Architettura completa Early Detector su Solana,

pronta per essere implementata in Python con **Postgres (Supabase)** e backtestabile.

Userai dati principalmente da:

- Birdeye
- DexScreener
- Pump.fun

---

1 FORMULA MATEMATICA COMPLETA (PRONTA PER CODICE)

🔹 Variabili Base

Per ogni token iii al tempo ttt:

- HtH_tHt = numero holder
- Vt5mV_t^{5m}Vt5m = volume ultimi 5 minuti
- PtP_tPt = prezzo
- LtL_tLt = liquidity pool value
- BtB_tBt = numero buy 5m
- StS_tSt = numero sell 5m
- SWtSW_tSWt = numero smart wallet attivi
- MCtMC_tMCt = market cap

🔹 1. Holder Acceleration

velocity1=Ht−Ht−10velocity_1 = H_t - H_{t-10}velocity1=Ht−Ht−10
velocity2=Ht−10−Ht−20velocity_2 = H_{t-10} - H_{t-20}velocity2=Ht−10−Ht−20
holder_acceleration=velocity1−velocity2holder_acceleration = velocity_1 - velocity_2holder_acceleration=velocity1−velocity2

Normalizzazione:

Hnorm=holder_accelerationHt+1H_{norm} = \frac{holder_acceleration}{H_t + 1}Hnorm=Ht+1holder_acceleration

---

🔹 2. Stealth Accumulation Score

unique_buyers20munique_buyers_{20m}unique_buyers20m sell_ratio=S20mB20m+1sell_ratio = \frac{S_{20m}}{B_{20m} + 1}sell_ratio=B20m+1S20m

price_stability=1−std(P20m)mean(P20m)price_stability = 1 - \frac{std(P_{20m})}{mean(P_{20m})}price_stability=1−mean(P20m)std(P20m)
SA=unique_buyers20m·(1−sell_ratio)·price_stabilitySA = unique_buyers_{20m} \cdot (1 - sell_ratio) \cdot price_stabilitySA=unique_buyers20m·(1−sell_ratio)·price_stability

---

🔹 3. Volatility Shift

vol20m=std(P20m)vol_{20m} = std(P_{20m})vol20m=std(P20m) vol5m=std(P5m)vol_{5m} = std(P_{5m})vol5m=std(P5m) VS=vol5mvol20m+ϵVS = \frac{vol_{5m}}{vol_{20m} + \epsilon}VS=vol20m+ϵvol5m

---

🔹 4. Smart Wallet Rotation Ratio

SWR=SWtsmart_wallets_active_global30m+1SWR = \frac{SW_t}{smart_wallets_active_global_{30m} + 1}SWR=smart_wallets_active_global30m +1SWt

---

🔹 5. Sell Pressure

sell_pressure=S5mB5m+S5m+1sell_pressure = \frac{S_{5m}}{B_{5m} + S_{5m} + 1}sell_pressure=B5m+S5m+1S5m

---

🔥 Instability Index Finale

II=2·Z(SA)+1.5·Z(Hnorm)+1.5·Z(VS)+2·Z(SWR)−2·Z(sell_pressure)II = 2 \cdot Z(SA) + 1.5 \cdot Z(H_{norm}) + 1.5 \cdot Z(VS) + 2 \cdot Z(SWR) - 2 \cdot Z(sell_pressure)II=2·Z(SA)+1.5·Z(Hnorm)+1.5·Z(VS)+2·Z(SWR)−2·Z(sell_pressure)

Dove Z(x)Z(x)Z(x) è lo z-score cross-sectional rispetto a tutti i token analizzati negli ultimi 60 minuti.

🔹 Trigger Finale

Segnale se:

- II>percentile95II > percentile_{95}II>percentile95
- Lt>40kL_t > 40kLt>40k
- MCt<3MMC_t < 3MMCt<3M
- top10_holder < 35%
- mint_authority disabilitata

2 SCHEMA DATABASE (Postgres – Supabase)

◆ Tabella: tokens

```
id (uuid)
address (text, unique)
name (text)
symbol (text)
created_at (timestamp)
first_seen_at (timestamp)
```

◆ Tabella: token_metrics_timeseries

```
id (bigserial)
token_id (uuid, fk)
timestamp (timestamp)

price (numeric)
marketcap (numeric)
liquidity (numeric)

holders (int)
volume_5m (numeric)
volume_1h (numeric)

buys_5m (int)
sells_5m (int)

top10_ratio (numeric)

smart_wallets_active (int)

instability_index (numeric)
```

Index:

```
CREATE INDEX idx_token_time
ON token_metrics_timeseries(token_id, timestamp DESC);
```

◆ Tabella: wallet_performance

```
wallet (text, pk)
avg_roi (numeric)
total_trades (int)
win_rate (numeric)
cluster_label (text)
last_active (timestamp)
```

Tabella: signals

```
id (bigserial)
token_id (uuid)
timestamp (timestamp)
instability_index (numeric)
entry_price (numeric)
liquidity (numeric)
marketcap (numeric)
```

---

3 PSEUDO-CODICE PYTHON COMPLETO

Architettura

```
collector.py
features.py
scoring.py
db.py
backtest.py
main.py
```

---

main.py

```
while True:

    tokens = get_recent_tokens()

    for token in tokens:
        metrics = fetch_metrics(token)

        features = compute_features(token, metrics)

        score = compute_instability_index(features)

        save_metrics(token, metrics, score)

    compute_cross_sectional_zscores()

    check_signals()

    sleep(60)
```

---

compute_instability_index()

```
def compute_instability_index(f):

    II = (
        2 * f["z_stealth_accum"]
        + 1.5 * f["z_holder_acc"]
        + 1.5 * f["z_vol_shift"]
        + 2 * f["z_smart_ratio"]
```

```
        - 2 * f["z_sell_pressure"]
    )

    return II
```

🔹 Z-score cross-sectional

```
def compute_zscore(series):
    return (series - series.mean()) / (series.std() + 1e-9)
```

Calcolato su tutti i token attivi negli ultimi 60 minuti.

🔹 Signal Detection

```
def check_signal(token):

    if token.instability_index > percentile_95:
        if token.liquidity > 40000 and token.marketcap < 3_000_000:
            create_signal(token)
```

---

4 SISTEMA DI BACKTEST SERIO SU SOLANA

🔹 Step 1 – Raccolta Storica

Scarica da Birdeye:

- Tutti i token nati negli ultimi 6 mesi
- Candles 1m
- Holder timeline
- Transaction log

Salva tutto nel DB.

🔹 Step 2 – Simulazione Real-Time

Non usare dati futuri.

Per ogni minuto storico:

1. Calcola features solo con dati disponibili fino a quel minuto
2. Calcola II

3. Se trigger → registra entry

🔹 Step 3 – Regole di Uscita

Testa 3 modelli:

A)

- TP 100%
- SL -30%

B)

- Trailing stop 40%

C)

- Exit quando smart wallet iniziano a vendere

🔹 Step 4 – Metriche Serie

Calcola:

- Win rate
- Profit factor
- Max drawdown
- Sharpe ratio
- Average time to peak
- % token che fanno 2x entro 2h

🔹 Step 5 – Walk-Forward Validation

Dividi periodo:

- 70% training (ottimizza pesi formula)
- 30% out-of-sample

Ricalibra pesi con regressione logistica:

```
target = 1 se token fa 2x entro 120 min
```

🎯 RISULTATO ATTESO

Early Detector serio su Solana tipicamente:

- Win rate 25–40%
- Ma R:R alto (1:3 o superiore)
- Edge dipende da execution speed

⚠️ Realtà

Il vero vantaggio non è la formula.
È:

- velocità
- qualità smart wallet list
- pulizia dati
- disciplina nel risk management