ML for Crypto Markets
From Data to Decisions
Day 4 of 5

Prof. Jörg Osterrieder

MSc Seminar: Digital Finance

Spring 2026

Where We Are

Days 1–3 Recap

- Crypto pricing and market structure
- DeFi mechanics and AMM math
- Game theory of consensus and fees
- MEV, EIP-1559, tokenomics

Today's Roadmap

1. The ML pipeline for financial prediction
2. Feature engineering for crypto
3. Model selection and evaluation
4. SHAP explainability
5. Backtesting pitfalls and reality checks

The Backtest That Fooled Wall Street

Two Quant Funds, Two Outcomes

Fund A: Backtested strategy shows 200% annual return on crypto. Team raises $50M. Launches live.

Fund B (Renaissance Technologies): Medallion Fund averages 66%/year before fees over 30 years. Most successful quant fund in history.

One year later:

- Fund A: -40%. Investors pull out. Fund closes.
- Fund B: Another profitable year.

The Question

Both used ML. Both had PhDs. What did Fund A get wrong?

Why 90% of Backtested Strategies Fail Live

Common sins:

1. **Lookahead bias:** using future data in feature construction
2. **Survivorship bias:** only testing on tokens that survived
3. **Data snooping:** testing 1,000 strategies, reporting the best
4. **Ignoring costs:** slippage, spread, market impact
5. **Overfitting:** model memorizes noise, not signal

The backtest–live gap:

- Backtest: zero market impact, instant fills, no slippage
- Reality: your trade *moves the market*
- Crypto: wide spreads (20–50 bps on altcoins), thin order books, high volatility
- A 200% backtest becomes -40% after costs and regime changes

The Crystal Ball Salesman

[Cartoon Placeholder]

"The Crystal Ball Salesman"

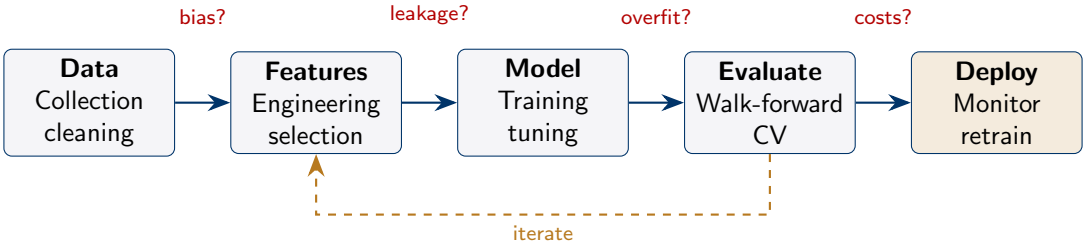
A trader at a market stall buys an ML model.
Fine print on the receipt: *"Backtested only. Past performance is literally all there is."*

Today: how to build ML pipelines that don't fool you.

Today: The Complete ML Pipeline Done Right

1. The ML Pipeline
2. Model Selection & Evaluation
3. Explainability: SHAP
4. Backtesting Pitfalls
5. Hands-On: Crypto Prediction Pipeline

The ML Pipeline for Crypto Prediction



Each stage has its own failure modes. Today we build the pipeline end-to-end, with emphasis on what goes wrong.

- Every arrow is a potential source of data leakage
- The feedback loop is where overfitting hides
- Deployment is where reality humbles your model

Feature Engineering: Four Categories

1. Technical
Returns, volume, momentum, RSI, Bollinger, MACD

2. Microstructure
Bid-ask spread, Amihud illiquidity, order flow imbalance

3. On-Chain
Active addresses, exchange flows, miner/validator revenue

4. Sentiment
News tone, social media volume, Fear & Greed Index

Crypto is unique: categories 3 and 4 are not available for traditional equities. On-chain data is a genuine informational edge.

Technical Features

Standard return-based features:

$$r_t = \ln(P_t/P_{t-1}) \quad \text{log return} \tag{1}$$

$$\sigma_{t,k} = \sqrt{\frac{1}{k}\sum_{i=0}^{k-1}(r_{t-i} - \bar{r})^2} \quad \text{rolling volatility } (k\text{-day}) \tag{2}$$

$$\text{RSI}_t = 100 - \frac{100}{1 + \text{AvgGain}_k/\text{AvgLoss}_k} \quad \text{Relative Strength Index} \tag{3}$$

$$\text{Mom}_{t,k} = P_t/P_{t-k} - 1 \quad k\text{-day momentum} \tag{4}$$

Volume features:

- $V_t/\bar{V}_{20}$: volume ratio (relative to 20-day average)
- $\text{OBV}_t = \text{OBV}_{t-1} + \text{sign}(r_t) \cdot V_t$: on-balance volume

Warning: All features must be computed using *only past data* at each point. No peeking at the future!

Microstructure Features

Liquidity and order flow:

$$\text{Spread}_t = \frac{\text{Ask}_t - \text{Bid}_t}{\text{Mid}_t} \qquad \text{relative bid-ask spread} \tag{5}$$

$$\text{Amihud}_t = \frac{|r_t|}{V_t \cdot P_t} \qquad \text{Amihud illiquidity ratio} \tag{6}$$

$$\text{OFI}_t = \frac{V_t^{\text{buy}} - V_t^{\text{sell}}}{V_t^{\text{buy}} + V_t^{\text{sell}}} \qquad \text{order flow imbalance} \tag{7}$$

Why these matter for crypto:

- Amihud illiquidity: high $\Rightarrow$ price moves easily, predicts higher future volatility [3]
- Order flow imbalance: strong buy pressure $\Rightarrow$ short-term positive returns
- Spread widens before large moves (informed trading signal)

On-Chain Features

Unique to crypto: blockchain data is public and real-time.

Activity metrics:

- Active addresses (daily unique)
- Transaction count and total value
- New address creation rate
- Smart contract interactions

Exchange flows:

- Net exchange inflow/outflow
- Large transfers ("whale alerts")
- Exchange reserve changes

Miner/Validator metrics:

- Hash rate (PoW) / staking ratio (PoS)
- Miner revenue and outflows
- MEV extracted per block

Signal Example

Large exchange outflows (coins leaving exchanges) historically precede price increases: holders move to cold storage = reduced sell pressure.

Sentiment Features

Data sources:

- **News:** CryptoCompare, GDELT, LexisNexis
- **Social media:** Twitter/X, Reddit (r/cryptocurrency), Telegram groups
- **Search trends:** Google Trends for "Bitcoin"
- **Aggregated indices:** Fear & Greed Index (0–100)

Processing:

- NLP sentiment scores (-1 to $+1$)
- Volume of mentions (buzz)
- Sentiment *change* (not level) predicts returns

[Cartoon]

"Feature Buffet"

Decision trees lined up at an all-you-can-eat buffet of features. Some plates are labeled "noise."

Classification vs. Regression

Classification (predict direction):

- Target: $y_t = \begin{cases} 1 & \text{if } r_{t+1} > 0 \\ 0 & \text{if } r_{t+1} \leq 0 \end{cases}$
- Simpler, more robust
- Natural for long/short signals
- Metrics: accuracy, precision, recall, F1

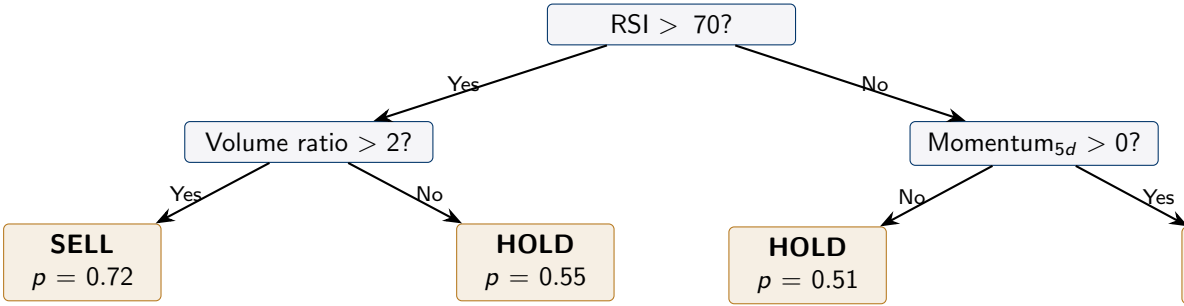
Regression (predict magnitude):

- Target: $y_t = r_{t+1}$
- Harder—must predict both direction and size
- Useful for position sizing
- Metrics: RMSE, MAE, R^2

MSc Recommendation

Start with classification. If your model can't beat 52% directional accuracy out-of-sample, regression won't help.

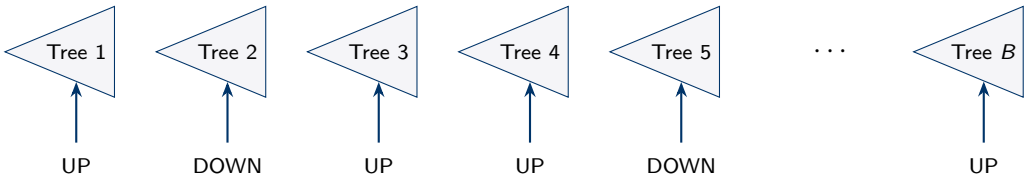
Decision Trees for Crypto Prediction



Pros: interpretable, captures non-linear interactions, no scaling needed.
Cons: high variance, prone to overfitting, unstable (small data change $\Rightarrow$ different tree).

Random Forests: Ensemble Wisdom

Idea: Train B trees on bootstrap samples, each using a random subset of features. Average their predictions.



Majority vote: UP (e.g., 65 of 100 trees)

Variance reduction: $\mathrm{Var}(\bar{f}) = \frac{\sigma^2}{B}(1-\rho) + \rho\sigma^2$, where ρ is the average pairwise correlation between trees. Feature subsampling reduces ρ.

Gradient Boosting and XGBoost

Key difference from Random Forest: trees are built *sequentially*, each correcting the errors of the previous ensemble.

Algorithm:

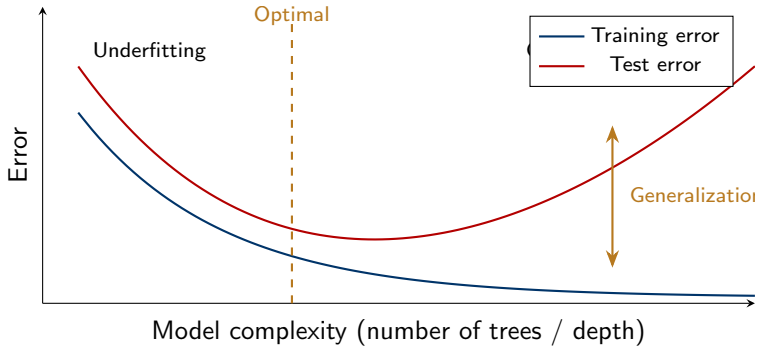
1. Start with a constant prediction: $\hat{y}^{(0)} = \bar{y}$
2. For $m = 1, \ldots, M$:
   - Compute residuals: $e_i^{(m)} = y_i - \hat{y}_i^{(m-1)}$
   - Fit a tree h_m to the residuals
   - Update: $\hat{y}^{(m)} = \hat{y}^{(m-1)} + \eta \cdot h_m(\mathbf{x})$

where $\eta \in (0, 1]$ is the learning rate (shrinkage).

XGBoost adds:

- Regularization: $\Omega(h) = \gamma T + \frac{1}{2}\lambda \sum_j w_j^2$ (penalize tree complexity)
- Second-order gradient information for better splits
- Built-in handling of missing values

The Overfitting Trap



In crypto, noise-to-signal ratio is extreme. A model with 95% training accuracy but 51% test accuracy has memorized noise, not learned patterns.

Checkpoint

Quick Question

Your model has 92% training accuracy and 53% test accuracy. Is this model useful? Why or why not?

Checkpoint

Quick Question

Your model has 92% training accuracy and 53% test accuracy. Is this model useful? Why or why not?

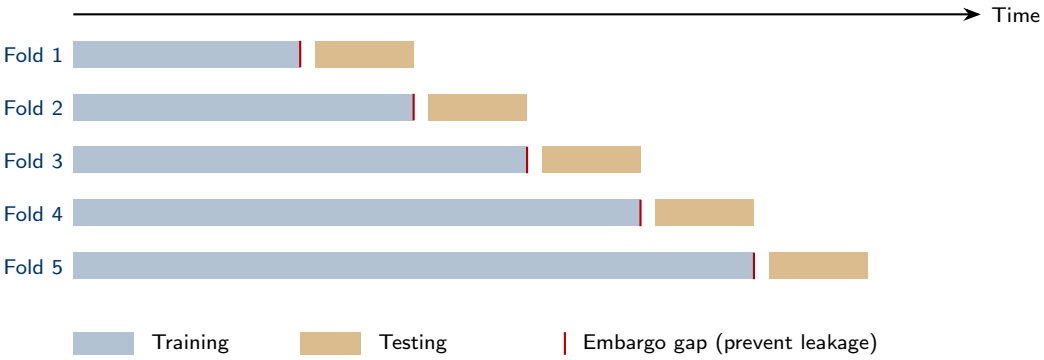
Answer

No. The 39-percentage-point gap between training and test accuracy is a classic sign of **overfitting**: the model memorized the training data (including noise) rather than learning generalizable patterns. At 53% test accuracy, it is barely better than a coin flip and would not survive transaction costs in live trading.

Time Series CV: NEVER Random Split

Cardinal Rule

In financial ML, random train/test splits create lookahead bias. Future data leaks into training. Use **expanding** or **rolling** windows only.



Walk-Forward Optimization Protocol

1. **Initial training window:** e.g., 2 years of daily data
2. **Validation window:** next 3 months
3. **Retrain:** expand (or roll) training window, advance test window
4. **Repeat:** until you reach the present

Concrete Example

- Train: Jan 2020 – Dec 2021 (730 days)
- Test: Jan 2022 – Mar 2022 (90 days)
- Retrain: Jan 2020 – Mar 2022 (expanding)
- Test: Apr 2022 – Jun 2022
- ... continue until Dec 2024

Aggregate all out-of-sample predictions $\Rightarrow$ true performance estimate.

Key: Hyperparameters must be tuned *within* each training window, never on the test set.

Evaluation Metrics: Beyond Accuracy

Classification metrics:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \qquad \text{of predicted UPs, how many were right?} \tag{8}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \qquad \text{of actual UPs, how many did we catch?} \tag{9}$$

$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \qquad \text{harmonic mean} \tag{10}$$

Financial metrics (the ones that matter):

$$\text{Sharpe Ratio} = \frac{\mathbb{E}[r_{\text{strategy}}] - r_f}{\sigma_{\text{strategy}}} \cdot \sqrt{252}$$

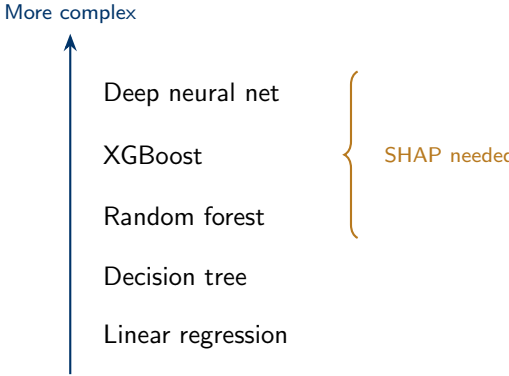
- 54% accuracy with Sharpe 1.2 $>$ 60% accuracy with Sharpe 0.5
- **Must include transaction costs** in Sharpe calculation
- AUC-ROC useful for comparing models, but Sharpe decides if you trade

The Black-Box Problem

Why we need explainability:

1. **Regulatory:** MiFID II, EU AI Act require explanation of automated decisions
2. **Trust:** portfolio managers won't trade signals they don't understand
3. **Debugging:** is the model learning real patterns or data artifacts?
4. **Risk management:** what drives extreme predictions?

Model interpretability spectrum:



SHAP: Shapley Additive Explanations [ ]

From cooperative game theory to ML:

Each feature i receives a **Shapley value** measuring its contribution to the prediction:

$$\phi_i = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!\,(n - |S| - 1)!}{n!} \Big[\underbrace{f(S \cup \{i\})}_{\text{with feature } i} - \underbrace{f(S)}_{\text{without feature } i} \Big]$$

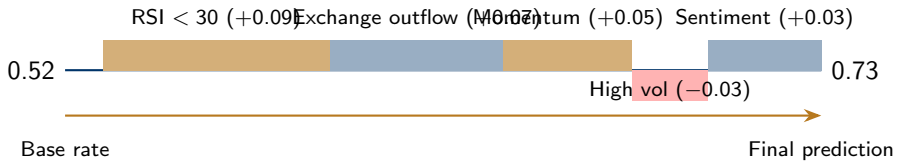
- $N = \{1, \ldots, n\}$: set of all features
- S: every possible subset excluding feature i
- $f(S)$: model prediction using only features in S
- We average the marginal contribution of feature i over **all possible orderings**

Properties: efficiency ($\sum_i \phi_i = f(x) - f(\emptyset)$), symmetry, linearity, null player.

SHAP Intuition: Allocating Credit Fairly

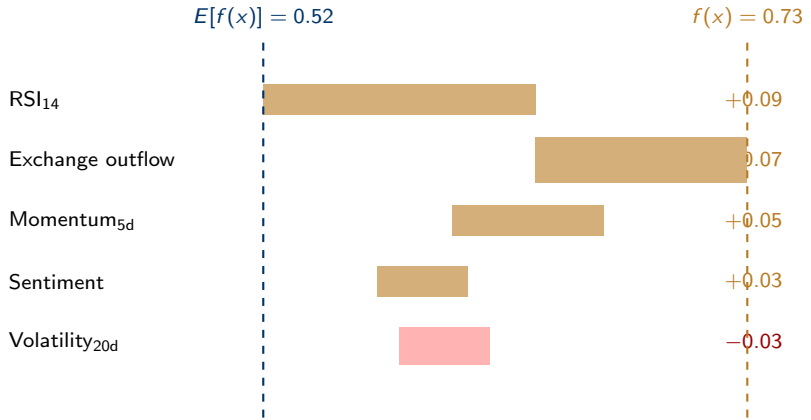
Example: Model predicts BTC will go UP with probability 0.73.
Base rate (average prediction): 0.52.

Question: Which features pushed the prediction from 0.52 to 0.73?



Each bar is a SHAP value ϕ_i. They sum to the difference: $0.73 - 0.52 = 0.21$.

SHAP Waterfall Plot (Conceptual)



Each bar shows how much a feature pushes the prediction away from the base rate. Gold = positive contribution; red = negative.

Global Explanations: Beeswarm and Dependence Plots

Beeswarm plot:

- One dot per observation per feature
- x-axis: SHAP value (impact on prediction)
- Color: feature value (blue=low, red=high)
- Shows which features matter *globally*
- Example: high RSI (red dots) consistently pushes predictions down

Dependence plot:

- x-axis: feature value
- y-axis: SHAP value for that feature
- Reveals non-linear relationships
- Example: SHAP of RSI is near zero for RSI $\in [30, 70]$ but strongly negative for RSI > 80 (overbought)
- Color by an interaction feature

Practical use: After training, generate SHAP plots for each prediction. If the top feature is "data index" or "timestamp," you have a data leakage bug.

Three Deadly Biases [ ]

1. **Lookahead bias:**
   - Using data not available at prediction time
   - Example: normalizing returns using the full sample mean
   - Fix: all transformations must use rolling/expanding windows
2. **Survivorship bias:**
   - Testing only on tokens that still exist today
   - Example: training on top-50 coins (2024 list)—ignores LUNA, FTT, etc.
   - Fix: include delisted/failed tokens in your dataset
3. **Data snooping (multiple testing):**
   - Testing 1,000 strategies, reporting the best one
   - With 1,000 trials at 5% significance: expect 50 "significant" results by chance
   - Fix: Bonferroni correction, or better: hold-out a final test set never touched during development

Transaction Costs: The Strategy Killer

Components of trading cost in crypto:

Cost Component	Typical Range	Impact
Exchange fee	5–30 bps	Fixed per trade
Bid-ask spread	5–50 bps	Varies by token/venue
Slippage	10–100 bps	Scales with trade size
Market impact	20–200 bps	Permanent price change
Funding rate (perps)	1–10 bps/8h	Continuous cost for shorts

Example: A strategy that trades daily on an altcoin:

- Round-trip cost: $\approx$60–200 bps
- 252 trading days $\times$ 100 bps $=$ 252% annual drag
- Your backtest must overcome this just to break even

Rule of thumb: Always include $\geq$30 bps round-trip cost in backtests. 50 bps for altcoins.

Reality Check: What's Achievable?

Red flags (likely overfit):

- Sharpe > 3 on daily crypto
- Accuracy $> 60\%$ out-of-sample
- Drawdown $< 10\%$ over $3+$ years
- Works on *all* tokens
- No losing months

Realistic performance:

- Sharpe 0.5–1.5 after costs
- Accuracy 52–56%
- Drawdowns of 20–40% expected
- Works on liquid tokens (BTC, ETH)
- 40% of months are losing

The Deflation Test

If your backtest Sharpe is 2.0, apply haircuts:

- Transaction costs: -0.3 to -0.5
- Data snooping (tried multiple strategies): $\div 1.5$
- Regime change: $\times 0.7$
- **Expected live Sharpe:** ≈ 0.6–0.9

Hands-On Session

Build a Crypto Prediction Pipeline

Python notebooks on course platform

Step 1: Load Data and Create Features

Data

```
import pandas as pd
import numpy as np

# Load BTC daily OHLCV (2019-2024)
df = pd.read_csv('btc_daily.csv', parse_dates=['date'])
df = df.set_index('date').sort_index()
```

Create features (all using only past data):

1. Log returns: $r_t = \ln(P_t/P_{t-1})$
2. Rolling volatility: 5, 10, 20-day windows
3. RSI (14-day), momentum (5, 10, 20-day)
4. Volume ratio: $V_t/\bar{V}_{20}$
5. Amihud illiquidity: $|r_t|/V_t$

Target: $y_t = \mathbb{1}_{r_{t+1}>0}$ (binary: next-day UP/DOWN)

Step 2: Feature Selection via Correlation

Remove redundant features

```
corr_matrix = features.corr().abs()
upper = corr_matrix.where(
    np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))
to_drop = [c for c in upper.columns
           if any(upper[c] > 0.9)]
features = features.drop(columns=to_drop)
```

Guidelines:

- Drop features with $|\rho| > 0.9$ (multicollinearity)
- Check for target leakage: no feature should correlate > 0.3 with the target (suspiciously predictive)
- Keep 10–20 features for Random Forest, 5–15 for XGBoost

Step 3: Rolling Cross-Validation

Walk-forward CV setup

```
from sklearn.model_selection import TimeSeriesSplit

tscv = TimeSeriesSplit(n_splits=5, gap=5)  # 5-day embargo
results = []

for train_idx, test_idx in tscv.split(X):
    X_train, X_test = X.iloc[train_idx], X.iloc[test_idx]
    y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]
    # ... train and evaluate
```

Key settings:

- gap=5: embargo of 5 days between train and test
- Minimum training size: 500 observations
- Store predictions for each fold $\Rightarrow$ aggregate later

Step 4: Train Random Forest and XGBoost

Model training

```
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier

rf = RandomForestClassifier(
    n_estimators=200, max_depth=5,
    min_samples_leaf=20, random_state=42)

xgb = XGBClassifier(
    n_estimators=200, max_depth=3,
    learning_rate=0.05, reg_lambda=1.0,
    subsample=0.8, colsample_bytree=0.8)
```

Deliberate conservatism:

- Shallow trees (`max_depth` 3–5) to avoid overfitting

Step 5: Evaluate and Compare

For each model, compute out-of-sample:

Classification metrics:

- Accuracy, precision, recall, F1
- AUC-ROC curve
- Confusion matrix

Financial metrics:

- Annualized return
- Sharpe ratio (with 30 bps cost)
- Maximum drawdown
- Win rate by month

Quick Sharpe calculation

```
# Convert predictions to positions
position = 2 * predictions - 1  # +1 long, -1 short
strategy_ret = position * actual_returns - 0.003  # costs
sharpe = strategy_ret.mean() / strategy_ret.std() * np.sqrt(252)
```

Step 6: SHAP Analysis

Generate SHAP values

```
import shap

explainer = shap.TreeExplainer(xgb_model)
shap_values = explainer.shap_values(X_test)

# Summary plot (beeswarm)
shap.summary_plot(shap_values, X_test)

# Waterfall for single prediction
shap.waterfall_plot(
    shap.Explanation(values=shap_values[0],
                     base_values=explainer.expected_value,
                     data=X_test.iloc[0],
                     feature_names=X_test.columns))
```

Convert Predictions to Strategy Returns

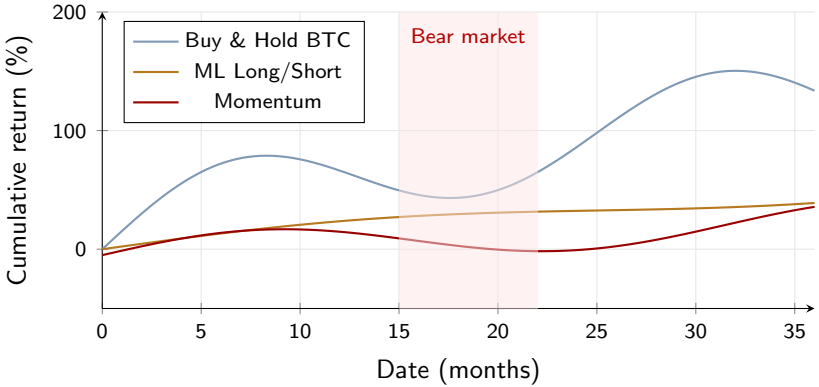
Three strategies to compare:

1. **ML long/short:** go long when model predicts UP, short when DOWN
2. **Buy-and-hold:** always long BTC
3. **Momentum:** long if 10-day return > 0, else short

Metric	ML L/S	Buy & Hold	Momentum
Annual return	18%	45%	12%
Sharpe (after costs)	0.82	0.71	0.45
Max drawdown	-28%	-73%	-35%
Accuracy	54.1%	—	52.3%

ML strategy has lower return than buy-and-hold but much better risk-adjusted performance and smaller drawdowns.

Cumulative Returns: ML vs. Benchmarks



The ML strategy's value shows during drawdowns: hedging reduces losses while buy-and-hold suffers the full crash.

Discussion: Would You Trade This Strategy?

Your ML model achieves:

- 54% directional accuracy (out-of-sample)
- Sharpe ratio: 0.8 after transaction costs
- Maximum drawdown: -28%
- Losing months: 42%

Arguments FOR trading it:

- Risk-adjusted return beats buy-and-hold
- Consistent out-of-sample over 5 folds
- SHAP features are economically interpretable
- Drawdown is manageable

Arguments AGAINST:

- Sharpe < 1 suggests weak edge
- 3 years of data may not cover all regimes
- Model could degrade as markets evolve
- Ethical: profiting from detected patterns?

Discussion: What additional tests would you run before committing capital?

Ethics: When ML Meets Markets

1. **Information asymmetry:** ML traders detect patterns retail investors cannot. Is this fair?
2. **Market stability:** if everyone uses similar ML models, do they amplify crashes?
3. **Front-running:** ML-detected patterns often exploit slower traders. How is this different from MEV (Day 3)?
4. **Responsibility:** when an algorithm causes losses, who is accountable?

No Easy Answers

Traditional finance has rules (insider trading laws, best execution). Crypto largely does not. As ML becomes standard, should crypto adopt similar regulations?

Day 4 Summary

1. **ML pipeline:** data $\rightarrow$ features $\rightarrow$ model $\rightarrow$ evaluate $\rightarrow$ deploy (every stage has pitfalls)
2. **Features:** technical, microstructure, on-chain, sentiment; crypto-specific data is an edge
3. **Models:** Random Forest and XGBoost are workhorses; shallow trees, strong regularization
4. **Evaluation:** walk-forward CV only; Sharpe ratio $>$ accuracy
5. **SHAP:** explains predictions, catches data leakage, builds trust
6. **Reality:** 90% of backtested strategies fail live; include costs, expect Sharpe < 1.5

Day 5 Preview: Risk, Regulation, and the Future

Fat-tailed risk measures, portfolio construction with crypto, DeFi systemic risk, MiCA regulation, and the road ahead.

References I

[1] Marcos Lopez de Prado. *Advances in Financial Machine Learning*. Wiley, 2018.

[2] Scott M. Lundberg and Su-In Lee. "A Unified Approach to Interpreting Model Predictions". In: *Advances in Neural Information Processing Systems 30 (NeurIPS 2017)*. 2017, pp. 4765–4774.

[3] Igor Makarov and Antoinette Schoar. "Trading and Arbitrage in Cryptocurrency Markets". In: *Journal of Financial Economics* 135.2 (2020), pp. 293–319.