Kaggle Playground

Problem Statement / Real World Implementations

1. Importing Libraries

In [1]:
```
# Core Data Science Libraries
import numpy as np
import pandas as pd
import warnings

# Visualization Libraries
import plotly.express as px
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.graph_objects as go
from plotly.subplots import make_subplots

# Scikit-Learn for Preprocessing and Modeling
from sklearn.model_selection import KFold, train_test_split
from sklearn.preprocessing import OrdinalEncoder, StandardScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Machine Learning Models
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor

# Hyperparameter Tuning
import optuna

# Notebook settings
warnings.filterwarnings('ignore')
pd.set_option('display.max_columns', None)
```

2. Loading Dataset

In [2]:
```
# Define file paths
TRAIN_PATH = "/kaggle/input/playground-series-s5e11/train.csv"
TEST_PATH = "/kaggle/input/playground-series-s5e11/test.csv"
SUBMISSION_PATH = "/kaggle/input/playground-series-s5e11/sample_submission.csv"

# Load the datasets into pandas DataFrames
train_df = pd.read_csv(TRAIN_PATH)
test_df = pd.read_csv(TEST_PATH)
submission_df = pd.read_csv(SUBMISSION_PATH)
```

Refining the Original dataset

In [3]:
```
print("Train shape:", train_df.shape)
print("Test shape:", test_df.shape)
```

```
Train shape: (593994, 13)
Test shape: (254569, 12)
```

In [4]:
```
df=train_df
df.head(5)
```

Out[4]:

	id	annual_income	debt_to_income_ratio	credit_score	loan_amount	interest_rate	gend
0	0	29367.99	0.084	736	2528.42	13.67	Fema
1	1	22108.02	0.166	636	4593.10	12.92	Ma
2	2	49566.20	0.097	694	17005.15	9.76	Ma
3	3	46858.25	0.065	533	4682.48	16.10	Fema
4	4	25496.70	0.053	665	12184.43	10.21	Ma

In [5]:
```
print(df["gender"].unique())
print(df["marital_status"].unique())
print(df["education_level"].unique())
print(df["employment_status"].unique())
print(df["loan_purpose"].unique())
print(df["grade_subgrade"].unique())
```

```
['Female' 'Male' 'Other']
['Single' 'Married' 'Divorced' 'Widowed']
['High School' "Master's" "Bachelor's" 'PhD' 'Other']
['Self-employed' 'Employed' 'Unemployed' 'Retired' 'Student']
['Other' 'Debt consolidation' 'Home' 'Education' 'Vacation' 'Car'
 'Medical' 'Business']
['C3' 'D3' 'C5' 'F1' 'D1' 'D5' 'C2' 'C1' 'F5' 'D4' 'C4' 'D2' 'E5' 'B1'
 'B2' 'F4' 'A4' 'E1' 'F2' 'B4' 'E4' 'B3' 'E3' 'B5' 'E2' 'F3' 'A5' 'A3'
 'A1' 'A2']
```

In [6]:
```
df.isna().sum()
```

Out[6]:
```
id                      0
annual_income           0
debt_to_income_ratio    0
credit_score            0
loan_amount             0
interest_rate           0
gender                  0
marital_status          0
education_level         0
employment_status       0
loan_purpose            0
grade_subgrade          0
loan_paid_back          0
dtype: int64
```

In [7]:
```
df.head(10)
```

Out[7]:

	id	annual_income	debt_to_income_ratio	credit_score	loan_amount	interest_rate	gend
0	0	29367.99	0.084	736	2528.42	13.67	Fema
1	1	22108.02	0.166	636	4593.10	12.92	Ma
2	2	49566.20	0.097	694	17005.15	9.76	Ma
3	3	46858.25	0.065	533	4682.48	16.10	Fema
4	4	25496.70	0.053	665	12184.43	10.21	Ma
5	5	44940.30	0.058	653	12159.92	12.24	Ma
6	6	61574.16	0.042	696	16907.71	13.52	Oth
7	7	45953.31	0.100	654	10111.62	12.82	Fema
8	8	30592.29	0.132	713	7522.36	9.48	Ma
9	9	17342.45	0.121	548	9653.48	16.04	Fema

4. EDA

In [8]:

```python
# Select only numeric columns for correlation matrix
numerical_cols = train_df.select_dtypes(include=np.number).columns.tolist()
numerical_cols.remove('id')
numerical_cols.remove('loan_paid_back')

numeric_df = train_df[numerical_cols + ['loan_paid_back']]
corr_matrix = numeric_df.corr()

# Create the interactive heatmap
fig = go.Figure(data=go.Heatmap(
                   z=corr_matrix.values,
                   x=corr_matrix.columns,
                   y=corr_matrix.columns,
                   colorscale='RdBu_r',
                   zmin=-1, zmax=1,
                   text=corr_matrix.round(2).values,
                   texttemplate="%{text}",
                   hoverongaps=False))

fig.update_layout(
    title='Correlation Heatmap of Numerical Features',
    width=800, height=800
)
fig.show()
```

3. Normalization of data

```
In [9]:
# Save the test IDs for your submission file
test_ids = test_df['id']

# Drop the 'id' column from both, as it's not a feature
train_df = train_df.drop('id', axis=1)
test_df = test_df.drop('id', axis=1)
```

```
In [10]:
def create_financial_features(df):
    """
    Creates new financial features from the existing columns.
    """
    # Create monthly_income
    df['monthly_income'] = df['annual_income'] / 12

    # Create total_monthly_debt
    df['total_monthly_debt'] = df['debt_to_income_ratio'] * df['monthly_income']

    # Create available_income (disposable income)
    df['available_income'] = df['monthly_income'] - df['total_monthly_debt']

    # Create loan_to_income_ratio
    df['loan_to_income_ratio'] = df['loan_amount'] / df['annual_income']

    # Create loan_to_available_income
    # We will replace inf/-inf with NaN later
    df['loan_to_available_income'] = df['loan_amount'] / df['available_income']

    return df

# Assuming 'train_df' and 'test_df' are already loaded

print("Applying features to train_df...")
train_df = create_financial_features(train_df)

print("Applying features to test_df...")
test_df = create_financial_features(test_df)
```

```
Applying features to train_df...
Applying features to test_df...
```

In [11]:

```
import pandas as pd
import numpy as np

def process_risk_features(df):
    """
    Applies ordinal mapping and binning to risk features.
    (Corrected version)
    """

    # 1. Map grade_subgrade
    # This map is correct and complete.
    grades = ['A', 'B', 'C', 'D', 'E', 'F', 'G']
    subgrades = ['1', '2', '3', '4', '5']
    grade_map = {}
    i = 0
    for g in grades:
        for s in subgrades:
            grade_map[g + s] = i
            i += 1

    df['grade_subgrade_ordinal'] = df['grade_subgrade'].map(grade_map)

    # 2. Map education_level (FIXED)
    # This map now includes 'PhD' and 'Other'
    # We place them in a logical ordinal scale.
    education_map = {
        'Other': 0,          # Treat 'Other' as the baseline
        'High School': 1,
        'Bachelor\'s': 2,
        'Master\'s': 3,
        'PhD': 4            # 'PhD' is the highest
    }
    df['education_level_ordinal'] = df['education_level'].map(education_map)

    # 3. Bin credit_score
    # This logic is correct.
    score_bins = [300, 579, 669, 739, 799, 850]
    score_labels = ['Poor', 'Fair', 'Good', 'Very Good', 'Excellent']

    df['credit_score_bin'] = pd.cut(df['credit_score'],
                                    bins=score_bins,
                                    labels=score_labels,
                                    include_lowest=True)

    # 4. Drop original columns
    cols_to_drop = ['grade_subgrade', 'education_level', 'credit_score']
    df = df.drop(columns=cols_to_drop)

    return df
```

In [12]:
```
cols_to_drop = ['annual_income', 'debt_to_income_ratio']

# --- Clean Training Data ---
train_df = train_df.drop(columns=cols_to_drop)
train_df.replace([np.inf, -np.inf], np.nan, inplace=True)

# --- Clean Test Data ---
test_df = test_df.drop(columns=cols_to_drop)
test_df.replace([np.inf, -np.inf], np.nan, inplace=True)


print("Feature creation complete. Redundant columns dropped.")
print("\n--- train_df columns ---")
print(list(train_df.columns))

print("\n--- test_df columns ---")
print(list(test_df.columns))
```

```
Feature creation complete. Redundant columns dropped.

--- train_df columns ---
['credit_score', 'loan_amount', 'interest_rate', 'gender', 'marital_status',
'education_level', 'employment_status', 'loan_purpose', 'grade_subgrade', 'loan_paid_
'monthly_income', 'total_monthly_debt', 'available_income', 'loan_to_income_ratio',
'loan_to_available_income']

--- test_df columns ---
['credit_score', 'loan_amount', 'interest_rate', 'gender', 'marital_status',
'education_level', 'employment_status', 'loan_purpose', 'grade_subgrade', 'monthly_in
'total_monthly_debt', 'available_income', 'loan_to_income_ratio',
'loan_to_available_income']
```

In [13]:
```
# --- Apply to train_df ---
print("Processing train_df...")
train_df = process_risk_features(train_df)

# --- Apply to test_df ---
print("Processing test_df...")
test_df = process_risk_features(test_df)
```

```
print("\nFeature creation complete. Original columns dropped.")

# --- Show New Columns ---
print("\n--- train_df columns ---")
print(list(train_df.columns))

print("\n--- test_df columns ---")
print(list(test_df.columns))

# --- Check the new binned values ---
print("\n--- Value counts for new 'credit_score_bin' (from train_df) ---")
print(train_df['credit_score_bin'].value_counts())
```

```
Processing train_df...
Processing test_df...

Feature creation complete. Original columns dropped.

--- train_df columns ---
['loan_amount', 'interest_rate', 'gender', 'marital_status', 'employment_status',
'loan_purpose', 'loan_paid_back', 'monthly_income', 'total_monthly_debt',
'available_income', 'loan_to_income_ratio', 'loan_to_available_income',
'grade_subgrade_ordinal', 'education_level_ordinal', 'credit_score_bin']

--- test_df columns ---
['loan_amount', 'interest_rate', 'gender', 'marital_status', 'employment_status',
'loan_purpose', 'monthly_income', 'total_monthly_debt', 'available_income',
'loan_to_income_ratio', 'loan_to_available_income', 'grade_subgrade_ordinal',
'education_level_ordinal', 'credit_score_bin']

--- Value counts for new 'credit_score_bin' (from train_df) ---
credit_score_bin
Good         275585
Fair         209954
Very Good     71337
Poor          27266
Excellent      9852
Name: count, dtype: int64
```

In [14]:

```
from sklearn.preprocessing import OneHotEncoder

categorical_cols = [
    'gender',
    'marital_status',
    'employment_status',
    'loan_purpose',
    'credit_score_bin'
]

ohe = OneHotEncoder(handle_unknown='ignore', sparse_output=False, dtype='int')

print("Fitting One-Hot Encoder on train_df...")
# Note: We must convert .astype(str) because 'credit_score_bin' is a category
ohe.fit(train_df[categorical_cols].astype(str))

# 3. TRANSFORM both train_df and test_df
print("Transforming train_df...")
# --- Process train_df ---
# Get the new encoded columns
```

```
train_encoded = ohe.transform(train_df[categorical_cols].astype(str))
# Get the new column names
encoded_feature_names = ohe.get_feature_names_out(categorical_cols)
# Create a new DataFrame with the encoded features
train_encoded_df = pd.DataFrame(train_encoded, columns=encoded_feature_names, ir
# Drop the original text columns
train_processed = train_df.drop(columns=categorical_cols)
# Add the new numeric columns
train_processed = pd.concat([train_processed, train_encoded_df], axis=1)


print("Transforming test_df...")
# --- Process test_df ---
# Use the SAME fitted encoder to transform test_df
test_encoded = ohe.transform(test_df[categorical_cols].astype(str))
# Create a new DataFrame with the encoded features
test_encoded_df = pd.DataFrame(test_encoded, columns=encoded_feature_names, inde
# Drop the original text columns
test_processed = test_df.drop(columns=categorical_cols)
# Add the new numeric columns
test_processed = pd.concat([test_processed, test_encoded_df], axis=1)


print("\n--- Encoding Complete ---")
print("train_processed is ready for training.")
print("test_processed is ready for prediction.")
```

```
Fitting One-Hot Encoder on train_df...
Transforming train_df...
Transforming test_df...

--- Encoding Complete ---
train_processed is ready for training.
test_processed is ready for prediction.
```

In [15]:
```
train_df.head(5)
```

Out[15]:

	loan_amount	interest_rate	gender	marital_status	employment_status	loan_purpose	lo
0	2528.42	13.67	Female	Single	Self-employed	Other	
1	4593.10	12.92	Male	Married	Employed	Debt consolidation	
2	17005.15	9.76	Male	Single	Employed	Debt consolidation	
3	4682.48	16.10	Female	Single	Employed	Debt consolidation	
4	12184.43	10.21	Male	Married	Employed	Other	

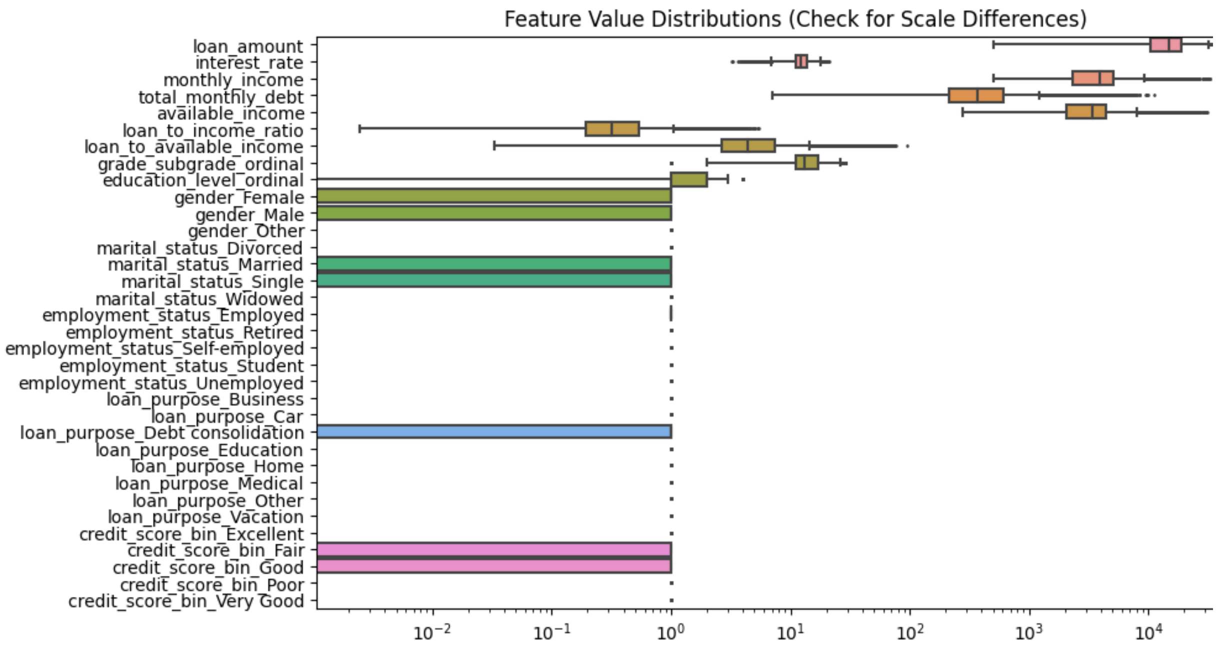
In [16]:
```
# Exclude target column if present
features = train_processed.drop(columns=['loan_paid_back'], errors='ignore')

# 1. Check summary statistics
print("Summary Statistics:\n")
display(features.describe())

# 2. Check for large differences in scale
```

```python
range_df = features.max() - features.min()
print("\nFeature Ranges:\n")
print(range_df.sort_values(ascending=False))

# 3. Visualize distribution of feature scales
plt.figure(figsize=(10, 6))
sns.boxplot(data=features, orient='h', fliersize=1)
plt.title("Feature Value Distributions (Check for Scale Differences)")
plt.xscale('log')
plt.show()

# 4. Correlation check
corr_matrix = features.corr()
high_range_features = range_df[range_df > range_df.mean()].index.tolist()
print(f"\nFeatures with significantly higher ranges: {high_range_features}")

# 5. Quick rule-based decision
if range_df.max() / range_df.min() > 10:
    print("\n✅ Feature scaling is likely necessary (large scale differences det
else:
    print("\n❌ Feature scaling might not be strictly necessary (features on sim
```

```
Summary Statistics:
```

	loan_amount	interest_rate	monthly_income	total_monthly_debt	available_income
count	593994.000000	593994.000000	593994.000000	593994.000000	593994.000000
mean	15020.297629	12.356345	4017.683581	485.002319	3532.681262
std	6926.530568	2.008959	2225.995173	415.297292	1979.316733
min	500.090000	3.200000	500.202500	7.014620	274.692110
25%	10279.620000	10.990000	2327.866667	216.130683	2037.879680
50%	15000.220000	12.370000	3879.806667	367.691262	3386.199475
75%	18858.580000	13.680000	5081.776667	615.139751	4450.237361
max	48959.950000	20.990000	32781.811667	11151.413863	31477.877736

```
Feature Ranges:

loan_amount                         48459.860000
monthly_income                      32281.609167
available_income                    31203.185626
total_monthly_debt                  11144.399243
loan_to_available_income               94.058593
grade_subgrade_ordinal                 29.000000
interest_rate                          17.790000
loan_to_income_ratio                    5.458652
education_level_ordinal                 4.000000
gender_Female                           1.000000
gender_Male                             1.000000
gender_Other                            1.000000
marital_status_Divorced                 1.000000
marital_status_Married                  1.000000
marital_status_Single                   1.000000
marital_status_Widowed                  1.000000
employment_status_Employed              1.000000
employment_status_Retired               1.000000
employment_status_Self-employed         1.000000
employment_status_Student               1.000000
employment_status_Unemployed            1.000000
loan_purpose_Business                   1.000000
loan_purpose_Car                        1.000000
loan_purpose_Debt consolidation         1.000000
loan_purpose_Education                  1.000000
loan_purpose_Home                       1.000000
loan_purpose_Medical                    1.000000
loan_purpose_Other                      1.000000
loan_purpose_Vacation                   1.000000
credit_score_bin_Excellent              1.000000
credit_score_bin_Fair                   1.000000
credit_score_bin_Good                   1.000000
credit_score_bin_Poor                   1.000000
credit_score_bin_Very Good              1.000000
dtype: float64
```



```
Features with significantly higher ranges: ['loan_amount', 'monthly_income',
'total_monthly_debt', 'available_income']

✓ Feature scaling is likely necessary (large scale differences detected).
```

Train test split

In [17]:
```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler, RobustScaler, Pc
import numpy as np

# Use encoded data for model training
X = train_processed.drop("loan_paid_back", axis=1)
y = train_processed["loan_paid_back"]

# Ensure all columns are numeric
X = X.select_dtypes(include=[np.number])

# Train-test split
X_train, X_val, y_train, y_val = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Choose scaling method
selected_method = 'Standard Scaling'

# Apply the best scaling method
if selected_method == 'Min-Max Scaling':
    scaler = MinMaxScaler()
elif selected_method == 'Standard Scaling':
    scaler = StandardScaler()
elif selected_method == 'Robust Scaling':
    scaler = RobustScaler()
elif selected_method == 'Power Transformation':
    scaler = PowerTransformer(method='yeo-johnson')
else:
    scaler = None

# Perform scaling
if scaler is not None:
    # Fit on training data
    X_train_scaled = scaler.fit_transform(X_train)
    # Only transform validation data
    X_val_scaled = scaler.transform(X_val)
else:
    X_train_scaled = X_train
    X_val_scaled = X_val

print("Scaling for tuning complete.")
```

```
Scaling for tuning complete.
```

In [18]:
```python
import optuna
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.metrics import log_loss, accuracy_score, roc_auc_score # <-- Correc
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier

# ✓ Enable GPU for both XGBoost and LightGBM
# --- Hyperparameter tuning for XGBClassifier ---
def objective_xgb(trial):
    param = {
        'tree_method': 'gpu_hist',
        'predictor': 'gpu_predictor',
        'gpu_id': 0,
        'objective': 'binary:logistic', # <-- CORRECTED: Set objective for class
        'lambda': trial.suggest_loguniform('lambda', 1e-3, 10.0),
        'alpha': trial.suggest_loguniform('alpha', 1e-3, 10.0),
        'colsample_bytree': trial.suggest_categorical('colsample_bytree', [0.3,
        'subsample': trial.suggest_categorical('subsample', [0.5, 0.6, 0.7, 0.8,
        'learning_rate': trial.suggest_float('learning_rate', 0.005, 0.05, log=T
        'n_estimators': trial.suggest_int('n_estimators', 200, 1000, step=100),
        'max_depth': trial.suggest_int('max_depth', 3, 12),
        'min_child_weight': trial.suggest_int('min_child_weight', 1, 300),
        'random_state': 42
    }

    model = XGBClassifier(**param, verbosity=0) # <-- CORRECTED: Classifier
    model.fit(X_train_scaled, y_train)
    y_pred_proba = model.predict_proba(X_val_scaled)[:, 1] # <-- Get probabiliti
    ll = log_loss(y_val, y_pred_proba) # <-- CORRECTED: Use log_loss
    return ll


# --- Hyperparameter tuning for LGBMClassifier ---
def objective_lgbm(trial):
    param = {
        'device': 'gpu',
        'gpu_platform_id': 0,
        'gpu_device_id': 0,
        'boosting_type': 'gbdt',
        'objective': 'binary', # <-- CORRECTED: Set objective for classification
        'metric': 'binary_logloss', # <-- CORRECTED: Set metric
        'lambda_l1': trial.suggest_float('lambda_l1', 1e-5, 1.0, log=True),
        'lambda_l2': trial.suggest_float('lambda_l2', 1e-5, 1.0, log=True),
        'num_leaves': trial.suggest_int('num_leaves', 16, 256),
        'feature_fraction': trial.suggest_float('feature_fraction', 0.5, 1.0),
        'bagging_fraction': trial.suggest_float('bagging_fraction', 0.5, 1.0),
        'bagging_freq': trial.suggest_int('bagging_freq', 1, 7),
        'min_child_samples': trial.suggest_int('min_child_samples', 10, 100),
        'learning_rate': trial.suggest_float('learning_rate', 0.005, 0.3, log=Tr
        'n_estimators': trial.suggest_int('n_estimators', 200, 1000, step=100),
        'max_depth': trial.suggest_int('max_depth', 3, 12),
        'random_state': 42,
        'verbosity': -1
    }

    model = LGBMClassifier(**param) # <-- CORRECTED: Classifier
    model.fit(X_train_scaled, y_train)
    y_pred_proba = model.predict_proba(X_val_scaled)[:, 1] # <-- Get probabiliti
    ll = log_loss(y_val, y_pred_proba) # <-- CORRECTED: Use log_loss
    return ll

# --- Run GPU-accelerated Optuna optimization ---
print("🚀 Tuning XGBClassifier (GPU)...")
study_xgb = optuna.create_study(direction='minimize') # Minimize log_loss
study_xgb.optimize(objective_xgb, n_trials=50, timeout=600)
best_params_xgb = study_xgb.best_params
print(f"✅ Best XGBClassifier parameters: {best_params_xgb}")
print("\n🚀 Tuning LGBMClassifier (GPU)...")
study_lgbm = optuna.create_study(direction='minimize') # Minimize log_loss
study_lgbm.optimize(objective_lgbm, n_trials=50, timeout=600)
best_params_lgbm = study_lgbm.best_params
print(f"✅ Best LGBMClassifier parameters: {best_params_lgbm}")


# --- Initialize models with tuned GPU parameters ---
xgb_model = XGBClassifier(**best_params_xgb, tree_method='gpu_hist', predictor='
lgbm_model = LGBMClassifier(**best_params_lgbm, device='gpu', objective='binary'

models = [
    ("XGBClassifier (GPU)", xgb_model),
    ("LGBMClassifier (GPU)", lgbm_model)
]

print("\n⚡ Evaluating Tuned Models on GPU...\n")
logloss_scores = []
model_names = []

for name, model in models:
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_val_scaled)
    y_pred_proba = model.predict_proba(X_val_scaled)[:, 1]

    # --- CORRECTED Metrics ---
    ll = log_loss(y_val, y_pred_proba)
    acc = accuracy_score(y_val, y_pred)
    auc = roc_auc_score(y_val, y_pred_proba)

    logloss_scores.append(ll)
    model_names.append(name)

    print(f"{name:<30} | LogLoss: {ll:.5f} | Accuracy: {acc:.5f} | ROC AUC: {auc
```

```
[I 2025-11-03 07:15:12,165] A new study created in memory with name: no-
name-8afcf609-3efc-4631-a3bc-8b843a3cd5f2
```

```
🚀 Tuning XGBClassifier (GPU)...
```

```
[I 2025-11-03 07:15:17,864] Trial 0 finished with value: 0.2645231073170942 and paran
{'lambda': 0.0361936500047033, 'alpha': 0.007568899354734025, 'colsample_bytree': 1.0
'subsample': 0.6, 'learning_rate': 0.009507472541685866, 'n_estimators': 300, 'max_de
11, 'min_child_weight': 121}. Best is trial 0 with value: 0.2645231073170942.
[I 2025-11-03 07:15:26,473] Trial 1 finished with value: 0.2573499981158369 and paran
{'lambda': 0.09118575472550126, 'alpha': 0.006382495033513585, 'colsample_bytree': 0.
'subsample': 0.8, 'learning_rate': 0.020666723707219982, 'n_estimators': 900, 'max_de
8, 'min_child_weight': 58}. Best is trial 1 with value: 0.2573499981158369.
[I 2025-11-03 07:15:32,359] Trial 2 finished with value: 0.2592545912919665 and paran
{'lambda': 0.012995029534803772, 'alpha': 0.6651609154168596, 'colsample_bytree': 1.0
'subsample': 0.7, 'learning_rate': 0.013830255474092062, 'n_estimators': 1000, 'max_c
4, 'min_child_weight': 202}. Best is trial 1 with value: 0.2573499981158369.
[I 2025-11-03 07:15:37,303] Trial 3 finished with value: 0.2587124057571616 and paran
{'lambda': 0.04113972357896702, 'alpha': 4.417408109316141, 'colsample_bytree': 0.7,
'subsample': 0.7, 'learning_rate': 0.025955877404414594, 'n_estimators': 400, 'max_de
11, 'min_child_weight': 218}. Best is trial 1 with value: 0.2573499981158369.
[I 2025-11-03 07:15:40,227] Trial 4 finished with value: 0.2611889991191748 and paran
{'lambda': 0.004734241446974353, 'alpha': 0.003013944723008013, 'colsample_bytree': 0
'subsample': 0.6, 'learning_rate': 0.01710684720815652, 'n_estimators': 300, 'max_dep
6, 'min_child_weight': 167}. Best is trial 1 with value: 0.2573499981158369.
[I 2025-11-03 07:15:45,933] Trial 5 finished with value: 0.2576091697124722 and paran
{'lambda': 0.12166336022200093, 'alpha': 0.014304734825826672, 'colsample_bytree': 0.
'subsample': 0.7, 'learning_rate': 0.0434660234370753, 'n_estimators': 600, 'max_dept
'min_child_weight': 91}. Best is trial 1 with value: 0.2573499981158369.
[I 2025-11-03 07:15:53,635] Trial 6 finished with value: 0.2608557041858083 and paran
{'lambda': 0.026980012268408095, 'alpha': 0.0018446481854280735, 'colsample_bytree':
'subsample': 0.7, 'learning_rate': 0.013614857336067543, 'n_estimators': 700, 'max_de
9, 'min_child_weight': 185}. Best is trial 1 with value: 0.2573499981158369.
[I 2025-11-03 07:15:58,081] Trial 7 finished with value: 0.26273495940873975 and
parameters: {'lambda': 6.5082357772219535, 'alpha': 9.317028254420542, 'colsample_byt
1.0, 'subsample': 0.6, 'learning_rate': 0.009085498824438855, 'n_estimators': 700,
'max_depth': 4, 'min_child_weight': 64}. Best is trial 1 with value: 0.257349998115831
[I 2025-11-03 07:16:05,311] Trial 8 finished with value: 0.2579580561236633 and paran
{'lambda': 0.003380654226157312, 'alpha': 0.01703146881244035, 'colsample_bytree': 0.
'subsample': 1.0, 'learning_rate': 0.04202674720091878, 'n_estimators': 700, 'max_dep
10, 'min_child_weight': 286}. Best is trial 1 with value: 0.2573499981158369.
[I 2025-11-03 07:16:10,779] Trial 9 finished with value: 0.25816211054741106 and
parameters: {'lambda': 0.0031406438367416823, 'alpha': 0.12918290371363503,
'colsample_bytree': 0.9, 'subsample': 0.7, 'learning_rate': 0.024981132832192814,
'n_estimators': 800, 'max_depth': 5, 'min_child_weight': 181}. Best is trial 1 with v
0.2573499981158369.
[I 2025-11-03 07:16:20,395] Trial 10 finished with value: 0.2594719159297115 and
parameters: {'lambda': 1.5661425752614788, 'alpha': 0.07643494139376109,
'colsample_bytree': 0.7, 'subsample': 0.8, 'learning_rate': 0.005489057691074062,
'n_estimators': 1000, 'max_depth': 7, 'min_child_weight': 5}. Best is trial 1 with va
0.2573499981158369.
[I 2025-11-03 07:16:25,532] Trial 11 finished with value: 0.2577697358008288 and
parameters: {'lambda': 0.24326017984851894, 'alpha': 0.025526427065941656,
'colsample_bytree': 0.5, 'subsample': 0.5, 'learning_rate': 0.04906657470555916,
'n_estimators': 500, 'max_depth': 8, 'min_child_weight': 90}. Best is trial 1 with va
0.2573499981158369.
[I 2025-11-03 07:16:34,323] Trial 12 finished with value: 0.25750789239913274 and
parameters: {'lambda': 0.24133162318144, 'alpha': 0.0010088575703394705,
'colsample_bytree': 0.7, 'subsample': 0.8, 'learning_rate': 0.030039663735225014,
'n_estimators': 900, 'max_depth': 8, 'min_child_weight': 32}. Best is trial 1 with va
0.2573499981158369.
[I 2025-11-03 07:16:42,472] Trial 13 finished with value: 0.2574224778365216 and
parameters: {'lambda': 0.5266193908344308, 'alpha': 0.0014151126501476, 'colsample_by
0.7, 'subsample': 0.8, 'learning_rate': 0.027853433914135396, 'n_estimators': 900,
'max_depth': 7, 'min_child_weight': 2}. Best is trial 1 with value: 0.257349998115836
```

```
[I 2025-11-03 07:16:49,652] Trial 14 finished with value: 0.25754753481300513 and
parameters: {'lambda': 1.0566261705611366, 'alpha': 0.004510060596708759,
'colsample_bytree': 0.7, 'subsample': 0.8, 'learning_rate': 0.018133034864827927,
'n_estimators': 900, 'max_depth': 6, 'min_child_weight': 13}. Best is trial 1 with va
0.2573499981158369.
[I 2025-11-03 07:17:02,748] Trial 15 finished with value: 0.2587000782480725 and
parameters: {'lambda': 0.9219592386752085, 'alpha': 0.07262882153129689,
'colsample_bytree': 0.7, 'subsample': 0.8, 'learning_rate': 0.03258283887759407,
'n_estimators': 900, 'max_depth': 12, 'min_child_weight': 50}. Best is trial 1 with v
0.2573499981158369.
[I 2025-11-03 07:17:09,232] Trial 16 finished with value: 0.2578573695826229 and
parameters: {'lambda': 5.095116237085027, 'alpha': 0.33996098929077445, 'colsample_by
0.5, 'subsample': 0.8, 'learning_rate': 0.019407058782810696, 'n_estimators': 800,
'max_depth': 6, 'min_child_weight': 123}. Best is trial 1 with value: 0.2573499981158
[I 2025-11-03 07:17:14,771] Trial 17 finished with value: 0.2585088610381899 and
parameters: {'lambda': 0.42624188613345126, 'alpha': 0.0010980232948895675,
'colsample_bytree': 0.7, 'subsample': 0.5, 'learning_rate': 0.02216773352409775,
'n_estimators': 1000, 'max_depth': 3, 'min_child_weight': 1}. Best is trial 1 with va
0.2573499981158369.
[I 2025-11-03 07:17:24,777] Trial 18 finished with value: 0.25798091772046844 and
parameters: {'lambda': 0.0809909688327199, 'alpha': 0.005547614572827067,
'colsample_bytree': 0.7, 'subsample': 1.0, 'learning_rate': 0.010389433504091137,
'n_estimators': 800, 'max_depth': 9, 'min_child_weight': 73}. Best is trial 1 with va
0.2573499981158369.
[I 2025-11-03 07:17:30,364] Trial 19 finished with value: 0.25731562046172163 and
parameters: {'lambda': 2.746270697088073, 'alpha': 0.03333123632829805, 'colsample_by
0.7, 'subsample': 0.8, 'learning_rate': 0.03643335365651437, 'n_estimators': 600,
'max_depth': 7, 'min_child_weight': 37}. Best is trial 19 with value: 0.2573156204617
[I 2025-11-03 07:17:36,278] Trial 20 finished with value: 0.2576358598043774 and
parameters: {'lambda': 0.010053749681438472, 'alpha': 0.03977407477009539,
'colsample_bytree': 0.3, 'subsample': 0.8, 'learning_rate': 0.03574734490967888,
'n_estimators': 500, 'max_depth': 9, 'min_child_weight': 126}. Best is trial 19 with
0.25731562046172163.
[I 2025-11-03 07:17:41,886] Trial 21 finished with value: 0.2574151157221354 and
parameters: {'lambda': 2.8584161451684267, 'alpha': 0.00891615835995151,
'colsample_bytree': 0.7, 'subsample': 0.8, 'learning_rate': 0.02896741425641675,
'n_estimators': 600, 'max_depth': 7, 'min_child_weight': 44}. Best is trial 19 with v
0.25731562046172163.
[I 2025-11-03 07:17:47,481] Trial 22 finished with value: 0.2573876680389996 and
parameters: {'lambda': 3.345061080070869, 'alpha': 0.010009968495511242,
'colsample_bytree': 0.7, 'subsample': 0.8, 'learning_rate': 0.03449612395708449,
'n_estimators': 600, 'max_depth': 7, 'min_child_weight': 37}. Best is trial 19 with v
0.25731562046172163.
[I 2025-11-03 07:17:51,355] Trial 23 finished with value: 0.25781124024494334 and
parameters: {'lambda': 9.669034384991654, 'alpha': 0.2190456458994139, 'colsample_byt
0.7, 'subsample': 0.8, 'learning_rate': 0.038297716415108665, 'n_estimators': 500,
'max_depth': 5, 'min_child_weight': 35}. Best is trial 19 with value: 0.2573156204617
[I 2025-11-03 07:17:56,932] Trial 24 finished with value: 0.25778624646035775 and
parameters: {'lambda': 2.6037127357009724, 'alpha': 0.0337677110336313, 'colsample_by
0.7, 'subsample': 0.8, 'learning_rate': 0.020751912960964312, 'n_estimators': 600,
'max_depth': 7, 'min_child_weight': 87}. Best is trial 19 with value: 0.2573156204617
[I 2025-11-03 07:18:00,451] Trial 25 finished with value: 0.2589067255803117 and
parameters: {'lambda': 0.001044885706690124, 'alpha': 0.010674149868349192,
'colsample_bytree': 0.5, 'subsample': 0.5, 'learning_rate': 0.035043824089197415,
'n_estimators': 200, 'max_depth': 10, 'min_child_weight': 59}. Best is trial 19 with
0.25731562046172163.
[I 2025-11-03 07:18:04,105] Trial 26 finished with value: 0.25757803826289094 and
parameters: {'lambda': 0.08264939587100727, 'alpha': 0.003181821772459335,
'colsample_bytree': 0.7, 'subsample': 0.8, 'learning_rate': 0.04948259538542322,
'n_estimators': 400, 'max_depth': 6, 'min_child_weight': 26}. Best is trial 19 with v
```

```
0.25731562046172163.
[I 2025-11-03 07:18:09,906] Trial 27 finished with value: 0.25828610667991514 and
parameters: {'lambda': 3.073690481478437, 'alpha': 1.259177716837584, 'colsample_bytr
0.7, 'subsample': 1.0, 'learning_rate': 0.023023357426613246, 'n_estimators': 600,
'max_depth': 8, 'min_child_weight': 246}. Best is trial 19 with value: 0.2573156204617
[I 2025-11-03 07:18:15,085] Trial 28 finished with value: 0.2587965005045791 and
parameters: {'lambda': 0.19783695725050118, 'alpha': 0.03998320452885442,
'colsample_bytree': 0.7, 'subsample': 0.8, 'learning_rate': 0.01486421346139551,
'n_estimators': 700, 'max_depth': 5, 'min_child_weight': 111}. Best is trial 19 with
0.25731562046172163.
[I 2025-11-03 07:18:20,400] Trial 29 finished with value: 0.2598700920807726 and
parameters: {'lambda': 0.5333715889110444, 'alpha': 0.008349225767380257,
'colsample_bytree': 1.0, 'subsample': 0.6, 'learning_rate': 0.011817633530455503,
'n_estimators': 400, 'max_depth': 10, 'min_child_weight': 141}. Best is trial 19 with
value: 0.25731562046172163.
[I 2025-11-03 07:18:27,596] Trial 30 finished with value: 0.27193903588524276 and
parameters: {'lambda': 0.03497097150162013, 'alpha': 0.020900653667307725,
'colsample_bytree': 0.5, 'subsample': 0.8, 'learning_rate': 0.006343452402425456,
'n_estimators': 500, 'max_depth': 9, 'min_child_weight': 100}. Best is trial 19 with
0.25731562046172163.
[I 2025-11-03 07:18:33,213] Trial 31 finished with value: 0.25740248782501746 and
parameters: {'lambda': 3.562625192591287, 'alpha': 0.007805314110200777,
'colsample_bytree': 0.7, 'subsample': 0.8, 'learning_rate': 0.028884255018318167,
'n_estimators': 600, 'max_depth': 7, 'min_child_weight': 44}. Best is trial 19 with v
0.25731562046172163.
[I 2025-11-03 07:18:39,626] Trial 32 finished with value: 0.25739958318073713 and
parameters: {'lambda': 1.39482674157152, 'alpha': 0.0028708926724178496,
'colsample_bytree': 0.7, 'subsample': 0.8, 'learning_rate': 0.03190897995172127,
'n_estimators': 700, 'max_depth': 7, 'min_child_weight': 24}. Best is trial 19 with v
0.25731562046172163.
[I 2025-11-03 07:18:47,071] Trial 33 finished with value: 0.2575988145313977 and
parameters: {'lambda': 1.493698394869845, 'alpha': 0.0026893895290535097,
'colsample_bytree': 1.0, 'subsample': 0.8, 'learning_rate': 0.04217122951641485,
'n_estimators': 800, 'max_depth': 8, 'min_child_weight': 72}. Best is trial 19 with v
0.25731562046172163.
[I 2025-11-03 07:18:52,695] Trial 34 finished with value: 0.25737800681748224 and
parameters: {'lambda': 1.7132032645818234, 'alpha': 0.0053424955562438, 'colsample_by
0.7, 'subsample': 0.8, 'learning_rate': 0.03252593950740572, 'n_estimators': 700,
'max_depth': 6, 'min_child_weight': 30}. Best is trial 19 with value: 0.2573156204617
[I 2025-11-03 07:18:57,073] Trial 35 finished with value: 0.25808442532015785 and
parameters: {'lambda': 9.391317787021466, 'alpha': 0.005218083909051403,
'colsample_bytree': 0.9, 'subsample': 0.6, 'learning_rate': 0.024241465833981526,
'n_estimators': 500, 'max_depth': 6, 'min_child_weight': 23}. Best is trial 19 with v
0.25731562046172163.
[I 2025-11-03 07:18:59,557] Trial 36 finished with value: 0.25916812554118107 and
parameters: {'lambda': 4.909784171471608, 'alpha': 0.013490854289304358,
'colsample_bytree': 0.7, 'subsample': 0.8, 'learning_rate': 0.03953841169880165,
'n_estimators': 300, 'max_depth': 4, 'min_child_weight': 75}. Best is trial 19 with v
0.25731562046172163.
[I 2025-11-03 07:19:05,344] Trial 37 finished with value: 0.25924028289561596 and
parameters: {'lambda': 0.05715600687294148, 'alpha': 0.05806199432114279,
'colsample_bytree': 0.3, 'subsample': 0.7, 'learning_rate': 0.01730464958553354,
'n_estimators': 800, 'max_depth': 5, 'min_child_weight': 54}. Best is trial 19 with v
0.25731562046172163.
[I 2025-11-03 07:19:10,693] Trial 38 finished with value: 0.2582211941306502 and
parameters: {'lambda': 0.01549896014674224, 'alpha': 0.018020943075521705,
'colsample_bytree': 0.9, 'subsample': 1.0, 'learning_rate': 0.034835410475615995,
'n_estimators': 700, 'max_depth': 6, 'min_child_weight': 223}. Best is trial 19 with
0.25731562046172163.
[I 2025-11-03 07:19:16,408] Trial 39 finished with value: 0.25824831990058866 and
```

```
parameters: {'lambda': 0.879383651614276, 'alpha': 0.005041122652933428,
'colsample_bytree': 1.0, 'subsample': 0.5, 'learning_rate': 0.026443544364382947,
'n_estimators': 600, 'max_depth': 8, 'min_child_weight': 145}. Best is trial 19 with
0.25731562046172163.
[I 2025-11-03 07:19:22,063] Trial 40 finished with value: 0.25758404343100055 and
parameters: {'lambda': 0.12084691734055211, 'alpha': 0.0021322887924655498,
'colsample_bytree': 0.7, 'subsample': 0.6, 'learning_rate': 0.043627974503275185,
'n_estimators': 700, 'max_depth': 6, 'min_child_weight': 14}. Best is trial 19 with v
0.25731562046172163.
[I 2025-11-03 07:19:28,417] Trial 41 finished with value: 0.25740751772889653 and
parameters: {'lambda': 1.53067226017384, 'alpha': 0.003228818747108632, 'colsample_by
0.7, 'subsample': 0.8, 'learning_rate': 0.031779294830742454, 'n_estimators': 700,
'max_depth': 7, 'min_child_weight': 36}. Best is trial 19 with value: 0.2573156204617
[I 2025-11-03 07:19:34,907] Trial 42 finished with value: 0.2573833416910459 and
parameters: {'lambda': 1.8903837220166977, 'alpha': 0.001821357211682189,
'colsample_bytree': 0.7, 'subsample': 0.8, 'learning_rate': 0.032855631974691185,
'n_estimators': 700, 'max_depth': 7, 'min_child_weight': 20}. Best is trial 19 with v
0.25731562046172163.
[I 2025-11-03 07:19:40,871] Trial 43 finished with value: 0.2575609414293694 and
parameters: {'lambda': 2.197432379630974, 'alpha': 0.0017090459750847516,
'colsample_bytree': 0.7, 'subsample': 0.8, 'learning_rate': 0.04580523617648304,
'n_estimators': 600, 'max_depth': 8, 'min_child_weight': 61}. Best is trial 19 with v
0.25731562046172163.
[I 2025-11-03 07:19:48,157] Trial 44 finished with value: 0.2576033590333954 and
parameters: {'lambda': 0.39895164471356576, 'alpha': 0.012248725855203401,
'colsample_bytree': 0.7, 'subsample': 0.7, 'learning_rate': 0.03898593799710853,
'n_estimators': 800, 'max_depth': 7, 'min_child_weight': 15}. Best is trial 19 with v
0.25731562046172163.
[I 2025-11-03 07:19:57,175] Trial 45 finished with value: 0.2570974993002868 and
parameters: {'lambda': 5.886232957483388, 'alpha': 0.025940388011077873,
'colsample_bytree': 0.3, 'subsample': 0.8, 'learning_rate': 0.026569438393859245,
'n_estimators': 900, 'max_depth': 8, 'min_child_weight': 46}. Best is trial 45 with v
0.2570974993002868.
[I 2025-11-03 07:20:06,653] Trial 46 finished with value: 0.2572943246155355 and
parameters: {'lambda': 7.553103188892479, 'alpha': 0.13438983425704323, 'colsample_by
0.3, 'subsample': 0.8, 'learning_rate': 0.02615716099718383, 'n_estimators': 900,
'max_depth': 9, 'min_child_weight': 100}. Best is trial 45 with value: 0.257097499300
[I 2025-11-03 07:20:18,265] Trial 47 finished with value: 0.25734227315159125 and
parameters: {'lambda': 5.8455752130841905, 'alpha': 0.1231452832023838, 'colsample_by
0.3, 'subsample': 0.8, 'learning_rate': 0.020840511602705575, 'n_estimators': 1000,
'max_depth': 10, 'min_child_weight': 101}. Best is trial 45 with value: 0.25709749930
[I 2025-11-03 07:20:30,917] Trial 48 finished with value: 0.2574147253434204 and
parameters: {'lambda': 6.346579316448319, 'alpha': 0.13384077799292762, 'colsample_by
0.3, 'subsample': 0.8, 'learning_rate': 0.020161665431804185, 'n_estimators': 1000,
'max_depth': 11, 'min_child_weight': 101}. Best is trial 45 with value: 0.25709749930
[I 2025-11-03 07:20:41,257] Trial 49 finished with value: 0.25833729942246175 and
parameters: {'lambda': 5.240139445349718, 'alpha': 0.7204094288481924, 'colsample_byt
0.3, 'subsample': 0.7, 'learning_rate': 0.015534267841260203, 'n_estimators': 1000,
'max_depth': 9, 'min_child_weight': 170}. Best is trial 45 with value: 0.257097499300
[I 2025-11-03 07:20:41,258] A new study created in memory with name: no-
name-24d9e47d-39ff-4841-9e2b-2c86755c9e2a
```

```
✅ Best XGBClassifier parameters: {'lambda': 5.886232957483388, 'alpha':
0.025940388011077873, 'colsample_bytree': 0.3, 'subsample': 0.8, 'learning_rate':
0.026569438393859245, 'n_estimators': 900, 'max_depth': 8, 'min_child_weight': 46}

🚀 Tuning LGBMClassifier (GPU)...
```

```
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
1 warning generated.
[I 2025-11-03 07:20:57,851] Trial 0 finished with value: 0.2596210099951706 and param
{'lambda_l1': 0.002716602839774946, 'lambda_l2': 0.0003836313790088529, 'num_leaves'
'feature_fraction': 0.6543029519801753, 'bagging_fraction': 0.5318032723554944,
'bagging_freq': 4, 'min_child_samples': 56, 'learning_rate': 0.024324010658667954,
'n_estimators': 300, 'max_depth': 5}. Best is trial 0 with value: 0.2596210099951706
[I 2025-11-03 07:21:16,912] Trial 1 finished with value: 0.2577424765191688 and param
{'lambda_l1': 0.06680066069657667, 'lambda_l2': 0.00028671332725573625, 'num_leaves'
'feature_fraction': 0.9744573764305161, 'bagging_fraction': 0.6848279818999312,
'bagging_freq': 3, 'min_child_samples': 13, 'learning_rate': 0.06056701781170928,
'n_estimators': 500, 'max_depth': 5}. Best is trial 1 with value: 0.2577424765191688
[I 2025-11-03 07:21:27,768] Trial 2 finished with value: 0.263052686808262 and param
{'lambda_l1': 0.0020222387998461194, 'lambda_l2': 0.1947390221610764, 'num_leaves':
'feature_fraction': 0.9732074134274739, 'bagging_fraction': 0.7059110221226814,
'bagging_freq': 1, 'min_child_samples': 74, 'learning_rate': 0.2176739552662446,
'n_estimators': 300, 'max_depth': 7}. Best is trial 1 with value: 0.2577424765191688
[I 2025-11-03 07:21:58,601] Trial 3 finished with value: 0.25743496045312714 and
parameters: {'lambda_l1': 0.0013133751348417238, 'lambda_l2': 0.00042179946325548203
'num_leaves': 115, 'feature_fraction': 0.8019997291128946, 'bagging_fraction':
0.741481947208054, 'bagging_freq': 6, 'min_child_samples': 71, 'learning_rate':
0.058702360489712495, 'n_estimators': 900, 'max_depth': 5}. Best is trial 3 with valu
0.25743496045312714.
[I 2025-11-03 07:23:08,269] Trial 4 finished with value: 0.25745679287189865 and
parameters: {'lambda_l1': 0.0002621732001780611, 'lambda_l2': 0.001887306687168051,
'num_leaves': 179, 'feature_fraction': 0.6913051859410282, 'bagging_fraction':
0.9659991435169543, 'bagging_freq': 7, 'min_child_samples': 76, 'learning_rate':
0.01594148477545415, 'n_estimators': 900, 'max_depth': 11}. Best is trial 3 with val
0.25743496045312714.
```

```
[I 2025-11-03 07:23:28,097] Trial 5 finished with value: 0.25771091148873165 and
parameters: {'lambda_l1': 0.015182241770413525, 'lambda_l2': 0.0015200752229676073,
'num_leaves': 246, 'feature_fraction': 0.8880621678379127, 'bagging_fraction':
0.6964318809356786, 'bagging_freq': 5, 'min_child_samples': 80, 'learning_rate':
0.1749633527686237, 'n_estimators': 800, 'max_depth': 3}. Best is trial 3 with value
0.25743496045312714.
[I 2025-11-03 07:23:39,098] Trial 6 finished with value: 0.2581490067176458 and param
{'lambda_l1': 0.26673972807827945, 'lambda_l2': 0.0054649190527836485, 'num_leaves':
'feature_fraction': 0.6762239561642109, 'bagging_fraction': 0.8174726291630314,
'bagging_freq': 7, 'min_child_samples': 32, 'learning_rate': 0.0770329904540322,
'n_estimators': 400, 'max_depth': 3}. Best is trial 3 with value: 0.2574349604531271
[I 2025-11-03 07:24:04,648] Trial 7 finished with value: 0.2616319007377422 and param
{'lambda_l1': 0.0245656669975974, 'lambda_l2': 0.0032108559885385577, 'num_leaves': 1
'feature_fraction': 0.8898431880657955, 'bagging_fraction': 0.7350873042996415,
'bagging_freq': 5, 'min_child_samples': 93, 'learning_rate': 0.08686248711625927,
'n_estimators': 400, 'max_depth': 12}. Best is trial 3 with value: 0.257434960453127
[I 2025-11-03 07:24:27,728] Trial 8 finished with value: 0.2585985096333796 and param
{'lambda_l1': 0.0027781886246329474, 'lambda_l2': 0.00037221988853414494, 'num_leaves
249, 'feature_fraction': 0.8363847875841872, 'bagging_fraction': 0.6422222655715286,
'bagging_freq': 1, 'min_child_samples': 93, 'learning_rate': 0.012170551574233664,
'n_estimators': 800, 'max_depth': 5}. Best is trial 3 with value: 0.2574349604531271
[I 2025-11-03 07:24:41,252] Trial 9 finished with value: 0.26294327825373964 and
parameters: {'lambda_l1': 0.857595416585917, 'lambda_l2': 0.00018306083147395157,
'num_leaves': 178, 'feature_fraction': 0.7276619708739973, 'bagging_fraction':
0.6314031142147365, 'bagging_freq': 2, 'min_child_samples': 81, 'learning_rate':
0.01549029823847191, 'n_estimators': 300, 'max_depth': 5}. Best is trial 3 with value
0.25743496045312714.
[I 2025-11-03 07:25:27,399] Trial 10 finished with value: 0.26119359790080376 and
parameters: {'lambda_l1': 2.0925040665937483e-05, 'lambda_l2': 1.5476848641847573e-05
'num_leaves': 21, 'feature_fraction': 0.5396067837676868, 'bagging_fraction':
0.8506396331888332, 'bagging_freq': 6, 'min_child_samples': 52, 'learning_rate':
0.006048999174428362, 'n_estimators': 1000, 'max_depth': 9}. Best is trial 3 with val
0.25743496045312714.
[I 2025-11-03 07:26:21,918] Trial 11 finished with value: 0.2582849761729365 and
parameters: {'lambda_l1': 0.00014127218084153097, 'lambda_l2': 0.054740335783379945,
'num_leaves': 126, 'feature_fraction': 0.7879224442292634, 'bagging_fraction':
0.9691918108332408, 'bagging_freq': 7, 'min_child_samples': 61, 'learning_rate':
0.0345184775996353, 'n_estimators': 1000, 'max_depth': 12}. Best is trial 3 with valu
0.25743496045312714.
[I 2025-11-03 07:27:22,171] Trial 12 finished with value: 0.2596104828034562 and
parameters: {'lambda_l1': 0.00026919936857296015, 'lambda_l2': 0.015559082245543743,
'num_leaves': 98, 'feature_fraction': 0.5681303773228598, 'bagging_fraction':
0.9799202441381428, 'bagging_freq': 6, 'min_child_samples': 69, 'learning_rate':
0.006497042299170708, 'n_estimators': 800, 'max_depth': 10}. Best is trial 3 with val
0.25743496045312714.
[I 2025-11-03 07:28:11,850] Trial 13 finished with value: 0.25725828269692314 and
parameters: {'lambda_l1': 0.0002940013123952954, 'lambda_l2': 4.5225371776859564e-05,
'num_leaves': 207, 'feature_fraction': 0.6462374794303218, 'bagging_fraction':
0.8567269768323934, 'bagging_freq': 7, 'min_child_samples': 42, 'learning_rate':
0.020079871353289785, 'n_estimators': 900, 'max_depth': 7}. Best is trial 13 with val
0.25725828269692314.
[I 2025-11-03 07:28:44,157] Trial 14 finished with value: 0.2574534143995856 and
parameters: {'lambda_l1': 1.3517869679106238e-05, 'lambda_l2': 1.5632886280360352e-05
'num_leaves': 229, 'feature_fraction': 0.606454224560812, 'bagging_fraction':
0.857848207904601, 'bagging_freq': 6, 'min_child_samples': 41, 'learning_rate':
0.039045441981621194, 'n_estimators': 600, 'max_depth': 7}. Best is trial 13 with val
0.25725828269692314.
[I 2025-11-03 07:29:28,154] Trial 15 finished with value: 0.2642831285010989 and
parameters: {'lambda_l1': 7.3516609299707e-05, 'lambda_l2': 5.248591374988818e-05,
'num_leaves': 201, 'feature_fraction': 0.7692467630158706, 'bagging_fraction':
```

```
0.7879759882575366, 'bagging_freq': 5, 'min_child_samples': 30, 'learning_rate':
0.10412394927042785, 'n_estimators': 700, 'max_depth': 8}. Best is trial 13 with valu
0.25725828269692314.
[I 2025-11-03 07:30:11,957] Trial 16 finished with value: 0.25723239986063656 and
parameters: {'lambda_l1': 0.0006130434418029652, 'lambda_l2': 6.376572790581942e-05,
'num_leaves': 107, 'feature_fraction': 0.6120751671862034, 'bagging_fraction':
0.8940030237623764, 'bagging_freq': 4, 'min_child_samples': 45, 'learning_rate':
0.03996637371759323, 'n_estimators': 900, 'max_depth': 6}. Best is trial 16 with valu
0.25723239986063656.
[I 2025-11-03 07:31:02,547] Trial 17 finished with value: 0.25710944888518794 and
parameters: {'lambda_l1': 0.0006676400275768641, 'lambda_l2': 6.882280051749269e-05,
'num_leaves': 155, 'feature_fraction': 0.5063358344178233, 'bagging_fraction':
0.9060032142363916, 'bagging_freq': 3, 'min_child_samples': 43, 'learning_rate':
0.026077518808694123, 'n_estimators': 700, 'max_depth': 8}. Best is trial 17 with val
0.25710944888518794.
✅ Best LGBMClassifier parameters: {'lambda_l1': 0.0006676400275768641, 'lambda_l2':
6.882280051749269e-05, 'num_leaves': 155, 'feature_fraction': 0.5063358344178233,
'bagging_fraction': 0.9060032142363916, 'bagging_freq': 3, 'min_child_samples': 43,
'learning_rate': 0.026077518808694123, 'n_estimators': 700, 'max_depth': 8}

⚡ Evaluating Tuned Models on GPU...

XGBClassifier (GPU)            | LogLoss: 0.25703 | Accuracy: 0.90212 | ROC AUC: 0.91
LGBMClassifier (GPU)           | LogLoss: 0.25734 | Accuracy: 0.90231 | ROC AUC: 0.91
```

In [19]:
```python
# Select best model
best_idx = np.argmin(logloss_scores) # <-- Select based on lowest log_loss
best_model_name = model_names[best_idx]
best_model = models[best_idx][1]
print(f"\n✅ Best Model Based on LogLoss: {best_model_name}")

# Evaluate final model
y_pred = best_model.predict(X_val_scaled)
y_pred_proba = best_model.predict_proba(X_val_scaled)[:, 1]

ll_final = log_loss(y_val, y_pred_proba)
acc_final = accuracy_score(y_val, y_pred)
auc_final = roc_auc_score(y_val, y_pred_proba)

print("\n✅ Final Model Evaluation (on validation set):")
print(f"LogLoss    : {ll_final:.5f}")
print(f"Accuracy   : {acc_final:.5f}")
print(f"ROC AUC Score: {auc_final:.5f}")
```

```
✅ Best Model Based on LogLoss: XGBClassifier (GPU)

✅ Final Model Evaluation (on validation set):
LogLoss    : 0.25703
Accuracy   : 0.90212
ROC AUC Score: 0.91283
```

Selecting best model and Generating Submission

In [20]:
```python
print("\n🚀 Retraining the best model on full training data...")

# Prepare full training features and target
X_full = train_processed.drop(columns=['loan_paid_back'], errors='ignore')
y_full = train_processed['loan_paid_back']

# Ensure all columns are numeric
```

```
X_full = X_full.select_dtypes(include=[np.number])

# Scale full data using the same scaler
if scaler is not None:
    # We must re-fit the scaler on the FULL training data
    X_full_scaled = scaler.fit_transform(X_full)
else:
    X_full_scaled = X_full

# Retrain best model on the full scaled dataset
best_model.fit(X_full_scaled, y_full)

print(f"✅ Model retrained successfully: {best_model_name} ({best_model.__class_
```

```
🚀 Retraining the best model on full training data...
✅ Model retrained successfully: XGBClassifier (GPU) (XGBClassifier)
```

Submission creation

In [21]:
```
X_submission = test_processed.select_dtypes(include=[np.number])

# Ensure column order matches the training data
X_submission = X_submission[X_full.columns]

# Scale using the scaler that was fit on X_full
if scaler is not None:
    X_submission_scaled = scaler.transform(X_submission)
else:
    X_submission_scaled = X_submission
```

In [22]:
```
print("🔮 Generating predictions using the best model...")
# Predict probabilities for the submission
submission_preds = best_model.predict_proba(X_submission_scaled)[:, 1]

# Optional: clip predictions to valid range [0, 1]
submission_preds = np.clip(submission_preds, 0, 1)
```

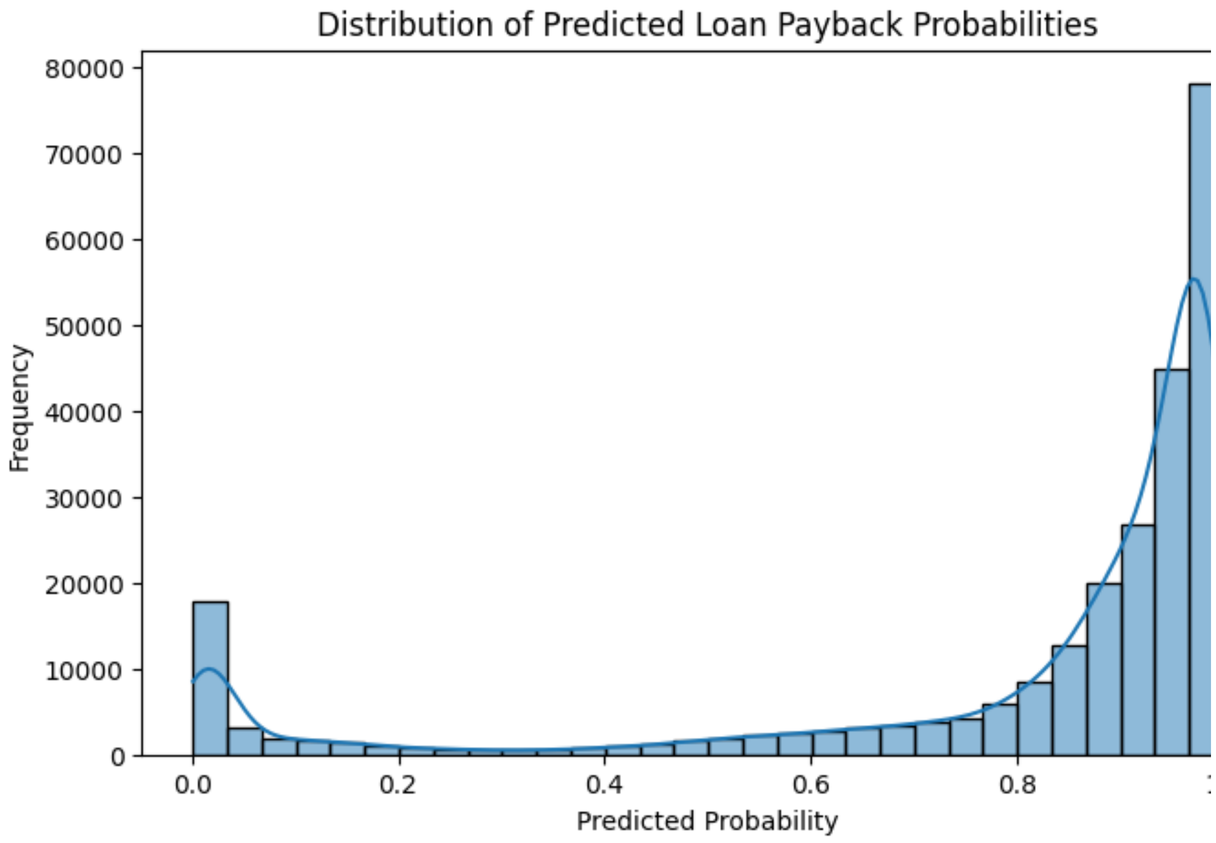
```
🔮 Generating predictions using the best model...
```

In [23]:
```
submission = pd.DataFrame({
    'id': test_ids,
    'loan_paid_back': submission_preds
})
submission.to_csv('submission.csv', index=False)
print("\n✅ Submission file 'submission.csv' generated successfully!")
display(submission.head())
```

```
✅ Submission file 'submission.csv' generated successfully!
```

	id	loan_paid_back
0	593994	0.944431
1	593995	0.955119
2	593996	0.431062
3	593997	0.962783
4	593998	0.978027

```
In [24]: plt.figure(figsize=(8, 5))
         sns.histplot(submission['loan_paid_back'], bins=30, kde=True)
         plt.title('Distribution of Predicted Loan Payback Probabilities')
         plt.xlabel('Predicted Probability')
         plt.ylabel('Frequency')
         plt.show()
```



```
In [ ]:
```