**1. Core Trading System Logic**

**Confluence Scoring System**

In your trading strategy, **confluence** refers to the alignment of multiple indicators or factors that give you confidence in a trade. Here's how it works:

* **PRIMARY\_CONFS:** The core indicators/signals you use to determine a trade’s strength (e.g., structure, break of structure, candlestick patterns, support/resistance zones).
* **SECONDARY\_CONFS:** These are additional indicators that add confirmation to the trade but are not as critical as the primary ones.

**Computing Confluence Scores:**

PRIMARY\_CONFS = ['conf\_structure', 'conf\_bos\_or\_choch', 'conf\_candle', 'conf\_sr\_zone']

SECONDARY\_CONFS = ['conf\_psych\_level', 'conf\_fib\_zone', 'bias\_bull']

def compute\_confluence\_scores(df):

df['primary\_score'] = df[PRIMARY\_CONFS].sum(axis=1)

df['secondary\_score'] = df[SECONDARY\_CONFS].sum(axis=1)

df['total\_confluence'] = df['primary\_score'] + df['secondary\_score']

return df

* This function calculates **confluence scores** based on primary and secondary factors.
* The **primary\_score** and **secondary\_score** are summed up to give a **total\_confluence** score, which indicates how strong the trade setup is.

**Trade Signal Logic:**

def trade\_signal\_logic(row, entry\_thresh=2):

return row['primary\_score'] >= entry\_thresh

* A trade signal is triggered if the **primary score** meets or exceeds a set threshold (in this case, entry\_thresh=2).
* If the score is high, this suggests a strong trade setup and will trigger an entry signal.

**Position Size Logic:**

def position\_size(row, base\_risk=1.0):

multiplier = 1.0 + 0.2 \* row['secondary\_score']

return base\_risk \* multiplier

* The **position size** is adjusted based on the **secondary score** (the higher the secondary score, the larger the position size).
* This ensures that stronger trade setups (higher confluence) are allocated more capital.

**2. Dynamic and Intelligent Stop Loss / Take Profit Logic**

**SL/TP Logic:**

* **ATR-based SL/TP:** The **Average True Range (ATR)** is used to dynamically set stop loss and take profit levels based on the market's volatility.
* **Trailing Stop:** As the trade moves in your favor, the stop is adjusted (trailing stop) to lock in profits.

# Example: ATR-based stop-loss and take-profit logic (simplified)

def calculate\_atr(data, period=14):

return data['High'].rolling(window=period).max() - data['Low'].rolling(window=period).min()

def dynamic\_sl\_tp(entry\_price, atr\_value, multiplier=2):

stop\_loss = entry\_price - atr\_value \* multiplier

take\_profit = entry\_price + atr\_value \* multiplier

return stop\_loss, take\_profit

* **Breakeven Rule:** Once the price moves a certain distance in your favor (e.g., after making X pips), you set your stop loss to **break-even** to eliminate risk.

**3. Regime Filter (Trend/Volatility Detection)**

**Trend-based Entry Logic:**

* **Bullish/Bearish Regime:** The trading system only takes **long trades** when the market is in a **bullish** trend (above key moving averages or other trend filters).
* If the model signals a **long** trade, but the market is in a **bearish regime**, the trade is either skipped or size is reduced.

def regime\_filter(price\_data, moving\_avg):

if price\_data[-1] > moving\_avg: # Bullish market regime

return True # Allow long trades

return False # Skip trades in a bearish market

**4. Signal Quality Filter**

**Only High-Quality Entries:**

* **Confirmation Needed:** The model requires **high confluence** or a strong pattern score to enter a trade.
* **Confirmation Bars:** Signals that immediately go negative (e.g., price drops within 2-3 bars) should be ignored or filtered out.
* **Second-Chance Entry:** If a trade closes poorly, a second entry signal might be waited for, allowing more confirmation.

def signal\_quality\_filter(row, quality\_threshold=3):

return row['total\_confluence'] >= quality\_threshold

**5. Post-Entry Management**

**Managing Trades After Entry:**

* **Breakeven After X Pips:** Once the trade is up by a certain amount (1R, meaning 1x risk), the stop-loss is moved to break-even to lock in no-loss territory.
* **Stagnation Exit:** If the price stagnates (i.e., no movement after N bars), exit the trade early or reduce position size.

def post\_entry\_management(entry\_price, current\_price, stop\_loss, profit\_threshold=1.0):

if current\_price - entry\_price >= profit\_threshold: # After X pips in profit

stop\_loss = entry\_price # Move stop loss to breakeven

return stop\_loss

**6. Retraining with Pattern Filtering**

* **Add Trend/Regime Filters into Feature Engineering:**  
  These filters will be incorporated into your model's features for retraining.
* **Forensic Review of Losses:**  
  Every loss (e.g., stop-loss hit) is logged with context (e.g., trend, pattern) for post-analysis.

def log\_losses(entry, exit, context\_data):

loss\_details = {

'entry': entry,

'exit': exit,

'context': context\_data

}

# Store loss details for forensic review

return loss\_details

**Summary of Key Features in Your Trading System:**

1. **Confluence Scoring:**  
   Combines multiple signals (structure, pattern, zones) to assess trade quality.
2. **Dynamic SL/TP:**  
   Uses ATR for flexible stop-loss and take-profit levels, with breakeven adjustment.
3. **Regime Filter:**  
   Ensures trades are taken only in favorable market conditions (e.g., bullish market for long trades).
4. **Signal Quality Filter:**  
   Adds a layer of confirmation, requiring strong signals before entry and additional confirmation during trade.
5. **Post-Entry Management:**  
   Moves stop loss to breakeven once a trade goes in favor, and exits if the market stagnates.

# Confluence Scoring Logic

PRIMARY\_CONFS = ['conf\_structure', 'conf\_bos\_or\_choch', 'conf\_candle', 'conf\_sr\_zone']

SECONDARY\_CONFS = ['conf\_psych\_level', 'conf\_fib\_zone', 'bias\_bull']

def compute\_confluence\_scores(df):

"""

Compute confluence scores based on primary and secondary factors.

Args:

df (pandas.DataFrame): DataFrame containing the confluence data.

Returns:

pandas.DataFrame: DataFrame with computed confluence scores.

"""

# Primary score is the sum of all primary confs

df['primary\_score'] = df[PRIMARY\_CONFS].sum(axis=1)

# Secondary score is the sum of all secondary confs

df['secondary\_score'] = df[SECONDARY\_CONFS].sum(axis=1)

# Total confluence score is the sum of primary and secondary scores

df['total\_confluence'] = df['primary\_score'] + df['secondary\_score']

return df

def trade\_signal\_logic(row, entry\_thresh=2):

"""

Generate a trade signal based on primary confluence score.

Args:

row (pandas.Series): Data for a single row of the DataFrame.

entry\_thresh (int): Threshold for trade entry based on primary score.

Returns:

bool: Whether the trade signal is triggered.

"""

return row['primary\_score'] >= entry\_thresh

def position\_size(row, base\_risk=1.0):

"""

Calculate position size based on secondary confluence score.

Args:

row (pandas.Series): Data for a single row of the DataFrame.

base\_risk (float): Base risk per trade (default = 1.0).

Returns:

float: Calculated position size.

"""

multiplier = 1.0 + 0.2 \* row['secondary\_score']

return base\_risk \* multiplier