TECHNICAL REPORT

AlphaGlyph: A Backtesting Laboratory That Measures Its Own Credibility

Walk-forward simulation, Deflated Sharpe, and combinatorial purged cross-validation — and a 72% probability that its own best strategy is overfit.

Daniel Lichtenberger · August 2026

alphaglyph.org · Source on GitHub

A backtesting laboratory built to measure how much its own results should be believed.

Summary

Any strategy that looks good in a backtest looks good for one of two reasons: it found something real, or it was the luckiest of the many variants that were tried. The backtest itself cannot tell you which. That ambiguity is not a detail — it is the central problem of quantitative finance, and most tooling ignores it entirely.

AlphaGlyph is a backtesting engine with the second question built in. It simulates strategies on real historical prices, then subjects every result to the statistical tests that quantify selection bias: the Probabilistic and Deflated Sharpe Ratios, Combinatorially-Symmetric Cross-Validation for the Probability of Backtest Overfitting, Fama-French factor decomposition, and a forward-only prediction ledger that cannot be retroactively edited.

Pointed at its own strategy grid, the system reports a **72% probability of backtest overfitting** (SPY, run 2026-08-01) — the in-sample winner lands in the bottom half out-of-sample more often than not. That is the headline result, and it is a negative one.

- 15 REST endpoints, 195 automated tests in CI
- Indicators implemented from first principles (SMA, EMA, RSI, MACD, ADX, Bollinger)
- Live at alphaglyph.org

1. The problem: a backtest is a contaminated experiment

A backtest looks like an experiment, but it usually violates the rule that makes experiments meaningful: the hypothesis is chosen after seeing the data.

The mechanism is simple. Suppose you test 42 variants of a moving-average strategy and keep the one with the best Sharpe ratio. Even if none of the 42 has any genuine edge, the *maximum* of 42 noisy estimates is systematically above zero. You will always find a winner. Its in-sample Sharpe is not an estimate of its true performance — it is an estimate of true performance *plus a selection premium*, and nothing in the backtest separates the two.

The standard response is to report the winner's Sharpe ratio and move on. AlphaGlyph treats that as the bug, and the rest of this report describes the instrumentation built to measure it.

---

2. The demonstration: the Data-Mining Lab

Before arguing that selection bias is severe, the system demonstrates it on strategies that are *known* to have no edge.

`backend/datamining.py` generates N random long/flat timing strategies on a real price series. Each goes long on a random ~50% of days. By construction, none of them contains any information about future prices — every one is a coin flip.

The lab then backtests all N, keeps the single best Sharpe ratio, and reports two verdicts side by side:

Verdict	What it assumes	Typical result
Naive PSR	the winner was the only strategy tried	often "significant"
Deflated Sharpe	N strategies were tried; corrects for it	"not significant"

The gap between those two numbers is the entire lesson. The winning strategy is 100% luck — this is guaranteed by how it was generated — yet naive statistics frequently call it significant. The expected maximum Sharpe of N noise strategies is drawn on the histogram, and the winner sits exactly where chance predicts.

This is the argument the rest of the system is built to act on: **a Sharpe ratio is meaningless without knowing how many were tried to find it.**

---

3. Probabilistic and Deflated Sharpe Ratios

Implemented in `backend/stats.py`, following López de Prado (2014).

3.1 PSR

The naive Sharpe ratio assumes returns are normal and the sample is large. Financial returns are neither — they are skewed and fat-tailed, which makes the Sharpe estimate noisier than it appears. PSR returns the probability that the *true* Sharpe exceeds a benchmark, correcting for both:

```
PSR(SR*) = Φ[ (SR̂ − SR*) · √(T−1) / √(1 − γ₃·SR̂ + (γ₄−1)/4 · SR̂²) ]
```

where γ_3 is skewness and γ_4 raw kurtosis. Since SciPy reports *excess* kurtosis, the implementation uses `(excess + 2)/4`, which is the same quantity. All Sharpe values are held in daily units internally and converted only at the boundary — a detail that is easy to get wrong and silently changes the answer.

3.2 DSR

The Deflated Sharpe Ratio is PSR with the benchmark raised to the Sharpe you would expect from the luckiest of N independent trials:

```
SR*_annual = E[max Z_k] · √(252 / T)
```

`E[max Z_k]` uses the standard Euler-Mascheroni approximation to the expected maximum of N unit normals.

The `√(252/T)` scaling matters and is frequently omitted in third-party implementations. The expected maximum is expressed in units of the *sampling distribution* of the Sharpe estimate, whose width depends on how many observations you have. Without rescaling for T, the deflation is applied at the wrong magnitude and the test is miscalibrated. With `n_strategies = 1`, DSR reduces exactly to PSR against a zero benchmark, which is the correct degenerate case and is asserted in the tests.

---

4. CPCV and the Probability of Backtest Overfitting

`backend/cpcv.py`. Method from Bailey, Borwein, López de Prado & Zhu (2017), via Combinatorially-Symmetric Cross-Validation.

4.1 The strategy grid

The grid is deliberately made of strategies people actually tune, not strawmen:

- **Trend-following** — long when fast SMA > slow SMA, else flat. Fast ∈ {5, 10, 15, 20, 30, 40}, slow ∈ {50, 80, 100, 150, 200}, keeping only fast < slow → 30 variants.
- **Mean-reversion** — stateful RSI entry/exit: enter below a buy threshold, exit above 55. Period ∈ {7, 14, 21}, threshold ∈ {25, 30, 35, 40} → 12 variants.

42 variants total. Positions are shifted one day, so a decision made on the close of day t is held over $t \to t+1$. There is no look-ahead. The first 205 rows are dropped to clear the longest indicator's warm-up.

4.2 The procedure

1. Build a T×N matrix of per-period returns, one column per variant.

2. Split the T rows into S contiguous groups (S even, clamped to 6–12; C(12,6) = 924 combinations is the computational ceiling).
3. For every way of choosing S/2 groups as in-sample:
    - pick the variant with the best in-sample Sharpe, `n*`
    - find `n*`'s **rank** among all variants out-of-sample
    - ω = rank / (N+1); logit $\lambda = \ln(\omega / (1-\omega))$
4. **PBO** = the fraction of splits where $\lambda \leq 0$ — i.e. the in-sample champion lands in the bottom half out-of-sample.

A PBO near 0.5 means selection told you nothing at all.

4.3 Purging and embargo

Financial returns are serially correlated, so an in-sample block sitting immediately adjacent to an out-of-sample block can leak its edge straight across the seam. The implementation trims `embargo` rows from each end of every in-sample block — and critically, **only** from the in-sample side. Trimming both would discard out-of-sample data that is legitimately available and bias the result optimistically.

4.4 Degenerate columns

A variant that stays flat across an entire block has zero variance and therefore an undefined Sharpe. These are assigned `-inf` so they can never be selected as the in-sample champion, and are filtered out before the degradation regression, medians and scatter. Silently letting a no-trade outcome count as a performance figure would distort every downstream statistic — this is the kind of edge case that turns a correct method into a wrong number.

4.5 What it also reports

- **Performance degradation** — OLS of OOS Sharpe on IS Sharpe. A slope well below 1 means the in-sample edge mostly evaporates.
- **Probability of OOS loss** — how often the selected strategy actually loses money.
- Logit histogram, IS-vs-OOS scatter, and a split diagram for the interface.

---

5. Factor decomposition

A strategy can post real returns and still have no alpha, if it is merely capturing well-known risk premia that a passive factor ETF sells for a few basis points. `fama_french_decomposition` regresses daily excess returns on the three Fama-French factors:

```
R_p − R_f = α + β_mkt(R_m − R_f) + β_smb·SMB + β_hml·HML + ε
```

and reports Jensen's alpha annualised, factor loadings, R^2, and per-coefficient t-statistics. Factors are the real daily series from Ken French's data library, not a proxy.

An engineering note worth recording: this test kept intermittently reporting "n/a" in production. Two causes, both environmental rather than statistical — Ken French had moved the file from `/data_library/` to `/ftp/`, and Dartmouth blocks the default `python-requests` user agent from some cloud IPs. The fix was a browser-like UA, both URLs tried in order, retries with backoff, and a disk cache of the last good copy as an offline fallback. A statistical test is only as reliable as its data pipeline, and this one was failing for reasons that had nothing to do with statistics.

---

6. Calibration and the forward ledger

Two components address a claim the backtest cannot make: whether the forecaster's *probabilities* are honest.

`calibration.py` — the model emits statements like "~58% chance of an up move." A forecaster is well-calibrated when, across every day it said 60%, the market actually rose about 60% of the time. Predictions are binned into deciles to produce a reliability diagram and scored with Brier score and log loss, each against a "always predict the base rate" baseline so the number means something.

`ledger.py` — the harder test. Every directional call is written to an append-only table *at the moment it is made*, with price, p_up, direction and horizon. Once the horizon elapses, the row is graded against what actually happened. **Nothing is ever edited or deleted.** The resulting hit rate and Brier score are a genuine forward record that cannot be retro-fitted.

Calibration measures the model on historical out-of-sample data already in hand; the ledger measures it on the future, one real day at a time. Grading is lazy — a plain GET matures any elapsed rows, so the record stays current without a cron job.

---

7. Results

> **Reproduce before citing.** CPCV runs live against a rolling window of fetched price data, so PBO is a function of the window it was run on. Every figure below carries the date it was produced.

CPCV on SPY — 42 variants, 10 groups, 252 splits, 1,050 rows after warm-up, embargo 5. Run 2026-08-01:

Reproducible from committed data: the window this run used (SPY closes, 2021-08-02 → 2026-07-31) is committed at `docs/data/SPY_close_2021-08-02_2026-07-31.csv`, and `python tools/pbo_snapshot.py` recomputes every figure in this table from it, through the same `cpcv_from_prices` the live endpoint calls. Re-downloading the same dates would not be the same data: Yahoo revises adjusted closes as later dividends are paid.

Metric	Value	Reading
Probability of backtest overfitting	**72%**	selection is worse than a coin flip
Median in-sample Sharpe	**1.26**	looks like a good strategy
Median out-of-sample Sharpe	**0.65**	~52% of the edge survives
IS→OOS degradation slope	**−0.59**	see below
Probability of OOS loss	**5%**	it rarely *loses* money
ML forecaster test AUC	**≈0.51**	a coin flip on next-week direction

The window-dependence is itself a result

An earlier run of the identical procedure reported **PBO 63%** with **~38%** Sharpe retention. This run reports **72%** and **~52%**. Same code, same grid, same ticker — a different rolling window.

That instability is not a defect in the measurement; it is the measurement working. A single PBO number is a statement about one window, and anyone quoting it as a fixed property of a strategy has made the same category error the metric exists to catch. This report therefore dates every figure.

The degradation slope is negative

The regression of out-of-sample Sharpe on in-sample Sharpe has a slope of **−0.59**. A slope near 1 would mean in-sample performance predicts out-of-sample performance; a slope near 0 would mean it predicts nothing.

A *negative* slope means the relationship is inverted — across these splits, the variants that looked best in-sample tended to do **worse** out-of-sample. Selecting on in-sample Sharpe was actively worse than selecting at random.

What the 5% OOS-loss rate means

Worth stating precisely, because it is easy to over-claim. The selected strategy loses money out-of-sample only about 5% of the time. The failure is not that these strategies are ruinous — it is that **choosing among them on in-sample performance does not identify the good ones.** The money is mostly fine; the selection procedure is what's worthless.

The one clean positive, with its conditions: a dip-buying strategy on AAPL/MSFT from 2021-01-01 to 2024-01-01 returned **+21.6%** in ranging markets against SPY's **+5.6%** over the same days — an excess of +16.1 percentage points. Both conditions are load-bearing. The regime one is the obvious one; the window is the other: extending the run to 2025-01-01 shrinks the ranging-market excess to **+3.6** points, because 2024's ranging days rewarded simply holding the index. It is not a claim that the strategy works in general. (Rerun on 2026-10-27; the July figures, +21.8% vs +5.6%, differed only by data revisions.)

The AUC of 0.51 is worth stating plainly. The machine-learning forecaster, which took the most work to build, cannot predict next-week direction. It is reported at 0.51 rather than quietly dropped, because a system whose entire premise is honest measurement does not get to hide its own null results.

---

8. Limitations

- **PBO is window-dependent.** A single PBO figure describes one data window and one grid. It is not a universal constant.
- **The grid is only two strategy families.** Trend and mean-reversion are representative of what retail practitioners tune, but a wider grid could produce a different number.
- **CSCV assumes reasonably comparable variants.** With a grid mixing wildly different exposure profiles, rank-based comparison is less meaningful.
- **Transaction costs are modelled, not measured.** Real fills, slippage and market impact are not simulated.
- **The forward ledger is young.** A track record is only as good as its length, and this one accrues one day at a time.

---

9. Reproducing

```
git clone https://github.com/Danny-397/Alphaglyph
cd Alphaglyph/backend
pip install -r requirements.txt
pytest                      # 195 tests
python app.py               # http://localhost:5000
```

Architecture: the browser holds all UI state and the backend is a stateless compute layer; the ML model is trained offline and shipped as a portable ONNX artifact. Data resolves through a three-stage cascade — Tiingo, then yfinance, then Stooq — so a single upstream outage does not take the system down.

---

10. References

1. Bailey, D. H., Borwein, J. M., López de Prado, M., & Zhu, Q. J. *The Probability of Backtest Overfitting.* **Journal of Computational Finance, 20(4), 39–70 (2017).** Circulated as an SSRN working paper from 2014; the 2014 date commonly cited refers to that preprint, not the journal publication.
2. Bailey, D. H., & López de Prado, M. (2014). *The Deflated Sharpe Ratio: Correcting for Selection Bias, Backtest Overfitting and Non-Normality.* **Journal of Portfolio Management, 40(5), 94–107.**
3. Fama, E. F., & French, K. R. (1993). *Common Risk Factors in the Returns on Stocks and Bonds.* **Journal of Financial Economics, 33(1), 3–56.**

Citations verified against publication records on 2026-08-01.