this is around vwap footprint study, apparently hedgefunds look at historical return patterns to predict the returns for real time. Is this possible?

Aye, let's break this down in a clear, realistic, *front-office quant* way — Scottish accent as requested GB 🏴.

Short answer: **Yes, it's possible — but only under specific conditions, and only for short-horizon microstructure effects.**
It is *not* the same as forecasting mid-term returns like a factor model. It's much closer to *microstructure pattern recognition*.

✅ 1. What hedge funds actually do

Hedge funds (esp. HFT / market-making firms like Citadel, Jane Street, Optiver) *absolutely* use historical return patterns to predict short-horizon returns *in real time*.
But they don't use simple returns — they use **conditional microstructure states**:

✔ Lagged return patterns

e.g.,

- If price drifts up when queue is thin + buy pressure is high,
- Then the probability of further uptick in the next 50–500ms is positive.

✔ "State-dependent" patterns

They don't look at unconditional returns.
They look at patterns *conditional* on:

State variable	Meaning
Depth imbalance	Bid/ask size imbalance
OFI (order flow imbalance)	Market buy vs sell pressure
Queue dynamics	Queue depletion / replenishment
Short-lag autocorrelation	Momentum at 50ms, 100ms, 500ms
Volatility regime	Is market whippy or stable
Spread regime	Wide vs narrow
Event state	Auction, open/close, news

The idea is:

> "Given the current state of the order book + recent flow, what are the odds the next

micro-return is up or down?"

This is absolutely feasible and used everywhere: **HFT alpha modelling**, **market making**, **signal engines**, **VWAP/TWAP footprint detection**, etc.

✅ 2. Why this relates to your VWAP footprint study

You're asking:

> "If we trade today, is the market reacting the same way as historically?
> Can others detect our footprint by matching real-time behaviour to historical patterns?"

This is exactly what HFT firms do:

They continuously compare:

(A) Real-time micro-returns

vs

(B) Historical "normal behaviour" under similar states

vs

(C) Historical "when big flow is present" behaviour

If today's behaviour resembles (C) more than (A), they infer:

> "There's someone trading behind the scenes — probably VWAP/POV/IS flow."

This is detection of "flow influence".

✅ 3. Is it theoretically possible?

Yes — because:

✔ Market microstructure has *short-term memory*

Returns over 50ms–2s are **not** IID.

They show:

- Autocorrelation
- Mean reversion
- Order-flow persistence
- Queue depletion patterns

This makes micro-returns *predictable in expectation*, even if only for tiny horizons.

✔ Order flow exhibits persistence

Institutional flow (VWAP/POV) produces a **train of similarly-signed trades**.

This creates detectable statistical signatures:

- Signed volume autocorrelation
- Price impact shape
- Slow reaction of LOB liquidity providers
- Spread dynamics

✔️ HFT firms exploit this

They learn patterns like:

- "After a sequence of medium-sized buys at the ask, the next few trades have +X bp drift."
- "If a queue is repeatedly eaten at a constant cadence, there is a VWAP slicing behind it."

They can match your child order signature *to their historical library of flow signatures*.

✅ 4. Examples of pattern recognition models used

These are **real models** used on dealer desks, HFT, and hedge funds:

📌 1. ACF of signed returns (micro-momentum)

If:

- r_t = price change
- sign_t = +1 for buy-initiated trades, −1 for sells

Then hedge funds compute:

- ACF(sign_t)
- ACF(r_t)
- XCF(sign_t, r_t+lag)
- Autocorrelation of OFI

These show whether incoming flow has predictable direction.

📌 2. Fourier-based pattern matching

This is common for:

- Detecting periodic child order slices
- Identifying constant-cadence VWAP/POV algos

If your slices arrive at a semi-regular interval, the FFT will show peaks at that frequency.

But:
FFT works only when:

- Slicing cadence is stable
- There's low noise
- There's consistent behaviour

VWAP tends to have *variable slicing intervals*, so FFT is sometimes weaker than ACF/XCF.

📌 3. Machine learning “state-to-return” predictors

Typical models:

- Logistic regression on order-book states
- Random forests on OFI, spread, depth
- LSTMs on event sequences
- Gradient boosted trees (LightGBM / XGBoost)
- HMMs (hidden Markov models) for latent liquidity regimes

Inputs typically:

- Depth imbalance
- Quote replenishment
- Immediate past returns
- Order arrival intensity

HFT desks run these in real-time at 100µs–10ms frequencies.

✅ 5. Bottom line — is it possible to predict returns in real time?

✔️ **Yes — for microstructure horizons (10ms to ~5s).**

✔️ **Yes — especially when flow is strong or systematic.**

✔️ **Yes — by conditioning on order-book states.**

❌ **No — for longer horizons (minutes–hours), markets revert to noisy / semi-random.**

VWAP footprints absolutely can be detected like this.

That’s why your study is interesting:

You’re basically recreating an **HFT flow-detection system** from your algo POV.

✅ 6. If you'd like, I can give you one of the following next:

1. **A full statistical pipeline to replicate what HFT firms do**
2. **KDB+ code to compute state-conditioned returns, ACF, XCF, OFI**
3. **A detection model to see if your VWAP slices are identifiable**
4. **FFT approach vs ACF/XCF — detailed comparison and when each works**
5. **How to construct a “flow signature” library**
6. **How to test if real-time returns match historical footprint states**

Which one do you want, mate?

give me all

Thought for 19s >

Nice, we're going full "build an HFT footprint lab in-house" mode then 😄

I'll go through **all 6** pieces you asked for, but keep each one tight and usable.

1️⃣ Full statistical pipeline (what a hedge fund would actually do)

Goal:

Given your child orders + full-market L1/L2 + trades, measure whether:

- You can **predict short-horizon returns** from order-book state, and
- Your VWAP flow leaves a **detectable signature** others could exploit.

1. Data you need

Per symbol, per venue (ideally):

- **Quotes**: `time`, `sym`, `bid`, `bidsz`, `ask`, `asksz`, maybe depth levels.
- **Trades**: `time`, `sym`, `px`, `qty`, `side` (or aggressor flag).
- **Your child orders**: `time`, `sym`, `side`, `qty`, `parent_id`, `child_id`, `algo` (VWAP/POV/etc).

2. Build microstructure features

On a unified event stream (e.g. trade+quote updates):

For each event:

- Midprice: `mid_t = (bid + ask) / 2`
- Spread: `spr_t = ask - bid`
- Depth imbalance:
  `imb_t = (bidsz - asksz) / (bidsz + asksz)`
- Order Flow Imbalance (simplified):
  For trade events:
  - `sign_t = +1` if trade at/near ask, `-1` if at/near bid
  - `oflow_t = sign_t * qty_t`
- Volatility proxy: rolling `σ_t` of mid returns over a short window (e.g. 30s).

3. Define returns you want to predict

Typical short-horizon returns:

- **Event time**:
  `r_t(+k) = (mid_{t+k} - mid_t) / mid_t` for k = 1,2,... events

- **Clock time**:
  `r_t(Δ) = (mid_{t+Δ} - mid_t) / mid_t` for $\Delta \in \{100\text{ms}, 1\text{s}, 5\text{s}\}$

You can look at:

- Plain returns
- Spread-normalised returns: `r_t / spr_t`
- Vol-normalised returns: `r_t / σ_t`

4. Baseline vs trade-day (your previous idea formalised)

For each symbol & time-of-day:

- **Baseline**: use **N days when you did not trade** that symbol
  - Estimate conditional expectations:
    `E[r_t(Δ) | state_t = s]` where `state_t` is a vector of features (imbalance, OFI, spread, etc.)
- **Trade days**: same symbol + time-of-day, but days you *did* trade.
  - Estimate the same conditional expectation.

Then compare:

> Does `E[r_t(Δ) | state, trade_day]` deviate significantly from `E[r_t(Δ) | state, baseline]` ?

If yes, your flow is changing micro-returns relative to "normal".

5. Models to estimate conditional expectations

Start simple:

- **Non-parametric buckets**:
  Bucket by `imbalance` (e.g. [-1,-0.5), ...), `spread`, `vol_regime` and compute average `r_t(Δ)`.
- **Linear / logistic regression**:
  `r_t(Δ)` or `sign(r_t(Δ))` on features: imbalance, OFI, spread, last few returns, etc.
- **Tree models (LightGBM/XGBoost)** for richer interactions.

You can run **separate models** for:

- Baseline periods (no your flow)
- Days with your flow

Then:

- Compare coefficients / feature importances
- Compare predicted vs actual returns
- Test whether residuals behave differently when you trade.

6. Outputs & diagnostics

- **Signal curves**: `E[r_t(Δ)]` by OFI / imbalance buckets.
- **ACF/XCF of signed returns / OFI**.
- **Impact shape**: return vs time since your child order.

- "**Detectability score**": how well a classifier can distinguish "your-flow-present" vs "no-flow".

---

2 KDB+/q code skeleton for key microstructure stats

Assume:

- `quote` : `time` , `sym` , `bid` , `bidsz` , `ask` , `asksz`
- `trade` : `time` , `sym` , `px` , `qty` , `side` (`"B"` buyer-initiated, `"S"` seller-initiated)
- `child` : `time` , `sym` , `side` , `qty` , `parentId` , `childId`

2.1 Basic mid, spread, imbalance from quotes

q Copy code

```q
/ Mid, spread, depth imbalance
q)quoteEnriched: update
    mid: (bid + ask) % 2f,
    spr: ask - bid,
    imb: (bidsz - asksz) % (bidsz + asksz)
  from quote
```

2.2 Trade sign and signed volume

q Copy code

```q
/ +1 for buy-initiated, -1 for sell-initiated
q)tradeEnriched: update
    sign: $[side = "B"; 1; -1],
    signedQty: sign * qty
  from trade
```

2.3 Simple OFI (event-time, trade-only version)

q Copy code

```q
/ Aggregate trades to, say, 1s buckets per sym
q)tbars: select
    qty: sum qty,
    signedQty: sum signedQty,
    ntrades: count i,
    vwap: sum qty*px % sum qty
  by sym, time: 0D00:00:01 xbar time
  from tradeEnriched;

/ OFI ~ signed volume per bar
q)tbars: update
    ofi: signedQty
  from tbars;
```

You can join in quote features at bar open or close:

q Copy code

```
/ Get last quote in each bar
q)qbars: select by sym, time:0D00:00:01 xbar time from quoteEnriched;

/ Join tbars with qbars on sym,time
q)bars: lj[`sym`time; tbars; qbars];
```

Now `bars` has `mid` , `spr` , `imb` , `ofi` , etc.

2.4 Returns over horizons

Example: 5s forward mid return:

q Copy code

```
/ Assume bars sorted by sym,time
q)bars: `sym`time xasc bars;

/ 5s horizon: use shifted mid per sym
/ number of bars for 5s if bar is 1s = 5
q)fwd5:{[tab;h]
  tab update
    r5: (neg[h] mid - mid) % mid
  by sym
  from tab
 }[bars;5];

q)bars: fwd5;
```

2.5 ACF and XCF functions in q (event-time / bar-time)

Say you want ACF of signed returns:

q Copy code

```
/ Simple helper: autocorrelation up to maxlag
/ corr is standard correlation in q
q)acf:{[x;maxlag]
  {cor[x[0 + til count x - y]; x[y + til count x - y]]}
    each x enlist til maxlag
 };

/ Example per sym on signedQty
q)acfs: select acfVals: enlist acf[signedQty;20] by sym from tbars;
```

Cross-correlation between OFI and returns:

q Copy code

```
/ Cross-correlation with lags: x -> y lagged by k
q)xcf:{[x;y;maxlag]
  {cor[x[0 + til count x - y]; y[y + til count x - y]]}
    each x enlist til maxlag
 };

/ Example: ofi vs r5
q)xcfVals: select xcfVals: enlist xcf[ofi; r5; 20] by sym from bars;
```

2.6 Join your child orders and compute impact

```q
/ Nearest mid at child send time (previous quote)
q)childWithMid: lj[`sym`time;
    child;
    select last mid by sym, time: 0D00:00:01 xbar time from quoteEnriched
  ];

/ 5s post mid for each child: via as-of join to bars with r5
q)childImpact: aj[`sym`time; childWithMid; select sym,time,r5 from bars];

/ Now childImpact has r5 = 5s mid return post child slice
```

You can then bucket `r5` by spread/OFI/imbalance.

---

3 Detection model: “Is there VWAP-like flow present?”

Think of this as a **classification problem**.

3.1 Labels

Define per bar (or per second):

- `y_t = 1` if your algo is active in that symbol at that time (you have child orders overlapping that bar)
- `y_t = 0` otherwise (baseline periods)

3.2 Features

Per (sym, time bar):

- OFI, depth imbalance
- Spread, mid-level
- Recent returns (e.g. r1, r5, r10)
- Trade intensity (trades per second, volume)
- Buy/sell imbalance (fraction of buy-initiated trades)
- Quote update intensity
- Your parent direction, if known (for your own internal tests)

3.3 Model choices

Start **very simple**:

- Logistic regression:
  - `P(flow_active=1 | features)`
- Gradient boosting (LightGBM/XGBoost) for non-linearities
- Optional: HMM/regime models if you want "algo active vs not" as latent.

3.4 Evaluation

- Train on historical data where you *know* when you traded.
- Check:
  - ROC-AUC for predicting `y_t`.
  - Precision / recall at reasonable thresholds.
  - Whether model generalises to new days.

If an external HFT had the same data, and your model can easily separate `y=1` vs `0`, then **your flow is detectable**.

4️⃣ FFT vs ACF/XCF – when and why

You'd asked this explicitly before; cleaned-up answer:

ACF / XCF (time-domain)

- **ACF**: correlation of a signal with itself at different lags.
  - Good for:
    - Detecting **serial dependence** (momentum / mean reversion).
    - "How persistent is signed volume, returns, OFI?"
- **XCF**: correlation between two signals at different lags.
  - Good for:
    - "Does OFI predict returns 1–10s ahead?"
    - "Do our child order timestamps align with spikes in OFI/returns?"

Pros:

- Directly linked to economic intuition.
- Easy to interpret (e.g. positive at lag +5 → our flow affects returns after 5 bars).
- Works well with irregular and noisy flows.

Cons:

- You must pick max lags.
- Doesn't highlight *strict* periodicities as crisply as FFT.

FFT (frequency-domain)

FFT decomposes your time series into sinusoidal components.

- Good for:
  - Detecting **periodicity** / regular slicing cadence.

- e.g. child order every 2s → strong spectral peak at 0.5 Hz.

Pros:

- Great when the algo slices with quasi-constant cadence.
- Very clear for “metronomic” TWAP, some POVs, some DMA clients.

Cons:

- Sensitive to irregular timing and noise.
- VWAP often has **variable slice size and timing** → spectral lines blur.
- Harder to link to economic microstructure interpretation.

Which is more practical for your VWAP footprint work?

- **For impact / predictability of returns →**
  ACF/XCF is more practical.
- **For detecting a fixed slicing frequency pattern →**
  FFT can help, but only if your flow is quite regular.

If you must prioritise:

1. Start with **ACF/XCF** of:
   - trade sign
   - OFI
   - returns
   - your child order indicator
2. Add FFT later if you explicitly suspect quasi-periodic slicing.

---

5 Flow “signature library” – how to build it

Think like an HFT:

> For each *type of flow* (VWAP buy, VWAP sell, POV, IS, etc.), what does the local market typically look like before / during / after?

5.1 Define signature objects

For each (algo type, side, symbol/sector, volatility regime), build:

- **Impact curve**: average `r(Δ)` for Δ ∈ {1s,5s,30s,300s}.
- **Spread response**: average change in spread around your child orders.
- **Depth response**: how L1/L2 depth changes before vs after.
- **OFI profile**: signed volume pattern around your flow.
- **Autocorrelation profile**: ACF of signed trades around parent order.

Store these as templates:

text Copy code

```
signature[algo="VWAP", side="Buy", volRegime="High", symBucket="LargeCap"]
   .impactCurve[Δ]
```

```
.spreadCurve[Δ]
.ofiCurve[Δ]
.acfSign[lag]
.acfReturn[lag]
```

5.2 How to build them

For every historical parent order:

1. Align time to `t=0` at **first child slice**.
2. For Δ ∈ [-T_pre,…,T_post]:
   - Record features: mid, spread, OFI, depth, etc.
3. Normalise:
   - Express returns in bps or spread units.
   - Maybe normalise by volatility regime.
4. Average across many parents (within each bucket).

This gives you **mean curves + confidence bands**.

6️⃣ Real-time matching of current behaviour to historical signatures

Now the fun bit: "Can someone detect us in real time?"

6.1 Build a live feature window

At each timestamp `t` (e.g. every second):

- Build a **feature vector** from recent history `[t-T, t]`:
  - Sequence of OFI, mid returns, spread, depth imbalance.
  - Maybe compressed into summary stats:
    - cumulative OFI
    - realised volatility
    - skewness of returns
    - fraction of buy-initiated trades
    - trend in mid over last N seconds

6.2 Compare against signature templates

Two broad approaches:

(a) Distance / similarity

For each template (VWAP-buy, VWAP-sell, etc.):

- Compute distance between:
  - **Current feature profile** and
  - **Template mean curve**,

using e.g.:

- **Euclidean distance**
- **Cosine similarity**
- Dynamic Time Warping (DTW) if timing is a bit misaligned.

If distance is small and persistent over several seconds, label:

> "Likely VWAP-like buying flow present."

You can do this internal to see how "obvious" your flow looks.

(b) Supervised classifier (more HFT-like)
You already defined labels (`flow_active` vs `baseline`).

- Train model on historical data to predict `flow_active` .
- In **real time**, feed in the live feature vector → get `P(flow_active=1 | state_t)` .

If this probability is high and stable for a bit, then:

- A hypothetical external HFT with similar data could identify your flow.

How this all ties back to VWAP footprint & "is it possible?"

Putting it together:

1. **Yes, it's possible** to use historical patterns to predict short-horizon returns and detect flow.
2. The **pipeline** above is essentially how:
   - You learn from historical microstructure behaviour (baseline vs your-flow),
   - Build **signatures**, and
   - Run **real-time state-to-return** + **flow-detection** models.
3. If your own internal models can easily spot your flow / predict micro-returns due to it, then external HFTs with comparable data can too.

If you want, next step I can:

- Turn this into a **concrete notebook plan** (Python + PyKX) with:
  - precise table schemas,
  - step-by-step feature engineering,
  - ACF/XCF plots,
  - a simple logistic model for detection;
- or a **kdb-only "lab script"** you can run on a subset of names to prototype quickly.

yes

Cool, let's actually *build the lab* now rather than just talk about it 😄

I'll give you **both**:

1. A **Python notebook plan** (with code) – pandas + sklearn, optionally PyKX entry point.
2. A **kdb+/q "lab" script** to run directly on your tick and child-order tables.

You can then wire to your own infra.

1️⃣ Python notebook: VWAP footprint & flow detection lab

1. Imports & basic setup

python Copy code

```python
# Core
import numpy as np
import pandas as pd

# Stats / modelling
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score, precision_recall_curve
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

# Plotting
import matplotlib.pyplot as plt

# Optional: statsmodels for ACF
from statsmodels.tsa.stattools import acf

# Optional: PyKX if you pull from kdb directly
# import pykx as kx
```

---

2. Data model assumptions

Assume you can bring data into pandas in roughly these shapes:

python Copy code

```python
# Quotes: one row per quote update
# time is ns or ms resolution; convert to pandas.Timestamp
quotes = pd.DataFrame({
    # 'time': ...,
    # 'sym': ...,
    # 'bid': ...,
    # 'bidsz': ...,
    # 'ask': ...,
    # 'asksz': ...
})

# Trades: one row per trade
trades = pd.DataFrame({
    # 'time': ...,
    # 'sym': ...,
    # 'px': ...,
```

```
    # 'qty': ...,
    # 'side': ...  # 'B' buy initiated, 'S' sell initiated
})

# Your child orders
child = pd.DataFrame({
    # 'time': ...,
    # 'sym': ...,
    # 'side': ...,    # 'B' or 'S' for parent direction or child direction
    # 'qty': ...,
    # 'parent_id': ...,
    # 'child_id': ...,
    # 'parent_algo': ...   # 'VWAP', 'POV', etc (optional but nice)
})
```

Convert time to index / datetime:

python Copy code

```python
for df in [quotes, trades, child]:
    df['time'] = pd.to_datetime(df['time'])
```

Filter to one symbol to prototype:

python Copy code

```python
sym = 'AAPL'  # example
q_sym = quotes[quotes['sym'] == sym].copy()
t_sym = trades[trades['sym'] == sym].copy()
c_sym = child[child['sym'] == sym].copy()
```

---

3. Enrich quotes: mid, spread, imbalance

python Copy code

```python
q_sym['mid'] = (q_sym['bid'] + q_sym['ask']) / 2.0
q_sym['spr'] = q_sym['ask'] - q_sym['bid']
q_sym['imb'] = (q_sym['bidsz'] - q_sym['asksz']) / (q_sym['bidsz'] + q_sym['asksz
```

---

4. Enrich trades: sign, signed volume

python Copy code

```python
t_sym['sign'] = np.where(t_sym['side'] == 'B', 1, -1)
t_sym['signed_qty'] = t_sym['sign'] * t_sym['qty']
```

---

5. Build 1-second bars combining trades & quotes

You can do this many ways; here's a straightforward approach:

python Copy code

```python
# Set index for resampling
q_sym = q_sym.set_index('time').sort_index()
t_sym = t_sym.set_index('time').sort_index()

# Quote snapshot per second: take *last* quote in the bucket
q_bar = q_sym.resample('1S').last()[['mid', 'spr', 'imb']]

# Trade aggregates per second
def vwap(px, qty):
    return (px * qty).sum() / qty.sum() if qty.sum() > 0 else np.nan

agg = {
    'px': vwap,
    'qty': 'sum',
    'signed_qty': 'sum',
}

t_bar = t_sym.resample('1S').apply(agg)
t_bar.rename(columns={'px': 'vwap', 'qty': 'vol', 'signed_qty': 'ofi'}, inplace=T

# Combine into a single DataFrame
bars = q_bar.join(t_bar, how='outer')
bars['sym'] = sym

# Fill forward quotes where missing; leave trades as is
bars[['mid', 'spr', 'imb']] = bars[['mid', 'spr', 'imb']].ffill()
bars['vol'] = bars['vol'].fillna(0.0)
bars['ofi'] = bars['ofi'].fillna(0.0)
```

Now `bars` has:

- `mid`, `spr`, `imb`, `vwap`, `vol`, `ofi` for each second.

---

6. Compute short-horizon returns

Example: `r_1s`, `r_5s`, `r_30s` forward mid returns.

python Copy code

```python
for horizon, label in [(1, 'r_1s'), (5, 'r_5s'), (30, 'r_30s')]:
    fwd_mid = bars['mid'].shift(-horizon)
    bars[label] = (fwd_mid - bars['mid']) / bars['mid']
    # Normalise by spread if you like:
    bars[f'{label}_spr'] = bars[label] / bars['spr']
```

7. Label bars as “flow active” vs baseline

We map child orders to 1-second bars and set a binary label.

python Copy code

```python
# Child orders to 1S buckets
c_sym = c_sym.set_index('time').sort_index()
c_bar = c_sym.resample('1S').agg({'qty': 'sum'})

# Label whether we were active in that second
bars['flow_active'] = (c_bar['qty'] > 0).reindex(bars.index, fill_value=False)

# Optional: also keep parent direction
# If you want direction = sign of net parent/child qty that second:
bars['flow_dir'] = np.sign(c_bar['qty']).reindex(bars.index)
```

---

8. Split baseline vs trade-day populations

You can do this by days where you trade vs days you don’t, or by `flow_active` .

Simple version:

python Copy code

```python
baseline = bars[~bars['flow_active']].copy()
trade_period = bars[bars['flow_active']].copy()
```

For your original *study* concept, you’d probably:

- Identify *days* with no flow in that symbol → baseline days.
- Identify days with flow → treatment days.
- Tag each bar with `day_has_flow` flag and use that as the grouping.

---

9. ACF/XCF analysis

9.1 ACF of returns or OFI

python Copy code

```python
# Drop NaNs
ret_series = bars['r_1s'].dropna()
ofi_series = bars['ofi'].loc[ret_series.index]

max_lag = 20
acf_ret = acf(ret_series, nlags=max_lag, fft=True)
acf_ofi = acf(ofi_series, nlags=max_lag, fft=True)
```

Plot:

python Copy code

```python
lags = np.arange(max_lag + 1)

plt.figure()
plt.stem(lags, acf_ret, use_line_collection=True)
plt.xlabel('Lag (seconds)')
plt.ylabel('ACF of r_1s')
plt.title(f'ACF of 1s returns – {sym}')
plt.show()
```

9.2 XCF (OFI → returns)

No built-in, but you can roll your own:

python Copy code

```python
def xcf(x, y, max_lag):
    x = np.asarray(x)
    y = np.asarray(y)
    n = len(x)
    x = (x - x.mean()) / x.std()
    y = (y - y.mean()) / y.std()
    corrs = []
    for lag in range(max_lag + 1):
        corrs.append(np.mean(x[:n-lag] * y[lag:]))
    return np.array(corrs)

ret = ret_series.values
ofi = ofi_series.values

xcf_ofi_ret = xcf(ofi, ret, max_lag=20)

plt.figure()
plt.stem(np.arange(21), xcf_ofi_ret, use_line_collection=True)
plt.xlabel('Lag (seconds)')
plt.ylabel('XCF: OFI → r_1s')
plt.title(f'Cross-correlation OFI to 1s returns – {sym}')
plt.show()
```

Do this for **baseline** vs **trade-period** and compare curves.

---

10. Simple flow detection classifier (hedge-fund style)

Predict `flow_active` from state variables. If this works well, your flow is detectable.

python Copy code

```python
# Feature matrix X, target y
feature_cols = [
    'spr',
    'imb',
    'ofi',
    'vol',
    'r_1s',  # previous returns - use lagged versions
]

# You should lag returns so you're not using forward info:
bars['r_1s_lag1'] = bars['r_1s'].shift(1)
feature_cols = ['spr', 'imb', 'ofi', 'vol', 'r_1s_lag1']

df_model = bars.dropna(subset=feature_cols + ['flow_active'])
X = df_model[feature_cols].values
y = df_model['flow_active'].astype(int).values

X_train = X  # for prototype, you can do a true train/test split by day later
y_train = y
```

Model:

python Copy code

```python
clf = Pipeline([
    ('scaler', StandardScaler()),
    ('logit', LogisticRegression(max_iter=1000))
])

clf.fit(X_train, y_train)
y_score = clf.predict_proba(X_train)[:, 1]

auc = roc_auc_score(y_train, y_score)
print(f"ROC-AUC for detecting your flow: {auc:.3f}")
```

If `AUC >> 0.5` (e.g. 0.7–0.9), then **given your features**, your flow is statistically detectable.

---

11. Distance / similarity to historical "flow signatures"

Suppose you've built **average OFI + return profiles** around your flow (precomputed templates). For now, just show how to compute a rolling feature vector and compare with a template.

11.1 Build rolling feature window

For each time `t`, for the last `T` seconds, stack OFI & returns into a vector:

python Copy code

```python
window = 10  # seconds
```

```python
# Example narrow features: last 10s OFI and returns
ofi_seq = np.stack([
    bars['ofi'].shift(lag)
    for lag in range(window)
], axis=1)

r1_seq = np.stack([
    bars['r_1s'].shift(lag)
    for lag in range(window)
], axis=1)

feature_seq = np.hstack([ofi_seq, r1_seq])
feature_seq = pd.DataFrame(
    feature_seq,
    index=bars.index,
    columns=[f'ofi_t-{k}' for k in range(window)] + [f'r1_t-{k}' for k in range(w
)
```

11.2 Create a template vector from known VWAP intervals

Say you know intervals where you traded with a VWAP parent; you can average their sequences:

python Copy code

```python
# Use only times when flow_active is True to define your signature
sig_df = feature_seq[bars['flow_active']].dropna()

template_vec = sig_df.mean(axis=0).values  # 1D vector
template_norm = template_vec / np.linalg.norm(template_vec)
```

11.3 Compute cosine similarity in real-time

python Copy code

```python
# Cosine similarity of each window to the template
seq = feature_seq.dropna()
seq_arr = seq.values
seq_norm = seq_arr / np.linalg.norm(seq_arr, axis=1, keepdims=True)

cos_sim = (seq_norm @ template_norm)

cos_sim_series = pd.Series(cos_sim, index=seq.index)
```

You can now:

- Plot `cos_sim_series` over time.
- Flag intervals where cos_sim > threshold (e.g. 0.8) as “VWAP-like flow signature”.

---

2 kdb+/q “VWAP footprint lab” script

Below is a q script you can adapt. It assumes:

- `quote` and `trade` tables exist for some date range.
- `child` table exists with your child orders.

You can run it symbol by symbol or across a universe.

2.1 Basic table schema assumptions

q Copy code

```
/ Quotes
/ schema: quote:([] time:`timestamp$(); sym:`symbol$(); bid:`float$(); bidsz:`flo

/ Trades
/ trade:([] time:`timestamp$(); sym:`symbol$(); px:`float$(); qty:`float$(); side

/ Child orders
/ child:([] time:`timestamp$(); sym:`symbol$(); side:`symbol$(); qty:`float$(); pa
```

2.2 Enrich quotes

q Copy code

```
/ mid, spread, imbalance
quoteEnriched: update
    mid:(bid + ask) % 2f,
    spr:ask - bid,
    imb:(bidsz - asksz) % (bidsz + asksz)
  from quote;
```

2.3 Enrich trades: sign & signed volume

q Copy code

```
/ sign: +1 for buy-initiated, -1 for sell-initiated
tradeEnriched: update
    sign:$[side=`B; 1f; -1f],
    signedQty: sign * qty
  from trade;
```

2.4 1-second bars from trades

q Copy code

```
/ Helper VWAP aggregate
vwap:{[px;qty] $[0f=sum qty; (px * qty) wsum qty; 0n]}

/ Aggregate trades to 1-second buckets
tbars: select
    vwap:vwap[px;qty],
    vol:sum qty,
    ofi:sum signedQty,
    ntrades:count i
  by sym, time:0D00:00:01 xbar time
  from tradeEnriched;
```

2.5 Quote per-second snapshots

q Copy code

```
/ Last quote in each 1-second bucket
qbars: select last mid, last spr, last imb
  by sym, time:0D00:00:01 xbar time
  from quoteEnriched;
```

2.6 Combine quote + trade bars

q Copy code

```
/ Left join trades onto quotes or vice-versa - I'll use full outer via uj
/ But we'll keep one canonical table 'bars'.

bars: uj[
    select sym,time,mid,spr,imb from qbars;
    select sym,time,vwap,vol,ofi,ntrades from tbars
 ];

/ Sort for lagging
bars: `sym`time xasc bars;

/ Fill forward quotes, default 0 for vol/ofi/ntrades
bars: update
    mid: mid ffill,
    spr: spr ffill,
    imb: imb ffill,
    vol: 0f ^ vol,
    ofi: 0f ^ ofi,
    ntrades: 0 ^ ntrades
  from bars;
```

2.7 Forward returns in q

Example: 1s, 5s, 30s forward returns by sym.

q Copy code

```
/ Helper: forward horizons in bars (assuming 1-second bars)
/ h is integer number of bars
fwdRet:{[t;h]
  update r:(neg[h] mid - mid) % mid
  by sym
  from t
 };

/ Compute forward returns in place
bars: update r1s:(neg[1] mid - mid) % mid by sym from bars;
bars: update r5s:(neg[5] mid - mid) % mid by sym from bars;
bars: update r30s:(neg[30] mid - mid) % mid by sym from bars;

/ Optional: spread-normalised
bars: update r1s_spr: r1s % spr from bars;
```

2.8 Label bars with your flow activity

Map child orders to the 1-second grid.

q Copy code

```
/ Aggregate child orders into 1-second buckets
childBars: select
    childVol: sum qty,
    childDir: sum $[side=`B; qty; -qty]   / net directional volume
  by sym, time:0D00:00:01 xbar time
  from child;

/ Join into bars
bars: lj[`sym`time; bars; childBars];

/ Flag flow active
bars: update
    flowActive: 0f < childVol,        / bool
    flowDir: $[childVol=0f; 0f; sign childDir] / +1 buy, -1 sell, 0 none
  from bars;
```

2.9 ACF/XCF functions in q

q Copy code

```
/ Autocorrelation up to maxlag (returns list of length maxlag+1)
acf:{[x;maxlag]
  x: x where not null x;
  n: count x;
  m: avg x;
  xs: x - m;
  denom: sum xs*xs;
  / lag 0..maxlag
  { (sum xs[til n-y]*xs[y+til n-y]) % denom } each til maxlag+1
};

/ Cross-correlation x -> y
xcf:{[x;y;maxlag]
  / align & drop nulls
  x: x where not null x;
  y: y where not null y;
  n: min[count each (enlist x; enlist y)];
  x: x til n;
  y: y til n;
  mx: avg x;
  my: avg y;
  xs: x - mx;
  ys: y - my;
  denom: sqrt sum xs*xs * sum ys*ys;
  { (sum xs[til n-y]*ys[y+til n-y]) % denom } each til maxlag+1
};
```

Use it:

```q
/ Example per sym ACF of 1s returns, OFI, etc.
maxlag: 20;

retAcf: select acfVals: enlist acf[r1s; maxlag] by sym from bars;
ofiAcf: select acfVals: enlist acf[ofi; maxlag] by sym from bars;

/ Example XCF: OFI -> r1s
ofiR1xcf: select xcfVals: enlist xcf[ofi; r1s; maxlag] by sym from bars;
```

2.10 Impact around child orders (VWAP footprint style)

Compute average return after your child slices.

```q
/ Function: impact around child orders at given horizon h (bars)
impactAt:{[bars;h]
  / pick timestamps where flowActive just turned on (or where childVol>0)
```

```
  co: select from bars where flowActive & 0=prev flowActive by sym;
  / for each such event, look up forward r1s or r5s etc.
/ Using r5s as example
  rname:`$"r",string h,"s"; / e.g. `r5s
  select avgRet: avg .[bars; (;rname); ()] by sym from
    select sym,time,ret:.[bars; `sym`time!(sym; time); rname] from co
};

/ Example: 5s impact
impact5: impactAt[bars;5];
```

If you want a full curve (`Δ` = 1,2,5,10,30 seconds) you can:

- Precompute `r1s,r2s,r5s,r10s,r30s`
- Aggregate avg per horizon & per side.

2.11 Simple "detectability" score in q (bucket-level)

Instead of full logistic regression in q, you can:

1. Bucket features (e.g. OFI and imbalance).
2. For each bucket, compute:
   - Fraction of time flow is active.
   - Conditional returns.

If the empirical `P(flowActive=1 | bucket)` varies a lot across buckets, your flow is conditionally predictable.

q Copy code

```q
/ Example: bucket OFI and imbalance
bucket:{[x;edges] edges bin x}

/ define edges
ofiEdges:-1 -0.5 -0.1 0 0.1 0.5 1;
imbEdges:-1 -0.5 -0.1 0 0.1 0.5 1;

bars: update
   ofiBucket: bucket[ofi; ofiEdges],
   imbBucket: bucket[imb; imbEdges]
  from bars;

/ Group stats
detectStats: select
    pFlow: avg flowActive,              / probability flow active
    avgRet1s: avg r1s,
    avgOfi: avg ofi
  by sym, ofiBucket, imbBucket
  from bars;
```

```
/ You can inspect detectStats buckets where pFlow is high vs low.
```

If there are very distinct “high pFlow” buckets that line up with certain imbalance/OFI states, then someone who knows that mapping can infer your presence.

What you have now

You now have:

- ✅ **Python notebook skeleton** for:
  - Bar construction
  - Feature engineering
  - ACF/XCF
  - Flow detection via logistic regression
  - Cosine-similarity “signature” matching
- ✅ **kdb+/q lab script** to:
  - Build 1s bars
  - Compute mid/spread/imbalance/OFI
  - Compute forward returns