In [1]:
```
# --- Setup Project Path ---
import sys
import os

# Add the project root to the Python path to allow imports from 'src'
project_root = os.path.abspath(os.path.join(os.getcwd(), '..'))
if project_root not in sys.path:
    sys.path.append(project_root)

# --- General Imports ---
import yaml
import pandas as pd

# --- Custom Project Imports ---
from src.training_pipeline import TrainingPipeline
from src.backtester import VectorizedBacktester

# --- Load Configuration ---
config_path = os.path.join(project_root, 'configs', 'config.yaml')
print(f"Loading configuration from: {config_path}")
with open(config_path, 'r') as file:
    config = yaml.safe_load(file)
print("Configuration loaded successfully.")
```

```
Loading configuration from: C:\Projetos_Python\gld_lstm_strategy\configs\config.yaml
Configuration loaded successfully.
```

In [2]:
```
# --- 1. Run Pipeline with LSTM Model ---

# Instantiate the main pipeline
pipeline = TrainingPipeline(config=config, project_root=project_root)

print("="*50)
print("RUNNING PIPELINE FOR LSTM MODEL")
print("="*50)

# Run the static test specifically for the 'lstm' model type
lstm_results = pipeline.run_static_test(model_type='lstm')
```

```python
print("\n\n✅ --- LSTM Pipeline Finished! --- ✅")
```

```
Random seeds set to 2025 for reproducibility.
==================================================
RUNNING PIPELINE FOR LSTM MODEL
==================================================



===== Starting STATIC Test Run for model_type='lstm' =====
===== Step 1: Loading and Preparing Full Dataset =====
--- Loading Main Asset Data ---
Loading GLD data from local cache: C:\Projetos_Python\gld_lstm_strategy\data\gld_data.csv

--- Loading Macroeconomic Data ---
Loading DX-Y.NYB data from local cache: C:\Projetos_Python\gld_lstm_strategy\data\dx-y.nyb_data.csv
Loading ^TNX data from local cache: C:\Projetos_Python\gld_lstm_strategy\data\^tnx_data.csv
Loading ^VIX data from local cache: C:\Projetos_Python\gld_lstm_strategy\data\^vix_data.csv
Loading CL=F data from local cache: C:\Projetos_Python\gld_lstm_strategy\data\cl=f_data.csv
Loading SI=F data from local cache: C:\Projetos_Python\gld_lstm_strategy\data\si=f_data.csv
Loading TIP data from local cache: C:\Projetos_Python\gld_lstm_strategy\data\tip_data.csv
Loading HG=F data from local cache: C:\Projetos_Python\gld_lstm_strategy\data\hg=f_data.csv

===== Starting Feature Engineering Pipeline =====
Step 1: Creating custom OHLCV features...
Step 2: Applying 'All' technical indicator strategy from pandas_ta...
```

```
130it [00:26,  4.86it/s]
```

```
 -> Dropped 2 redundant TA columns.
Step 3: Creating custom interaction and ratio features...
Step 4: Creating lagged and momentum features...
Step 5: Merging macroeconomic features...
 -> Macro features merged and forward-filled.
Step 6: Defining target variable...

Pipeline complete. Dropped 77 rows with NaN values.
Final dataset shape: (2438, 247)
===============================================



--- Splitting data chronologically ---
Train set size: 1761, Validation set size: 311, Test set size: 366

--- Running Feature Selection ---
Selected 37 features via BorutaPy.

--- Scaling data ---

--- Preparing data and training LSTM Model ---
Epoch 1/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 40s 524ms/step - accuracy: 0.5030 - loss: 0.7019 - val_accuracy: 0.4180 - val_loss: 0.7305
Epoch 2/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 40s 496ms/step - accuracy: 0.5585 - loss: 0.6879 - val_accuracy: 0.4180 - val_loss: 0.7121
Epoch 3/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 29s 572ms/step - accuracy: 0.5489 - loss: 0.6861 - val_accuracy: 0.6557 - val_loss: 0.6872
Epoch 4/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 22s 435ms/step - accuracy: 0.5691 - loss: 0.6838 - val_accuracy: 0.4180 - val_loss: 0.6974
Epoch 5/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 48s 567ms/step - accuracy: 0.5838 - loss: 0.6729 - val_accuracy: 0.4180 - val_loss: 0.7160
Epoch 6/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 23s 463ms/step - accuracy: 0.6016 - loss: 0.6685 - val_accuracy: 0.4180 - val_loss: 0.7056
Epoch 7/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 25s 502ms/step - accuracy: 0.6108 - loss: 0.6620 - val_accuracy: 0.4590 - val_loss: 0.7094
Epoch 8/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 22s 438ms/step - accuracy: 0.6115 - loss: 0.6596 - val_accuracy: 0.4426 - val_loss: 0.7162
Epoch 9/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 42s 459ms/step - accuracy: 0.6270 - loss: 0.6525 - val_accuracy: 0.4918 - val_loss: 0.7050
Epoch 10/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 22s 445ms/step - accuracy: 0.6184 - loss: 0.6473 - val_accuracy: 0.5082 - val_loss: 0.7017
```

```
Epoch 11/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 39s 404ms/step - accuracy: 0.6291 - loss: 0.6392 - val_accuracy: 0.5164 - val_loss: 0.7020
Epoch 12/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 22s 430ms/step - accuracy: 0.6591 - loss: 0.6368 - val_accuracy: 0.5902 - val_loss: 0.6677
Epoch 13/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 39s 387ms/step - accuracy: 0.6671 - loss: 0.6273 - val_accuracy: 0.6311 - val_loss: 0.6537
Epoch 14/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 16s 322ms/step - accuracy: 0.6855 - loss: 0.6150 - val_accuracy: 0.5984 - val_loss: 0.6434
Epoch 15/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 22s 448ms/step - accuracy: 0.6800 - loss: 0.6257 - val_accuracy: 0.5410 - val_loss: 0.6906
Epoch 16/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 43s 484ms/step - accuracy: 0.6736 - loss: 0.6041 - val_accuracy: 0.5738 - val_loss: 0.6412
Epoch 17/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 24s 484ms/step - accuracy: 0.7016 - loss: 0.5977 - val_accuracy: 0.6721 - val_loss: 0.6043
Epoch 18/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 33s 310ms/step - accuracy: 0.7139 - loss: 0.5822 - val_accuracy: 0.6230 - val_loss: 0.6172
Epoch 19/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 15s 301ms/step - accuracy: 0.7208 - loss: 0.5667 - val_accuracy: 0.6721 - val_loss: 0.6169
Epoch 20/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 18s 354ms/step - accuracy: 0.7158 - loss: 0.5691 - val_accuracy: 0.6803 - val_loss: 0.6158
Epoch 21/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 23s 412ms/step - accuracy: 0.7198 - loss: 0.5646 - val_accuracy: 0.6885 - val_loss: 0.5929
Epoch 22/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 14s 278ms/step - accuracy: 0.7396 - loss: 0.5445 - val_accuracy: 0.6148 - val_loss: 0.6275
Epoch 23/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 14s 275ms/step - accuracy: 0.7159 - loss: 0.5560 - val_accuracy: 0.6803 - val_loss: 0.6010
Epoch 24/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 15s 289ms/step - accuracy: 0.7295 - loss: 0.5453 - val_accuracy: 0.7049 - val_loss: 0.5656
Epoch 25/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 14s 273ms/step - accuracy: 0.7227 - loss: 0.5308 - val_accuracy: 0.6803 - val_loss: 0.6030
Epoch 26/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 21s 276ms/step - accuracy: 0.7489 - loss: 0.5219 - val_accuracy: 0.7213 - val_loss: 0.5546
Epoch 27/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 264ms/step - accuracy: 0.7563 - loss: 0.5015 - val_accuracy: 0.6885 - val_loss: 0.5792
Epoch 28/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 268ms/step - accuracy: 0.7274 - loss: 0.5211 - val_accuracy: 0.7131 - val_loss: 0.5323
Epoch 29/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 14s 273ms/step - accuracy: 0.7863 - loss: 0.4833 - val_accuracy: 0.6721 - val_loss: 0.5674
Epoch 30/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 259ms/step - accuracy: 0.7763 - loss: 0.4730 - val_accuracy: 0.7049 - val_loss: 0.5492
Epoch 31/100
```

```
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 261ms/step - accuracy: 0.7885 - loss: 0.4418 - val_accuracy: 0.7377 - val_loss: 0.5541
Epoch 32/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 262ms/step - accuracy: 0.7878 - loss: 0.4426 - val_accuracy: 0.7213 - val_loss: 0.5174
Epoch 33/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 20s 411ms/step - accuracy: 0.7949 - loss: 0.4380 - val_accuracy: 0.7787 - val_loss: 0.4662
Epoch 34/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 14s 282ms/step - accuracy: 0.8123 - loss: 0.4249 - val_accuracy: 0.7295 - val_loss: 0.5130
Epoch 35/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 14s 272ms/step - accuracy: 0.8041 - loss: 0.4283 - val_accuracy: 0.7131 - val_loss: 0.5576
Epoch 36/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 14s 286ms/step - accuracy: 0.8196 - loss: 0.4213 - val_accuracy: 0.7459 - val_loss: 0.5098
Epoch 37/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 14s 278ms/step - accuracy: 0.8138 - loss: 0.4153 - val_accuracy: 0.6967 - val_loss: 0.5521
Epoch 38/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 14s 274ms/step - accuracy: 0.8091 - loss: 0.4115 - val_accuracy: 0.7131 - val_loss: 0.5675
Epoch 39/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 20s 264ms/step - accuracy: 0.8183 - loss: 0.4031 - val_accuracy: 0.7459 - val_loss: 0.5175
Epoch 40/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 14s 273ms/step - accuracy: 0.8211 - loss: 0.4185 - val_accuracy: 0.7049 - val_loss: 0.5543
Epoch 41/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 14s 287ms/step - accuracy: 0.8385 - loss: 0.3914 - val_accuracy: 0.7295 - val_loss: 0.5626
Epoch 42/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 14s 279ms/step - accuracy: 0.8278 - loss: 0.3902 - val_accuracy: 0.7295 - val_loss: 0.5882
Epoch 43/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 19s 237ms/step - accuracy: 0.8073 - loss: 0.3991 - val_accuracy: 0.7295 - val_loss: 0.6361
Epoch 44/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 269ms/step - accuracy: 0.8337 - loss: 0.3714 - val_accuracy: 0.7459 - val_loss: 0.5831
Epoch 45/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 14s 274ms/step - accuracy: 0.8271 - loss: 0.3825 - val_accuracy: 0.7459 - val_loss: 0.5808
Epoch 46/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 258ms/step - accuracy: 0.8444 - loss: 0.3550 - val_accuracy: 0.7705 - val_loss: 0.6124
Epoch 47/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 264ms/step - accuracy: 0.8418 - loss: 0.3579 - val_accuracy: 0.7705 - val_loss: 0.6309
Epoch 48/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 258ms/step - accuracy: 0.8472 - loss: 0.3670 - val_accuracy: 0.6967 - val_loss: 0.6306
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 142ms/step

--- Evaluating Model on Unseen Test Data ---


✅ --- LSTM Pipeline Finished! --- ✅
```

In [3]:
```python
# --- 2. Run Pipeline with XGBoost Model ---

# We can reuse the same pipeline instance
print("="*50)
print("RUNNING PIPELINE FOR XGBOOST MODEL")
print("="*50)

# Run the static test specifically for the 'xgboost' model type
xgboost_results = pipeline.run_static_test(model_type='xgboost')

print("\n\n✅ --- XGBoost Pipeline Finished! --- ✅")
```

```
==================================================
RUNNING PIPELINE FOR XGBOOST MODEL
==================================================


===== Starting STATIC Test Run for model_type='xgboost' =====
===== Step 1: Loading and Preparing Full Dataset =====
--- Loading Main Asset Data ---
Loading GLD data from local cache: C:\Projetos_Python\gld_lstm_strategy\data\gld_data.csv

--- Loading Macroeconomic Data ---
Loading DX-Y.NYB data from local cache: C:\Projetos_Python\gld_lstm_strategy\data\dx-y.nyb_data.csv
Loading ^TNX data from local cache: C:\Projetos_Python\gld_lstm_strategy\data\^tnx_data.csv
Loading ^VIX data from local cache: C:\Projetos_Python\gld_lstm_strategy\data\^vix_data.csv
Loading CL=F data from local cache: C:\Projetos_Python\gld_lstm_strategy\data\cl=f_data.csv
Loading SI=F data from local cache: C:\Projetos_Python\gld_lstm_strategy\data\si=f_data.csv
Loading TIP data from local cache: C:\Projetos_Python\gld_lstm_strategy\data\tip_data.csv
Loading HG=F data from local cache: C:\Projetos_Python\gld_lstm_strategy\data\hg=f_data.csv

===== Starting Feature Engineering Pipeline =====
Step 1: Creating custom OHLCV features...
Step 2: Applying 'All' technical indicator strategy from pandas_ta...
```

```
130it [00:12, 10.80it/s]
```

```
 -> Dropped 2 redundant TA columns.
Step 3: Creating custom interaction and ratio features...
Step 4: Creating lagged and momentum features...
Step 5: Merging macroeconomic features...
 -> Macro features merged and forward-filled.
Step 6: Defining target variable...

Pipeline complete. Dropped 77 rows with NaN values.
Final dataset shape: (2438, 247)
===============================================


--- Splitting data chronologically ---
Train set size: 1761, Validation set size: 311, Test set size: 366

--- Running Feature Selection ---
Selected 37 features via BorutaPy.

--- Scaling data ---

--- Preparing data and training XGBoost Model ---
```

```
C:\Users\rbert\.venv\lib\site-packages\xgboost\callback.py:386: UserWarning:

[14:18:19] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.
```

```
--- Evaluating Model on Unseen Test Data ---


✅ --- XGBoost Pipeline Finished! --- ✅
```

In [4]:

```
# --- 3. Run Backtests for Both Models ---

def run_backtest_for_results(results_dict, config, model_name):
    """
    Helper function to run the backtest for a given results dictionary.
    """
    print("\n" + "="*50)
    print(f"RUNNING BACKTEST FOR {model_name.upper()} MODEL")
```

```
    print("="*50)

    # Extract necessary data from the results dictionary
    processed_data = results_dict['processed_data_for_backtest']
    y_pred_proba = results_dict['pred_probas']

    # Recreate X_test to get the prices and the correct date index
    X = processed_data.drop(columns=[config['TARGET_NAME']])
    train_val_size = int(len(X) * (1 - config['TEST_SIZE']))
    X_test = X.iloc[train_val_size:]

    # Convert probabilities to binary signals
    test_predictions = (y_pred_proba > 0.5).astype(int)

    # Create a pandas Series for the signals with the correct date index
    if model_name.lower() == 'lstm':
        signal_dates = X_test.index[config['TIME_STEPS']:]
    else: # XGBoost uses 2D data, so no offset is needed for the index
        signal_dates = X_test.index

    signals_series = pd.Series(test_predictions, index=signal_dates, name="signal")

    # Get the price data for the same period
    price_data_for_backtest = X_test.loc[signal_dates]

    # Instantiate and run the backtester
    backtester = VectorizedBacktester(
        price_data=price_data_for_backtest,
        signals=signals_series,
        config=config
    )

    portfolio = backtester.run(commission=0.001, slippage=0.001)
    return portfolio

# Run the backtest for each model's results
lstm_portfolio = run_backtest_for_results(lstm_results, config, "LSTM")
xgboost_portfolio = run_backtest_for_results(xgboost_results, config, "XGBoost")
```

```
==================================================
RUNNING BACKTEST FOR LSTM MODEL
==================================================

===== Starting Vectorized Backtest with vectorbt =====

--- Backtest Performance Stats ---
Start                          2024-04-18 00:00:00
End                            2024-12-30 00:00:00
Period                           177 days 00:00:00
Start Value                                  100.0
End Value                                114.64216
Total Return [%]                          14.64216
Benchmark Return [%]                        9.2085
Max Gross Exposure [%]                       100.0
Total Fees Paid                           2.349421
Max Drawdown [%]                          3.349057
Max Drawdown Duration             36 days 00:00:00
Total Trades                                    11
Total Closed Trades                             11
Total Open Trades                                0
Open Trade PnL                                 0.0
Win Rate [%]                             72.727273
Best Trade [%]                            4.984358
Worst Trade [%]                          -0.796262
Avg Winning Trade [%]                     1.972687
Avg Losing Trade [%]                     -0.622551
Avg Winning Trade Duration         5 days 06:00:00
Avg Losing Trade Duration          3 days 16:00:00
Profit Factor                             8.454068
Expectancy                                1.331105
Sharpe Ratio                              2.994728
Calmar Ratio                              9.718887
Omega Ratio                               2.018344
Sortino Ratio                             6.045033
dtype: object

--- Plotting Equity Curve and Drawdowns ---
```

```
===== Backtest Finished =====

==================================================
RUNNING BACKTEST FOR XGBOOST MODEL
==================================================

===== Starting Vectorized Backtest with vectorbt =====

--- Backtest Performance Stats ---
Start                                2023-07-19 00:00:00
End                                  2024-12-30 00:00:00
Period                                 366 days 00:00:00
Start Value                                        100.0
End Value                                     177.072868
Total Return [%]                               77.072868
Benchmark Return [%]                           31.012145
Max Gross Exposure [%]                             100.0
Total Fees Paid                                10.306133
Max Drawdown [%]                                2.602918
Max Drawdown Duration                   21 days 00:00:00
Total Trades                                          38
Total Closed Trades                                   38
Total Open Trades                                      0
Open Trade PnL                                       0.0
Win Rate [%]                                   86.842105
Best Trade [%]                                  7.627185
Worst Trade [%]                                -0.769631
Avg Winning Trade [%]                           1.803887
Avg Losing Trade [%]                           -0.242435
Avg Winning Trade Duration    5 days 21:49:05.454545454
Avg Losing Trade Duration                2 days 09:36:00
Profit Factor                                   61.57335
Expectancy                                      2.028233
Sharpe Ratio                                    5.081341
Calmar Ratio                                   29.504051
Omega Ratio                                     2.474839
Sortino Ratio                                   11.73365
dtype: object

--- Plotting Equity Curve and Drawdowns ---
```

```
===== Backtest Finished =====
```

In [5]:
```
# --- 4. Final Results Comparison ---

# Extract the statistics from both portfolio objects
lstm_stats = lstm_portfolio.stats()
xgboost_stats = xgboost_portfolio.stats()

# Define the key metrics we want to compare
metrics_to_compare = [
    'Total Return [%]',
    'Benchmark Return [%]',
    'Sharpe Ratio',
    'Sortino Ratio',
    'Max Drawdown [%]',
    'Win Rate [%]',
    'Profit Factor',
    'Total Trades'
]

# Create a comparison DataFrame
comparison_df = pd.DataFrame({
    'LSTM': lstm_stats[metrics_to_compare],
    'XGBoost': xgboost_stats[metrics_to_compare]
})
```

```
print("\n\n" + "="*50)
print("MODEL BENCHMARK COMPARISON")
print("="*50)
display(comparison_df.round(4))
```

```
==================================================
MODEL BENCHMARK COMPARISON
==================================================
```

	LSTM	XGBoost
Total Return [%]	14.64216	77.072868
Benchmark Return [%]	9.2085	31.012145
Sharpe Ratio	2.994728	5.081341
Sortino Ratio	6.045033	11.73365
Max Drawdown [%]	3.349057	2.602918
Win Rate [%]	72.727273	86.842105
Profit Factor	8.454068	61.57335
Total Trades	11	38

5. Benchmark Conclusion

The results from the side-by-side comparison provide a clear and conclusive winner. After running both models through the same rigorous pipeline, the data shows the following:

- **Total Return:** The **XGBoost** model generated a vastly superior **Total Return of 77.07%**, compared to 14.64% for the LSTM.
- **Risk-Adjusted Return:** The **XGBoost** model demonstrated exceptional risk-adjusted performance, with a **Sharpe Ratio of 5.08** and a **Sortino Ratio of 11.73**, both significantly higher than the LSTM's.

- **Risk Control:** The **XGBoost** model also proved to be better at preserving capital, achieving a lower **Max Drawdown of 2.60%** versus the LSTM's 3.35%.
- **Consistency:** With a **Win Rate of 86.8%** across 38 trades, the **XGBoost** model was far more consistent than the LSTM.

Verdict: Based on these results, the **XGBoost model is the unequivocal choice** for this trading strategy, as it delivers dramatically higher returns, superior risk-adjusted performance, greater consistency, and better risk control.

This outcome suggests that for this problem, the rich, tabular dataset created during the comprehensive feature engineering phase is best leveraged by a powerful tree-based ensemble like XGBoost, which excels at finding complex, non-linear relationships between predictive features.