Code Review: Campisi et al. (2024) Replication Script

Executive Summary

The code provides a robust skeleton for replicating the research paper. It successfully handles data acquisition via `yfinance`, implements the specific feature engineering (RVOL, Returns), and sets up a comprehensive suite of Machine Learning models (Linear, Tree-based, Ensemble).

However, the implementation suffers from **critical data leakage** in the preprocessing and feature selection stages. The standardization and feature selection are performed on the entire dataset *before* the cross-validation split. This means information from the test set (future) leaks into the training process, likely inflating performance metrics (Accuracy, AUC) and rendering the "Walk-Forward" validation invalid.

Critical Issues (Must Fix)

1. Data Leakage: Preprocessing (Standardization)

Location: `Phase 4: Feature Selection -> standardize_features(X)` **Issue:** The code standardizes the entire feature matrix `X` using `StandardScaler` before the Walk-Forward Validation loop begins.

```
# CURRENT (WRONG)
X_scaled, scaler = standardize_features(X) # Calculates mean/std using FUTURE dat
# ...
# Inside Loop:
model.fit(X_train, y_train) # Model learns from features scaled with test data ir
```

Impact: The model implicitly "knows" the global mean and variance of the data, including the volatility regimes of the future test period. In finance, where regime shifts are common, this is a severe violation of causality. **Correction:** The scaler must be fit **only** on `X_train` inside the CV loop, and then applied to `X_test`.

2. Data Leakage: Feature Selection (Lasso)

Location: `Phase 4: Feature Selection -> lasso_feature_selection(...)` **Issue:** Similar to standardization, Lasso feature selection is run on the entire dataset (`X_scaled`, `y_continuous`) before the validation loop. **Impact:** The features selected are those that correlate best with the target over the *entire* period (2011-2022). The model in 2012 is using features selected because they will be predictive in 2021. **Correction:** Feature selection must be performed inside the CV loop, using only the training data available at that specific iteration.

3. Missing Statistical Significance Testing

Location: `diebold_mariano_test` function. **Issue:** The function `diebold_mariano_test` is defined but **never called** in the main execution flow. **Impact:** The original paper focuses on "A comparison of machine learning methods." To claim one model is superior to another (e.g., Random Forest vs. Logistic Regression), you must test if the difference in loss is statistically significant. **Correction:** Collect the full array of predictions for all models after the loop finishes, and run pairwise DM tests between the best models.

Performance & Optimization

1. Computational Bottleneck

Issue: The `WalkForwardCV` implements a "Rolling Origin" evaluation with a step size of 1 (implicitly, by yielding every index).

- **Calculation:** ~2,100 training points. Predicting ~750 days.
- **Workload:** 11 Models × ~750 Iterations = **~8,250 training runs**.
- **Result:** Without `max_iterations`, this script will take an extremely long time to run, especially for Gradient Boosting and Forest models which are retrained from scratch every single day. **Suggestion:**

1. **Refit Frequency:** Instead of retraining every day, retrain every N days (e.g., every 30 days) and predict the next N days.
2. **Fast Arguments:** Ensure the default `max_iterations` in `parse_args` is set to a low number (e.g., 10) for debugging, or add a progress estimate.

Minor Issues & Best Practices

1. **Gap Logic:** The gap logic `train_end = test_idx - self.gap` (where `gap=30`) is **correct** (and slightly conservative) for preventing label leakage, assuming `test_idx` is the time of prediction. It ensures the target variable y (returns from t to $t+30$) for the last training point is fully realized before the test point.
2. **Global Variables:** Constants like `TICKERS`, `START_DATE`, and `RANDOM_STATE` are defined globally. It is cleaner to pass these into a configuration dictionary or class to make the code more testable and modular.
3. **RVOL Calculation:** The code calculates RVOL using a 30-day rolling window of *daily* returns.

```
df['RVOL'] = daily_returns.rolling(window=30).std() * np.sqrt(252) * 100
```

This is standard practice. However, ensure that the index alignment matches the paper. `rolling` results are assigned to the end of the window.

Refactoring Recommendation

Use `sklearn.pipeline.Pipeline` to automatically handle the leakage issue.

Suggested Code Structure:

```
from sklearn.pipeline import Pipeline

# Inside the CV loop:
for train_idx, test_idx in cv.split(X):
    X_train, y_train = X[train_idx], y[train_idx]
    X_test, y_test = X[test_idx], y[test_idx]

    # Create a pipeline that scales AND trains
    # Note: Feature selection should also technically be a step here
    pipe = Pipeline([
        ('scaler', StandardScaler()),
        # ('selector', SelectFromModel(Lasso(alpha=...))), # Optional: Automated
        ('classifier', model)
    ])

    pipe.fit(X_train, y_train)
    pred = pipe.predict(X_test)
```

Conclusion

The script is a high-quality "first draft" that correctly implements the mechanics of the models and data fetching. However, the **data leakage** in Phase 4 is a "show-stopper" for a research replication. The results produced by this script will likely be overly optimistic compared to a true out-of-sample implementation