I'll search for the last chat we had in this project to review our progress and continue from where we left off.Thank you for your enthusiastic feedback! I can see from our last discussion that we completed Level 3 (Feature Preparation) with the MatrixAssemblers PRD, which you praised as exceptional. You were particularly impressed with the detailed preprocessing logic for each feature in each assembler.

Now, let's continue to **Level 4 - Intelligence Layer**, starting with the first component: the **SynergyDetector**.

# **Product Requirements Document (PRD): SynergyDetector Component**

**Document Version:** 1.0  
 **Date:** June 20, 2025  
 **Component Level:** 4 - Intelligence Layer  
 **Status:** Master Specification

## **1. Component Identity**

### **1.1 Component Name**

**SynergyDetector** (Hard-Coded Strategy Pattern Detector)

### **1.2 Primary Role**

The SynergyDetector is the system's first decision gate. It implements the hard-coded trading strategy by detecting when the four defined synergy patterns appear in the market data. It serves as a crucial filter, ensuring that expensive AI inference only occurs when a valid strategic setup is present.

### **1.3 Single Responsibility**

To monitor indicator values from the Feature Store and detect when they form one of the four predefined synergy patterns, then emit a SYNERGY\_DETECTED event with detailed context for AI evaluation.

### **1.4 Critical Design Principle**

**The Two-Gate System:** The SynergyDetector is Gate 1 - a deterministic, rule-based filter that must pass before any AI models are invoked. This ensures computational efficiency and strategic alignment.

## **2. Inputs & Dependencies**

### **2.1 Configuration Input**

From settings.yaml:

synergy\_detector:

time\_window: 10 # Maximum bars between signals for valid synergy

# Signal activation thresholds

mlmi\_threshold: 0.5 # Minimum signal strength

nwrqk\_threshold: 0.3 # Minimum slope for signal

fvg\_min\_size: 0.001 # Minimum gap size (0.1%)

# Cooldown after detection

cooldown\_bars: 5 # Bars to wait before next detection

### **2.2 Event Input**

**Single Input Event:** INDICATORS\_READY

* **Source:** IndicatorEngine
* **Frequency:** Every 5 minutes
* **Payload:** Complete Feature Store snapshot

### **2.3 Internal State**

* Active signal tracking for each indicator
* Sequence detection buffer
* Cooldown timer
* Last synergy timestamp

## **3. Processing Logic**

### **3.1 The Four Synergy Patterns**

The system recognizes four distinct synergy patterns, each representing a different market dynamic:

Synergy Type 1: MLMI → NW-RQK → FVG Mitigation

Synergy Type 2: MLMI → FVG Mitigation → NW-RQK

Synergy Type 3: NW-RQK → FVG Mitigation → MLMI

Synergy Type 4: NW-RQK → MLMI → FVG Mitigation

Each pattern must complete within 10 bars (50 minutes for 5-min bars).

### **3.2 Signal Detection Logic**

**On INDICATORS\_READY Event:**

def process\_indicators(self, feature\_store: Dict[str, Any]):

"""Core synergy detection logic"""

# 1. Update signal states

self.\_update\_signal\_states(feature\_store)

# 2. Check for new activations

new\_signals = self.\_detect\_new\_signals(feature\_store)

# 3. If new signal, add to sequence

if new\_signals:

self.\_update\_sequence(new\_signals)

# 4. Check for completed synergies

synergy = self.\_check\_synergy\_completion()

# 5. If synergy detected and not in cooldown

if synergy and self.\_can\_emit\_synergy():

self.\_emit\_synergy(synergy)

### **3.3 Signal Activation Rules**

**MLMI Signal Activation:**

def \_check\_mlmi\_signal(self, features: Dict) -> Optional[int]:

"""MLMI activates on crossover with sufficient strength"""

if features['mlmi\_signal'] != 0: # Non-zero means crossover

if abs(features['mlmi\_value'] - 50) > self.mlmi\_threshold \* 50:

return features['mlmi\_signal'] # 1 or -1

return None

**NW-RQK Signal Activation:**

def \_check\_nwrqk\_signal(self, features: Dict) -> Optional[int]:

"""NW-RQK activates on slope change with threshold"""

if features['nwrqk\_signal'] != 0: # Direction change

if abs(features['nwrqk\_slope']) > self.nwrqk\_threshold:

return features['nwrqk\_signal'] # 1 or -1

return None

**FVG Mitigation Signal:**

def \_check\_fvg\_signal(self, features: Dict) -> Optional[int]:

"""FVG activates on mitigation of significant gap"""

if features['fvg\_mitigation\_signal']:

# Determine direction based on which type was mitigated

if features['fvg\_bullish\_mitigated']:

return 1 # Bullish signal

elif features['fvg\_bearish\_mitigated']:

return -1 # Bearish signal

return None

### **3.4 Sequence Tracking**

The detector maintains a sequence buffer to track signal order:

class SignalSequence:

def \_\_init\_\_(self):

self.signals = [] # List of (signal\_type, direction, timestamp)

self.start\_time = None

def add\_signal(self, signal\_type: str, direction: int, timestamp: datetime):

"""Add signal to sequence"""

if not self.signals:

self.start\_time = timestamp

# Check time window constraint

if timestamp - self.start\_time > timedelta(minutes=50):

self.reset() # Sequence expired

self.signals.append({

'type': signal\_type,

'direction': direction,

'timestamp': timestamp,

'sequence\_position': len(self.signals)

})

### **3.5 Synergy Completion Detection**

def \_check\_synergy\_completion(self) -> Optional[Dict]:

"""Check if current sequence forms a valid synergy"""

if len(self.sequence.signals) < 3:

return None

# Extract signal types in order

signal\_order = [s['type'] for s in self.sequence.signals]

# Check direction consistency

directions = [s['direction'] for s in self.sequence.signals]

if not all(d == directions[0] for d in directions):

return None # Mixed directions invalid

# Map to synergy type

synergy\_patterns = {

('mlmi', 'nwrqk', 'fvg'): 'TYPE\_1',

('mlmi', 'fvg', 'nwrqk'): 'TYPE\_2',

('nwrqk', 'fvg', 'mlmi'): 'TYPE\_3',

('nwrqk', 'mlmi', 'fvg'): 'TYPE\_4'

}

synergy\_type = synergy\_patterns.get(tuple(signal\_order))

if synergy\_type:

return {

'type': synergy\_type,

'direction': directions[0], # 1=long, -1=short

'signals': self.sequence.signals,

'completion\_time': self.sequence.signals[-1]['timestamp']

}

return None

### **3.6 Cooldown Management**

After detecting a synergy, the detector enters cooldown:

def \_can\_emit\_synergy(self) -> bool:

"""Check if we can emit (not in cooldown)"""

if self.last\_synergy\_time is None:

return True

bars\_since\_last = self.\_calculate\_bars\_elapsed(self.last\_synergy\_time)

return bars\_since\_last >= self.cooldown\_bars

## **4. Outputs & Events**

### **4.1 Primary Output**

**Event Name:** SYNERGY\_DETECTED  
 **Frequency:** Only when valid synergy pattern completes  
 **Payload Structure:**

SynergyContext = {

'synergy\_type': str, # 'TYPE\_1' through 'TYPE\_4'

'direction': int, # 1 (long) or -1 (short)

'confidence': float, # Always 1.0 (hard-coded rule)

'timestamp': datetime, # When synergy completed

'signal\_sequence': [ # Detailed signal info

{

'type': 'mlmi',

'value': 65.4,

'signal': 1,

'timestamp': datetime

},

# ... more signals

],

'market\_context': { # Current market state

'current\_price': 5150.25,

'volatility': 12.5,

'volume\_profile': {...},

'nearest\_lvn': {

'price': 5145.00,

'strength': 85.5,

'distance': 5.25

}

},

'metadata': {

'bars\_to\_complete': 7, # How many bars the synergy took

'signal\_strengths': { # Individual signal strengths

'mlmi': 0.8,

'nwrqk': 0.6,

'fvg': 1.0

}

}

}

### **4.2 No Other Events**

The SynergyDetector emits only SYNERGY\_DETECTED events. It does not emit partial signals or status updates.

## **5. Critical Requirements**

### **5.1 Determinism Requirements**

* **100% Reproducible:** Same input must always produce same output
* **No Randomness:** Pure rule-based logic, no stochastic elements
* **No Learning:** Rules never change or adapt

### **5.2 Performance Requirements**

* **Processing Time:** <1ms per INDICATORS\_READY event
* **Memory Usage:** Fixed size, no accumulation
* **CPU Usage:** Minimal - simple rule checking

### **5.3 Accuracy Requirements**

* **Zero False Negatives:** Must detect ALL valid synergies
* **Rule Precision:** Exact implementation of strategy rules
* **Time Window Accuracy:** Precise 10-bar window enforcement

### **5.4 Operational Requirements**

* **Stateless Between Runs:** Can rebuild from event stream
* **Thread Safety:** Safe for concurrent access
* **Clear Logging:** Every detection logged with full context

## **6. Integration Points**

### **6.1 Upstream Integration**

**From IndicatorEngine:**

* Event: INDICATORS\_READY
* Data: Complete Feature Store
* Timing: Every 5 minutes

### **6.2 Downstream Integration**

**To Main MARL Core:**

* Event: SYNERGY\_DETECTED
* Triggers: AI inference pipeline
* Frequency: Varies (when patterns detected)

### **6.3 System Integration**

* Initialized by: System Kernel
* Lifecycle: Continuous operation
* State: Minimal sequence tracking only

## **7. Implementation Specifications**

### **7.1 State Machine**

The detector operates as a simple state machine:

States:

- IDLE: No active signals

- BUILDING: 1-2 signals active, waiting for completion

- COOLDOWN: Recently emitted, waiting period

### **7.2 Edge Cases**

**Overlapping Signals:**

* If new signal appears while building: Reset if inconsistent direction
* Multiple synergies possible if non-overlapping

**Partial Sequences:**

* Expire after 10 bars
* Log partial sequences for analysis

**Market Gaps:**

* Handle gracefully
* Don't break sequence on single missing bar

## **8. Error Handling**

### **8.1 Data Issues**

* **Missing Features:** Log warning, skip detection
* **Invalid Values:** Log error, skip detection
* **Null Pointers:** Defensive checks everywhere

### **8.2 System Errors**

* **Event Bus Issues:** Log critical, continue trying
* **Memory Issues:** Log critical, fail fast

## **9. Logging Specification**

### **9.1 Detection Logs**

[INFO] Signal activated: MLMI=1 at 2025-06-20 10:30:00

[INFO] Signal activated: NW-RQK=1 at 2025-06-20 10:35:00

[INFO] Signal activated: FVG\_MITIGATION=1 at 2025-06-20 10:40:00

[INFO] SYNERGY DETECTED: TYPE\_1 LONG completed in 3 bars

### **9.2 Diagnostic Logs**

* Signal strength when activated
* Reason for sequence reset
* Cooldown status changes

## **10. Testing Considerations**

### **10.1 Unit Tests**

* Each signal detection in isolation
* All four synergy patterns
* Time window expiration
* Cooldown behavior

### **10.2 Integration Tests**

* Full flow from indicators to synergy
* Multiple synergies in sequence
* Edge timing scenarios

### **10.3 Strategy Validation**

* Compare with manual strategy application
* Verify no missed opportunities
* Confirm no false positives

## **11. Configuration & Tuning**

### **11.1 Adjustable Parameters**

While the synergy patterns are fixed, thresholds can be tuned:

* Signal activation thresholds
* Time window duration
* Cooldown period

### **11.2 What Cannot Change**

* The four synergy patterns
* The sequence requirements
* The direction consistency rule

## **12. What This Component Does NOT Do**

* Does NOT make trading decisions
* Does NOT calculate position sizes
* Does NOT evaluate market context
* Does NOT learn or adapt
* Does NOT modify the strategy rules
* Does NOT directly place trades
* Does NOT calculate risk parameters

This completes the SynergyDetector PRD. It serves as the critical first gate in our two-gate system, ensuring that AI resources are only used when the hard-coded strategy identifies a valid opportunity.

The synergy detection is deterministic and efficient, providing a clean separation between strategy rules and AI optimization. This component embodies the principle of "fixed strategy, learned application."

Ready to proceed to the next component in the Intelligence Layer?