KITA KOD

KITA KOD

DOMAIN 2 QUANTITATIVE TRADING

UM HACKATHON 2025

2025

Our Team

NG ERN YI

OOI WEI SHEN

WONG WEN HAO

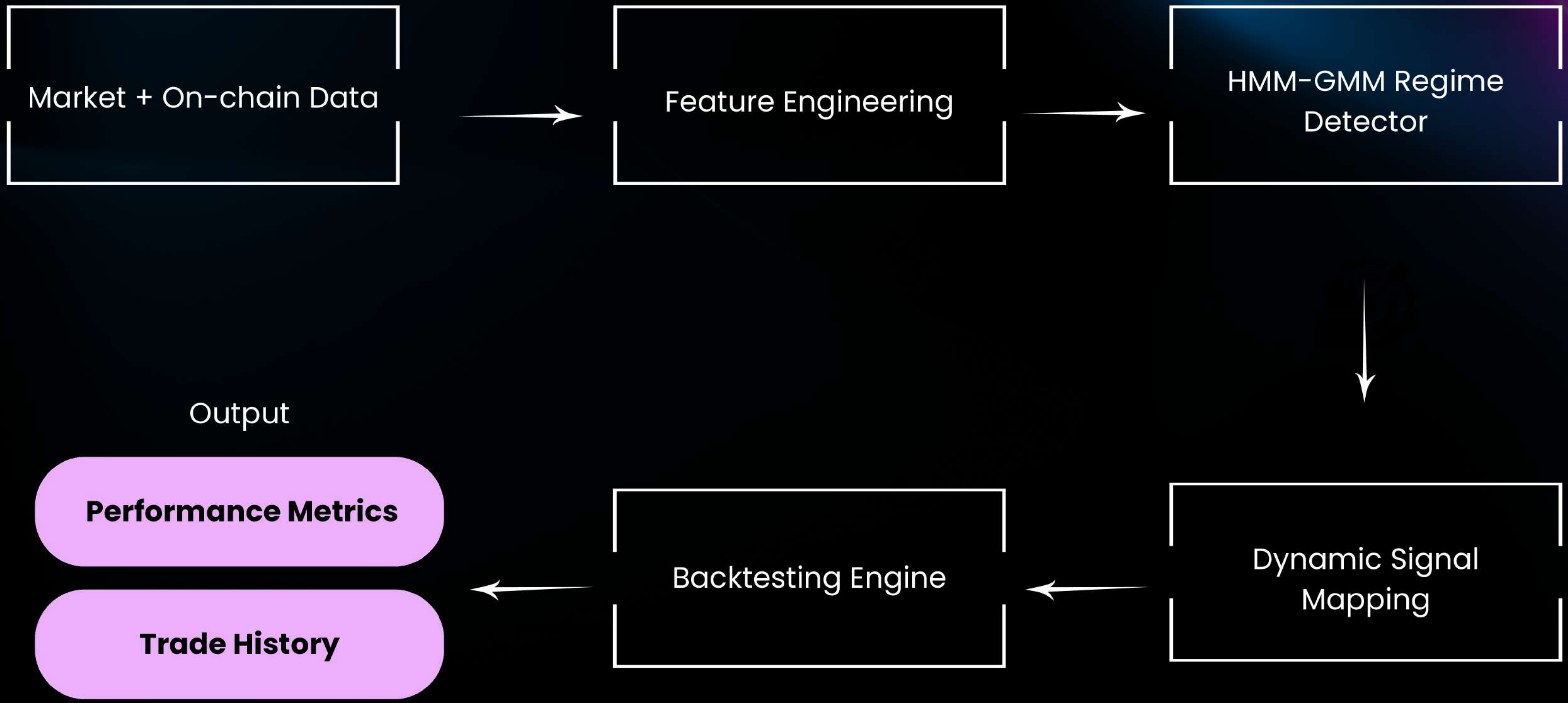
CHEW JIA HUI

TEOH ZHI YEE

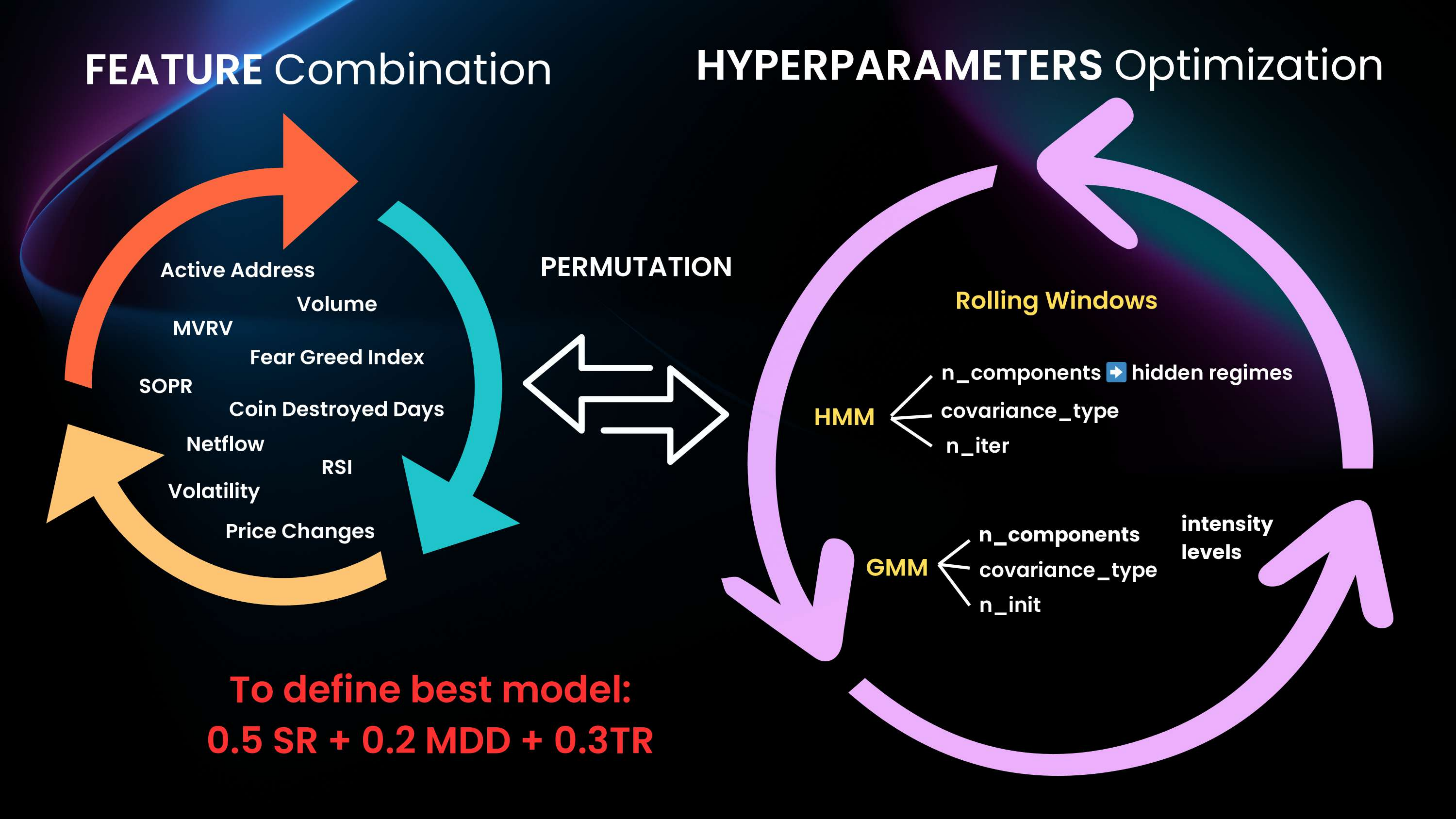
AutoQuant

"Automated. Regime-aware. Alpha-driven."

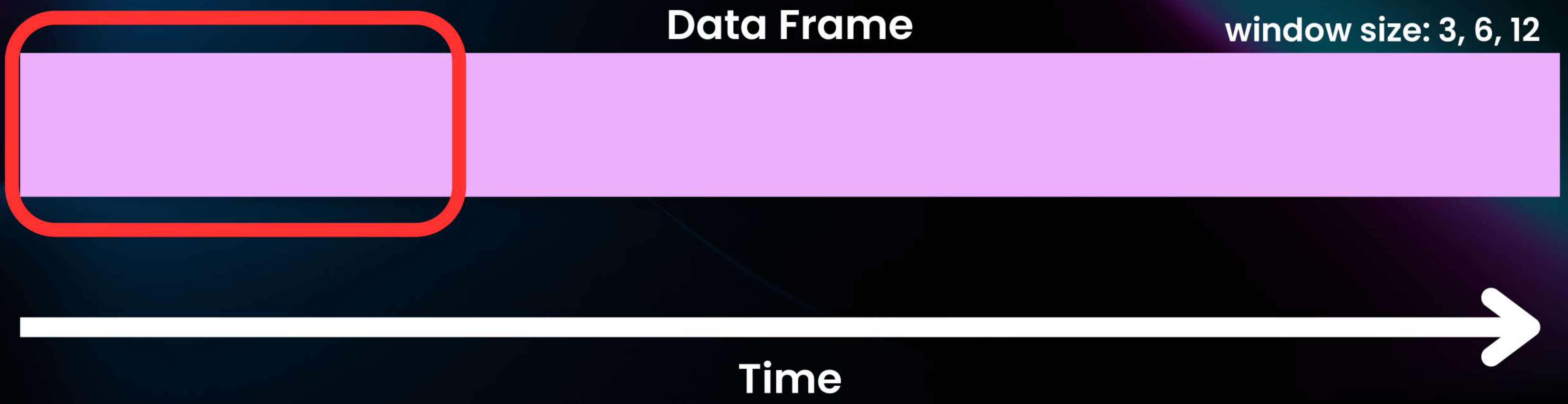
Automation & Modularity of the AutoQuant Pipeline



HMM-GMM Regime Detector & Signal Generator


FEATURE Combination
HYPERPARAMETERS Optimization
Active Address
Volume
MVRV
Fear Greed Index
SOPR
Coin Destroyed Days
Netflow
RSI
Volatility
Price Changes
PERMUTATION
Rolling Windows
HMM
n_components ➡️ hidden regimes
covariance_type
n_iter
GMM
n_components
covariance_type
n_init
intensity
levels
To define best model:
0.5 SR + 0.2 MDD + 0.3TR

Rolling Window (parameter)



HMMs are sensitive to noise — rolling averages make it easier to detect sustained patterns

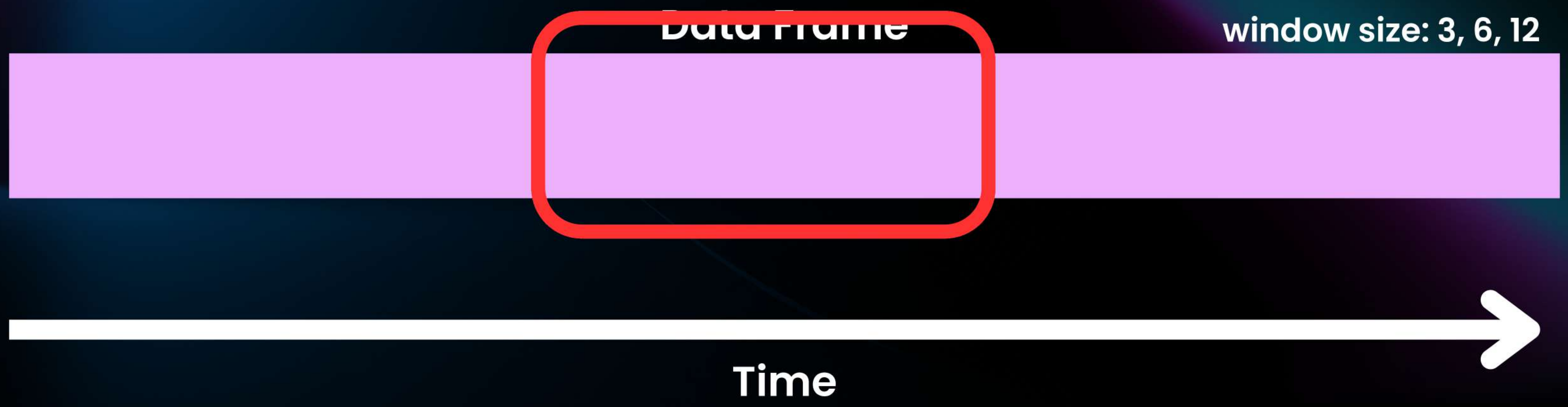
```
for window in window_sizes:
    print(f"\n🔁 Processing window size: {window}")

    rolled_train_df = train_df.copy()
    rolled_test_df = test_df.copy()
    for feature in all_features + [target_col]:
        rolled_train_df[feature] = train_df[feature].rolling(window=window, min_periods=1).mean()
        rolled_test_df[feature] = test_df[feature].rolling(window=window, min_periods=1).mean()
```

Rolling Window (parameter)

HMMs are sensitive to noise — rolling averages make it easier to detect sustained patterns

```
for window in window_sizes:
    print(f"\n🔄 Processing window size: {window}")

    rolled_train_df = train_df.copy()
    rolled_test_df = test_df.copy()
    for feature in all_features + [target_col]:
        rolled_train_df[feature] = train_df[feature].rolling(window=window, min_periods=1).mean()
        rolled_test_df[feature] = test_df[feature].rolling(window=window, min_periods=1).mean()
```

Rolling Window (parameter)



HMMs are sensitive to noise — rolling averages make it easier to detect sustained patterns

```
for window in window_sizes:
    print(f"\n🔁 Processing window size: {window}")

    rolled_train_df = train_df.copy()
    rolled_test_df = test_df.copy()
    for feature in all_features + [target_col]:
        rolled_train_df[feature] = train_df[feature].rolling(window=window, min_periods=1).mean()
        rolled_test_df[feature] = test_df[feature].rolling(window=window, min_periods=1).mean()
```

Rolling Window (parameter)

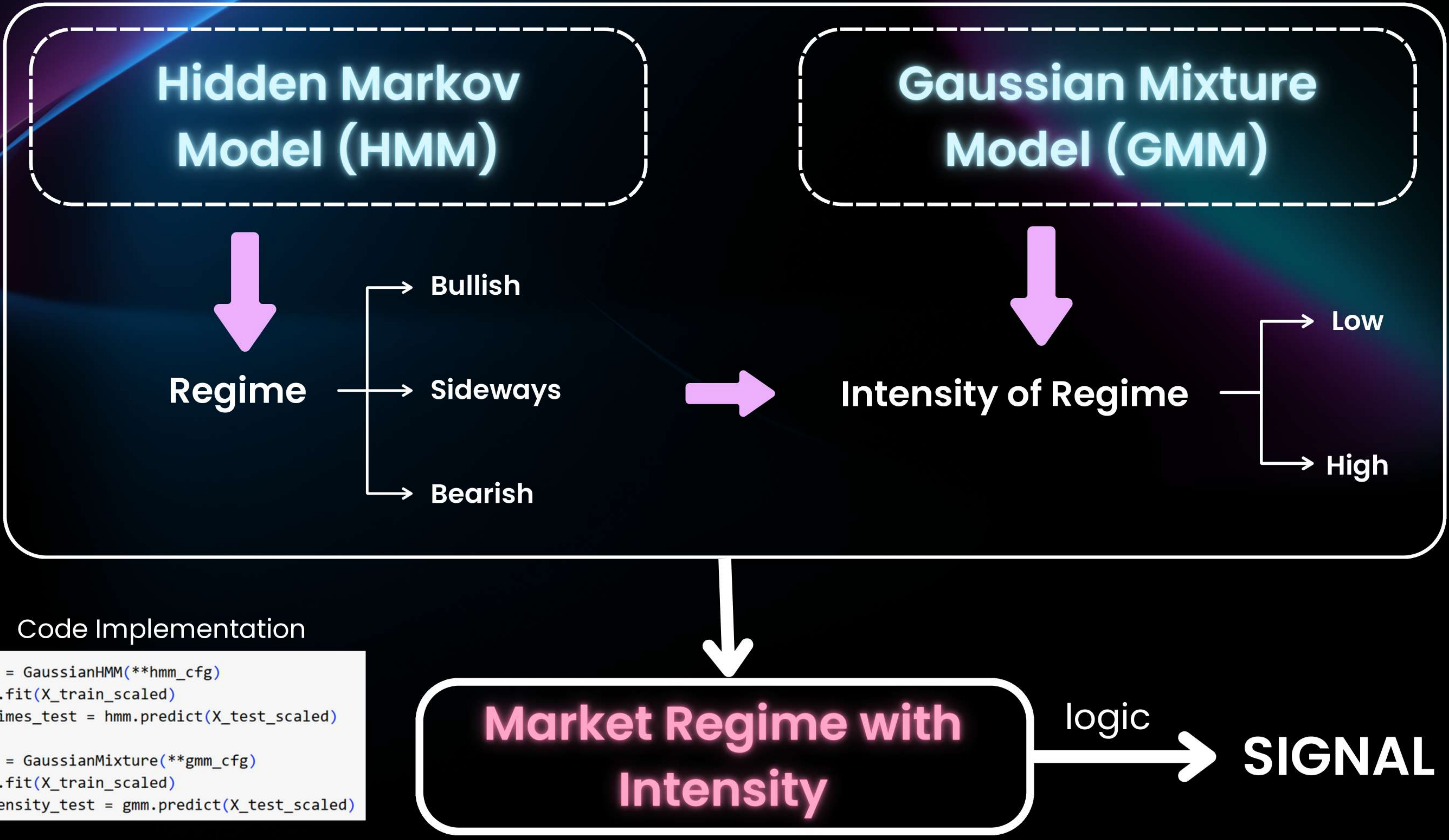
HMMs are sensitive to noise — rolling averages make it easier to detect sustained patterns

```
for window in window_sizes:
    print(f"\n🔁 Processing window size: {window}")

    rolled_train_df = train_df.copy()
    rolled_test_df = test_df.copy()
    for feature in all_features + [target_col]:
        rolled_train_df[feature] = train_df[feature].rolling(window=window, min_periods=1).mean()
        rolled_test_df[feature] = test_df[feature].rolling(window=window, min_periods=1).mean()
```

Rolling Window (parameter)

HMMs are sensitive to noise — rolling averages make it easier to detect sustained patterns

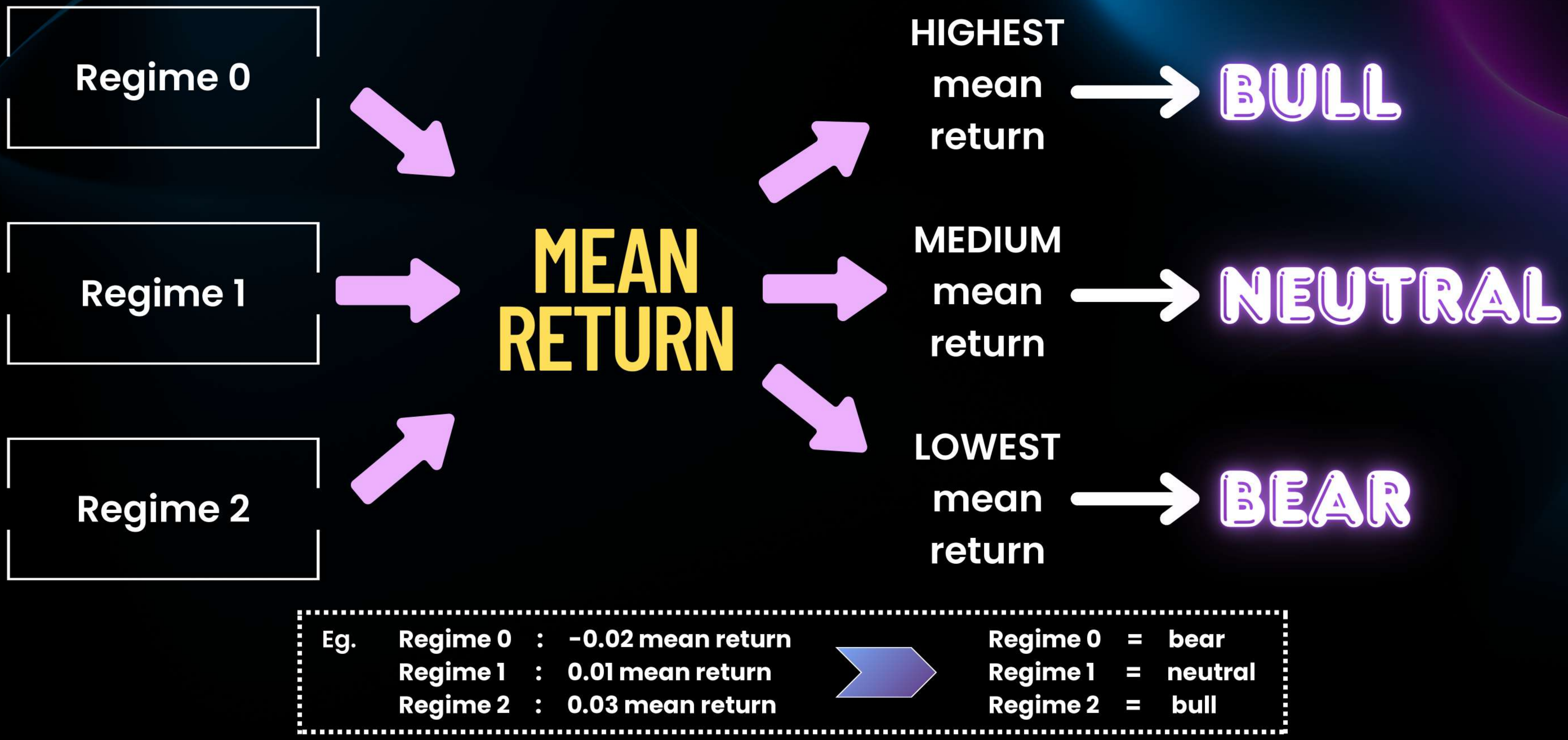
```
for window in window_sizes:
    print(f"\n🔁 Processing window size: {window}")

    rolled_train_df = train_df.copy()
    rolled_test_df = test_df.copy()
    for feature in all_features + [target_col]:
        rolled_train_df[feature] = train_df[feature].rolling(window=window, min_periods=1).mean()
        rolled_test_df[feature] = test_df[feature].rolling(window=window, min_periods=1).mean()
```



Code Implementation

```
hmm = GaussianHMM(**hmm_cfg)
hmm.fit(X_train_scaled)
regimes_test = hmm.predict(X_test_scaled)

gmm = GaussianMixture(**gmm_cfg)
gmm.fit(X_train_scaled)
intensity_test = gmm.predict(X_test_scaled)
```

Dynamic Signal Mapping Logic



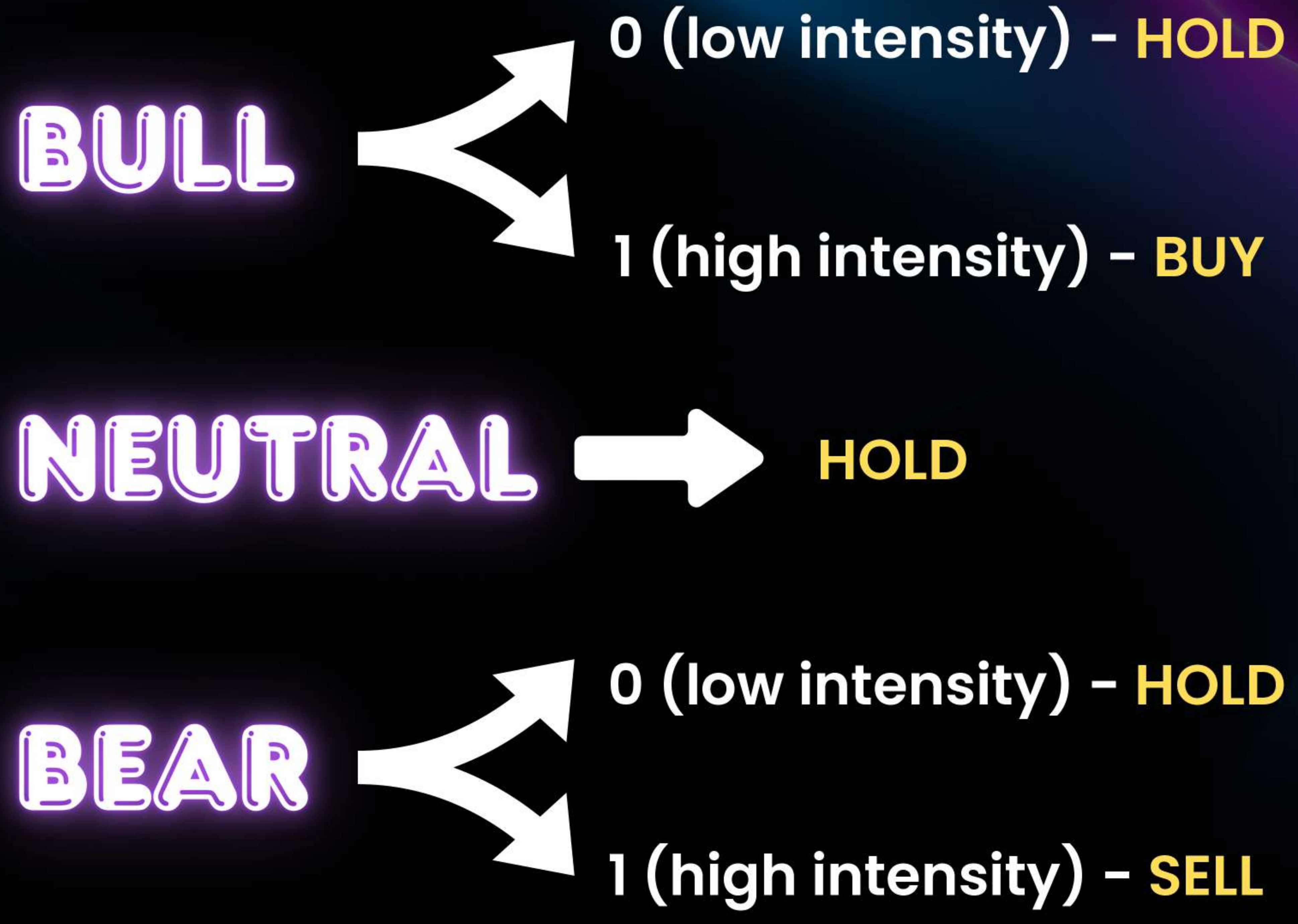
How We Know when to Buy, Sell or Hold

The Logic:

Only **BUY** when the regime has the **highest mean returns**, and intensity is high.

Only **SELL** when the regime has the **lowest mean returns**, and intensity is high.

Otherwise, just **HOLD**.



Regime label

"Signals are generated based on the underlying market regime, not arbitrary rules."

Actual average return of each regime to rank

```
regime_df = pd.DataFrame({'regime': regime_seq, 'return': price_changes_test})
regime_mean = regime_df.groupby('regime')['return'].mean().sort_values()
regime_rank = {reg: idx for idx, reg in enumerate(regime_mean.index)}
```

Regimes are not manually defined

Signal Execution

"Signals are only triggered when the regime classification has high intensity."

Avoids overtrading and only commits when the model is sure.

```python
signals = []
for r, i in zip(regime_seq, intensity_seq):
    r_rank = regime_rank[r]
    if r_rank == 0 and i == 1:
        signals.append(-1)  # SELL
    elif r_rank == len(regime_mean) - 1 and i == 1:
        signals.append(1)   # BUY
    else:
        signals.append(0)   # HOLD
return signals
```

If intensity is low no action is taken noise is filtered out

Performance Output
(Forward Test)

Sharpe Ratio (SR)	Maximum Drawdown (MDD)	Trade Frequency
2.7846	- 22.28%	19.54%
≥1.8	≥-40%	≥3%

Equity vs Price

Buy / Sell Signals on Price

Alternative Solution 1: Reinforcement Learning

Reinforcement Learning for Adaptive Crypto Trading with Regime Awareness

What we have?

1. **RL Agent -** Deep Q-Learning
2. **Progressive Backtester**
3. **Reward Function**

Why it's smart?

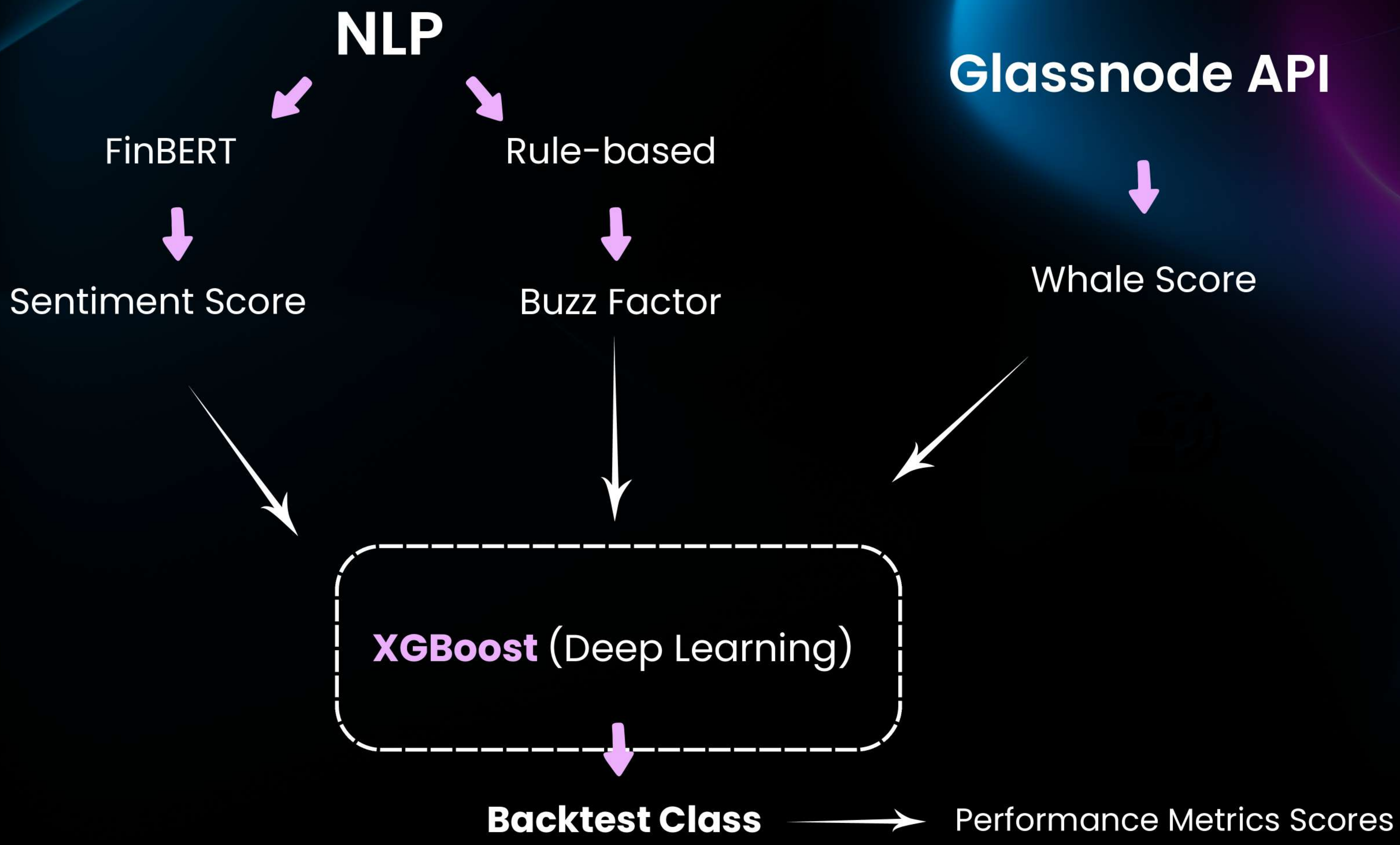
1. **Regime-aware**
2. **Dynamic reward shaping**
3. **Plug-and-play backtester**

Performance Output

```
Trade Frequency: 0.12998375203099613
Sharpe Ratio: 0.11602660942899881
Max Drawdown: -0.5300151821793874
Trade Frequency: 0.12845547582637337
Sharpe Ratio: 0.11227756011930386
Max Drawdown: -0.5300151821793874
Trade Frequency: 0.12696067288020005
Sharpe: 0.1123, MDD: -0.5300, Trade Freq: 0.1270
Sharpe Ratio: -0.4951080411384241
Max Drawdown: -0.22133490659054356
Trade Frequency: 0.08727272727272728

Test Results:
Sharpe: -0.4951
Max Drawdown: -0.2213
Trade Frequency: 0.0873
Training and testing completed.
```

```
Sharpe Ratio: -1.1885150663626685
Max Drawdown: -6.301956593727259
Trade Frequency: 0.8882700613775858
Sharpe: -1.1885, MDD: -6.3020, Trade Freq: 0.8883
Epoch 2/30
Sharpe Ratio: -0.06503528166771765
Max Drawdown: -0.26626376194347134
Trade Frequency: 0.8962075848303394
Sharpe Ratio: -0.5468472848982199
Max Drawdown: -0.3401327903934693
Trade Frequency: 0.8891108891108891
```

Alternative Solution 2: Deep Learning


NLP
FinBERT
Rule-based
Sentiment Score
Buzz Factor
Glassnode API
Whale Score
XGBoost (Deep Learning)
Backtest Class
Performance Metrics Scores

KITA KOD

DEMO SESSION

2025

UM HACKATHON 2025

KITA KOD

THANK YOU

2025

UM HACKATHON 2025

KITA KOD

QnA

2025

UM HACKATHON 2025