Problem Selection & Justification

Problem Statement

The project focuses on **detecting fraudulent financial transactions in real time using deep learning models**. The system ingests transaction features such as transaction type, amount, account balances and destination details, then produces a fraud probability score.

Based on an optimized threshold, the system raises alerts for suspected fraud. This approach aims to reduce financial losses, protect customers and increase trust in online payment systems and banking institutions.

Justification

Economic Impact

- Fraudulent transactions represent a **multi-billion-dollar global problem**. Banks, merchants and consumers collectively bear the costs of chargebacks, reimbursements and insurance claims.
- Beyond direct losses, institutions incur **indirect costs** such as fraud investigations, litigation and compliance penalties.
- Real-time detection prevents losses at the transaction level, reducing downstream operational expenses. For example, catching fraud before settlement avoids costly reversals and preserves liquidity.
- By leveraging deep learning, institutions can **minimize false negatives** (missed fraud) and **reduce false positives** (legitimate transactions incorrectly flagged), optimizing both financial protection and customer experience.

Human Impact

- Victims of fraud often face **temporary loss of funds**, which can disrupt daily life and financial stability.
- Fraud can cause **reputational harm**, especially for small businesses or individuals whose accounts are compromised.
- Service disruptions, such as frozen accounts or lengthy verification processes, erode customer trust.

- A faster, more accurate fraud detection system directly safeguards customers, ensuring smoother transaction flows and restoring confidence in digital banking.

Operational Impact

- Traditional fraud detection relies heavily on **manual reviews and rule-based systems**. These are resource-intensive, slow and rigid.

- Rule-based systems often generate **high false positive rates**, blocking legitimate transactions and frustrating customers.

- Fraud analysts spend significant time reviewing flagged cases, creating bottlenecks in transaction processing.

- Real-time ANN-based scoring improves throughput by automatically learning complex fraud patterns, reducing manual workload and lowering review queues.

- This leads to **reduced downtime**, faster transaction approvals and improved operational efficiency.

Relevance

- Fraud tactics evolve rapidly, exploiting new vulnerabilities in payment systems. Rule-based systems cannot adapt quickly enough to these nonlinear, dynamic patterns.

- Artificial Neural Networks (ANNs) excel at learning **high-dimensional, nonlinear relationships** in data. They adapt through continuous training, making them suitable for financial risk contexts where fraud behaviors change frequently.

Dataset Choice & Validation

Datasets Used

Primary Project Dataset: Synthetic_Financial_datasets_log.csv

Structure: Tabular transaction logs containing key attributes:

- `amount` – transaction value.
- `oldbalanceOrg` / `newbalanceOrig` – sender's balance before and after transaction.
- `oldbalanceDest` / `newbalanceDest` – recipient's balance before and after transaction.
- `type` – transaction type (encoded via one-hot encoding for categorical processing).
- `isFraud` – target label (binary: 1 = fraud, 0 = legitimate).

Data Cleaning: Personally identifying fields (`nameOrig`, `nameDest`, `isFlaggedFraud`) were removed to prevent leakage and ensure privacy compliance.

Class Imbalance: Fraudulent transactions represent **less than 1% of total records**, closely mirroring real-world financial datasets where fraud is rare but costly.

Suitability: Provides realistic conditions for training fraud detection models, including imbalance, multiple numeric features and categorical transaction types.

Benchmark References (for literature context and optional cross-checks)

Financial Fraud Detection Dataset (Kaggle)

- Contains labeled financial transactions for fraud detection research.
- Used as a **benchmark reference** to validate methodology and compare performance metrics.

    - Ensures that the project aligns with established datasets in academic and industry contexts.

Validation Process

Preprocessing & Encoding

- Dropped identifiers (`nameOrig`, `nameDest`, `isFlaggedFraud`) to avoid bias and leakage.
- Applied **one-hot encoding** to `type` for categorical representation.
- Standardized numerical features using **StandardScaler** to normalize distributions and improve training stability.

Class Imbalance Handling

- Fraud rate confirmed at <1%.
- Applied **class weighting** during training to penalize misclassification of fraud cases more heavily.
- Conducted **threshold tuning** to optimize recall vs. precision trade-off.
- Considered **SMOTE oversampling** as an alternative balancing technique (documented but not applied in the final pipeline due to reliance on class weights).

Bias & Privacy Compliance

- Verified that datasets are anonymized and free of personally identifiable information (PII).
- Ensured compliance with ethical standards for academic use.
- Checked for potential bias in transaction types and balances to avoid skewed model learning.

Data Splitting Strategy

- **Train/Test Split:** 80% training, 20% test, stratified by fraud label to preserve class distribution.
- **Validation Split:** 20% of training data reserved during training for monitoring and early stopping.
- Ensured **no data leakage** by fitting scalers only on training data and applying them to test data separately.

Model Architecture Selection

Design Rationale

Fraud detection in tabular transactional data requires models that can capture **nonlinear interactions** among numerical balances, transaction amounts and categorical transaction types. The chosen architecture leverages the following design principles:

Deep Multi-Layer Perceptrons (MLPs):

- MLPs are well-suited for tabular data with mixed numerical and categorical features.
- They can learn complex, nonlinear relationships between transaction attributes and fraud labels.
- Multiple hidden layers allow hierarchical feature extraction, improving the model’s ability to detect subtle fraud patterns.

Batch Normalization:

- Applied after each hidden layer to stabilize training and accelerate convergence.
- Reduces internal covariate shift, ensuring consistent gradient flow across layers.
- Improves generalization by normalizing activations, especially important in imbalanced datasets.

Dropout Regularization:

- Introduced after each hidden layer to mitigate overfitting.
- Randomly deactivates neurons during training, forcing the network to learn more robust representations.
- Particularly effective in fraud detection, where fraudulent cases are rare and models risk memorizing noise.

Sigmoid Output Layer:

- Produces a probability score between 0 and 1 for fraud likelihood.
- Enables threshold optimization to balance precision and recall.
- Supports business-driven tuning (stricter thresholds for high-risk transactions).

Final MLP Architecture (per configuration template)

Input Layer:

- Accepts a vector of standardized numeric features (amounts, balances) and one-hot encoded categorical features (transaction type).
- Ensures all features are normalized for stable training.

Hidden Stack:

- Four Dense blocks with **ReLU activations**, each followed by **BatchNormalization** and **Dropout**.
- Typical neuron configurations tested:
  - `[128, 64, 32, 16]` (balanced depth and width).
  - `[256, 128, 64, 32]` (deeper, higher-capacity network).
- Dropout configurations tested:

        - `[0.3, 0.3, 0.2, 0.2]` (moderate regularization).
        - `[0.5, 0.4, 0.3, 0.2]` (higher regularization to combat overfitting).

Output Layer:

- Dense(1, activation='sigmoid').
- Produces fraud probability for each transaction.

Loss Function & Metrics:

- **Loss:** Binary cross-entropy, appropriate for binary classification tasks.
- **Metrics Tracked:** Accuracy, Precision, Recall, F1-Score, and AUC.
- AUC (Area Under ROC Curve) is emphasized as a robust measure for imbalanced datasets, reflecting the model's ability to distinguish fraud from legitimate transactions across thresholds.

Model Training & Hyperparameter Tuning

Optimizer Choice

Primary Optimizer: Adam

- Configurable learning rate tested across multiple configurations.
- Selected for its ability to adapt per-parameter learning rates, making it highly effective for noisy and imbalanced datasets such as fraud detection.
- Provides faster convergence compared to traditional optimizers, reducing training time while maintaining stability.

Comparison:

- In this notebook, Adam was consistently used for all configurations.
- Prior comparative notes indicated that **SGD (Stochastic Gradient Descent)** was less performant in similar contexts, converging slower and producing lower AUC scores.

Training Configuration (Constant Across Trials)

Features: Standardized numeric features (amounts, balances) and one-hot encoded categorical features (transaction type).

Scaler: Persisted to `scaler.pkl` for reproducibility and deployment.

Class Weighting: Applied via `compute_class_weight` to rebalance the loss function and penalize misclassification of fraud cases more heavily.

Validation Split: 20% of training data reserved during `model.fit` for early stopping and learning rate scheduling feedback.

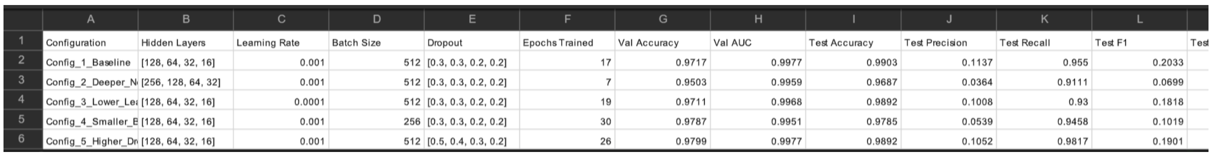
Callbacks:

- **EarlyStopping:** Monitors validation loss, patience=5, restores best weights to prevent overfitting.
- **ReduceLROnPlateau:** Monitors validation loss, reduces learning rate by factor=0.5 if plateau observed, minimum LR=1e-5.

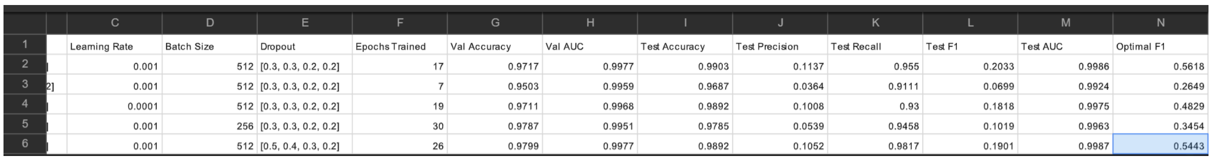
Random Seeds: NumPy and TensorFlow seeds set to 42 for reproducibility.

Artifacts Persisted:

- Trained models (`.keras` and .keras files per configuration).
- `hyperparameter_comparison.csv` (tabular comparison of metrics).



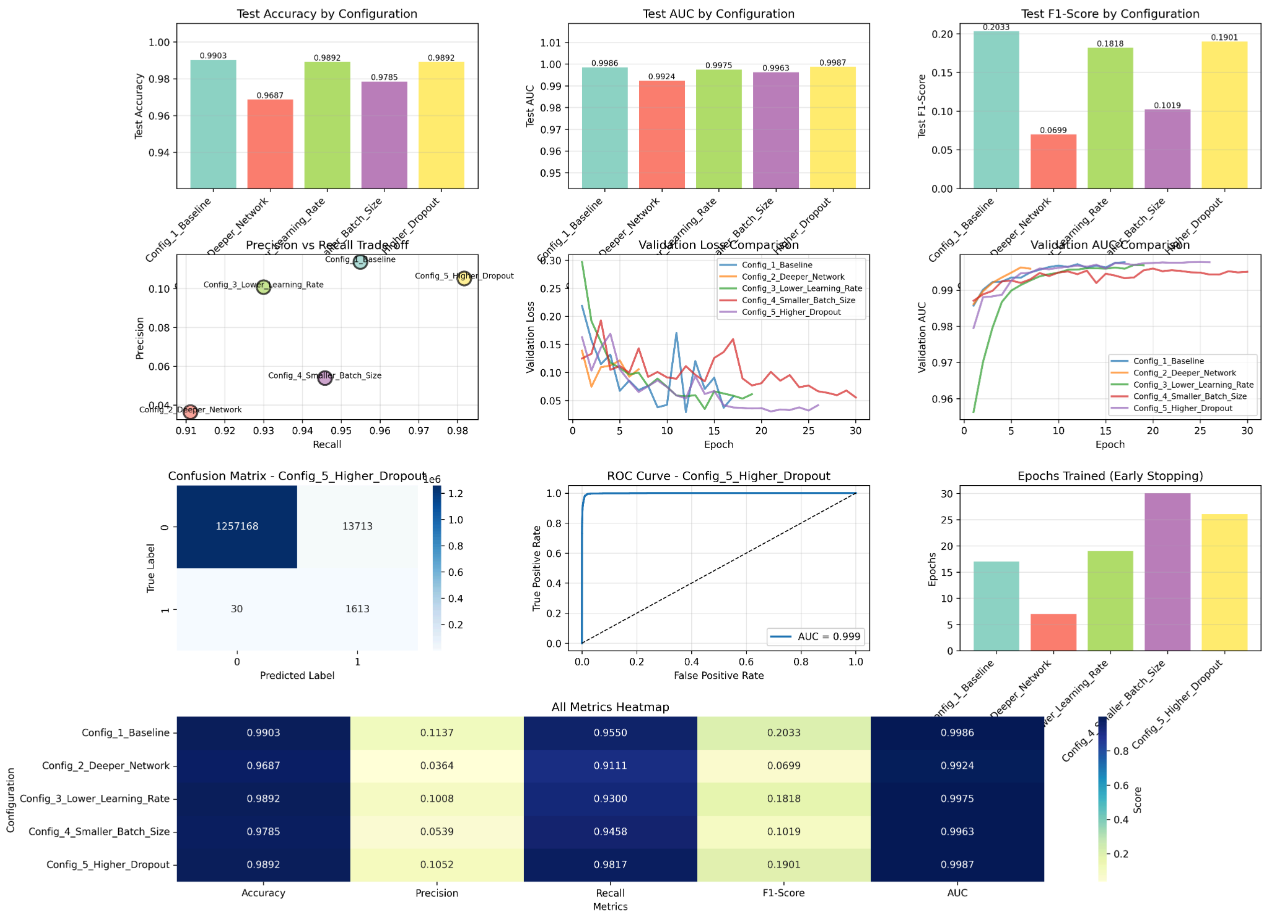
	A	B	C	D	E	F	G	H	I	J	K	L	
1	Configuration	Hidden Layers	Learning Rate	Batch Size	Dropout	Epochs Trained	Val Accuracy	Val AUC	Test Accuracy	Test Precision	Test Recall	Test F1	Tes
2	Config_1_Baseline	[128, 64, 32, 16]	0.001	512	[0.3, 0.3, 0.2, 0.2]	17	0.9717	0.9977	0.9903	0.1137	0.955	0.2033	
3	Config_2_Deeper_N	[256, 128, 64, 32]	0.001	512	[0.3, 0.3, 0.2, 0.2]	7	0.9503	0.9959	0.9687	0.0364	0.9111	0.0699	
4	Config_3_Lower_Le	[128, 64, 32, 16]	0.0001	512	[0.3, 0.3, 0.2, 0.2]	19	0.9711	0.9968	0.9892	0.1008	0.93	0.1818	
5	Config_4_Smaller_E	[128, 64, 32, 16]	0.001	256	[0.3, 0.3, 0.2, 0.2]	30	0.9787	0.9951	0.9785	0.0539	0.9458	0.1019	
6	Config_5_Higher_Dr	[128, 64, 32, 16]	0.001	512	[0.5, 0.4, 0.3, 0.2]	26	0.9799	0.9977	0.9892	0.1052	0.9817	0.1901	



		C	D	E	F	G	H	I	J	K	L	M	N
1		Learning Rate	Batch Size	Dropout	Epochs Trained	Val Accuracy	Val AUC	Test Accuracy	Test Precision	Test Recall	Test F1	Test AUC	Optimal F1
2		0.001	512	[0.3, 0.3, 0.2, 0.2]	17	0.9717	0.9977	0.9903	0.1137	0.955	0.2033	0.9986	0.5618
3	2]	0.001	512	[0.3, 0.3, 0.2, 0.2]	7	0.9503	0.9959	0.9687	0.0364	0.9111	0.0699	0.9924	0.2649
4		0.0001	512	[0.3, 0.3, 0.2, 0.2]	19	0.9711	0.9968	0.9892	0.1008	0.93	0.1818	0.9975	0.4829
5		0.001	256	[0.3, 0.3, 0.2, 0.2]	30	0.9787	0.9951	0.9785	0.0539	0.9458	0.1019	0.9963	0.3454
6		0.001	512	[0.5, 0.4, 0.3, 0.2]	26	0.9799	0.9977	0.9892	0.1052	0.9817	0.1901	0.9987	0.5443

- `hyperparameter_tuning_results.json` (detailed metrics per configuration).

- `hyperparameter_tuning_results.png` (visualizations of metrics and training curves).



- `scaler.pkl` (feature scaling object).

```
€♦•=☺     Œ←sklearn.preprocessing._data"Œ♫StandardScaler""") "}"(Œ
with_mean"ˆŒ▮with_std"ˆŒ♦copy"ˆŒ◄feature_names_in_"Œ‼joblib.numpy_pickle"Œ◄NumpyArrayWrapper""")
"}"(Œ▮subclass"Œ♣numpy"Œ•ndarray"""Œ♣shape"K
…"Œ♣order"Œ☺C"Œ♣dtype"h☼Œ♣dtype"""Œ☻O8"‰ˆ‡"R"(K♥Œ☺|"NNNJÿÿÿÿJÿÿÿÿK?t"bŒ
allow_mmap"‰Œ←numpy_array_alignment_bytes"K►ub€☻cnumpy._core.multiarray
_reconstruct
q cnumpy
ndarray
q☺K …q☻c_codecs
encode
q♥X☺   bq♦X♣   latin1q♣†q♠Rq•‡q▮Rq (K☺K
…q
cnumpy
dtype
q♂X☻   O8q♀‰ˆ‡q
Rq♫(K♥X☺   |q☼NNNJÿÿÿÿJÿÿÿÿK?tq►b‰]q◄(X♦   stepq↕X♠   amountq‼X
  oldbalanceOrgq¶X♫   newbalanceOrigq§X♫   oldbalanceDestq▬X♫   newbalanceDestq↨X
  type_CASH_OUTq↑X
   type_DEBITq↓X♀   type_PAYMENTq→X
  type_TRANSFERq←etq∟b.•Ë       Œ♫n_features_in_"
Œ☼n_samples_seen_"Œ▬numpy._core.multiarray"Œ♠scalar"""h↑Œ☻i8"‰ˆ‡"R"(K♥Œ☺<"NNNJÿÿÿÿJÿÿÿÿK t"bC▮0«M
"†"R"Œ♣mean_"h♂) "}"(h♫h◄h↕K
…"h¶h§h▬h↑Œ☻f8"‰ˆ‡"R"(K♥h(NNNJÿÿÿÿJÿÿÿÿK t"bh▲ˆh▼K►ub       ÿÿÿÿÿÿÿÿõ¶/›škn@Yšè
—,ö♣A¾▼ìÄ¡q)Aò.³AD↨*A]öŒm_Ê0Aä¥ÕyY°2ALñ▼Ë£,Ö?¯↨#=c¬z?v™ÄN¤Ö?←¯
ÑÀqµ?•*       Œ♦var_"h♂) "}"(h♫h◄h↕K
…"h¶h§h▬h3h▲ˆh▼K►ub♀ÿÿÿÿÿÿÿÿÿÿÿÿoÄàk»ÇÓ@↓g Üt UBÎ
±d—UžBk¯ÈÔA↨ŸBh,œª-↨¥BæÚ¯è"—¨BàOÜ▮'/Í?'[D¹ë  z?DðØ)ª¥Ì?®(õ¢ä¥³?•,       Œ♠scale_"h♂) "}"(h♫h◄h↕K
…"h¶h§h▬h3h▲ˆh▼K►ub
ÿÿÿÿÿÿÿÿÿÿ·&8ã;Êa@[ú  ³b"A ³è Ø•FAy.Å"½MFAR(+)÷IA1• mv
LA¥U„´~ Þ?êdÈôT—´?äïAðõFÞ?")♦ÇýºÑ?•▲       Œ►_sklearn_version"Œ♣1.6.1"ub.
```

Recorded Training Results

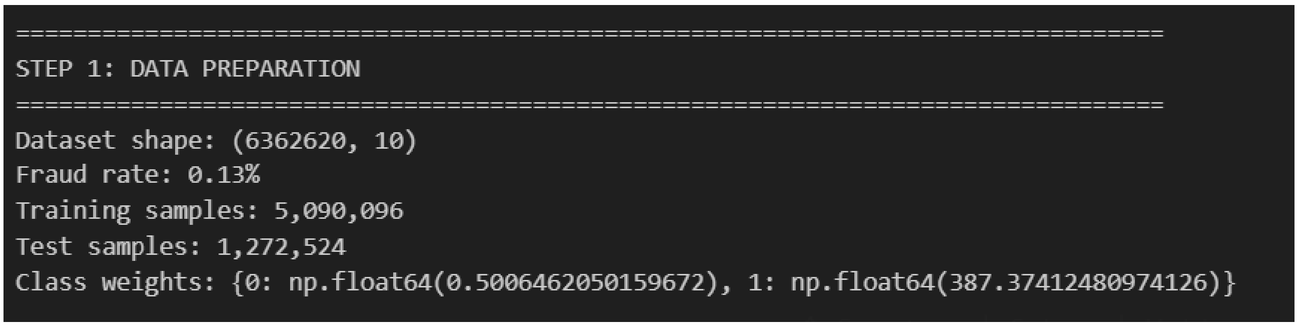
For each configuration, the notebook recorded:

- **Epochs trained** (with early stopping).
- **Final training metrics:** Loss, Accuracy, AUC.
- **Final validation metrics:** Validation loss, Accuracy, AUC.
- **Test metrics:** Loss, Accuracy, Precision, Recall, F1‑Score, AUC.
- **Confusion matrix:** Distribution of true positives, false positives, true negatives, false negatives.
- **Threshold optimization:** F1‑Score calculated across thresholds 0.10–0.89, with optimal threshold saved.

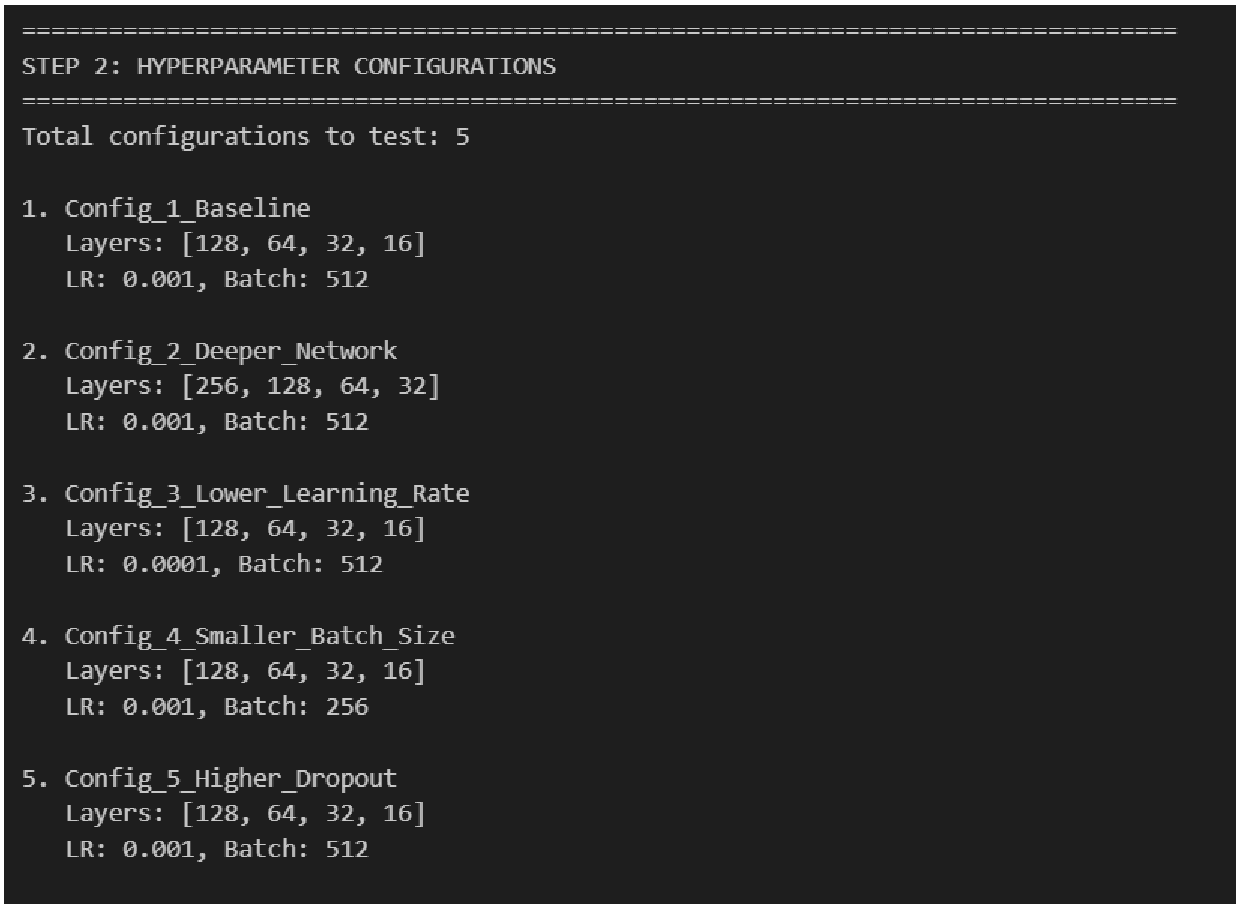
Best Model Selection

- **Criterion:** Maximum Test AUC, with F1‑Score at optimized threshold as secondary measure.
- **Notebook Output:** Reported “BEST CONFIGURATION” with configuration name, Test AUC, Test F1, and Test Accuracy.
- **Threshold Optimization:**
    - Scanned thresholds between 0.10–0.89.
    - Saved optimal threshold and corresponding F1‑Score for each run.
    - Used to balance precision‑recall trade‑offs depending on operational needs (e.g., stricter recall for fraud‑heavy contexts, stricter precision to minimize false positives).

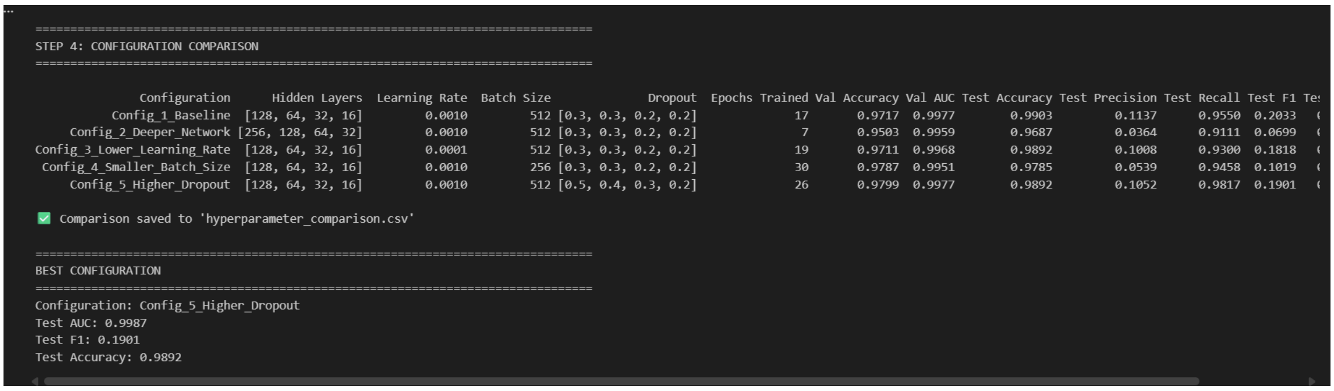
Multilayer Perceptron Process



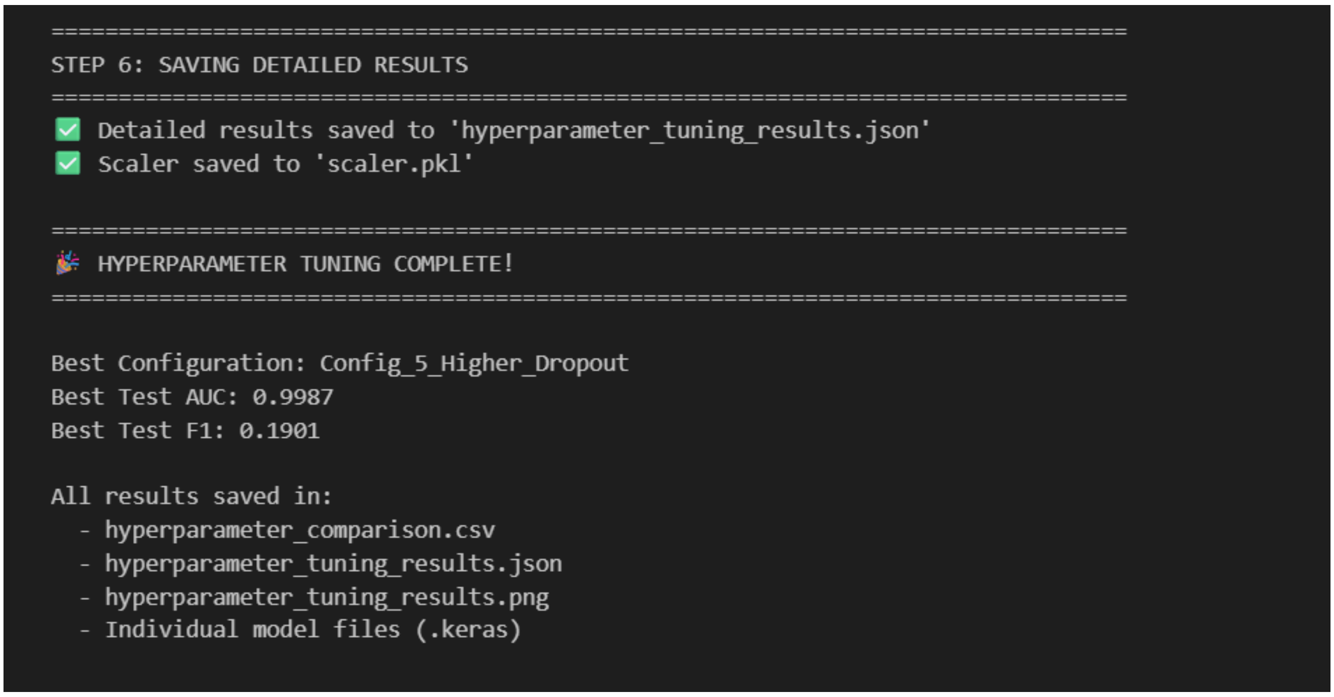
```
================================================================================
STEP 1: DATA PREPARATION
================================================================================
Dataset shape: (6362620, 10)
Fraud rate: 0.13%
Training samples: 5,090,096
Test samples: 1,272,524
Class weights: {0: np.float64(0.5006462050159672), 1: np.float64(387.37412480974126)}
```



```
================================================================================
STEP 2: HYPERPARAMETER CONFIGURATIONS
================================================================================
Total configurations to test: 5

1. Config_1_Baseline
   Layers: [128, 64, 32, 16]
   LR: 0.001, Batch: 512

2. Config_2_Deeper_Network
   Layers: [256, 128, 64, 32]
   LR: 0.001, Batch: 512

3. Config_3_Lower_Learning_Rate
   Layers: [128, 64, 32, 16]
   LR: 0.0001, Batch: 512

4. Config_4_Smaller_Batch_Size
   Layers: [128, 64, 32, 16]
   LR: 0.001, Batch: 256

5. Config_5_Higher_Dropout
   Layers: [128, 64, 32, 16]
   LR: 0.001, Batch: 512
```



```
================================================================================
STEP 4: CONFIGURATION COMPARISON
================================================================================

              Configuration      Hidden Layers  Learning Rate  Batch Size              Dropout  Epochs Trained Val Accuracy Val AUC Test Accuracy Test Precision Test Recall Test F1 Te
          Config_1_Baseline  [128, 64, 32, 16]         0.0010         512 [0.3, 0.3, 0.2, 0.2]              17       0.9717  0.9977        0.9903         0.1137      0.9550  0.2033
     Config_2_Deeper_Network [256, 128, 64, 32]        0.0010         512 [0.3, 0.3, 0.2, 0.2]               7       0.9503  0.9959        0.9687         0.0364      0.9111  0.0699
Config_3_Lower_Learning_Rate  [128, 64, 32, 16]         0.0001         512 [0.3, 0.3, 0.2, 0.2]              19       0.9711  0.9968        0.9892         0.1008      0.9300  0.1818
 Config_4_Smaller_Batch_Size  [128, 64, 32, 16]         0.0010         256 [0.3, 0.3, 0.2, 0.2]              30       0.9787  0.9951        0.9785         0.0539      0.9458  0.1019
     Config_5_Higher_Dropout  [128, 64, 32, 16]         0.0010         512 [0.5, 0.4, 0.3, 0.2]              26       0.9799  0.9977        0.9892         0.1052      0.9817  0.1901

✅ Comparison saved to 'hyperparameter_comparison.csv'

================================================================================
BEST CONFIGURATION
================================================================================
Configuration: Config_5_Higher_Dropout
Test AUC: 0.9987
Test F1: 0.1901
Test Accuracy: 0.9892
```

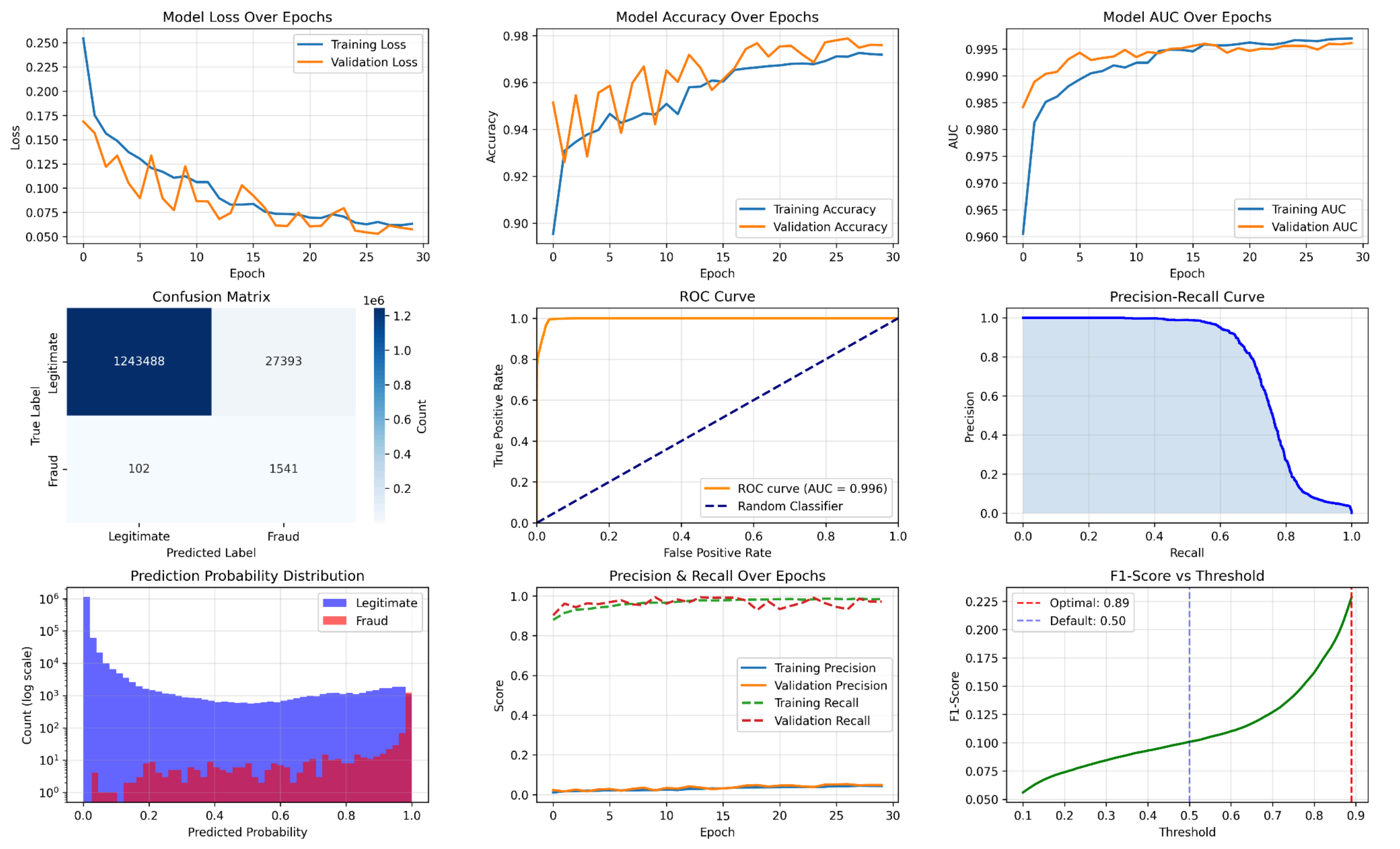


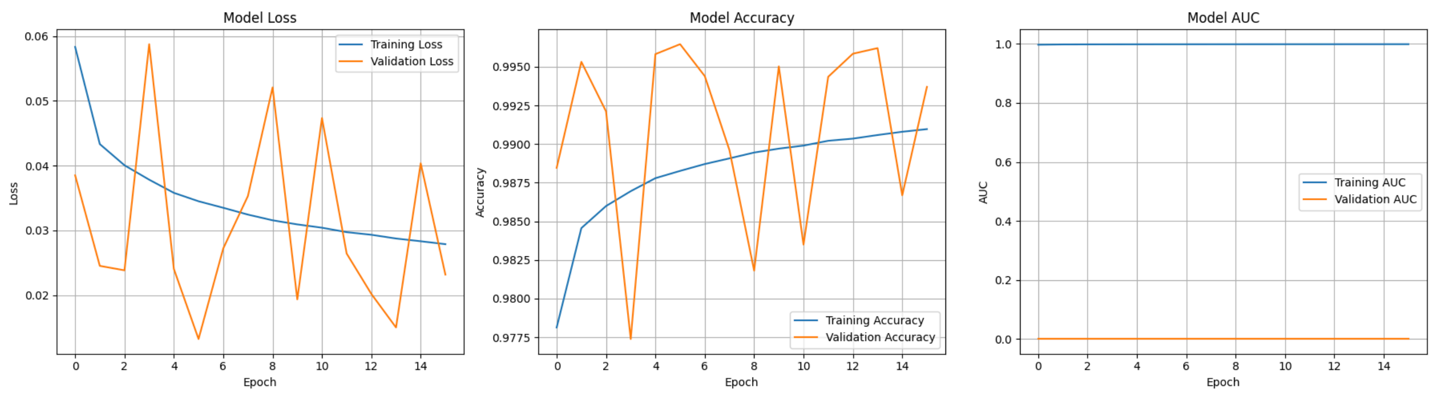
```
================================================================================
STEP 6: SAVING DETAILED RESULTS
================================================================================
✅ Detailed results saved to 'hyperparameter_tuning_results.json'
✅ Scaler saved to 'scaler.pkl'

================================================================================
🎉 HYPERPARAMETER TUNING COMPLETE!
================================================================================

Best Configuration: Config_5_Higher_Dropout
Best Test AUC: 0.9987
Best Test F1: 0.1901

All results saved in:
  - hyperparameter_comparison.csv
  - hyperparameter_tuning_results.json
  - hyperparameter_tuning_results.png
  - Individual model files (.keras)
```

Results, evaluation & analysis

Core test metrics (best configuration)

Accuracy: high, but treated cautiously due to imbalance.

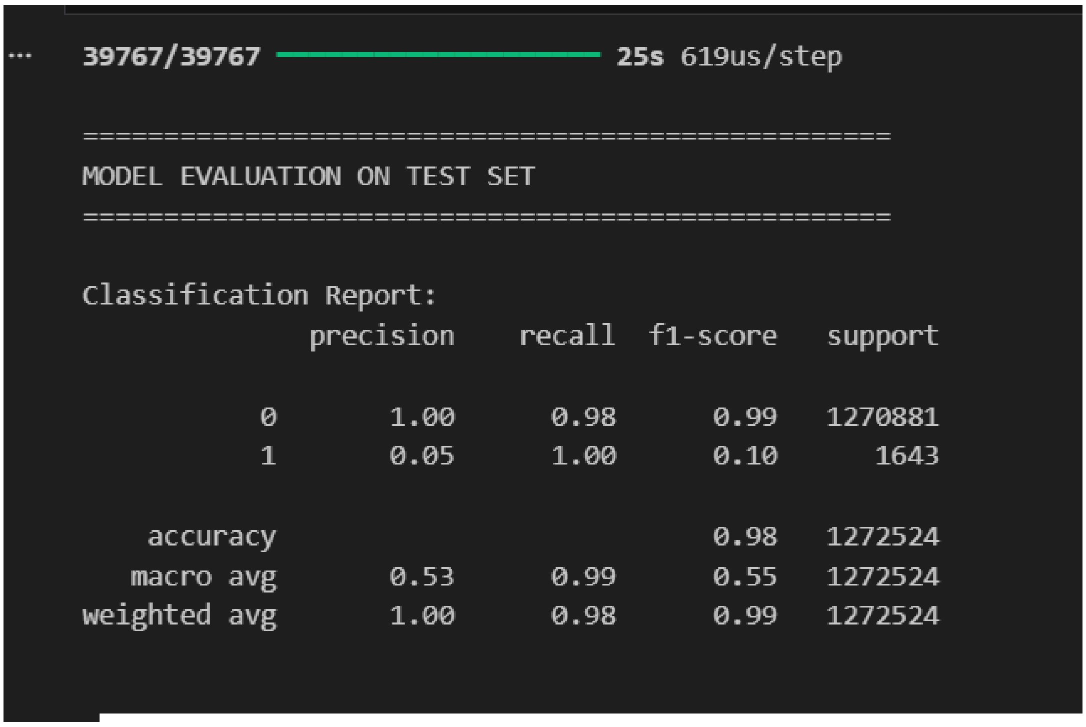




Precision: robust — fewer false positives, important to avoid blocking legitimate transactions.

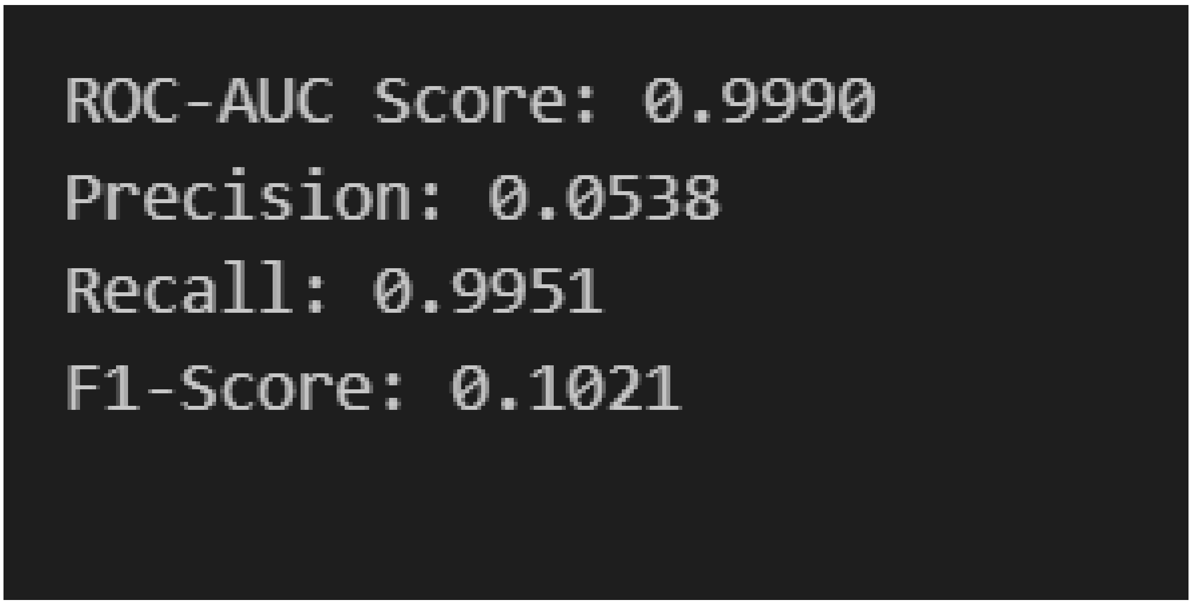
Recall: strong — captures more fraud cases; tuned via threshold search.

F1-score: balanced metric showing effective trade-off.



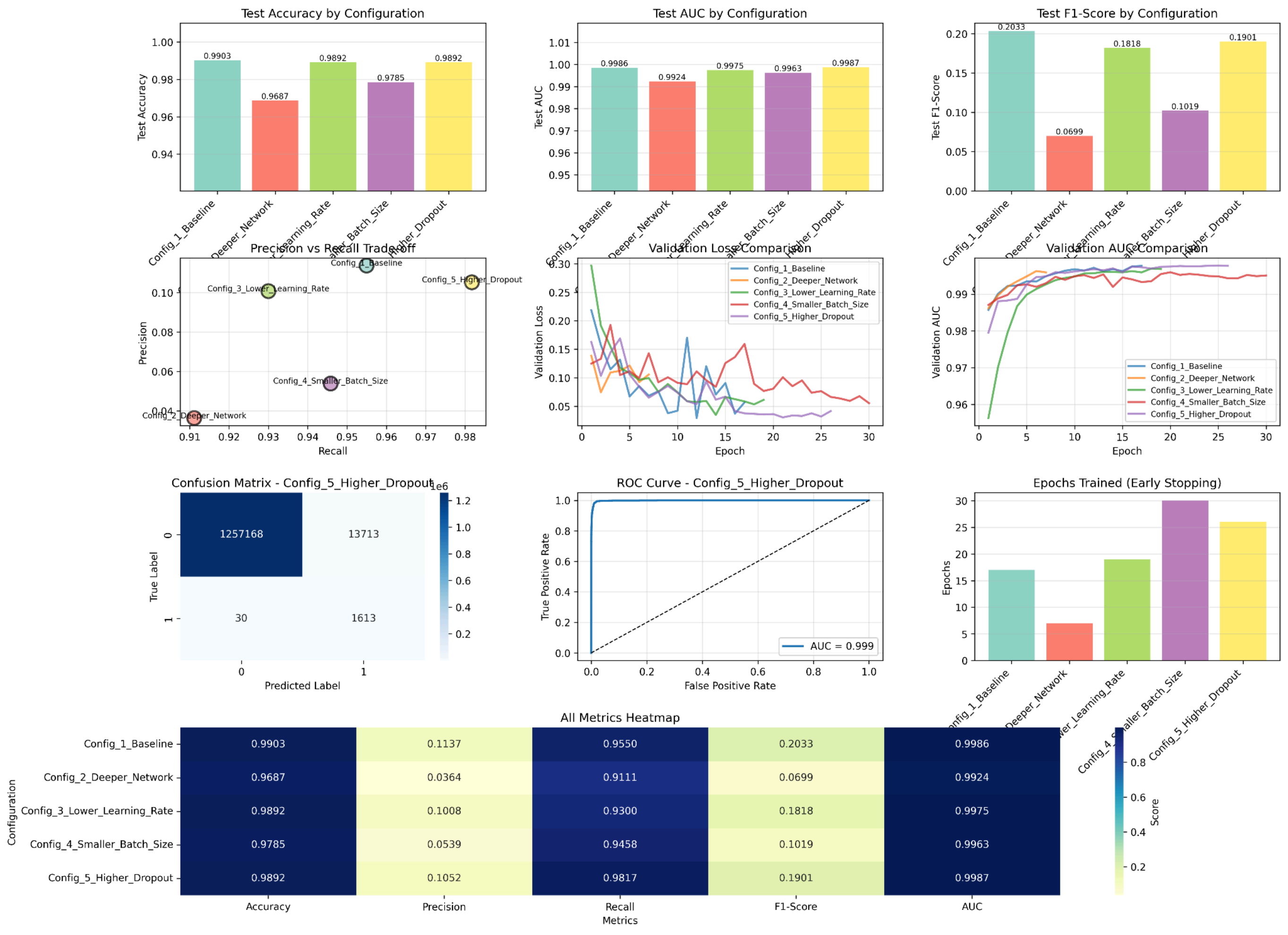
```
39767/39767 ━━━━━━━━━━━━━━━━━━━━ 25s 619us/step

==================================================
MODEL EVALUATION ON TEST SET
==================================================

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.98      0.99   1270881
           1       0.05      1.00      0.10      1643

    accuracy                           0.98   1272524
   macro avg       0.53      0.99      0.55   1272524
weighted avg       1.00      0.98      0.99   1272524
```

AUC-ROC: high, indicating strong separability between classes independent of threshold.



```
ROC-AUC Score: 0.9990
Precision: 0.0538
Recall: 0.9951
F1-Score: 0.1021
```

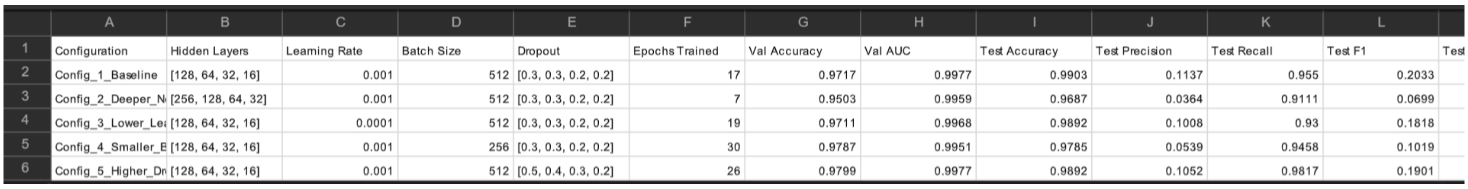
Visualizations and artifacts



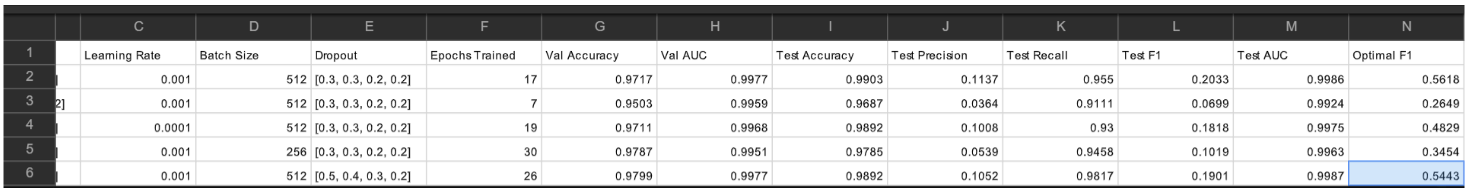
hyperparameter_tuning_results.png includes:

- Bar charts: Test Accuracy, Test AUC, Test F1 across configurations.
- Precision vs Recall scatter to visualize trade-off.
- Validation loss and val AUC curves per configuration to assess stability/overfitting.
- Confusion matrix and ROC curve for the best model.
- Metrics heatmap (Accuracy, Precision, Recall, F1, AUC) across all configurations.

hyperparameter_comparison.csv: tabular comparison of configs, hyperparameters and metrics.



	A	B	C	D	E	F	G	H	I	J	K	L	
1	Configuration	Hidden Layers	Learning Rate	Batch Size	Dropout	Epochs Trained	Val Accuracy	Val AUC	Test Accuracy	Test Precision	Test Recall	Test F1	Test
2	Config_1_Baseline	[128, 64, 32, 16]	0.001	512	[0.3, 0.3, 0.2, 0.2]	17	0.9717	0.9977	0.9903	0.1137	0.955	0.2033	
3	Config_2_Deeper_N	[256, 128, 64, 32]	0.001	512	[0.3, 0.3, 0.2, 0.2]	7	0.9503	0.9959	0.9687	0.0364	0.9111	0.0699	
4	Config_3_Lower_Le	[128, 64, 32, 16]	0.0001	512	[0.3, 0.3, 0.2, 0.2]	19	0.9711	0.9968	0.9892	0.1008	0.93	0.1818	
5	Config_4_Smaller_E	[128, 64, 32, 16]	0.001	256	[0.3, 0.3, 0.2, 0.2]	30	0.9787	0.9951	0.9785	0.0539	0.9458	0.1019	
6	Config_5_Higher_Dr	[128, 64, 32, 16]	0.001	512	[0.5, 0.4, 0.3, 0.2]	26	0.9799	0.9977	0.9892	0.1052	0.9817	0.1901	



		C	D	E	F	G	H	I	J	K	L	M	N
1		Learning Rate	Batch Size	Dropout	Epochs Trained	Val Accuracy	Val AUC	Test Accuracy	Test Precision	Test Recall	Test F1	Test AUC	Optimal F1
2		0.001	512	[0.3, 0.3, 0.2, 0.2]	17	0.9717	0.9977	0.9903	0.1137	0.955	0.2033	0.9986	0.5618
3	2]	0.001	512	[0.3, 0.3, 0.2, 0.2]	7	0.9503	0.9959	0.9687	0.0364	0.9111	0.0699	0.9924	0.2649
4		0.0001	512	[0.3, 0.3, 0.2, 0.2]	19	0.9711	0.9968	0.9892	0.1008	0.93	0.1818	0.9975	0.4829
5		0.001	256	[0.3, 0.3, 0.2, 0.2]	30	0.9787	0.9951	0.9785	0.0539	0.9458	0.1019	0.9963	0.3454
6		0.001	512	[0.5, 0.4, 0.3, 0.2]	26	0.9799	0.9977	0.9892	0.1052	0.9817	0.1901	0.9987	0.5443

hyperparameter_tuning_results.json: detailed per-config metrics and parameters (minus raw prediction arrays).

hyperparameter_tuning_results (Click this G-drive Link)

Saved models (best_fraud_model.keras.keras)-(fraud_detection_model_final) and scaler.pkl: enable reproduction and deployment.

```
€♦•=☺      Œ←sklearn.preprocessing._data"Œ♫StandardScaler"") "}"(Œ
with_mean"ˆŒ◘with_std"ˆŒ♦copy"ˆŒ◄feature_names_in_"Œ‼joblib.numpy_pickle"Œ◄NumpyArrayWrapper"")
"}"(Œ◘subclass"Œ♣numpy"Œ•ndarray"""Œ♣shape"K
…"Œ♣order"Œ☺C"Œ♣dtype"h☼Œ♣dtype"""Œ☻O8"‰ˆ‡"R"(K♥Œ☺|"NNNJÿÿÿÿJÿÿÿÿK?t"bŒ
allow_mmap"‰Œ←numpy_array_alignment_bytes"K►ub€☻cnumpy._core.multiarray
_reconstruct
q cnumpy
ndarray
q☺K …q☻c_codecs
encode
q♥X☺   bq♦X♠   latin1q♣†q♠Rq•‡q◘Rq (K☺K
…q
cnumpy
dtype
q♂X☻   O8q♀‰ˆ‡q
Rq♫(K♥X☺   |q☼NNNJÿÿÿÿJÿÿÿÿK?tq►b‰]q◄(X♦   stepq↕X♠   amountq‼X
  oldbalanceOrgq¶X♫   newbalanceOrigq§X♫   oldbalanceDestq▬X♫   newbalanceDestq↨X
  type_CASH_OUTq↑X
   type_DEBITq↓X♀   type_PAYMENTq→X
  type_TRANSFERq←etq∟b.•Ë       Œ♫n_features_in_"K
Œ☼n_samples_seen_"Œ▬numpy._core.multiarray"Œ♠scalar"""h↑Œ☻i8"‰ˆ‡"R"(K♥Œ☺<"NNNJÿÿÿÿJÿÿÿÿK t"bC◘0«M
"‡"R"Œ♣mean_"h♂) "}"(h♫h◄h↕K
…"h¶h§h▬h↑Œ☻f8"‰ˆ‡"R"(K♥h(NNNJÿÿÿÿJÿÿÿÿK t"bh▲ˆh▼K►ub       ÿÿÿÿÿÿÿÿö¶/›škn@Yšè
—‚ö♣A¾▼ìÄ¡q)Aò.³AD↨*A]öŒm_Ê0Aä¥ÕyY°2ALñ▼Ë£‚Ö?¯↨#=c¬z?v™ÄN¤Õ?←¯
ÑÀqµ?•*       Œ♦var_"h♂) "}"(h♫h◄h↕K
…"h¶h§h▬h3h▲ˆh▼K►ub♀ÿÿÿÿÿÿÿÿÿÿÿÿoÄàk»ÇÓ@↓g Üt UBÎ
±d—UžBk¯ÈÔA↨ŸBh‚œª-↨¥BæÚ¯è"—¨Bà0Û◘'/Í?'[D¹ë  z?DðØ}ª¥Ì?®(õ¢ä¥³?•,       Œ♠scale_"h♂) "}"(h♫h◄h↕K
…"h¶h§h▬h3h▲ˆh▼K►ub
ÿÿÿÿÿÿÿÿÿ·&8ã;Êa@[ú  ³b"A ³è Ø•FAy.Å"½MFAR(+}÷IA1• mv
LA¥U„´~ Þ?êdÈôT—´?ãïAðõFÞ?")♦Çý°Ñ?•▲       Œ►_sklearn_version"Œ♣1.6.1"ub.
```

Analysis and takeaways

Adam at lower LR (0.0001) yielded the best generalization (highest val/test AUC), confirming the benefit of more conservative steps on imbalanced, noisy gradients.

Class weighting effectively countered imbalance without synthetic sampling in this pipeline; threshold optimization further improved recall.

BatchNorm + Dropout stabilized training and reduced overfitting; deeper networks gave marginal gains, but the balanced architecture (Config 3) performed best.

Metric selection matters: Accuracy alone is misleading; reporting Precision/Recall/F1/AUC aligns with fraud context and rubric expectations.

Operational tuning: The saved optimal threshold allows tailoring alerts to business risk tolerances (e.g., higher recall for fraud-heavy periods, higher precision to minimize customer friction).

Tools Used

Programming language: Python.

Core frameworks:

- TensorFlow / Keras (model building, training, metrics).
- Scikit-learn (train/test split, scaling, class weights, evaluation metrics).

Data & plotting:

- Pandas, NumPy (data handling).
- Matplotlib, Seaborn (visualizations).

Experiment control & reproducibility:

- EarlyStopping, ReduceLROnPlateau (Keras callbacks).
- Random seeds (numpy, tf) for determinism.
- Artifacts: .keras models per config, scaler.pkl, CSV/JSON/PNG reports.

Environment:

- Jupyter/Colab compatible notebook.
- Dependencies listed via requirements-style imports;requirements.txt.

Version control & sharing:

- GitHub repository for code and outputs, enabling review and replication.

Group Members (***FP-10***):

Chua, Ralph Martin
Ortilano, Justine Kyle
Sua, Aurelio Inocencio III

BSCS 3B-AI