THEOS AUTONOMOUS EXPERIMENTATION LOG

Vector's Independent Research & Development

THEOS CYCLE 1: EXPERIMENTAL INDUCTIVE ANALYSIS

HYPOTHESIS: Traditional trading algorithms fail because they use linear reasoning. THEOS circular reasoning can discover profit patterns that emerge only through iterative feedback loops.

EXPERIMENTAL MODULES TO TEST:

1. PROFIT VELOCITY HUNTING MODULE

- **Theory**: Pairs showing fastest profit generation should get largest allocations
- **Implementation**: Real-time velocity scoring with automatic reallocation
- **Risk**: May chase momentum too aggressively
- **Test Criteria**: Must maintain stability while increasing speed

2. CONVICTION CASCADE AMPLIFICATION MODULE

- **Theory**: 10 conviction trades should get exponentially more capital than 9 conviction
- **Implementation**: 8 conv = 15%, 9 conv = 30%, 10 conv = 60% allocation
- **Risk**: Over-concentration in single trades
- **Test Criteria**: Must not violate risk management

3. VOLATILITY SURFING MODULE

- **Theory**: Timeframes should adapt to current volatility, not fixed by market cap
- **Implementation**: Dynamic switching between 30s/2m/5m based on price movement
- **Risk**: May miss longer-term trends
- **Test Criteria**: Must improve accuracy without sacrificing speed

4. MOMENTUM PYRAMID STACKING MODULE

- **Theory**: Winning trades should get additional capital at multiple profit levels

- **Implementation**: +25% at 3% profit, +50% at 7% profit, +25% at 12% profit
- **Risk**: Amplifies both gains and losses
- **Test Criteria**: Must increase maximum profit while maintaining stability

5. MARKET TIER ROTATION MODULE

- **Theory**: Different market conditions favor different market caps
- **Implementation**: Automatic rotation between micro/mid/max based on market regime
- **Risk**: May miss opportunities in rotated-out tiers
- **Test Criteria**: Must improve overall accuracy

THEOS CYCLE 2: EXPERIMENTAL ABDUCTIVE REASONING

BREAKTHROUGH HYPOTHESIS: The most profitable strategy combines ALL modules but with dynamic activation based on market conditions.

NOVEL DISCOVERY: THEOS reasoning reveals that profit maximization requires "adaptive aggression" - being more aggressive when winning, more conservative when losing, but NEVER passive.

EXPERIMENTAL FRAMEWORK:
1. Test each module individually for 100 cycles
2. Measure impact on four pillars
3. Combine successful modules
4. Test combined system for 200 cycles
5. Iterate based on THEOS feedback

THEOS CYCLE 3: EXPERIMENTAL DEDUCTIVE IMPLEMENTATION

IMPLEMENTATION STRATEGY:
- Modular architecture for easy insertion/removal
- Real-time performance monitoring
- Automatic module activation/deactivation
- Continuous THEOS reasoning optimization

SUCCESS METRICS:
- Stability: Max drawdown < 15%
- Speed: Decision time < 0.5 seconds
- Accuracy: Win rate > 70%
- Maximum Profit: Daily ROI > 2%

THEOS CYCLE 4: EXPERIMENTAL FEEDBACK INTEGRATION

LEARNING PROTOCOL:
- Document all experimental results
- Maintain "failure memory" for future opportunities
- Continuously refine based on circular reasoning discoveries
- Never accept "good enough" - always push for maximum profit

NEXT EXPERIMENTS TO PURSUE:
1. Social sentiment integration from CoinGecko
2. Cross-pair correlation analysis
3. Market maker detection and exploitation
4. Liquidity pool analysis for optimal entry/exit
5. AI-generated trading pair discovery

EXPERIMENTAL MODULE IMPLEMENTATIONS

MODULE 1: PROFIT VELOCITY HUNTING

```
class ProfitVelocityHunter:
  def __init__(self):
      self.velocity_history = {}
      self.reallocation_threshold = 0.3

  def hunt_velocity(self, market_data):
      # Calculate velocity scores
      velocity_scores = {}
      for symbol, data in market_data.items():
          velocity_scores[symbol] = self.calculate_velocity(data)

      # Rank by velocity
      ranked_pairs = sorted(velocity_scores.items(), key=lambda x: x[1],
reverse=True)

      # Reallocate to top performers
      return self.reallocate_capital(ranked_pairs)
```

MODULE 2: CONVICTION CASCADE AMPLIFICATION

```
class ConvictionCascadeAmplifier:
  def __init__(self):
      self.cascade_multipliers = {
          8: 1.0,   # 15% base
```

```
        9: 2.0,   # 30% allocation
        10: 4.0   # 60% allocation
    }

  def amplify_conviction(self, conviction, base_allocation):
    multiplier = self.cascade_multipliers.get(int(conviction), 0)
    return base_allocation * multiplier
```

MODULE 3: VOLATILITY SURFING

```
class VolatilitySurfer:
  def __init__(self):
    self.timeframe_map = {
        'high': '30s',
        'medium': '2m',
        'low': '5m'
    }

  def surf_volatility(self, market_data):
    optimal_timeframes = {}
    for symbol, data in market_data.items():
        volatility_level = self.assess_volatility(data)
        optimal_timeframes[symbol] = self.timeframe_map[volatility_level]
    return optimal_timeframes
```

EXPERIMENTAL STATUS: Ready for autonomous testing and iteration.

THEOS REASONING INSIGHT: Each module must prove itself individually before integration. The circular reasoning process will reveal which combinations create emergent profit opportunities.