📊 Detailed Technical Overview: How AlphaPulse Calculates Trading Factors

🏗️ System Architecture Overview

AlphaPulse uses a **multi-layered calculation pipeline** that processes market data through several specialized engines to generate trading signals. The system follows this flow:

```
Raw Market Data → Feature Engineering → Analysis Engines → Model Heads
→ Consensus → Signals
```

1️⃣ TECHNICAL ANALYSIS (TA) CALCULATION ENGINE

Enhanced Indicators Engine (`enhanced_indicators_engine.py`)

AlphaPulse uses **Polars** for ultra-fast vectorized calculations with pandas fallback:

Core Technical Indicators (calculated in <50ms)

A. Trend Indicators (40% weight):

- **RSI (Relative Strength Index):**
  - Formula: `RSI = 100 - (100 / (1 + RS))` where `RS = avg_gains / avg_losses`
  - Calculation: Rolling 14-period average of gains vs losses
  - Uses Polars vectorized operations: `pl.col("close").diff()` → separate gains/losses → rolling mean
- **MACD (Moving Average Convergence Divergence):**
  - Formula: `MACD = EMA(12) - EMA(26)`, Signal = `EMA(9) of MACD`
  - Histogram: `MACD - Signal`

- Polars: `pl.col("close").ewm_mean(span=12)` for fast EMA

- **SMA/EMA Crossovers:**
  - Multiple periods: 20, 50, 100, 200
  - Alignment scoring: `(SMA20 > SMA50 > SMA100 > SMA200) = bullish`
- **ADX (Average Directional Index):**
  - Measures trend strength (0-100 scale)
  - Calculation: True Range → Directional Movement (+DM, -DM) → DI+/DI- → DX → ADX
  - Strong trend: ADX > 25
- **Supertrend, HMA (Hull MA), Aroon, DEMA/TEMA, Ichimoku:**
  - Each calculated with specific formulas
  - Aggregated with individual weights (see indicator_aggregator.py)

B. Momentum Indicators (35% weight):

- **RSI, Stochastic, TSI (True Strength Index)**
- **Williams %R, CCI, CMO (Chande Momentum)**
- **PPO, TRIX, Ultimate Oscillator, Awesome Oscillator**

C. Volatility Indicators (25% weight):

- **Bollinger Bands:**
  - Formula: `BB_upper = SMA(20) + (2 × StdDev)`, `BB_lower = SMA(20) - (2 × StdDev)`
  - Position scoring: Price near upper = overbought, near lower = oversold
- **ATR (Average True Range):**
  - `TR = max(High-Low, |High-Close_prev|, |Low-Close_prev|)`
  - `ATR = MA(TR, 14)`
- **Donchian Channels, Keltner Channels, Mass Index, Chandelier Exit**

Technical Indicator Aggregator (`indicator_aggregator.py`)

Aggregation Strategy: 50+ Indicators → Single Score

```
# Weighted aggregation formula
technical_score = (
    trend_score × 0.40 +
    momentum_score × 0.35 +
    volatility_score × 0.25
)

# Direction classification
if technical_score >= 0.55: direction = "bullish"
elif technical_score <= 0.45: direction = "bearish"
else: direction = "neutral"
```

Confidence Calculation:

```
confidence = alignment_factor × strength_factor × consistency_factor
# Where:
# - alignment_factor: How many indicators agree
# - strength_factor: Signal strength magnitude
# - consistency_factor: Historical reliability
```

2️⃣ SENTIMENT ANALYSIS (SA) CALCULATION ENGINE

Enhanced Sentiment Analyzer (`enhanced_sentiment_analysis.py`)

Multi-Source Sentiment Aggregation:

A. News Sentiment Processing

```
# News sentiment pipeline
1. Text preprocessing (clean, tokenize)
2. Ensemble model analysis:
   - FinBERT (financial news-specific)
   - VADER (rule-based sentiment)
   - Custom crypto sentiment model
3. Weighted ensemble scoring:
```

```
   sentiment_score = Σ(model_score × model_weight)
4. Confidence calculation based on model agreement
```

Feature Extraction from News:

- Title/content length
- Sentiment score (-1 to +1)
- Source credibility score
- Temporal features (time of day, market hours)
- Market regime correlation
- Cross-source validation

B. Social Media Sentiment

Twitter/Reddit Analysis:

```
# Social sentiment workflow
1. Fetch posts from free APIs
2. Filter spam/bots (quality filtering)
3. Analyze each post:
   - Sentiment classification
   - Sarcasm detection
   - Topic classification (signal vs noise)
4. Aggregate with recency weighting:
   recent_posts_weight = exp(-time_decay × hours_old)
5. Volume spike detection (viral sentiment)
```

C. Market Sentiment Metrics

- **Fear & Greed Index:** 0-100 scale
  - <25: Extreme fear (contrarian buy)
  - 75: Extreme greed (contrarian sell)
- **Social Volume Analysis:**

- Spike detection: `volume > 2 × moving_average`
- Volume-sentiment correlation

D. Sentiment Aggregation

```
# Multi-source weighted aggregation
overall_sentiment = (
    news_sentiment × 0.40 × news_confidence +
    twitter_sentiment × 0.35 × twitter_confidence +
    reddit_sentiment × 0.25 × reddit_confidence
) / (total_weighted_confidence)

# Market regime adjustment
if market_regime == "high_volatility":
    sentiment_impact *= 1.2  # Amplify in volatile markets
if is_market_hours:
    sentiment_impact *= 1.1  # Boost during trading hours
```

3️⃣ FUNDAMENTAL ANALYSIS (FA) CALCULATION ENGINE

For Cryptocurrencies (`real_data_integration_service.py`)

Since traditional FA metrics (P/E, revenue) don't apply to crypto, AlphaPulse uses **crypto-specific fundamentals:**

A. Market Structure Metrics

```
# BTC Dominance Analysis
btc_dominance = btc_market_cap / total_crypto_market_cap
# >50%: BTC season (alt coins underperform)
# <40%: Alt season (alt coins outperform)

# Market Cap Ratios
total2 = total_market_cap - btc_market_cap  # All alts
```

```
total3 = total2 - eth_market_cap  # All alts except ETH
altcoin_strength = total2 / total3
```

B. On-Chain Metrics

- **Exchange Reserves:**
  - Formula: `Σ(coins held on exchanges)`
  - Low reserves = bullish (supply shock)
  - High inflows = bearish (selling pressure)
- **Whale Activity:**
  - Large transaction count (>$100k)
  - Whale accumulation score
- **Developer Activity:**
  - GitHub commits, contributors
  - Protocol upgrades

C. DeFi Metrics

- **TVL (Total Value Locked):**
  - Protocol-level: Individual DeFi project health
  - Chain-level: L1 vs L2 competition
- **Staking Ratios:**
  - `staked_supply / circulating_supply`
  - High staking = reduced liquid supply = bullish

D. Derivatives Metrics (See Crypto Metrics section)

4️⃣ VOLUME ANALYSIS ENGINE

Multi-Dimensional Volume Calculations

A. Volume Profile (`volume_profile_calculator.py`)

```
# Volume Profile Calculation
1. Divide price range into bins (e.g., 50 bins)
2. Aggregate volume at each price level
3. Identify key levels:
   - POC (Point of Control): Price with highest volume
   - VA (Value Area): 70% of volume distribution
   - HVN (High Volume Nodes): Support/resistance
   - LVN (Low Volume Nodes): Breakout zones
```

B. CVD (Cumulative Volume Delta)

```
# CVD calculation
for each candle:
    if close > open:
        delta = +volume  # Buying pressure
    else:
        delta = -volume  # Selling pressure

    cvd = Σ(delta)

# Divergence detection
if price_making_higher_highs and cvd_making_lower_highs:
    bearish_divergence = True  # Weak rally
```

C. Volume-Based Indicators

- **OBV (On-Balance Volume):**
  - Cumulative volume with direction
  - Confirms trend strength
- **VWAP (Volume Weighted Average Price):**
  - `VWAP = Σ(price × volume) / Σ(volume)`

- Institutional execution benchmark
- **Volume Ratios:**
  - `current_volume / avg_volume(20)`
  - 2.0: Significant activity

5️⃣ MARKET REGIME DETECTION ENGINE

Advanced Regime Classification (`market_regime_detection.py`)

```
# Multi-metric regime detection
def detect_regime(price_data, volume_data):
    # Calculate regime metrics
    metrics = {
        'volatility': calculate_realized_volatility(prices),
        'trend_strength': calculate_adx(prices),
        'momentum': calculate_rate_of_change(prices),
        'volume_trend': calculate_volume_ma_slope(volumes),
        'consolidation_score': calculate_price_compression(prices)
    }

    # Rule-based classification
    if metrics['trend_strength'] > 25 and metrics['momentum'] > 0.02:
        regime = "TRENDING_UP"
        confidence = 0.85
    elif metrics['volatility'] > volatility_threshold:
        if metrics['momentum'] > 0.05:
            regime = "BREAKOUT"
            confidence = 0.80
        else:
            regime = "VOLATILE"
            confidence = 0.75
    elif metrics['consolidation_score'] > 0.8:
        regime = "SIDEWAYS"
        confidence = 0.70
```

```
    else:
        regime = "RANGING"
        confidence = 0.60

    return regime, confidence
```

Regime Types:

- **STRONG_TREND_BULL/BEAR**: ADX > 30, clear direction
- **WEAK_TREND**: ADX 20-30
- **RANGING**: ADX < 20, low volatility
- **VOLATILE_BREAKOUT**: High ATR + volume spike
- **CHOPPY**: Conflicting signals

Adaptive Thresholds:

```
# Regime-specific threshold adjustment
if regime == "VOLATILE":
    min_confidence_threshold *= 1.2  # Require higher confidence
elif regime == "STRONG_TREND":
    min_confidence_threshold *= 0.9  # Lower threshold for trending
```

6️⃣ FEATURE ENGINEERING & FEATURE STORE

Advanced Feature Engineering (`advanced_feature_engineering.py`)

Feature Categories:

A. Technical Features

```
# Calculated from OHLCV data
technical_features = {
    'rsi_14', 'rsi_divergence', 'macd', 'macd_histogram',
    'bb_position': (close - bb_lower) / (bb_upper - bb_lower),
    'atr_normalized': atr / close,
```

```
    'price_vs_sma20': close / sma_20,
    'volume_ratio': volume / volume_ma_20
}
```

B. Price Action Features

```
price_action_features = {
    'higher_highs': detect_higher_highs(highs),
    'lower_lows': detect_lower_lows(lows),
    'swing_points': identify_swing_points(prices),
    'support_resistance': calculate_sr_levels(prices),
    'candlestick_patterns': detect_patterns(ohlc)
}
```

C. Time-Based Features

```
temporal_features = {
    'hour_of_day': timestamp.hour,
    'day_of_week': timestamp.dayofweek,
    'is_market_hours': 9 <= hour <= 16,
    'is_kill_zone': hour in [2,3,4, 8,9,10],  # ICT kill zones
    'session': identify_session(hour)  # Asian/London/NY
}
```

D. Derived Features

```
# Cross-feature combinations
derived_features = {
    'rsi_bb_combo': rsi × bb_position,
    'volume_momentum': volume_ratio × price_momentum,
    'trend_volatility': trend_strength × volatility,
    'regime_technical': regime_score × technical_score
}
```

Feature Store (`feature_store_timescaledb.py`)

Storage & Retrieval:

```
# Feature storage in TimescaleDB
1. Compute feature from raw data
2. Validate feature quality (null check, range check)
3. Store with timestamp (time-travel capability)
4. Cache for fast retrieval
5. Track feature drift over time

# Feature retrieval
def get_features(symbol, timestamp):
    # Check cache
    if cached:
        return cached_features

    # Query TimescaleDB
    features = query_features_at_timestamp(symbol, timestamp)

    # Fill missing features with computation
    for missing in missing_features:
        features[missing] = compute_feature(missing, symbol, timestamp)

    return features
```

7️⃣ CRYPTO-SPECIFIC METRICS ENGINE

10 Crypto-Native Indicators (from signal reference)

A. CVD (Cumulative Volume Delta)

- Calculation: Already covered in Volume Analysis
- Divergence detection for reversal signals

B. Altcoin Season Index

```
# Alt Season Index (0-100)
altcoin_season_index = (
    count(alts outperforming BTC in 90 days) / total_alts
) × 100

# Interpretation
if index > 75: "Alt Season" → Long alts
if index < 25: "BTC Season" → Long BTC, avoid alts
```

C. Long/Short Ratio

```
# Multi-exchange aggregation
ls_ratio = total_long_positions / total_short_positions

# Contrarian signals
if ls_ratio > 3.0:
    signal = "SHORT"  # Overcrowded long
    confidence = 0.85
elif ls_ratio < 0.33:
    signal = "LONG"  # Overcrowded short
    confidence = 0.85
```

D. Perpetual Premium (Funding Rate)

```
# Perp-spot premium
premium = (perp_price - spot_price) / spot_price × 100

# Interpretation
if premium > 0.5%:
    signal = "SHORT"  # Overleveraged longs
    confidence = 0.85
elif premium < -0.3%:
```

```
    signal = "LONG"  # Extreme fear
    confidence = 0.85
```

E. Liquidation Cascade Prediction

```
# Aggregate liquidation levels
liquidation_clusters = []
for exchange in exchanges:
    long_liq_levels = calculate_long_liquidations(exchange)
    short_liq_levels = calculate_short_liquidations(exchange)
    liquidation_clusters.extend([long_liq_levels, short_liq_levels])

# Risk assessment
if price approaching major liquidation_cluster:
    cascade_risk = "HIGH"
    # Reduce position size or avoid trade
```

F. Taker Flow (Buy/Sell Pressure)

```
taker_buy_ratio = taker_buy_volume / total_volume

if taker_buy_ratio > 0.60:
    signal = "LONG"  # Strong buying pressure
elif taker_buy_ratio < 0.40:
    signal = "SHORT"  # Strong selling pressure
```

G. Exchange Reserves

- Already covered in Fundamental Analysis

H. DeFi TVL

- Already covered in Fundamental Analysis

I. L1 vs L2 Dominance

```
l1_dominance = l1_market_cap / (l1_market_cap + l2_market_cap)

if l1_dominance increasing:
    long_l1_tokens = True
elif l2_dominance increasing:
    long_l2_tokens = True
```

J. Crypto Volatility Index

```
realized_volatility = std(returns) × sqrt(252)
implied_volatility = option_implied_vol

if rv < iv:
    expect_volatility_expansion = True
```

8️⃣ SIGNAL GENERATION: 9-HEAD CONSENSUS SYSTEM

Model Heads Architecture (`model_heads.py`)

Each head analyzes the market independently:

Head A: Technical Analysis (13% weight)

```
def analyze_technical(market_data, indicators):
    # Aggregate 50+ technical indicators
    agg_result = aggregate_technical_signals(df, indicators)

    # Convert to probability
    if agg_result.direction == "bullish":
        probability = agg_result.technical_score
        direction = "LONG"
    elif agg_result.direction == "bearish":
        probability = 1.0 - agg_result.technical_score
```

```
        direction = "SHORT"
    else:
        direction = "FLAT"

    return ModelHeadResult(
        head_type="HEAD_A",
        direction=direction,
        probability=probability,
        confidence=agg_result.confidence,
        reasoning=agg_result.reasoning
    )
```

Head B: Sentiment Analysis (9% weight)

```
def analyze_sentiment(market_data):
    # Aggregate news + social sentiment
    overall_sentiment = aggregate_all_sentiment(
        news_sentiment, twitter_sentiment, reddit_sentiment
    )

    # Convert sentiment to direction
    if overall_sentiment > 0.1:
        direction = "LONG"
        probability = 0.5 + (overall_sentiment / 2)
    elif overall_sentiment < -0.1:
        direction = "SHORT"
        probability = 0.5 + (abs(overall_sentiment) / 2)

    return ModelHeadResult(...)
```

Head C: Volume Analysis (13% weight)

```
def analyze_volume(market_data):
    # Volume profile + CVD + OBV analysis
    volume_score = calculate_volume_score(
```

```
        cvd_divergence, obv_trend, volume_profile_support
    )

    # Determine direction
    if increasing_volume and uptrend:
        direction = "LONG"
        confidence = 0.75
    elif volume_divergence:
        direction = "SHORT"  # Reversal warning
        confidence = 0.70

    return ModelHeadResult(...)
```

Head D: Rule-Based (9% weight)

```
def analyze_patterns(market_data):
    # Candlestick patterns + chart patterns
    patterns = detect_all_patterns(ohlc)

    # Pattern scoring
    for pattern in patterns:
        if pattern.type == "bullish_engulfing" and at_support:
            direction = "LONG"
            confidence = 0.70

    return ModelHeadResult(...)
```

Head E: ICT Concepts (13% weight)

```
def analyze_ict(market_data):
    # ICT-specific analysis
    in_ote_zone = check_ote_zone(price, fib_levels)  # 0.62-0.79 retracement
    kill_zone_active = check_kill_zone(current_hour)  # London/NY
    judas_swing = detect_judas_swing(price_action)
```

```
    # Scoring
    if in_ote_zone and kill_zone_active:
        confidence = 0.88  # Very high
        direction = determine_ict_direction(price_structure)

    # Kill zone multiplier
    if kill_zone_active:
        confidence *= 1.3

    return ModelHeadResult(...)
```

Head F: Wyckoff Methodology (13% weight)

```
def analyze_wyckoff(market_data):
    # Detect Wyckoff patterns
    spring_detected = detect_spring(price_action, volume)  # Final shakeout
    utad_detected = detect_utad(price_action, volume)  # Final pump

    # Spring/UTAD = highest confidence signals
    if spring_detected:
        direction = "LONG"
        confidence = 0.90  # 🔥 Best signal
    elif utad_detected:
        direction = "SHORT"
        confidence = 0.90  # 🔥 Best signal
    elif accumulation_phase:
        direction = "LONG"
        confidence = 0.70

    return ModelHeadResult(...)
```

Head G: Harmonic Patterns (9% weight)

```
def analyze_harmonic(market_data):
    # Detect harmonic patterns (Gartley, Butterfly, Bat, Crab)
```

```
    patterns = detect_harmonic_patterns(price_action)

    # Pattern completion at D point
    if gartley_complete:
        direction = "LONG/SHORT"  # Depends on pattern type
        confidence = 0.85

    return ModelHeadResult(...)
```

Head H: Market Structure (9% weight)

```
def analyze_market_structure(market_data):
    # Multi-timeframe alignment
    mtf_aligned = check_mtf_alignment([1m, 5m, 15m, 1h, 4h])

    # Premium/Discount zones
    in_discount_zone = price < 0.5 × (high - low) + low
    in_premium_zone = price > 0.5 × (high - low) + low

    # Scoring
    if mtf_aligned and in_discount_zone:
        direction = "LONG"
        confidence = 0.88
    elif mtf_aligned and in_premium_zone:
        direction = "SHORT"
        confidence = 0.88

    return ModelHeadResult(...)
```

Head I: Crypto Metrics (12% weight)

```
def analyze_crypto_metrics(market_data):
    # Aggregate all 10 crypto-specific indicators
    signals = []
```

```
    # CVD divergence
    if cvd_bullish_divergence:
        signals.append(("LONG", 0.85))

    # Alt season
    if alt_season_index > 75:
        signals.append(("LONG", 0.80))

    # Long/Short ratio
    if ls_ratio > 3.0:
        signals.append(("SHORT", 0.85))  # Contrarian

    # Perpetual premium
    if perp_premium > 0.5:
        signals.append(("SHORT", 0.85))

    # ... (all 10 indicators)

    # Aggregate signals
    if 3+ signals aligned:
        confidence = 0.80+
    if 5+ signals aligned:
        confidence = 0.85+

    return ModelHeadResult(...)
```

Consensus Mechanism (`consensus_manager.py`)

```
def check_consensus(model_head_results):
    """
    Consensus Requirements:
    - Minimum 4 out of 9 heads must agree (44% threshold)
    - Each agreeing head must have:
      - Probability ≥ 0.60
      - Confidence ≥ 0.70
```

```
"""

# Count votes for each direction
long_votes = []
short_votes = []
flat_votes = []

for result in model_head_results:
    if result.probability >= 0.60 and result.confidence >= 0.70:
        if result.direction == "LONG":
            long_votes.append(result)
        elif result.direction == "SHORT":
            short_votes.append(result)
        else:
            flat_votes.append(result)

# Check consensus
max_votes = max(len(long_votes), len(short_votes), len(flat_votes))

if max_votes >= 4:  # Consensus achieved
    # Determine winning direction
    if len(long_votes) == max_votes:
        consensus_direction = "LONG"
        agreeing_heads = long_votes
    elif len(short_votes) == max_votes:
        consensus_direction = "SHORT"
        agreeing_heads = short_votes
    else:
        consensus_direction = "FLAT"
        agreeing_heads = flat_votes

    # Calculate weighted consensus probability
    total_weight = 0.0
    weighted_probability = 0.0

    for result in agreeing_heads:
```

```
            weight = MODEL_WEIGHTS[result.head_type]
            weighted_probability += result.probability × weight
            total_weight += weight

        consensus_probability = weighted_probability / total_weight

        # Calculate consensus confidence
        consensus_confidence = calculate_consensus_confidence(
            agreeing_heads, total_heads=9
        )

        return ConsensusResult(
            consensus_achieved=True,
            direction=consensus_direction,
            probability=consensus_probability,
            confidence=consensus_confidence,
            agreeing_heads=len(agreeing_heads),
            total_heads=9,
            consensus_score=len(agreeing_heads) / 9
        )
    else:
        # No consensus
        return ConsensusResult(
            consensus_achieved=False,
            direction="FLAT",
            reason="Insufficient agreement (<4 heads)"
        )
```

Weighted Confidence Calculation:

```
def calculate_consensus_confidence(agreeing_heads, total_heads):
    # Base confidence from average
    base_confidence = mean([h.confidence for h in agreeing_heads])

    # Agreement bonus (more heads = higher confidence)
    agreement_bonus = (len(agreeing_heads) - 4) / (total_heads - 4) × 0.15
```

```
    # Strength bonus (high probability = higher confidence)
    avg_probability = mean([h.probability for h in agreeing_heads])
    strength_bonus = (avg_probability - 0.60) / 0.40 × 0.10

    # Final confidence
    consensus_confidence = base_confidence + agreement_bonus + strength_
bonus
    consensus_confidence = clip(consensus_confidence, 0.0, 1.0)

    return consensus_confidence
```

9️⃣ FINAL SIGNAL GENERATION & EXECUTION

Signal Generation Result (`sde_database_integration.py`)

```
def generate_final_signal(consensus_result, market_conditions):
    if not consensus_result.consensus_achieved:
        return None  # No trade

    # Base signal from consensus
    signal = TradingSignal(
        symbol=symbol,
        direction=consensus_result.direction,
        probability=consensus_result.probability,
        confidence=consensus_result.confidence,
        timestamp=datetime.now()
    )

    # Calculate entry/exit levels
    signal.entry_price = current_price
    signal.stop_loss = calculate_stop_loss(
        entry_price, atr, direction
    )
```

```
    signal.take_profit = calculate_take_profit(
        entry_price, risk_reward_ratio=2.5
    )

    # Calculate position size based on confidence
    signal.position_size = calculate_position_size(
        confidence=signal.confidence,
        risk_per_trade=0.02  # 2% risk
    )

    # Risk management checks
    if approaching_liquidation_cascade:
        signal.position_size *= 0.5  # Reduce size

    if extreme_leverage_market:
        signal = None  # Skip trade

    # Store signal in database
    store_signal_in_timescaledb(signal)

    return signal
```

🔢 KEY CALCULATION FORMULAS SUMMARY

Technical Score:

```
Technical_Score = (Trend × 0.40) + (Momentum × 0.35) + (Volatility × 0.25)
```

Sentiment Score:

```
Sentiment_Score = (News × 0.40 × Conf) + (Twitter × 0.35 × Conf) + (Reddit × 0.25 × Conf)
```

Consensus Probability:

```
Consensus_Probability = Σ(Head_Probability × Head_Weight) / Σ(Head_Weigh
t)
```

Final Confidence:

```
Confidence = Base_Confidence + Agreement_Bonus + Strength_Bonus
where:
  Agreement_Bonus = ((agreeing_heads - 4) / 5) × 0.15
  Strength_Bonus = ((avg_probability - 0.60) / 0.40) × 0.10
```

⚡ PERFORMANCE OPTIMIZATIONS

1. **Polars Vectorization**: All TA calculations use Polars for 10-50x speedup
2. **Redis Caching**: Indicators cached with 1-minute TTL
3. **TimescaleDB**: Time-series optimized storage for fast historical queries
4. **Parallel Processing**: All 9 model heads run concurrently (asyncio)
5. **Incremental Updates**: Only recalculate changed indicators
6. **GPU Acceleration**: Available for ML model inference (optional)

📈 DATA FLOW DIAGRAM

```
1. Market Data Collection (WebSocket + REST APIs)
   ↓
2. Real-Time Data Pipeline (TimescaleDB storage)
   ↓
3. Feature Engineering (50+ features calculated)
   ↓
4. Parallel Analysis:
   ├── Technical Analysis Engine → Head A
   ├── Sentiment Analysis Engine → Head B
```

```
    ├─ Volume Analysis Engine → Head C
    ├─ Rule-Based Engine → Head D
    ├─ ICT Analysis → Head E
    ├─ Wyckoff Analysis → Head F
    ├─ Harmonic Analysis → Head G
    ├─ Market Structure Analysis → Head H
    └─ Crypto Metrics Analysis → Head I
   ↓
5. Consensus Mechanism (4/9 heads required)
   ↓
6. Signal Generation (if consensus achieved)
   ↓
7. Risk Management Validation
   ↓
8. Position Sizing & Execution
```

This is the complete technical architecture of how AlphaPulse calculates trading factors. The system processes **100+ data points**, calculates **50+ technical indicators**, analyzes **multiple sentiment sources**, considers **10 crypto-specific metrics**, and uses a **9-head consensus mechanism** to generate high-confidence trading signals with an expected win rate of **65-85%** depending on signal strength.