# Chapter8 The ML4T Workflow

## From Model to Strategy Backtesting

YiLong Geng Yuting Jiao

Fintech Stu
Wuhan University

November 17, 2022

# Contents

- 1 How to backtest an ML-driven strategy

- 2 Backtesting pitfalls and how to avoid them

- 3 How a backtesting engine works

- 4 backtrader – a flexible tool for local backtests

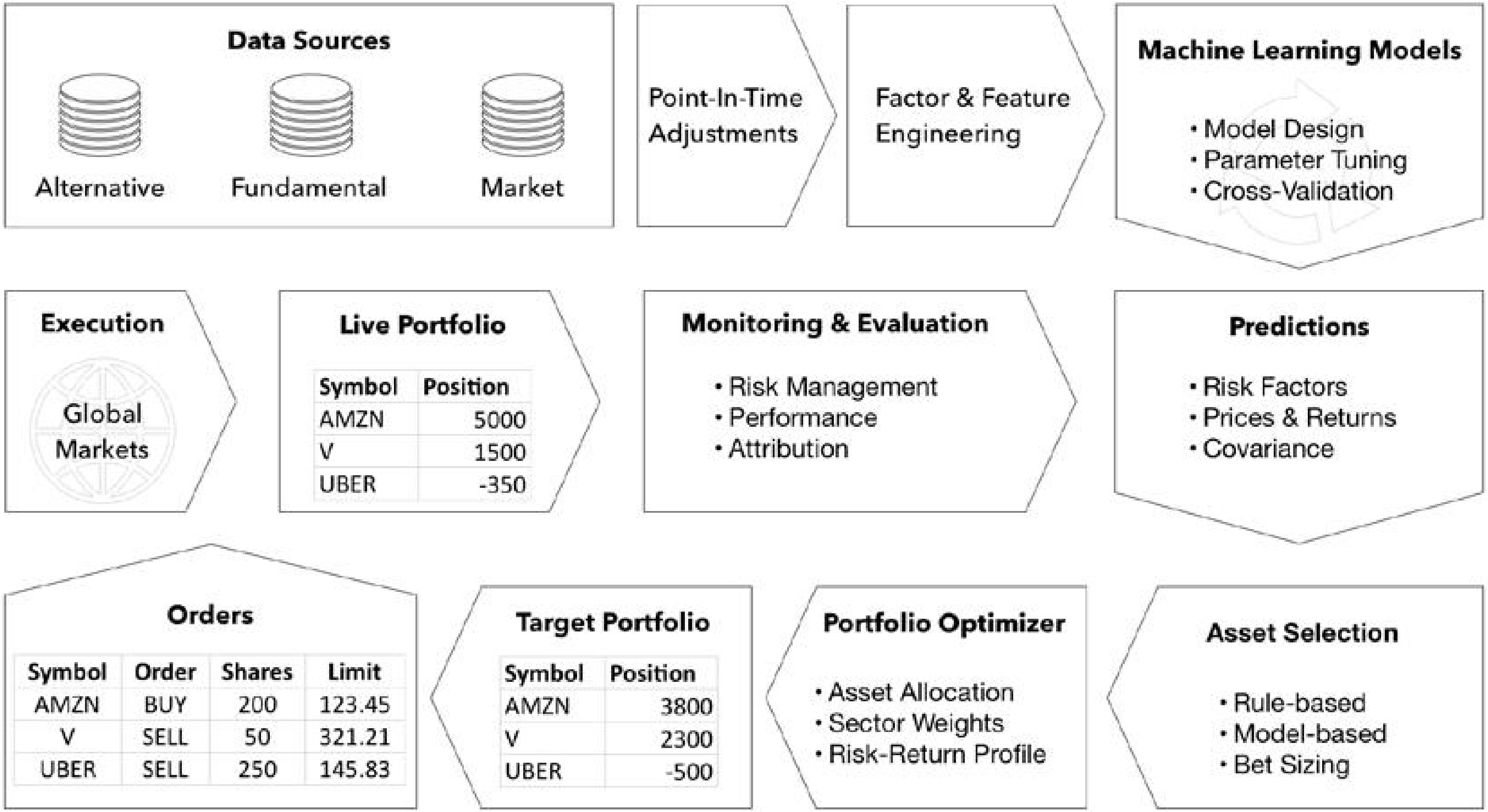
- 5 zipline – scalable back testing by Quantopian

# What is backtesting?

- Backtesting is the general method for seeing how well a strategy or model would have done ex-post. Backtesting assesses the viability of a trading strategy by discovering how it would play out using historical data. If backtesting works, traders and analysts may have the confidence to employ it going forward.

# 1 How to backtest an ML-driven strategy

## The ML4T Workflow



**Data Sources**

Alternative

Fundamental

Market

Point-In-Time Adjustments

Factor & Feature Engineering

**Machine Learning Models**

- Model Design
- Parameter Tuning
- Cross-Validation

**Predictions**

- Risk Factors
- Prices & Returns
- Covariance

**Monitoring & Evaluation**

- Risk Management
- Performance
- Attribution

**Live Portfolio**

| Symbol | Position |
|---|---|
| AMZN | 5000 |
| V | 1500 |
| UBER | -350 |

**Execution**

Global Markets

**Orders**

| Symbol | Order | Shares | Limit |
|---|---|---|---|
| AMZN | BUY | 200 | 123.45 |
| V | SELL | 50 | 321.21 |
| UBER | SELL | 250 | 145.83 |

**Target Portfolio**

| Symbol | Position |
|---|---|
| AMZN | 3800 |
| V | 2300 |
| UBER | -500 |

**Portfolio Optimizer**

- Asset Allocation
- Sector Weights
- Risk-Return Profile

**Asset Selection**

- Rule-based
- Model-based
- Bet Sizing

# 1 How to backtest an ML-driven strategy

1. Source and prepare market, fundamental, and alternative data
2. Engineer predictive alpha factors and features
3. Design, tune, and evaluate ML models to generate trading signals
4. Decide on trades based on these signals, for example, by applying rules
5. Size individual positions in the portfolio context
6. Simulate the resulting trades triggered using historical market data
7. Evaluate how the resulting positions would have performed

# 2 Backtesting pitfalls and how to avoid them

- **Getting the data right**
  - Look-ahead bias – use only point-in-time data
  - Survivorship bias – track your historical universe
  - Outlier control – do not exclude realistic extremes
  - Sample period – try to represent relevant future scenarios
- **Getting the simulation right**
  - Mark-to-market performance – track risks over time
  - Transaction costs – assume a realistic trading environment
  - Timing of decisions – properly sequence signals and trades
- **Getting the statistics right**
  - The minimum backtest length and the deflated SR
  - Optimal stopping for backtests

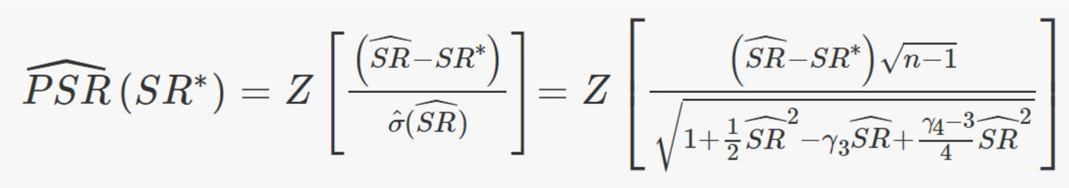
# 2 Backtesting pitfalls and how to avoid them

- Backtest Overfitting
  - multiple test
  - The minimum backtest length
  - Optimal stopping for backtests: math.ucla.edu

https://quantdare.com/deflated-sharpe-ratio-how-to-avoid-been-fooled-by-randomness/
https://www.davidhbailey.com/dhbpapers/deflated-sharpe.pdf
http://datagrid.lbl.gov/backtest/

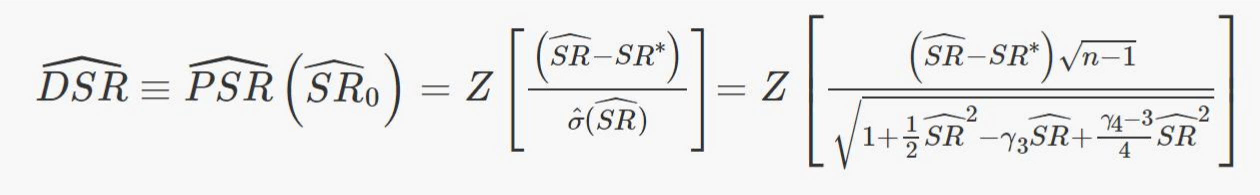
# 2 Backtesting pitfalls and how to avoid them

- Deflated Sharpe Ratio
  - Sharpe Ratio
  - Probalilistic Sharpe Ratio



$$\widehat{PSR}(SR^*) = Z\left[\frac{\left(\widehat{SR}-SR^*\right)}{\hat{\sigma}(\widehat{SR})}\right] = Z\left[\frac{\left(\widehat{SR}-SR^*\right)\sqrt{n-1}}{\sqrt{1+\frac{1}{2}\widehat{SR}^2-\gamma_3\widehat{SR}+\frac{\gamma_4-3}{4}\widehat{SR}^2}}\right]$$

  - Deflated Sharpe Ratio



$$\widehat{DSR} \equiv \widehat{PSR}\left(\widehat{SR}_0\right) = Z\left[\frac{\left(\widehat{SR}-SR^*\right)}{\hat{\sigma}(\widehat{SR})}\right] = Z\left[\frac{\left(\widehat{SR}-SR^*\right)\sqrt{n-1}}{\sqrt{1+\frac{1}{2}\widehat{SR}^2-\gamma_3\widehat{SR}+\frac{\gamma_4-3}{4}\widehat{SR}^2}}\right]$$

## references

https://quantdare.com/deflated-sharpe-ratio-how-to-avoid-been-fooled-by-randomness/

https://www.davidhbailey.com/dhbpapers/deflated-sharpe.pdf

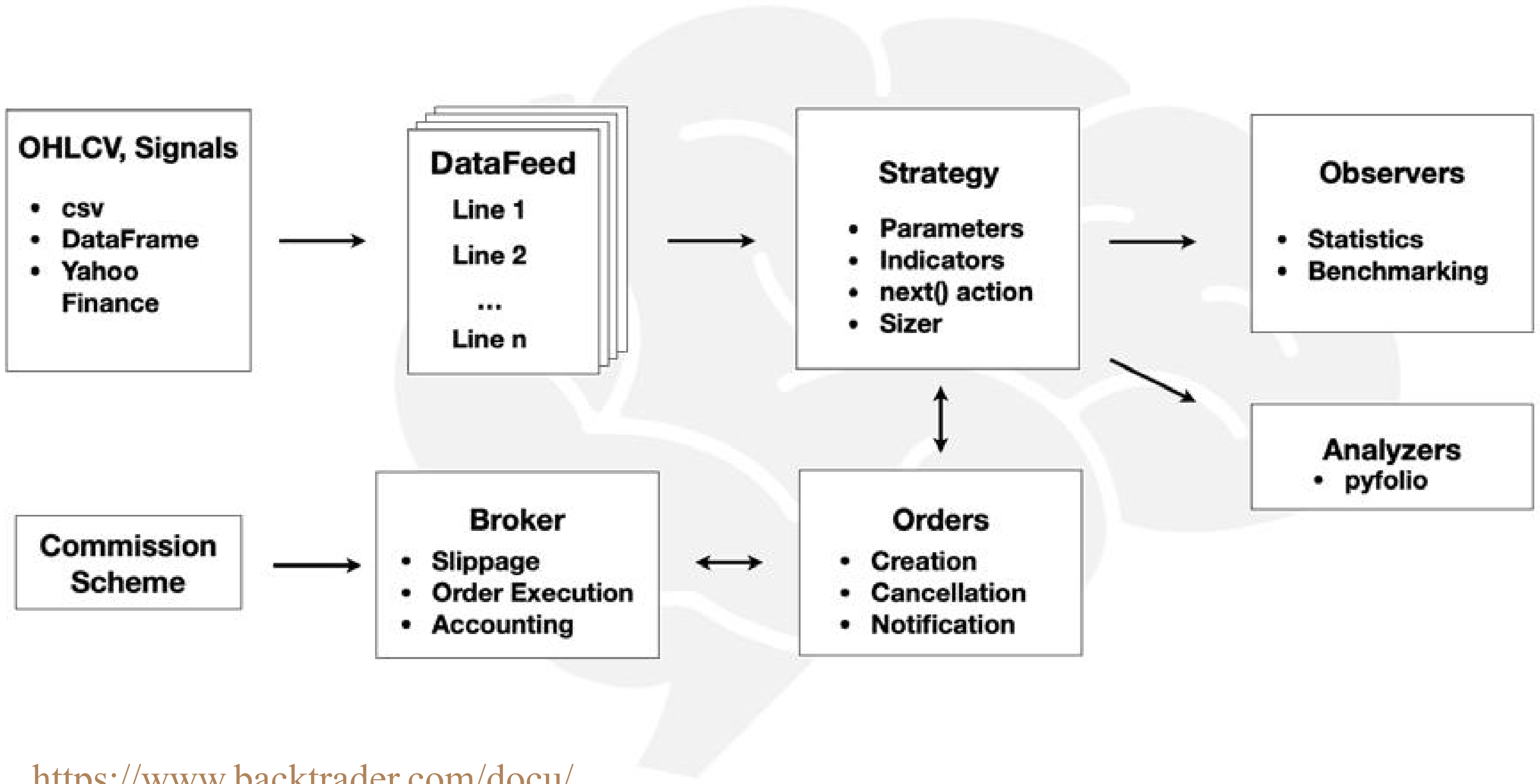
http://datagrid.lbl.gov/backtest/

# 3 How a backtesting engine works

- **Vectorized versus event-driven backtesting**
  - A vectorized backtest simply multiplies a signal vector that represents the target position size with a vector of returns for the investment horizon to compute the period performance.
  - An event-driven backtestingby design, can be used for both historical backtesting and live trading with minimal switch-out of components.
- **Key implementation aspects**
  - Data ingestion – format, frequency, and timing
  - Factor engineering – built-in factors versus libraries
  - ML models, predictions, and signals
  - Trading rules and execution
  - Performance evaluation

# 4 backtrader – a flexible tool for local backtests

**backtrader "Cerebro" Architecture**



OHLCV, Signals
- csv
- DataFrame
- Yahoo Finance

DataFeed
Line 1
Line 2
...
Line n

Strategy
- Parameters
- Indicators
- next() action
- Sizer

Observers
- Statistics
- Benchmarking

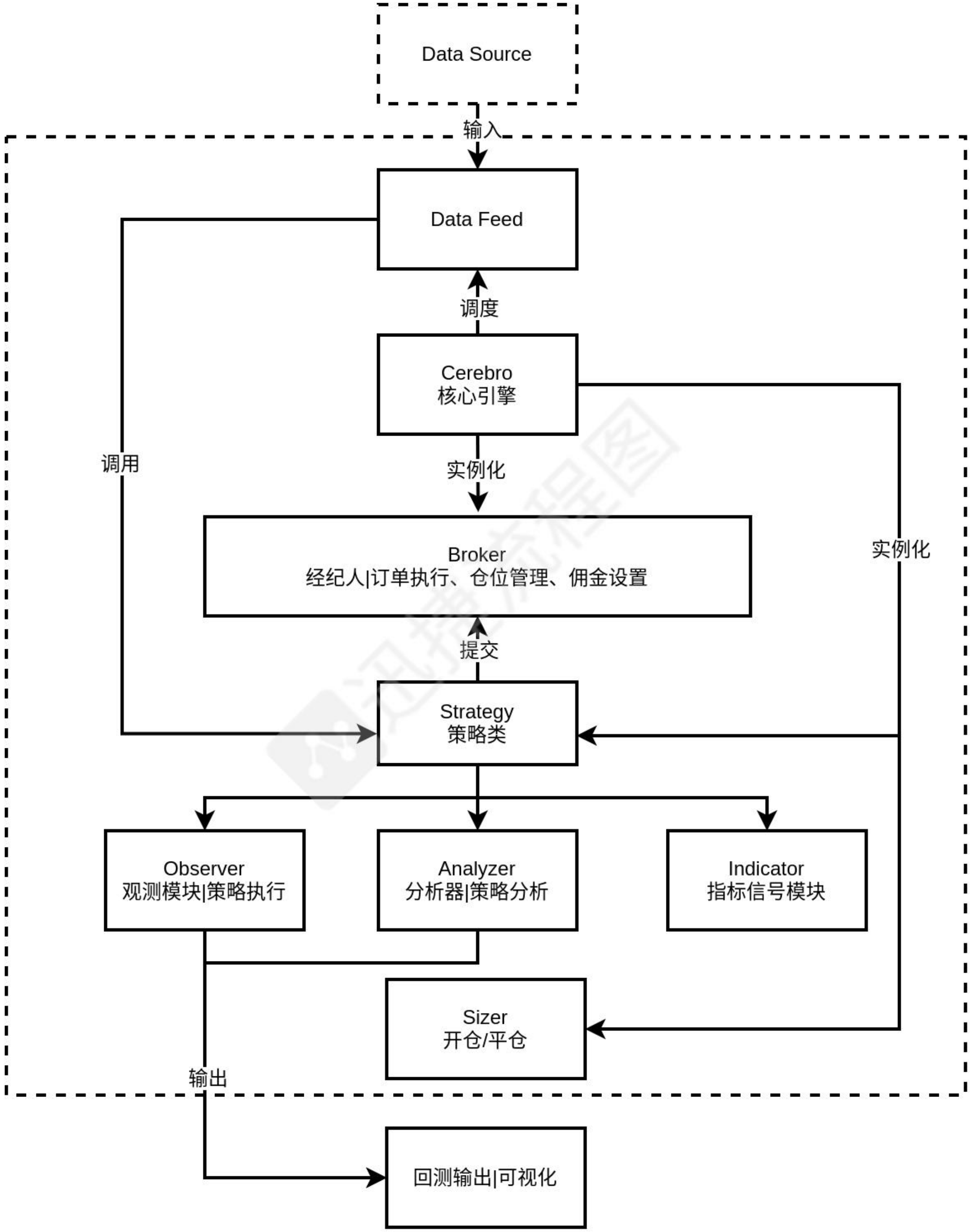
Analyzers
- pyfolio

Commission Scheme

Broker
- Slippage
- Order Execution
- Accounting

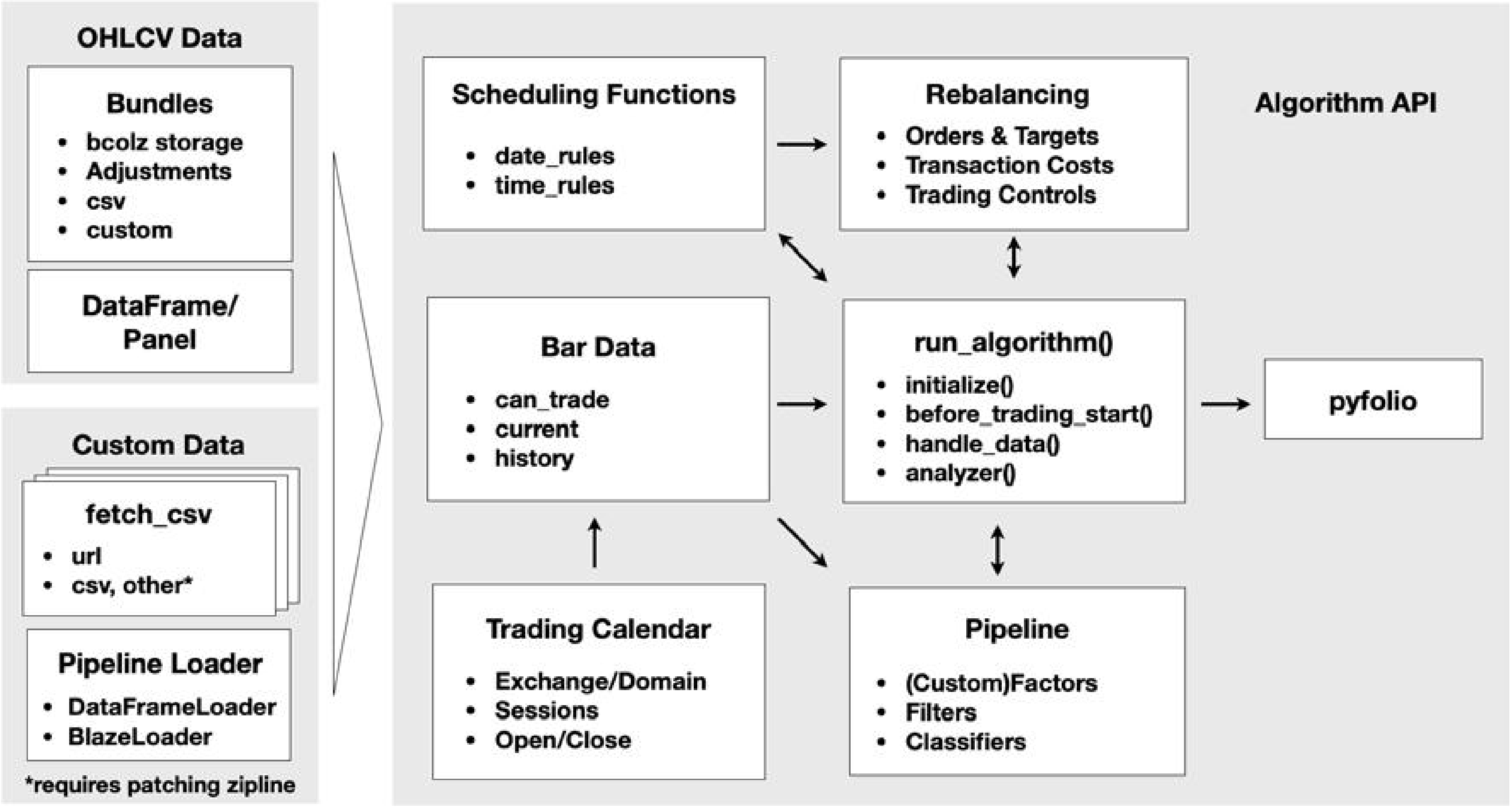
Orders
- Creation
- Cancellation
- Notification

https://www.backtrader.com/docu/


Data Source
输入
Data Feed
调度
Cerebro
核心引擎
调用
实例化
Broker
经纪人|订单执行、仓位管理、佣金设置
实例化
提交
Strategy
策略类
Observer
观测模块|策略执行
Analyzer
分析器|策略分析
Indicator
指标信号模块
Sizer
开仓/平仓
输出
回测输出|可视化

# 5 zipline - scalable back testing by Quantopian

## Zipline Architecture



**OHLCV Data**

**Bundles**
- bcolz storage
- Adjustments
- csv
- custom

**DataFrame/ Panel**

**Custom Data**

**fetch_csv**
- url
- csv, other*

**Pipeline Loader**
- DataFrameLoader
- BlazeLoader

*requires patching zipline

**Scheduling Functions**
- date_rules
- time_rules

**Rebalancing**
- Orders & Targets
- Transaction Costs
- Trading Controls

**Algorithm API**

**Bar Data**
- can_trade
- current
- history

**run_algorithm()**
- initialize()
- before_trading_start()
- handle_data()
- analyzer()

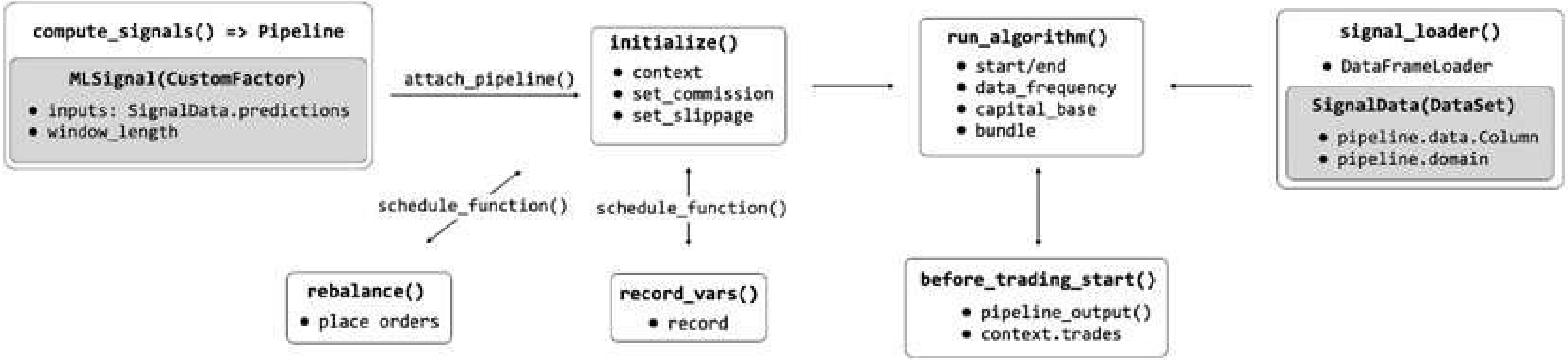
**pyfolio**

**Trading Calendar**
- Exchange/Domain
- Sessions
- Open/Close

**Pipeline**
- (Custom)Factors
- Filters
- Classifiers

https://pypi.org/project/zipline-reloaded/

# 5 zipline – scalable back testing by Quantopian



compute_signals() => Pipeline

MLSignal(CustomFactor)

- inputs: SignalData.predictions
- window_length

attach_pipeline()

initialize()

- context
- set_commission
- set_slippage

run_algorithm()

- start/end
- data_frequency
- capital_base
- bundle

signal_loader()

- DataFrameLoader

SignalData(DataSet)

- pipeline.data.Column
- pipeline.domain

schedule_function()

schedule_function()

rebalance()

- place orders

record_vars()

- record

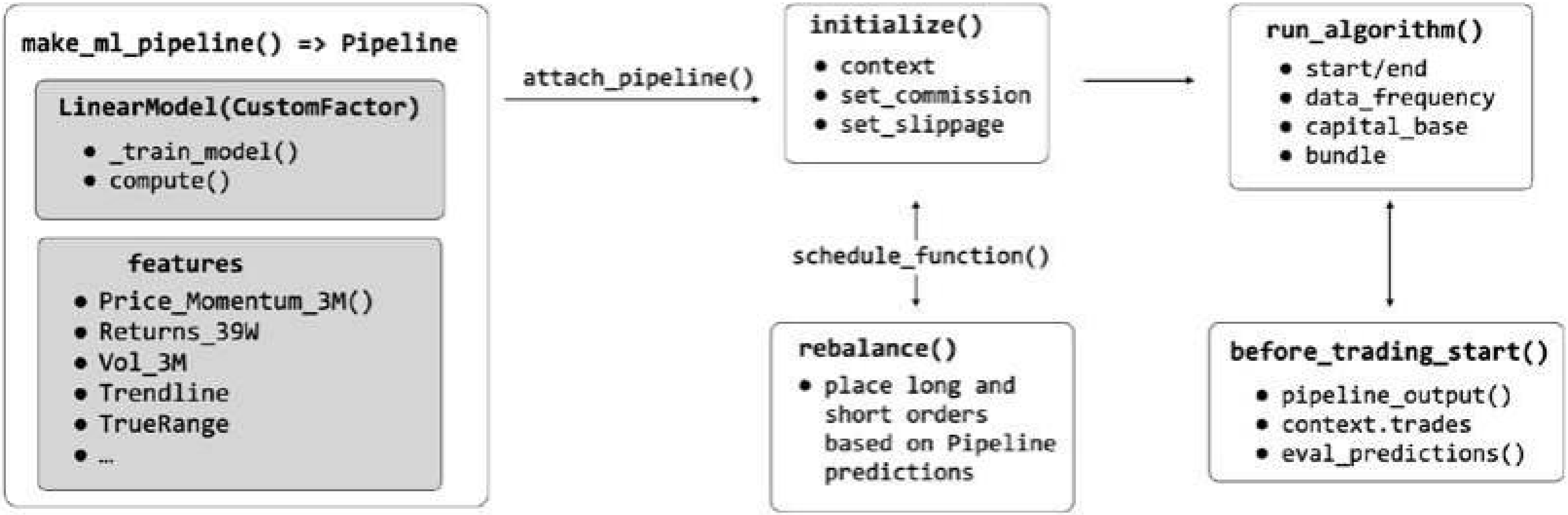
before_trading_start()

- pipeline_output()
- context.trades

ML signal backtest using Zipline's Pipeline API

# 5 zipline - scalable back testing by Quantopian

## How to train a model during the backtest



make_ml_pipeline() => Pipeline

LinearModel(CustomFactor)

- _train_model()
- compute()

features

- Price_Momentum_3M()
- Returns_39W
- Vol_3M
- Trendline
- TrueRange
- …

attach_pipeline()

initialize()

- context
- set_commission
- set_slippage

run_algorithm()

- start/end
- data_frequency
- capital_base
- bundle

schedule_function()

rebalance()

- place long and short orders based on Pipeline predictions

before_trading_start()

- pipeline_output()
- context.trades
- eval_predictions()

Flowchart of Zipline backtest with model training