In [1]:
```python
# --- 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]:
```python
# --- 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')
```

```
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 ---
File not found for SPY. Downloading data...
YF.download() has changed argument auto_adjust default to True
Data for SPY saved to C:\Projetos_Python\gld_lstm_strategy\data\spy_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 [00:11, 11.66it/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 25 features via BorutaPy.

--- Scaling data ---

--- Preparing data and training LSTM Model ---
Epoch 1/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 18s 276ms/step - accuracy: 0.5639 - loss: 0.6850 - val_accuracy: 0.6639 - val_loss: 0.6379
Epoch 2/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 266ms/step - accuracy: 0.6347 - loss: 0.6575 - val_accuracy: 0.6639 - val_loss: 0.6386
Epoch 3/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 250ms/step - accuracy: 0.6386 - loss: 0.6599 - val_accuracy: 0.6639 - val_loss: 0.6370
Epoch 4/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 252ms/step - accuracy: 0.6415 - loss: 0.6589 - val_accuracy: 0.6639 - val_loss: 0.6364
Epoch 5/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 258ms/step - accuracy: 0.6382 - loss: 0.6522 - val_accuracy: 0.6639 - val_loss: 0.6366
Epoch 6/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 253ms/step - accuracy: 0.6396 - loss: 0.6543 - val_accuracy: 0.6639 - val_loss: 0.6354
Epoch 7/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 14s 272ms/step - accuracy: 0.6375 - loss: 0.6565 - val_accuracy: 0.6639 - val_loss: 0.6348
Epoch 8/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 267ms/step - accuracy: 0.6382 - loss: 0.6501 - val_accuracy: 0.6639 - val_loss: 0.6345
Epoch 9/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 256ms/step - accuracy: 0.6387 - loss: 0.6494 - val_accuracy: 0.6639 - val_loss: 0.6345
Epoch 10/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 263ms/step - accuracy: 0.6397 - loss: 0.6519 - val_accuracy: 0.6639 - val_loss: 0.6316
```

```
Epoch 11/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 269ms/step - accuracy: 0.6389 - loss: 0.6475 - val_accuracy: 0.6639 - val_loss: 0.6297
Epoch 12/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 14s 278ms/step - accuracy: 0.6421 - loss: 0.6417 - val_accuracy: 0.6639 - val_loss: 0.6269
Epoch 13/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 261ms/step - accuracy: 0.6370 - loss: 0.6405 - val_accuracy: 0.6639 - val_loss: 0.6307
Epoch 14/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 14s 286ms/step - accuracy: 0.6466 - loss: 0.6400 - val_accuracy: 0.6639 - val_loss: 0.6316
Epoch 15/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 264ms/step - accuracy: 0.6568 - loss: 0.6312 - val_accuracy: 0.6885 - val_loss: 0.6389
Epoch 16/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 16s 329ms/step - accuracy: 0.6561 - loss: 0.6237 - val_accuracy: 0.6639 - val_loss: 0.6362
Epoch 17/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 22s 358ms/step - accuracy: 0.6721 - loss: 0.6261 - val_accuracy: 0.6393 - val_loss: 0.6675
Epoch 18/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 262ms/step - accuracy: 0.6515 - loss: 0.6213 - val_accuracy: 0.6803 - val_loss: 0.6206
Epoch 19/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 254ms/step - accuracy: 0.6797 - loss: 0.5959 - val_accuracy: 0.6148 - val_loss: 0.6715
Epoch 20/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 264ms/step - accuracy: 0.6853 - loss: 0.6062 - val_accuracy: 0.7049 - val_loss: 0.6124
Epoch 21/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 253ms/step - accuracy: 0.7023 - loss: 0.5763 - val_accuracy: 0.6639 - val_loss: 0.6162
Epoch 22/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 252ms/step - accuracy: 0.7105 - loss: 0.5803 - val_accuracy: 0.6475 - val_loss: 0.6418
Epoch 23/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 257ms/step - accuracy: 0.7186 - loss: 0.5753 - val_accuracy: 0.6557 - val_loss: 0.6045
Epoch 24/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 257ms/step - accuracy: 0.7174 - loss: 0.5590 - val_accuracy: 0.6639 - val_loss: 0.6014
Epoch 25/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 254ms/step - accuracy: 0.7301 - loss: 0.5409 - val_accuracy: 0.6803 - val_loss: 0.6178
Epoch 26/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 12s 247ms/step - accuracy: 0.7298 - loss: 0.5368 - val_accuracy: 0.6393 - val_loss: 0.6278
Epoch 27/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 254ms/step - accuracy: 0.7436 - loss: 0.5328 - val_accuracy: 0.6721 - val_loss: 0.6223
Epoch 28/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 259ms/step - accuracy: 0.7297 - loss: 0.5325 - val_accuracy: 0.6803 - val_loss: 0.6355
Epoch 29/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 251ms/step - accuracy: 0.7342 - loss: 0.5310 - val_accuracy: 0.6885 - val_loss: 0.6277
Epoch 30/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 12s 248ms/step - accuracy: 0.7544 - loss: 0.5127 - val_accuracy: 0.6721 - val_loss: 0.6119
Epoch 31/100
```

```
50/50 ━━━━━━━━━━━━━━━━━━━━ 12s 249ms/step - accuracy: 0.7621 - loss: 0.5110 - val_accuracy: 0.6721 - val_loss: 0.6063
Epoch 32/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 12s 246ms/step - accuracy: 0.7480 - loss: 0.5161 - val_accuracy: 0.6721 - val_loss: 0.6247
Epoch 33/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 12s 246ms/step - accuracy: 0.7548 - loss: 0.5072 - val_accuracy: 0.6721 - val_loss: 0.6205
Epoch 34/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 12s 247ms/step - accuracy: 0.7572 - loss: 0.5047 - val_accuracy: 0.6803 - val_loss: 0.5881
Epoch 35/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 260ms/step - accuracy: 0.7708 - loss: 0.4931 - val_accuracy: 0.6885 - val_loss: 0.5889
Epoch 36/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 253ms/step - accuracy: 0.7766 - loss: 0.4822 - val_accuracy: 0.6885 - val_loss: 0.5727
Epoch 37/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 12s 246ms/step - accuracy: 0.7875 - loss: 0.4718 - val_accuracy: 0.6803 - val_loss: 0.6367
Epoch 38/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 12s 246ms/step - accuracy: 0.7685 - loss: 0.4720 - val_accuracy: 0.6885 - val_loss: 0.5900
Epoch 39/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 259ms/step - accuracy: 0.7867 - loss: 0.4490 - val_accuracy: 0.6639 - val_loss: 0.6281
Epoch 40/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 254ms/step - accuracy: 0.7833 - loss: 0.4720 - val_accuracy: 0.6557 - val_loss: 0.6164
Epoch 41/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 252ms/step - accuracy: 0.7663 - loss: 0.4718 - val_accuracy: 0.6885 - val_loss: 0.6234
Epoch 42/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 12s 247ms/step - accuracy: 0.7703 - loss: 0.4666 - val_accuracy: 0.6967 - val_loss: 0.5892
Epoch 43/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 253ms/step - accuracy: 0.7749 - loss: 0.4687 - val_accuracy: 0.6721 - val_loss: 0.6437
Epoch 44/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 263ms/step - accuracy: 0.7817 - loss: 0.4506 - val_accuracy: 0.6967 - val_loss: 0.6298
Epoch 45/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 12s 248ms/step - accuracy: 0.7785 - loss: 0.4352 - val_accuracy: 0.6885 - val_loss: 0.6529
Epoch 46/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 258ms/step - accuracy: 0.7796 - loss: 0.4342 - val_accuracy: 0.6311 - val_loss: 0.6399
Epoch 47/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 251ms/step - accuracy: 0.7832 - loss: 0.4497 - val_accuracy: 0.6885 - val_loss: 0.6456
Epoch 48/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 12s 247ms/step - accuracy: 0.7947 - loss: 0.4348 - val_accuracy: 0.6639 - val_loss: 0.6748
Epoch 49/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 12s 247ms/step - accuracy: 0.8120 - loss: 0.4335 - val_accuracy: 0.6639 - val_loss: 0.6750
Epoch 50/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 254ms/step - accuracy: 0.8097 - loss: 0.4165 - val_accuracy: 0.6639 - val_loss: 0.6776
Epoch 51/100
50/50 ━━━━━━━━━━━━━━━━━━━━ 13s 251ms/step - accuracy: 0.7947 - loss: 0.4244 - val_accuracy: 0.6721 - val_loss: 0.6832
```

```
6/6 ━━━━━━━━━━━━━━━━━━━━ 2s 181ms/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 SPY data from local cache: C:\Projetos_Python\gld_lstm_strategy\data\spy_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 [00:09, 14.08it/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 25 features via BorutaPy.

--- Scaling data ---

--- Preparing data and training XGBoost Model ---
C:\Users\rbert\.venv\lib\site-packages\xgboost\callback.py:386: UserWarning:

[15:47:40] 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]:
```python
# --- 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                               128.792376
Total Return [%]                         28.792376
Benchmark Return [%]                     18.898143
Max Gross Exposure [%]                       100.0
Total Fees Paid                           1.858152
Max Drawdown [%]                           2.29424
Max Drawdown Duration             22 days 00:00:00
Total Trades                                     8
Total Closed Trades                              8
Total Open Trades                                0
Open Trade PnL                                 0.0
Win Rate [%]                                 100.0
Best Trade [%]                             6.57933
Worst Trade [%]                           1.864559
Avg Winning Trade [%]                     3.231731
Avg Losing Trade [%]                           NaN
Avg Winning Trade Duration        12 days 00:00:00
Avg Losing Trade Duration                      NaT
Profit Factor                                  inf
Expectancy                                3.599047
Sharpe Ratio                              5.049487
Calmar Ratio                             29.859026
Omega Ratio                               2.685281
Sortino Ratio                             9.870426
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                               155.118712
Total Return [%]                         55.118712
Benchmark Return [%]                     31.874076
Max Gross Exposure [%]                       100.0
Total Fees Paid                           7.263764
Max Drawdown [%]                          3.106266
Max Drawdown Duration             19 days 00:00:00
Total Trades                                    29
Total Closed Trades                             29
Total Open Trades                                0
Open Trade PnL                                 0.0
Win Rate [%]                             86.206897
Best Trade [%]                             6.11943
Worst Trade [%]                          -0.619491
Avg Winning Trade [%]                     1.825389
Avg Losing Trade [%]                     -0.242082
Avg Winning Trade Duration         8 days 01:55:12
Avg Losing Trade Duration          1 days 18:00:00
Profit Factor                            41.918888
Expectancy                                1.900645
Sharpe Ratio                              4.032945
Calmar Ratio                             17.684501
Omega Ratio                               2.065181
Sortino Ratio                             7.020935
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 [%]	28.792376	55.118712
Benchmark Return [%]	18.898143	31.874076
Sharpe Ratio	5.049487	4.032945
Sortino Ratio	9.870426	7.020935
Max Drawdown [%]	2.29424	3.106266
Win Rate [%]	100.0	86.206897
Profit Factor	inf	41.918888
Total Trades	8	29