predict(X_test_preprocessed)
```



Train test split

Pipeline design

SMOTE

BaseLine Model

ML Models

Models:

- Logistic Regression, Support Vector Machine Classifier, Random Forest
- Gradient Boosting Classifier
- Feedforward Neural Network
- Kolmogorov Arnolds Networks

Param search

```
param_grid = [
    {
        'classifier': [LogisticRegression(max_iter=500)],
        'classifier__C': [0.1, 1, 10]
    },
    {
        'classifier': [SVC()],
        'classifier__C': [0.1, 1, 10],
        'classifier__kernel': ['linear', 'rbf']
    },
    {
        'classifier': [RandomForestClassifier()],
        'classifier__n_estimators': [100, 200],
        'classifier__max_depth': [10, 20],
        'classifier__min_samples_split': [2, 5, 10],
        'classifier__min_samples_leaf': [1, 2, 4],
        'classifier__bootstrap': [True, False]
    },
    {
        'classifier': [GradientBoostingClassifier()],
        'classifier__n_estimators': [100, 200],
        'classifier__learning_rate': [0.01, 0.1, 0.2],
        'classifier__max_depth': [3, 5, 7],
        'classifier__min_samples_split': [2, 5],
        'classifier__min_samples_leaf': [1, 2]
    }
]
```

```
model_pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression())
])
```

```
sgf=StratifiedGroupKFold(n_splits=5)

grid_search = GridSearchCV(
    estimator=model_pipeline,
    param_grid=param_grid,
    cv=sgf,
    scoring='roc_auc',
    verbose=2,
    n_jobs=-1  # Use parallel processi
)
```

ML Models

Data size: >10m rows

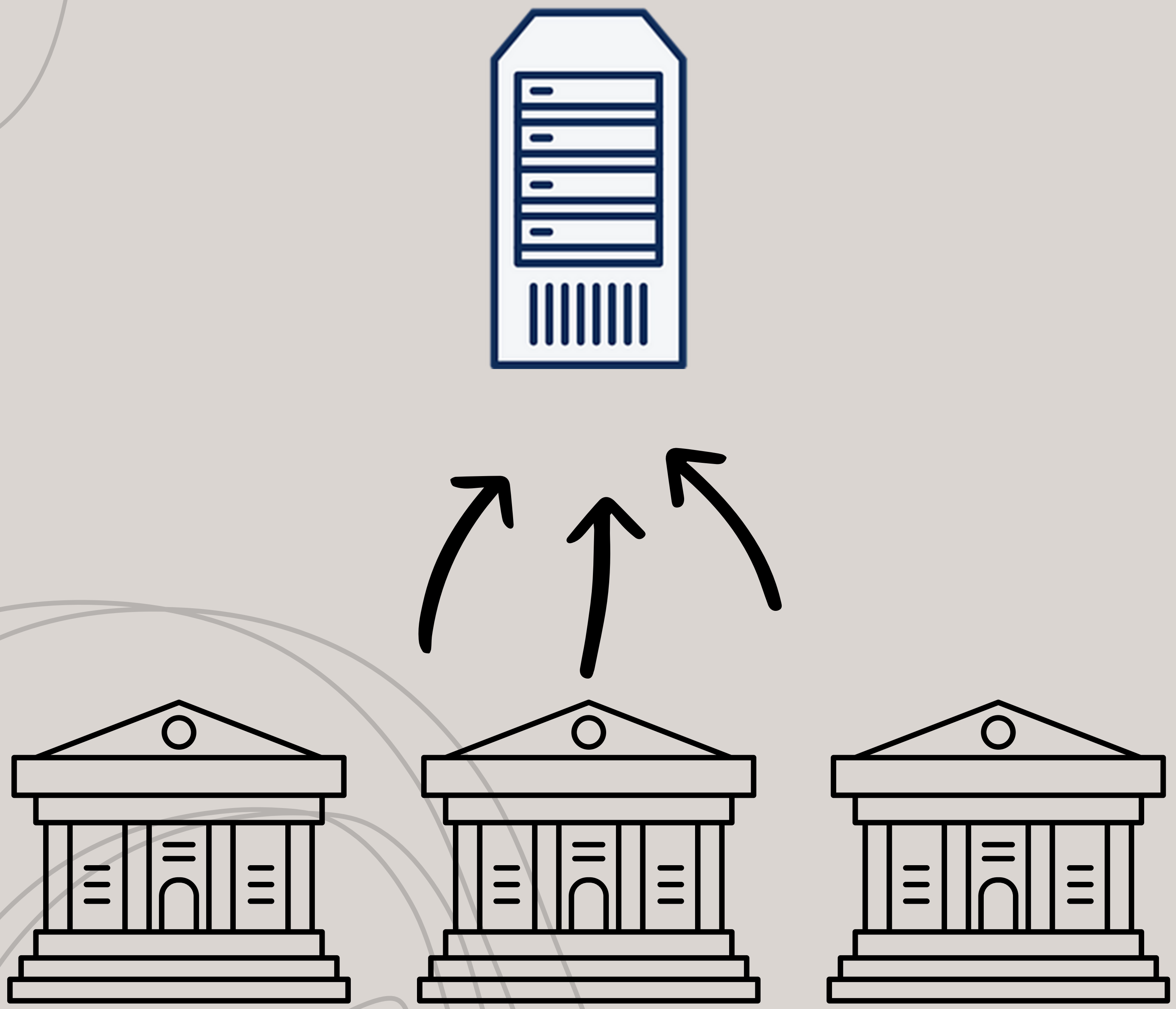
Grid search run took almost 50 hours

To avoid data leakage, on the fly preprocessing pipeline is given to each stratified kfold

Scoring focus is auc_roc

Also run gridsearch for each model to find best params per model

Federated Learning



- Decentralized Machine Learning
- Data Privacy Preserved
- Training Across Multiple Devices
- No Raw Data Sharing
- Collaborative Model Updates
- Secure Aggregation
- Ideal for Sensitive Data

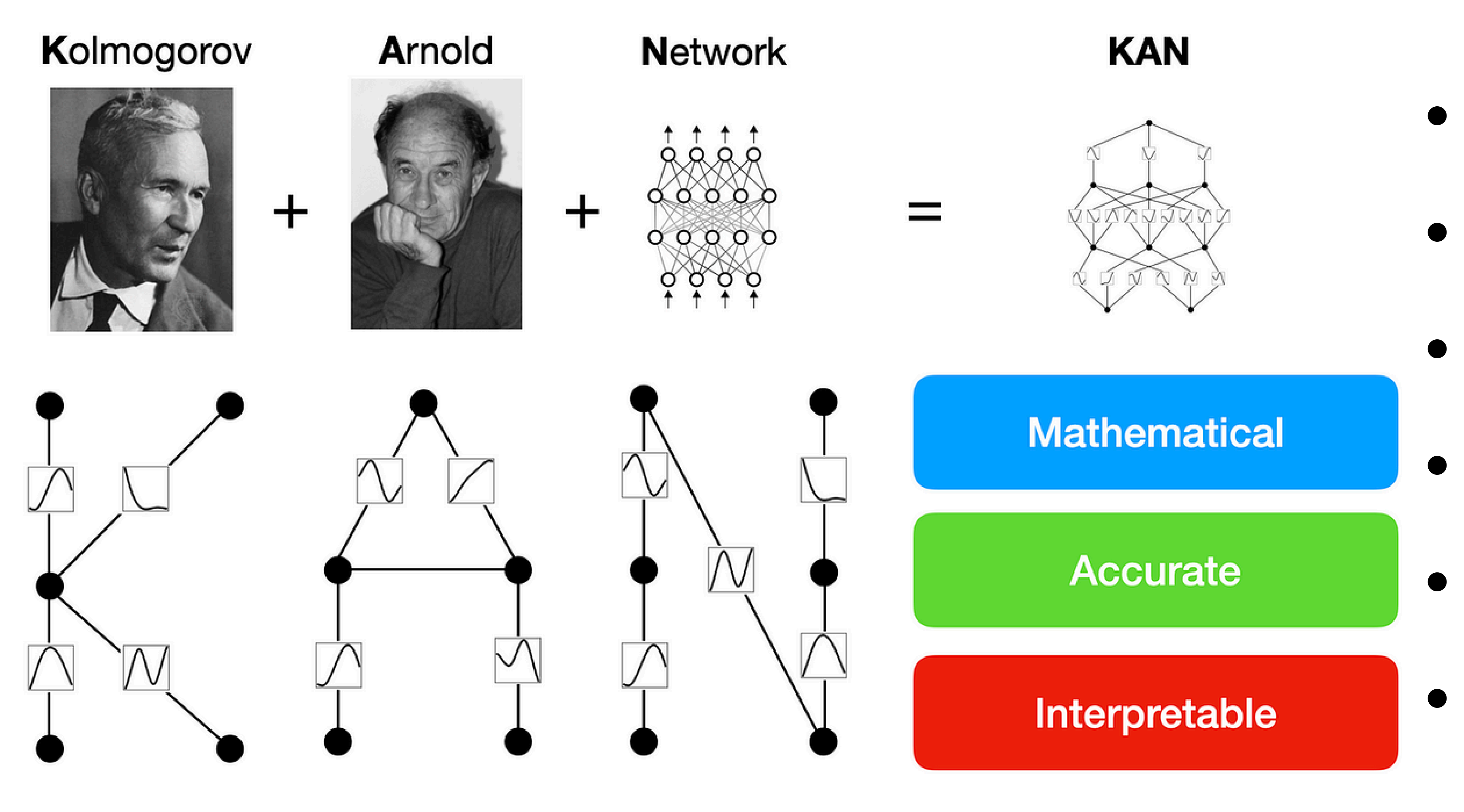
Neural network

```python
class Ff1(nn.Module):
    def __init__(self, num_features, embed_dim):
        super(Ff1, self).__init__()
        self.embedding = nn.Embedding(num_features, embed_dim)
        self.fc1 = nn.Linear(embed_dim, 64)
        self.fc2 = nn.Linear(64, 32)
        self.fc3 = nn.Linear(32, 2)

    def forward(self, x):
        x = self.embedding(x).view(-1, self.embed_dim)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

```python
class Ff2(nn.Module):
    def __init__(self, input_dim, hidden_dims=[64, 32], dropout=0.2):
        super(Ff2, self).__init__()
        self.layers = nn.ModuleList()
        self.batch_norms = nn.ModuleList()

        self.layers.append(nn.Linear(input_dim, hidden_dims[0]))
        self.batch_norms.append(nn.BatchNorm1d(hidden_dims[0]))

        for i in range(1, len(hidden_dims)):
            self.layers.append(nn.Linear(hidden_dims[i-1], hidden_dims[i]))
            self.batch_norms.append(nn.BatchNorm1d(hidden_dims[i]))

        self.output = nn.Linear(hidden_dims[-1], 2)

        self.dropout = nn.Dropout(dropout)
```

Tested several custom architectures

Kolmogorov Arnold Networks



- Inspired by Kolmogorov–Arnold Theorem
- Function Approximation Framework
- Decomposes Multivariate Functions
- High Theoretical Expressivity
- Layered Structure of Simple Functions
- Efficient for Complex Tasks

Model Evaluation

- Precision/Recall
- AUC-ROC
- Confussion matrix

Model Evaluation

Logistic Regression - C=10, num_epochs=1000

The Logistic Regression model achieved the following performance metrics:

AUC-ROC: 0.95

TABLE I

LOGISTIC REGRESSION RESULTS

Class	P	Recall	F1-Score	Support
0	0.9999	0.9462	0.9724	1,270,881
1	0.0225	0.9550	0.0439	1,643
Accuracy	0.9463			
Macro Avg	0.5112	0.9506	0.5081	1,272,524
Weighted Avg	0.9987	0.9463	0.9712	1,272,524

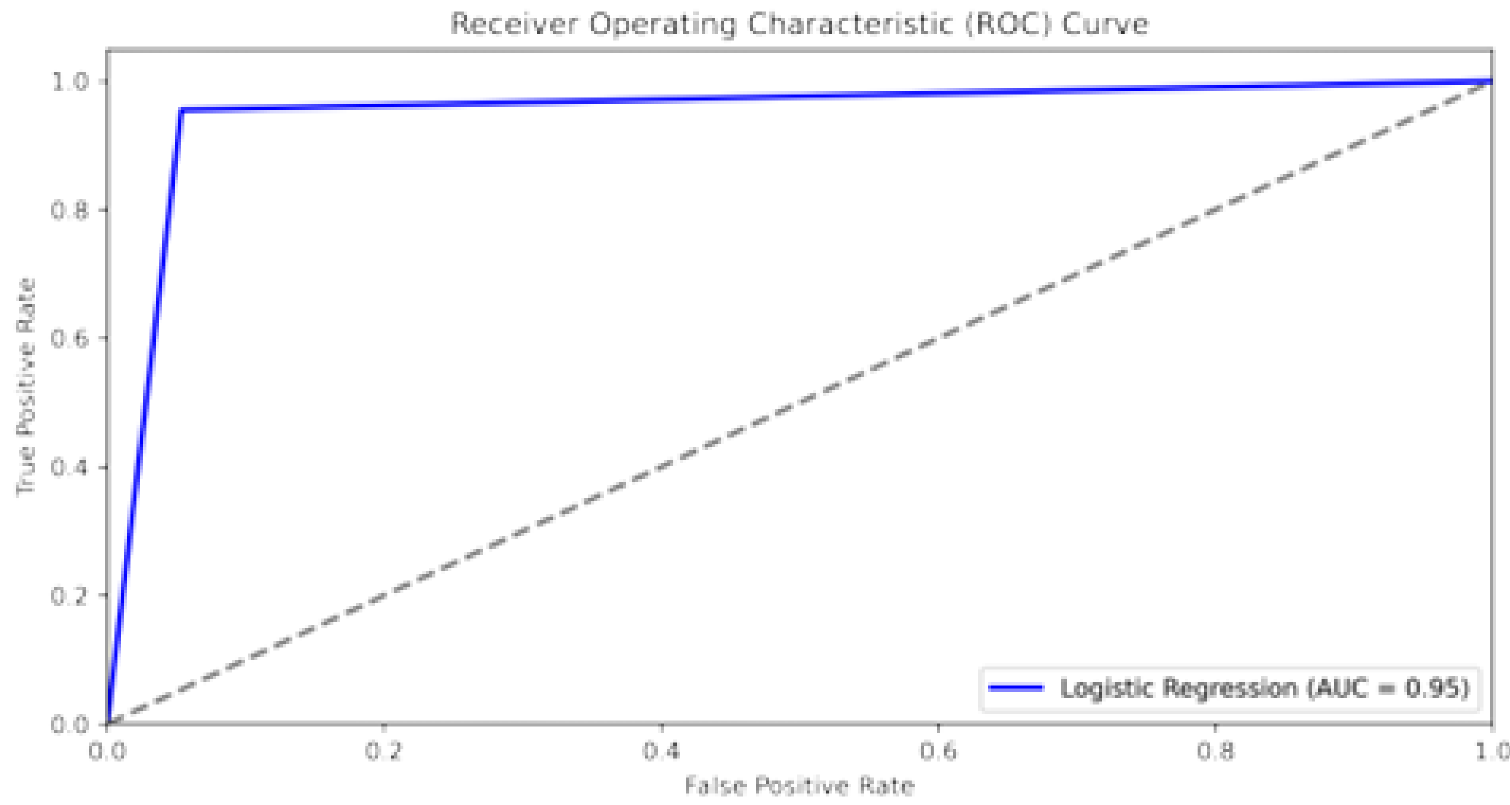


Fig. 7. ROC Curve for Baseline Logistic Regression Model

Model Evaluation

TABLE III

RANDOM FOREST RESULTS

Class	Precision	Recall	F1-Score	Support
0	0.9999	0.9835	0.9917	1,270,881
1	0.0723	0.9939	0.1348	1,643
Accuracy	0.9835			
Macro Avg	0.5361	0.9887	0.5632	1,272,524
Weighted Avg	0.9988	0.9835	0.9906	1,272,524

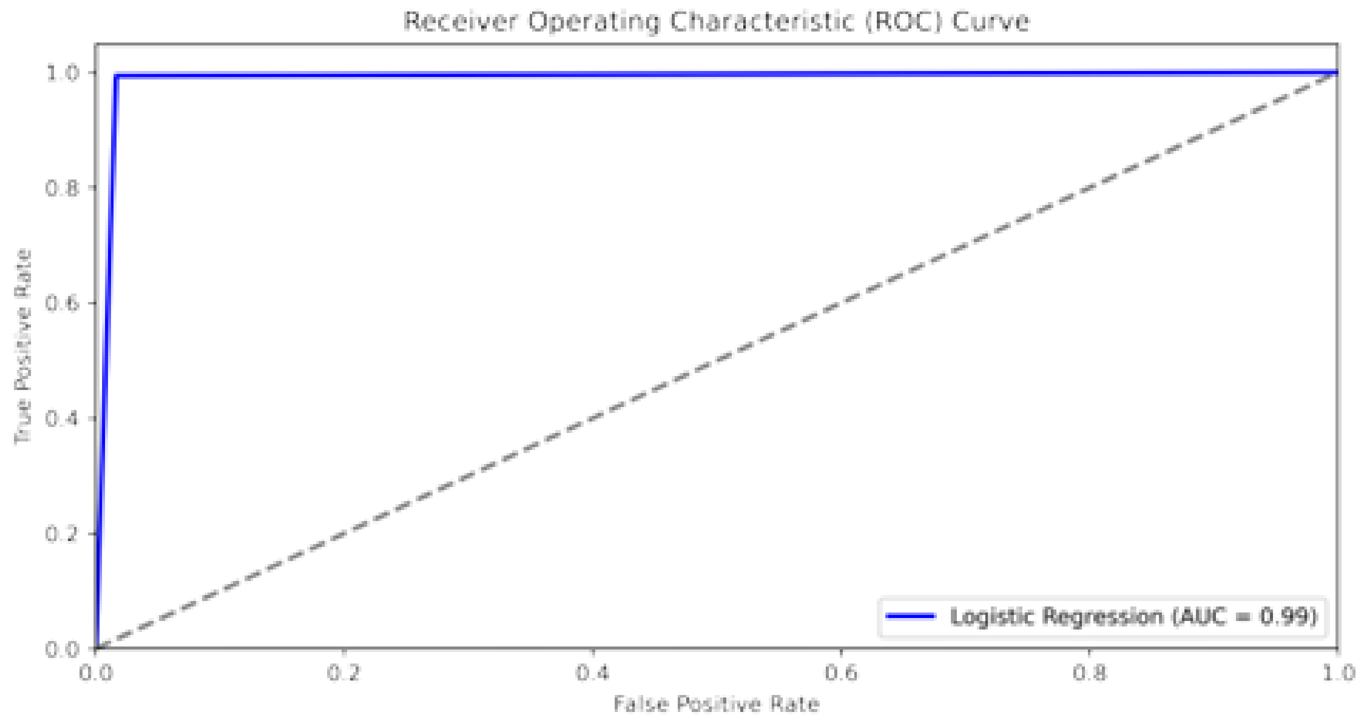


Fig. 10. ROC Curve for Random Forest Model

Random Forest with n_estimators=200, max_depth=10, min_split=2, min_leaf=2, bootstrap=True

Random Forest classifier achieved the second-best results, with the following metrics:

- AUC: 0.989

Model Evaluation

TABLE II

GRADIENT BOOSTING RESULTS

Class	Precision	Recall	F1-Score	Support
0	0.9999	0.9827	0.9913	1,270,881
1	0.0693	0.9976	0.1295	1,643
Accuracy	0.9827			
Macro Avg	0.5346	0.9901	0.5604	1,272,524
Weighted Avg	0.9988	0.9827	0.9901	1,272,524

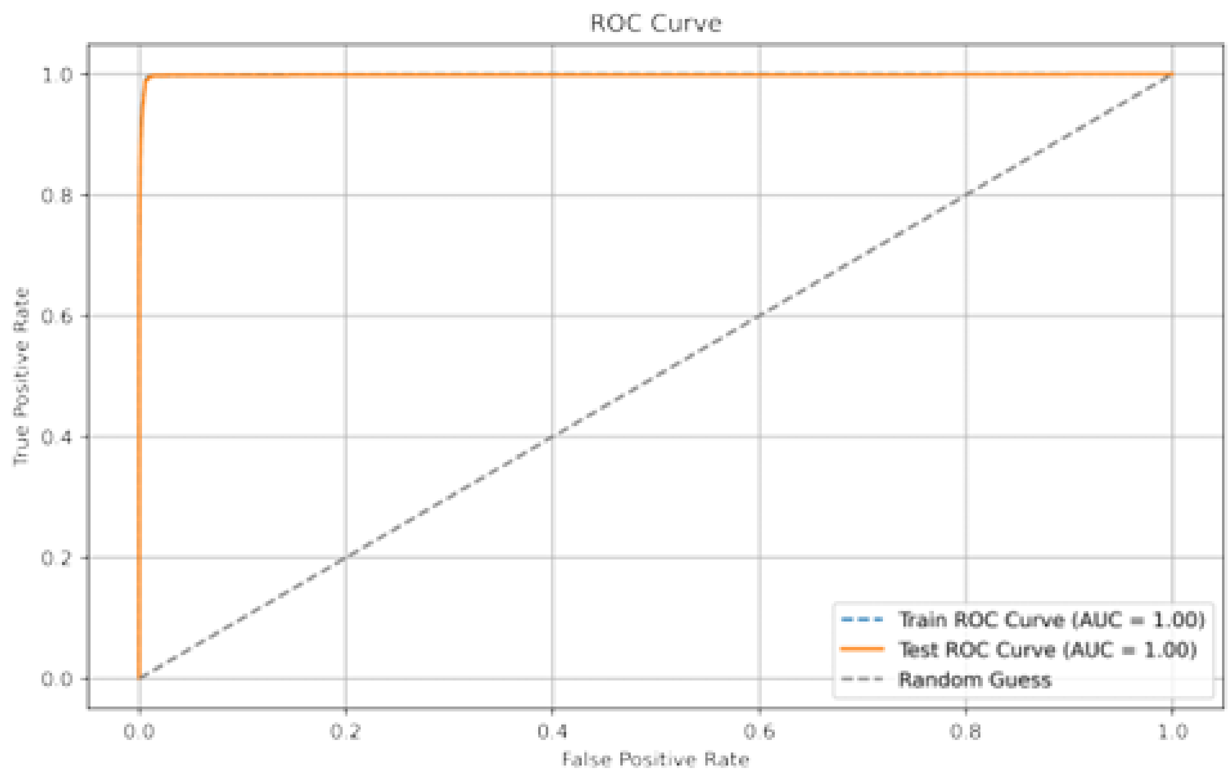


Fig. 8. ROC Curve for Gradient Boosting Model

Gradient Boosting with n_estimators=200, lr=0.01, max_depth=7, min_split=2, min_leaf=2

Gradient Boosting provided the best results, achieving superior performance on both training and test sets:

- Train AUC: 0.9989
- Test AUC: 0.9983

Model Evaluation

Using a Neural Feedforward Network within the Feder ated Learning framework, the model achieved the follow ing performance metrics:

• AUC-ROC: 0.9979
• Balanced Accuracy: 0.9781
• Accuracy: 0.9801

$$\begin{bmatrix} 1220320 & 50963 \\ 4768 & 1265712 \end{bmatrix}$$

NEURAL FEEDFORWARD NETWORK REPORT

Class	Precision	Recall	F1-Score	Support
0	0.9961	0.9601	0.9778	1,270,283
1	0.9613	0.9963	0.9785	1,270,480
Accuracy	0.9801			
Macro Avg	0.9787	0.9782	0.9781	2,540,763
Weighted Avg	0.9787	0.9801	0.9781	2,540,763

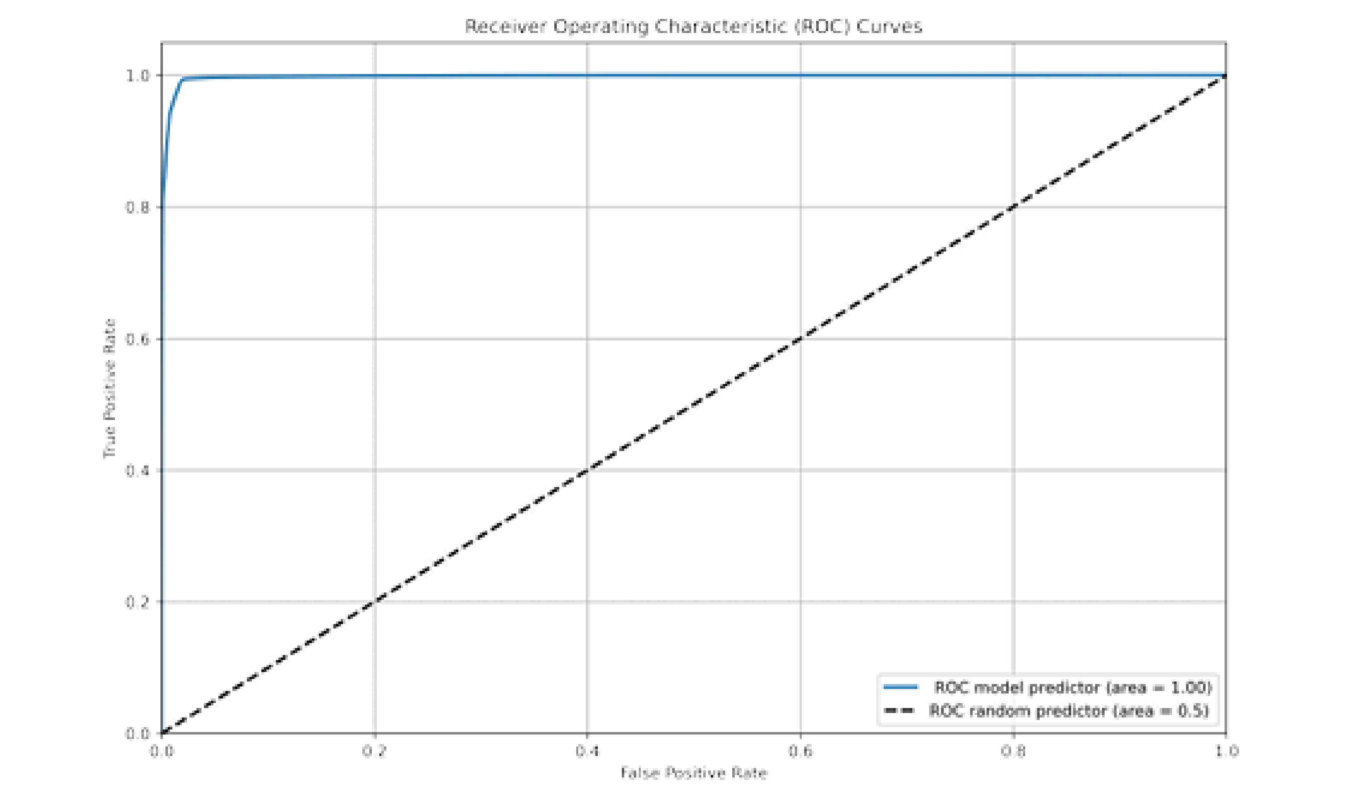


Fig. 11. ROC Curve for Neural Network with Federated Learning

Model Evaluation

Using Kolmogorov–Arnold Networks (KANs) within the Federated Learning framework, the model achieved the following performance metrics:

- AUC-ROC: 0.9992
- Precision: 0.1233
- Recall: 0.9970
- Accuracy: 0.9908

$$\begin{bmatrix} 1259231 & 11650 \\ 5 & 1638 \end{bmatrix}$$

TABLE V

FEDERATED LEARNING WITH KANS CLASSIFICATION REPORT

Class	Precision	Recall	F1-Score	Support
0	0.9999	0.9908	0.9954	1,270,881
1	0.1233	0.9970	0.2194	1,643
Accuracy	0.9908			
Macro Avg	0.5616	0.9939	0.6074	1,272,524
Weighted Avg	0.9989	0.9908	0.9944	1,272,524

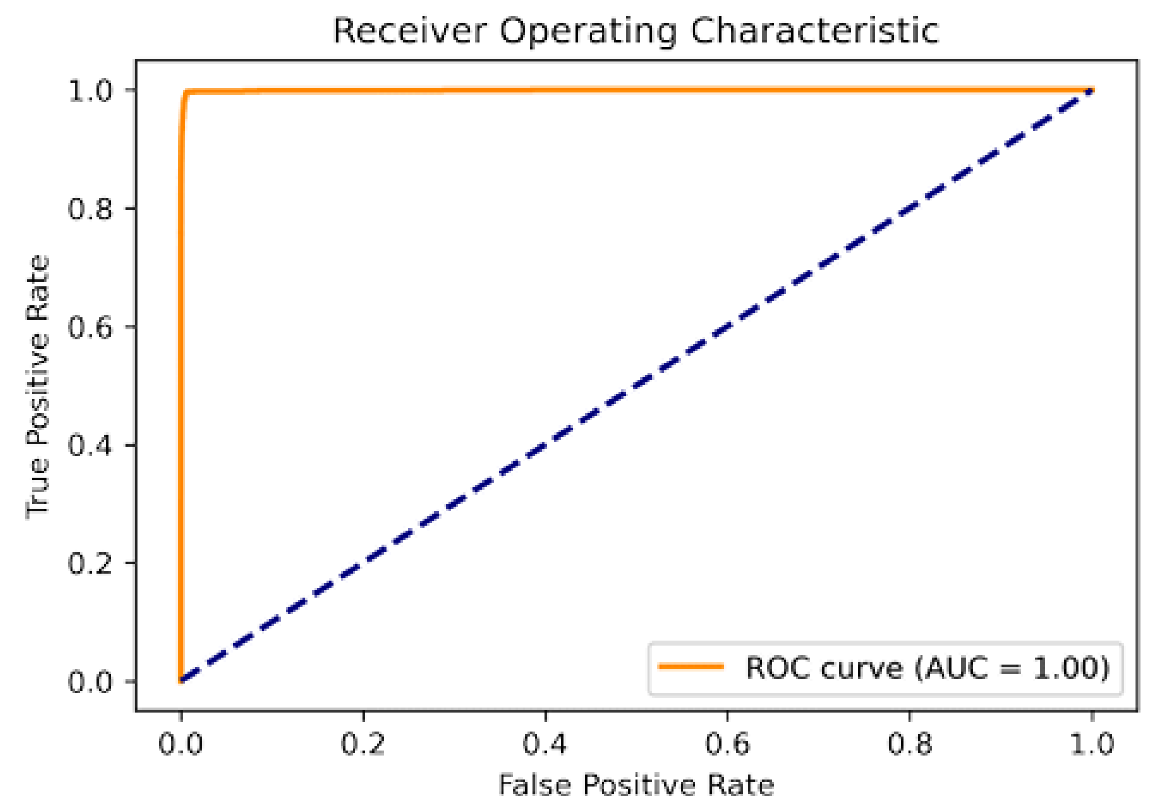


Fig. 14. Training Progress for Kolmogorov–Arnold Networks with Federated Learning

Model Evaluation

```python
formula1, formula2 = model.symbolic_formula()[0]
formula1
```

$$1.61\left(-\sin\left(0.55x_3+9.34\right)+0.12\tan\left(0.57x_1-9.4\right)+0.25\tanh\left(1.81x_2-4.89\right)-0.28\tanh\left(2.29x_4-4.25\right)+0.43\right)^2-153.19e^{0.12\sin\left(2.56x_1+3.0\right)+0.21\tanh\left(10.0x_4-8.8\right)-0.16\left|1.59x_3-6.98\right|}-0.01\sin\left(4.91\sin\left(1.43x_1+1.21\right)+0.01\tan\left(0.75x_4-3.48\right)-1.43\tanh\left(2.67x_2-10.0\right)+2.27\tanh\left(2.18x_3-9.12\right)+9.85\right)+63.66\tanh\left(248.65\left(1-0.16x_1\right)^4+0.36\sin\left(2.83x_4+1.75\right)-1.56\tanh\left(1.13x_3-3.4\right)-0.23\left|3.23x_2-9.88\right|+1.2\right)+14.34$$

$$1.61\left(-\sin\left(0.55x_3+9.34\right)+0.12\tan\left(0.57x_1-9.4\right)+0.25\tanh\left(1.81x_2-4.89\right)-0.28\tanh\left(2.29x_4-4.25\right)+0.43\right)^2-153.19e^{0.12\sin\left(2.56x_1+3.0\right)+0.21\tanh\left(10.0x_4-8.8\right)-0.16\left|1.59x_3-6.98\right|}-0.01\sin\left(4.91\sin\left(1.43x_1+1.21\right)+0.01\tan\left(0.75x_4-3.48\right)-1.43\tanh\left(2.67x_2-10.0\right)+2.27\tanh\left(2.18x_3-9.12\right)+9.85\right)+63.66\tanh\left(248.65\left(1-0.16x_1\right)^4+0.36\sin\left(2.83x_4+1.75\right)-1.56\tanh\left(1.13x_3-3.4\right)-0.23\left|3.23x_2-9.88\right|+1.2\right)+14.34$$

Discussions

TABLE VII

PERFORMANCE OF PAYSIM MODEL

Metric	Precision (Fraud)	Recall (Fraud)	F1-Score (Fraud)	AUC-ROC
PaySim Model	1.0000	0.0019	0.0039	0.5010

- Compare to Paysim Model
- Compare to Kaggle one of the best solutions

$$\begin{bmatrix} 6,354,407 & 0 \\ 8,197 & 16 \end{bmatrix}$$

Paysim dataset has isFraud and isFlaggedFraud

PaySim Model Analysis:

- Perfect precision (1.0000), but extremely low recall (0.0019).
- F1-score is very low (0.0039), indicating poor balance.
- AUC-ROC (0.5010) is close to random guessing.

Our Solution:
Significantly outperforms PaySim across all key metrics.

Discussions

https://www.kaggle.com/code/waleedfaheem/credit-card-fraud-detection-auc-0-9#-Model-Building-

- Compare to Paysim Model
- Compare to Kaggle one of the best solutions

TABLE IX

COMPARISON OF OUR SOLUTION AND KAGGLE SOLUTION

Model	Pr	Rec	F1-Score	AUC-ROC
Kaggle Solution	0.9700	0.7700	0.8600	0.9750
Neural Feedforward Network (FL)	0.9613	0.9963	0.9785	0.9979
KANs (FL)	0.1233	0.9970	0.2194	0.9992

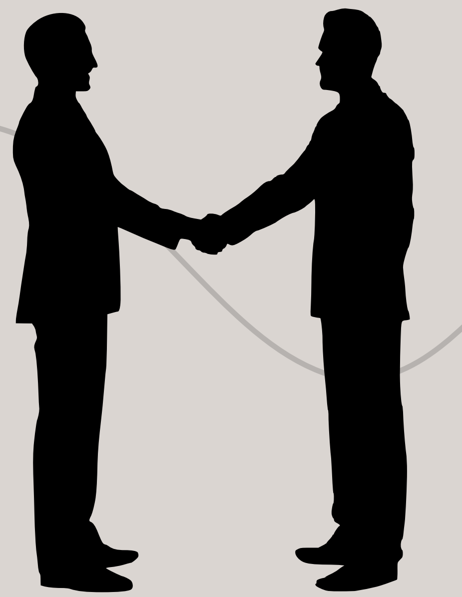
Kaggle solution, authored by Waleed Faheem, analysis:

- AUC-ROC: Neural Network (0.9979) and KANs (0.9992) outperform Kaggle solution (0.975).
- Recall: Neural Network (0.9963) and KANs (0.9970) excel vs. Kaggle (0.7700), identifying nearly all fraudulent cases.
- Precision: Kaggle (0.9700) slightly higher than Neural Network (0.9613), but recall and F1 improvements outweigh this.

Our Solution:
Kaggle model is strong, but our Neural Network and KANs deliver superior performance across key metrics, ensuring robust fraud detection.

Conclusion

- Developed and evaluated various models for fraud detection using the PaySim dataset.
- Federated Learning-based Models
- Achieved superior performance compared to baseline models and existing solutions.
- Implemented a user-friendly Web Platform (Model Zoo)
- KANs + Federated learning - new research approach not explored yet
- Allows users to select and utilize different trained models



Demo

https://risk-radar.online

Thank You

Presented by Suryansh & Ulugbek