EMA + RSI/MACD Coinbase Trading Bot

A lightweight, session-based crypto trading bot for Coinbase Advanced that trades multiple products using **EMA crossovers** as the captain signal and **RSI/MACD** as advisors (veto-only). It manages daily spend, cooldowns, portfolio P&L, and logs KPIs to CSV for post-run analysis.

> Built around Coinbase REST/WebSocket SDKs, with a focus on simple, explainable signals and safety rails.

How it works (high level)

1. **WebSocket ticker** subscribes to products and streams prices.

2. **Indicators** update per tick: short/long EMAs, RSI, and MACD.

3. **Signal** = short EMA crosses long EMA, with a small dead-band and N-tick confirmation to avoid whipsaws.

4. **Advisors (optional)** can veto obviously bad entries:

   - **RSI** blocks BUYs if overbought; blocks SELLs if oversold.

   - **MACD** histogram (normalized in bps) blocks BUYs that are too negative or SELLs that are too positive.

5. **Order placement** honors daily USD cap, cooldowns, and (optionally) a hard stop on unrealized losses.

6. **Maker-first** execution tries post-only limit orders using per-product offsets; otherwise falls back to market.

7. **Fills** are reconciled immediately (best-effort) and on startup, updating positions, cost basis, and realized P&L.

8. **Logs** and **CSV KPIs** are written to `.state/` for later review.

Project layout

```
bot/
  config.py        # Tunables (products, EMA lengths, advisors, caps, maker offsets, etc.)
  tradebot.py      # Core bot: WS loop, signals, orders, P&L, fills reconciliation, CSV KPIs
  indicators.py    # EMA, RSI, MACD implementations
  strategy.py      # AdvisorSettings + veto logic (RSI/MACD)
  orders.py        # Maker price/size math and rounding helpers
  persistence.py   # JSON state, rotating logs, spend & cooldown trackers
  constants.py     # .state file paths and shared constants
  utils.py         # Thin re-exports of persistence + constants
main.py            # Entry point with logging, env load, graceful signals
```

Features

- Multi-product EMA crossover with dead-band and confirmation

- Per-product EMA params & maker offsets

- RSI & MACD (normalized bps) **veto-only** advisors

- Daily **BUY** spend cap & per-product cooldowns
- Optional **hard stop** (sell if price drops X bps below cost basis)
- **Post-only maker** preference with precise tick/size rounding
- CSV KPI log: slippage, fees, liquidity, hold time, P&L per fill
- Startup **fills reconciliation** (lookback window) to sync portfolio
- Graceful shutdown with end-of-session P&L footer

Requirements

- Python 3.10+
- `coinbase` official SDK
- `python-dotenv` (for `APIkeys.env`)
- A Coinbase Advanced API key/secret (read & trade) and optional portfolio ID

Install:

```bash
pip install coinbase python-dotenv
```

Configuration

Key parameters live in `bot/config.py`. You can fork values globally and/or per product.

Core trading
- `product_ids`: list of `COIN-USD` products to trade

- `short_ema`, `long_ema`: global EMA periods (overridden per product)

- `min_ticks`: warmup ticks required before trading

- `confirm_ticks`: consecutive ticks required to confirm a cross

- `ema_deadband_bps`: small band to avoid flapping around the cross

Session & risk
- `dry_run`: simulate orders without sending to exchange

- `usd_per_order`: notional per order

- `max_usd_per_day`: **BUY** cap per UTC day

- `cooldown_sec`: min seconds between trades on the same product

- `hard_stop_bps`: if set, emergency market exit when price ≤ CB * (1 - bps/10,000)

Advisors (RSI/MACD)
- `enable_advisors` / `use_advisors`: master switch

- `rsi_period`, `rsi_buy_floor`, `rsi_sell_ceiling`

  - SELLs blocked if RSI < `rsi_buy_floor` (oversold)

  - BUYs blocked if RSI > `rsi_sell_ceiling` (overbought)

- `macd_fast`, `macd_slow`, `macd_signal`

- `macd_buy_min` (bps): BUYs require MACD_hist_bps ≥ this

- `macd_sell_max` (bps): SELLs require MACD_hist_bps ≤ this

Maker execution

- `prefer_maker`: default True; `prefer_maker_for_sells`: separate toggle for exits
- `maker_offset_bps`: default maker offset (bps)
- `maker_offset_bps_per_product`: per-product overrides

Per-product EMA overrides

```python
ema_params_per_product = {
  "BTC-USD": {"short_ema": 45, "long_ema": 150, "min_ticks": 220},
  ...
}
```

Environment & running

Create an `APIkeys.env` in the repo root (or set `ENV_PATH` to another path):

```
COINBASE_API_KEY=...
COINBASE_API_SECRET=...
PORTFOLIO_ID=...      # optional
```

Run the bot:

```bash
```

python -m main

or

python main.py

```

Graceful exits are handled (Ctrl+C / SIGTERM). On shutdown, a session footer is appended to the trade log.

## Files written to `.state/`

- `trade_log.txt` – human-readable trade lines and session P&L footers

- `daily_spend.json` – per-day BUY totals (enforces `max_usd_per_day`)

- `last_trades.json` – per-product timestamps for cooldowns

- `portfolio.json` – positions, cost basis, realized P&L

- `processed_fills.json` – dedupe set for seen fills

- `trades.csv` – KPI rows per fill: ts, side, size/price, quote USD, fee, liquidity, pnl, slippage, hold time

> The folder defaults to `<repo>/.state`. Override via `BOT_STATE_DIR` if desired.
```

Signal logic (details)

- **Warm-up:** wait until `min_ticks` per product.

- **Cross:** compute `rel = sign(short - long)` with a small `ema_deadband_bps` dead-band.

- **Prime phase:** the first time a product gets a `rel`, the bot *primes* and does not trade.

- **Confirm:** require `confirm_ticks` consecutive, consistent `rel` to count as a confirmed cross.

- **State change:** only trade when the new confirmed `rel` differs from the previous one.

- **Guards:** skip SELL if no position; apply `hard_stop_bps` if configured.

- **Advisors:** if enabled, veto BUY/SELL when RSI/MACD conditions fail.

- **Caps:** enforce per-day **BUY** cap and per-product cooldown.

Execution strategy

- **Maker-first (post-only) limit orders** compute price/size from:

  - reference = best bid/ask (or last) ± `maker_offset_bps` (per product)

  - size ≈ `usd_per_order / limit_price`, rounded to exchange increments

- **Market orders** are used if maker is disabled (or for hard stops).

- **SELL size** is clamped to your current position.

Both paths record an **intent snapshot** (price at signal) to compute **slippage** and other KPIs when fills are fetched.

Fills & P&L

- **Immediate fetch** after order placement: pulls fills for that order ID, updates position, cost basis, realized P&L, and logs CSV KPIs.

- **Startup reconciliation:** fetches recent fills over a configured lookback window and applies any missed fills to the local portfolio store.

Realized P&L is tracked across runs and logged both lifetime and per-run (relative to a baseline captured at startup).

Safety notes

- Use **dry_run** first to validate signals and CSV output.

- Start with small `usd_per_order` and low `max_usd_per_day`.

- Maker orders can **miss** fills during fast moves; consider `prefer_maker_for_sells=False` to exit faster.

- `hard_stop_bps` is a true emergency exit—size sells are sent at market.

Troubleshooting

- **No trading happening?** Ensure `min_ticks`/`confirm_ticks` aren’t too strict; verify WS prices are streaming; check advisors aren’t vetoing entries.

- **Daily cap reached early?** Increase `max_usd_per_day` or reduce `usd_per_order`.

- **CSV not created?** A fill must occur (or reconciliation must run) to add rows; check permissions on `.state/`.

- **Portfolio desynced?** Run the bot and let the **reconciliation** step pull recent fills.