A Reproducible Pairs-Trading Backtest in a Mixed ETF–Equity Universe: Implementation Decisions, Failure Modes, and Lessons Learned

Tan Le
Department of Mathematics and Computer Science, Knox College
Advisor: Andrew Leahy
tle@knox.edu

January 3, 2026

Abstract

This report documents a Colab-native, Drive-first pairs-trading backtest pipeline designed for reproducibility and defensible measurement under free-data constraints. We evaluate a mixed U.S. universe (sector ETFs, major ETFs, and large-cap equities) using walk-forward windows (12-month formation, 6-month trading, 1-month step), a distance/cointegration-style candidate process, z-score entry/exit rules, and conservative corporate-action gating.

On the final locked run (2011-01-05 to 2025-12-05), the baseline configuration delivers an annualized geometric return of 2.96% at 7.97% volatility (annualized Sharpe 0.41) with a maximum drawdown of 32.2%, underperforming a simple buy-and-hold benchmark (e.g., SPY) over the same period. Performance is regime-dependent: 2011–2019 is weak (Sharpe -0.14), while 2020–2025 is materially stronger (annualized geometric return 9.68%, Sharpe 1.07). A CSCV analysis over 620 parameter scenarios yields an estimated Probability of Backtest Overfitting (PBO) of approximately 0.28, suggesting non-trivial overfitting risk.

The central contribution is not a claim of alpha but an auditable research workflow: explicit data contracts separating signal vs execution, run-isolated artifact generation, and a clear record of failure modes and the fixes that improved validity.

Keywords: pairs trading, statistical arbitrage, sector ETFs, reproducibility, backtesting, walk-forward, lookahead bias, survivorship bias, corporate actions, overfitting control.

Contents

1 Executive Summary

This project builds a reproducible, bias-aware pairs-trading backtest pipeline in Google Colab for a broad, liquid universe that includes sector ETFs, other major ETFs, and S&P 500 constituents (Module 2). Although pairs trading is a classic strategy family
[3, 5], the core objective here is not to claim a new alpha but to produce a backtest that is *believable*: timing-consistent, contract-driven (signal vs execution separation), and robust to common "silent failure" modes that can inflate results.

What the system does end-to-end. For each walk-forward window, the pipeline (i) downloads and standardizes raw OHLC and corporate actions (Module 3), (ii) audits data coverage and constructs formation/trading windows with an eligibility gate computed *per window* (Module 4), (iii) generates candidate pairs via a Gatev-style distance filter (Module 6), (iv) validates candidates using formation-only diagnostics including Engle–Granger cointegration [2] and spread/OU-style diagnostics (Modules 5–7), (v) generates z-score-based position signals with explicit exit codes (Module 8), (vi) simulates fills and daily PnL using open/close execution with conservative data-quality gating for corporate actions (Module 9), and finally (vii) consolidates all per-window outputs into a single run artifact set under `RUN_DIR` (Module 9E). Paper-ready diagnostics are computed post-engine by reading consolidated outputs (Module 10). A full-engine optimizer with CSCV/PBO is included as a separate, resume-safe module (Module 11; planned to run after a final clean engine run).

What makes it defensible. Three design pillars keep the system scientifically and operationally defensible:

- **Separation of concerns (signal vs execution):** the notebook treats log-price econometrics as a research layer and open/close accounting as an execution layer, preventing accidental "vendor magic" from leaking into fills or corporate-action handling.
- **Bias controls via walk-forward + eligibility gating:** formation-only estimation and per-window eligibility gates reduce lookahead leakage; this specifically addresses a real failure encountered during development (see §3 and later postmortems).
- **Reproducibility engineering:** Drive-first persistence, run isolation, atomic file writes, checkpoint/resume, and anti-mixing rules reduce the chance that long Colab runs silently corrupt outputs or mix runs.

What failed early and what changed. Early development relied heavily on AI coding agents, which quickly produced more code than could be audited, leading to uncontrolled complexity and hard-to-trace bugs. The project was rebuilt in a single Colab notebook with explicit module boundaries and minimal, testable contracts. A major bias bug was discovered in the initial universe

validation approach: global integrity checks over a 15-year range removed later-listed assets (a survivorship/lookahead-style distortion). The fix was to compute eligibility within each window, not globally. Metric drift was also corrected: the system now defines Sharpe using daily returns with annualization, not PnL-to-vol ratios.

2 Introduction

2.1 Motivation

Pairs trading sits in a deceptive sweet spot: conceptually simple (trade mean reversion of a spread) yet implementation-fragile. Small mistakes in timeline alignment, window construction, corporate-action treatment, or evaluation methodology can create backtests that look impressive but are not reproducible or defensible. In practice, the hardest part is often not the econometrics; it is preventing subtle leakage and run bookkeeping errors from manufacturing false confidence.

This report treats the pairs-trading idea as a baseline [3] and focuses on building a pipeline that (i) separates research signals from execution accounting, (ii) enforces walk-forward conventions that reduce leakage, (iii) uses return-based performance definitions (including annualized Sharpe [4]), and (iv) records a complete, run-isolated artifact set suitable for auditing and re-analysis.

2.2 Scope and claims

This document is a *report paper* about implementation decisions, failure modes, and lessons learned. It makes the following claims and non-claims explicit:

- **Claim (engineering):** the notebook implements a Drive-safe, run-isolated pipeline with atomic writes and checkpoint/resume, producing a consistent set of consolidated artifacts under `RUN_DIR`.
- **Claim (methodological hygiene):** the pipeline adopts walk-forward formation/trading separation and per-window eligibility gating to reduce lookahead and survivorship-style leakage created by global filtering.
- **Claim (metrics):** the system reports return-based performance metrics, including an annualized Sharpe computed from daily returns, and avoids labeling PnL-to-vol ratios as Sharpe.
- **Non-claim (alpha):** this report does not claim a new trading strategy or universal profitability. The goal is believable measurement and reproducible evaluation.

2.3 Contributions

- A Colab-native, Drive-first backtest pipeline that is robust to runtime resets via checkpoint/resume and atomic writes.

- A clear data contract separating the *signal layer* (log-price panel) from the *execution layer* (open/close + corporate actions flags), reducing silent leakage between research convenience and accounting.
- A walk-forward window builder with an eligibility gate computed per window (formation/trading), explicitly addressing a discovered survivorship/lookahead-style failure.
- A conservative corporate-actions philosophy: dividends/splits are treated as data-quality events for gating and audit rather than as alpha.
- A paper-ready artifact set (per-window parts and consolidated outputs) enabling downstream diagnostics, plots, and optimizer analysis without rerunning the engine.

3 Notebook Map and Pipeline Overview (Module-by-Module)

The notebook is organized into eleven modules that mirror a production-style research workflow: environment configuration, universe definition, data ingestion/standardization, data validation and windowing, econometrics and candidate generation, validation, signal generation, execution and artifact streaming, consolidation, diagnostics, and finally a full-engine optimizer with overfitting risk estimation. Modules 9–11 are the computational core; earlier modules exist to prevent expensive computation from being performed on broken or biased inputs.

3.1 Module 1: Environment Setup & Research Configuration

Module 1 initializes Colab execution, imports the quantitative Python stack, and defines a single global `CONFIG` dictionary used throughout the pipeline. The notebook is designed for long Colab sessions, so this module also includes Drive detection logic: if Google Drive is present, `CONFIG["run_root"]` and `CONFIG["data_dir"]` are configured to write directly to Drive; otherwise, the notebook falls back to local storage. For reproducibility, the run date and dataset end date are explicitly controlled (in the provided notebook snapshot, `end_date` is set to `2025-12-25`).

3.2 Module 2: Define the Trading Universe (Stocks + ETFs, 2010+ Focus)

Module 2 constructs `full_tickers`, a broad, liquid universe intended to be practical for 2010–present research. The universe includes: (i) sector ETFs (and other major ETF families: broad market, style/factors, bonds, commodities, countries), and (ii) S&P 500 stocks fetched from a public constituents dataset with a fallback list to keep the notebook runnable.

This module also states the major conceptual limitation up front: because the S&P 500 list is based on current constituents, the universe is survivorship-biased for historical studies. The notebook accepts this limitation as a constraint of free data, and the report treats it explicitly as a limitation rather than as an ignored assumption.

3.3 Module 3: Data Ingestion & Standardization (Signals vs Execution)

Module 3 builds the two price layers used throughout the pipeline:

- **Execution layer:** raw OHLC panels and corporate actions, standardized into canonical `EXEC_*` panels (at minimum `EXEC_OPEN`, `EXEC_CLOSE`, plus `EXEC_DIV` and `EXEC_SPLIT` with safe default fills). This layer is the sole input to the backtest engine's accounting.
- **Signal layer:** a log-price panel `LOGP_SIGNAL` (aliased as `prices_df`) used for cointegration, OU-style diagnostics, and z-score signal generation. Log prices improve numerical stability for spread modeling and reduce scale sensitivity.

A key engineering feature in this module is a set of sanity checks that detect yfinance "short fallback" behavior (e.g., returning a short default range instead of the requested span). The module also standardizes indices (sorting, de-duplication, reindexing to `MASTER_CAL`) to prevent silent misalignment across panels.

UNCERTAIN note (explicit). In the current notebook snapshot, `LOGP_SIGNAL` is computed from `CLOSE_DF` with the comment that Yahoo close is "already split-adjusted." This assumption is not universally reliable across free-data conventions; therefore it remains an explicit `UNCERTAIN` item to be discussed later in the limitations section. The pipeline mitigates the risk operationally by treating split events as data-quality flags in the execution layer and applying conservative gating policies.

3.4 Module 4: Data Validation, Universe Audit, and Walk-Forward Eligibility Gate

Module 4 is the quality-control checkpoint before expensive pair selection and backtesting. It enforces a data contract on the signal panel (monotonic datetime index, finite log prices, and basic integrity checks), produces a universe audit table (coverage, start/end per ticker), and builds the walk-forward windows used for out-of-sample evaluation.

In the notebook snapshot, the walk-forward configuration is:

- formation length: 12 months,
- trading length: 6 months,
- step size: 1 month sliding,
- minimum coverage: 90% (window-specific).

Critical design decision: window-specific eligibility. Rather than filtering the universe once using global (2010–present) completeness, the module computes eligibility per formation window.

This directly addresses a major discovered failure mode: global integrity filtering can remove later-listed assets (IPO or later ETF launches), causing a survivorship/lookahead-style distortion where only early-start "survivors" remain. By gating eligibility within each window, the pipeline ensures that the tradable universe evolves over time in a way consistent with the information available at that point in history.

3.5 Module 5: Econometrics Engine (OLS-only, diagnostics-focused)

Module 5 defines the econometric primitives used in validation and signal construction:

- OLS hedge ratio estimation on formation data, producing `beta_ols` and `mu_ols` for a consistent spread definition.
- Engle–Granger cointegration testing via standard tools [2].
- Spread diagnostics such as zero-crossing rate to proxy "tradability" and avoid dead spreads. []
- OU/AR(1)-style mapping used to estimate mean-reversion speed (half-life) and a stationary scale for z-scoring.

A deliberate simplification in this notebook is to avoid strategy-specific threshold selection logic (e.g., Bertram-style threshold optimization). Instead, the econometrics module focuses on stable estimation and diagnostics that can be used inside a walk-forward system without overfitting the selection rule to a single sample.

3.6 Module 6: Distance Filter (Gatev-style) — Coarse Candidate Generation

Module 6 reduces the combinatorial explosion of possible pairs. With hundreds of symbols, the number of unordered pairs is on the order of $\binom{N}{2}$, making full validation across all pairs in all windows impractical. Inspired by Gatev-style screening [3], the notebook applies a coarse filter based on the sum of squared differences (SSD) between normalized price paths:

- Normalize each series by its initial value in the formation window.
- Compute pairwise SSD distances efficiently.
- Select the top-k closest pairs (preferred over hard thresholds for stability).

The output is a candidate set that is small enough to feed into formation-only validation tests without spending most compute on obviously unrelated pairs.

3.7 Module 7: Pair Validation Across Walk-Forward Windows (Anti-Clone + EG + OU)

Module 7 transforms coarse candidates into a validated set `VALIDATED_DF` by applying formation-only diagnostics within each walk-forward window. The validation logic is designed to prevent "cheap"

backtest inflation by requiring that a pair demonstrate statistical and practical properties before it is traded out-of-sample.

The notebook supports (and documents) multiple gates, including:

- cointegration screening (Engle–Granger p-value),
- optional ADF-style stationarity checks on the spread,
- half-life bounds to avoid extremely slow or noisy dynamics,
- minimum activity diagnostics such as zero-crossing rate,
- a clone-like pair detector/audit (high correlation and tiny tracking error) to reduce redundant exposure.

An important implementation choice noted in the notebook is that ranking and selection decisions are performed within each window using formation-only information, avoiding contamination from future trading outcomes.

3.8 Module 8: Signal Generation Core (Z-score → Position Signals)

Module 8 produces trading signals without computing PnL. For each validated pair-window row, it constructs the spread series, standardizes it into a z-score, and converts that z-score into a discrete position signal:

$$\texttt{pos\_sig} \in \{+1, 0, -1\},$$

with an explicit `exit_code_sig` that records the exit reason (mean-cross, band exit, stop, time stop, or missing-data forced exit).

The notebook's signal parameters are centralized in a `BT` configuration dictionary (entry z-threshold, exit mode and threshold, stop threshold, time-stop bars, execution lag bars, and an entry embargo). The module uses a strict timing convention: signals are generated with an explicit lag (`exec_lag_bars`), and execution occurs at the next open in the engine. Missing z-score values trigger conservative behavior (forced exit if currently in a position), preventing silent "carry" across data holes.

3.9 Module 9: Execution + Runner + Consolidation (Cells 9A–9E)

Module 9 is the heart of realized PnL generation and reproducible artifact production. It converts signals into executions using an open/close model and writes all intermediate and final outputs under a run-specific directory.

9A: Run context and directories. Cell 9A defines `RUN_ROOT`, `RUN_ID`, and `RUN_DIR`, creates part directories (results/daily/trades/events/episodes), and enforces an anti-mixing rule: old and new parts must not silently interleave under the same run root. This is critical in Colab, where partial runs are common.

9B: ExecPack builder. Cell 9B constructs an `exec_pack` aligned to the master calendar with arrays for open/close (and optional volume), plus corporate-actions panels and flags: `missing_open`, `missing_close`, `corp_event`, `split_event`, and an aggregate `dq_event`. Dividend-related gating is implemented using a conservative, no-lookahead proxy (dividend relative to prior close), and split events are explicitly surfaced as flags rather than hidden via implicit price adjustments.

9C: Backtest engine (pure). Cell 9C consumes a signal pack and exec pack and simulates fills, daily PnL, and event logs under explicit policies (costs/slippage assumptions and corporate-action gating policy). A key design rule is that the engine itself is "pure" with respect to I/O: it returns dataframes/rows; file persistence is handled by the runner.

9D: Runner with checkpoint/resume. Cell 9D loops over `VALIDATED_DF`, streams outputs to parquet parts, maintains checkpoints for resume safety, and writes commit markers so consolidation can cap reading at committed results. This directly addresses long-run failure modes (disconnects and partial writes).

9E: Consolidator and portfolio daily. Cell 9E reads parts (capped by committed results), deduplicates and concatenates outputs into consolidated parquet files, builds trade episodes, and produces `portfolio_daily.parquet`. Portfolio returns follow : daily portfolio return is defined as portfolio PnL divided by portfolio gross exposure, with a conservative no-exposure rule to keep the return series well-defined.

3.10 Module 10: Diagnostics & Reporting (Paper-Ready)

Module 10 reads consolidated outputs and generates analysis artifacts intended for inclusion in the report: equity and drawdown curves, annualized return/volatility/Sharpe, rolling metrics, distribution diagnostics (histogram/QQ/ACF), calendar/regime breakdowns, exposure and activity diagnostics (gross exposure and number of active windows), trade/episode diagnostics (win-rate, holding period, exit reason counts), and data-quality gate statistics (corporate actions and split flags).

Importantly, Module 10 is post-engine: it does not change positions or fills. Its role is to convert consolidated data into paper-ready plots and tables while enforcing the report's metric definitions (return-based series, annualization consistency).

3.11 Module 11: Optimizer + CSCV/PBO

Module 11 implements a full-engine parameter search and an overfitting-risk assessment layer. Rather than optimizing on a proxy metric disconnected from execution details, this module reuses the exact same core components as the main run: `make_signal_pack` (Module 8), `z_to_pos_and_exitcode` (Module 8), `make_exec_pack` (Cell 9B), and `backtest_engine` (Cell 9C). This choice ensures that

"optimized" configurations are evaluated under the same corporate-action gating, missing-data rules, and accounting conventions as the baseline.

The optimizer enumerates scenarios over entry/exit/stop/time-stop parameters (with seeded randomness for reproducibility), evaluates each scenario across the walk-forward set, and writes scenario-level outputs under `RUN_DIR`. To reduce the risk of reporting a spurious best configuration produced by multiple testing, Module 11 is designed to compute cross-validation-style estimates of overfitting risk (CSCV) and the probability of backtest overfitting (PBO) following the project's stated methodology plan [1]. In this report, Module 11 is treated as a reproducible artifact generator whose final quantitative conclusions will be inserted after the final clean engine run and diagnostics refresh.

4 Data and Assumptions

4.1 Data source and panels

All market data in this project is sourced from Yahoo Finance via `yfinance`, using daily bars (`interval="1d"`). The downloader is configured in **raw mode** (`auto_adjust=False`) and requests corporate actions (`actions=True`) so that dividends and splits are available as explicit panels rather than being silently folded into prices.

The notebook's source-of-truth download step (Cell 3) produces the following raw panels in memory:

- `OPEN_DF, HIGH_DF, LOW_DF, CLOSE_DF` (daily OHLC),
- `ADJ_CLOSE_DF` (vendor-adjusted close as provided by Yahoo),
- `VOLUME_DF`,
- `DIV_DF` (dividends),
- `SPLIT_DF` (stock splits),
- `MASTER_CAL` (the master trading calendar, taken from the close index).

The configured research horizon is:

$$\texttt{start\_date} = \texttt{2010-01-01}, \qquad \texttt{end\_date} = \texttt{2025-12-25}.$$

To reduce Colab fragility, the pipeline supports Drive-first persistence: raw panels and the signal panel can be saved as parquet under `CONFIG["data_dir"]`. The downloader is also chunked by ticker lists (with retries) to reduce yfinance failure rates for large universes.

4.2 Signal prices vs execution prices (Critical data contract)

A central design principle of this notebook is an explicit *two-layer contract*:

Execution layer (accounting truth). The execution layer is standardized immediately after download (Cell 3B) into canonical panels:

$$\texttt{EXEC\_OPEN} \leftarrow \texttt{OPEN\_DF}, \qquad \texttt{EXEC\_CLOSE} \leftarrow \texttt{CLOSE\_DF}.$$

Corporate actions are aligned to the execution calendar and filled safely:

- `EXEC_DIV` is a dense panel aligned to `EXEC_CLOSE` with missing values filled by 0.0.
- `EXEC_SPLIT` is a dense panel aligned to `EXEC_CLOSE` with missing values filled by 0.0 (later interpreted as "no split" by the execution logic).

These `EXEC_*` panels are the only inputs used for fills, PnL accounting, and corporate-action flags in the backtest engine (Module 9).

Signal layer (research convenience). The signal layer is a log-price panel used for econometrics and z-score signal generation (Cell 3C):

$$\texttt{LOGP\_SIGNAL} = \log(\texttt{CLOSE\_SPLIT\_ADJ}),$$

and the notebook aliases:

$$\texttt{prices\_df} \equiv \texttt{LOGP\_SIGNAL}.$$

This separation ensures the research layer can standardize and transform prices (e.g., log transform) without contaminating the accounting layer. It also makes it easier to reason about timing conventions: signals are generated from `prices_df`, while execution is performed using `EXEC_OPEN`/`EXEC_CLOSE`.

Explicit `UNCERTAIN` assumption: split adjustment in signal prices. In the current notebook snapshot, `LOGP_SIGNAL` is computed directly from `CLOSE_DF` with the comment that Yahoo close is already split-adjusted. This assumption is not guaranteed across all tickers and vendor conventions. Therefore, the report treats this as `UNCERTAIN`: results should be interpreted under the possibility that the signal layer contains split discontinuities for some instruments. The pipeline mitigates the operational risk by (i) surfacing split events as explicit flags in the execution layer and (ii) applying conservative data-quality gating policies around corporate actions in the engine (Module 9), rather than trusting implicit vendor adjustments.

4.3 Free-data limitations and mitigations

This project intentionally accepts the constraints of free data and compensates through conservative engineering and explicit limitations.

- **Corporate-action timing noise (`UNCERTAIN`).** Dividend series may not perfectly match the economically relevant ex-date timing for all tickers. The engine treats corporate actions as

data-quality events and uses conservative gating rather than attempting to model cashflows precisely.

- **Split-adjust conventions (`UNCERTAIN`).** Vendor split adjustments may be inconsistent across instruments. The notebook explicitly logs split events and can guard against "double counting" by keeping split handling policy-driven in the execution layer.
- **Missingness and heterogeneous instruments.** ETFs and single stocks differ in microstructure and corporate actions; missing opens/closes exist and are surfaced via flags rather than silently imputed in the engine.
- **Universe bias (major).** Any universe built from current ticker lists (e.g., current S&P 500 constituents) is survivorship-biased for historical research. The notebook acknowledges this up front and treats it as a limitation rather than an ignored assumption.

5 Walk-Forward Design and Bias Controls

5.1 Windowing: formation vs trading

The evaluation framework is explicitly walk-forward. For each window, the pipeline splits the timeline into:

- a **formation** period used for estimation (pair selection, hedge ratio estimation, spread diagnostics),
- a **trading** period used for out-of-sample execution and PnL measurement.

In the current configuration (Module 4), the rolling windows are:

- formation length: **12 months**,
- trading length: **6 months**,
- slide step: **1 month**,
- minimum coverage inside formation: **90%**.

A subtle but important engineering detail is that windows are **trading-day aligned**. The window builder maps calendar anchors into the master trading calendar (`prices_df.index`) using "first trading day on/after date" and "next trading day after date" logic. This avoids creating windows whose boundaries fall on non-trading days and prevents accidental off-by-one shifts that can produce lookahead leakage.

5.2 Eligibility gate (the critical fix)

The pipeline uses an explicit *eligibility gate* computed **within each formation window**. For a formation-window log-price panel $P_{t,j}$ (ticker j), the coverage inside that window is:

$$\text{cov}_j \;=\; \frac{1}{T}\sum_{t=1}^{T} \mathbf{1}\{P_{t,j} \text{ is observed}\},$$

and a ticker is eligible if:

$$\text{cov}_j \geq 0.90.$$

This design directly addresses a major failure mode discovered during development: **global** integrity filtering over a long horizon (e.g., 15 years) can remove assets that begin trading later (IPOs, later-launched ETFs). That behavior effectively conditions on future availability and creates a survivorship/lookahead-style distortion. Computing eligibility per window ensures the tradable universe evolves over time as it would have historically, and prevents "future existence" from becoming a hidden filter.

In addition to window eligibility, Module 4 produces a **universe audit table** that records each ticker's coverage fraction, number of observed days, and first/last valid date, and writes it to CSV. This audit is used as an operational guardrail: it makes data problems visible early, before expensive downstream computations.

5.3 Anti-lookahead conventions (timing and information sets)

The notebook enforces several conventions intended to keep information sets honest:

- **Formation-only inference.** Hedge ratios, cointegration tests, and spread diagnostics are computed using formation data only.
- **Lagged execution.** Signal generation is designed to support explicit lagging (signal computed from the past; execution at the next open), so that trades are not triggered by information from the same bar being traded.
- **Conservative corporate-action detection.** Dividend-related gating uses a prior-close normalization proxy (dividend relative to close$(t-1)$) to avoid relying on same-day "knowledge" in a way that could inadvertently introduce lookahead.

These conventions do not guarantee profitability; they guarantee that the backtest is harder to "accidentally cheat."

6 Econometrics and Pair Construction

6.1 OLS hedge ratio and residual definition

Pairs trading in this notebook is built around a consistent spread definition estimated by ordinary least squares (OLS) in the formation window (Module 5). Let $p_t^{(y)}$ and $p_t^{(x)}$ denote log prices for legs y and x in formation. The OLS regression estimates:

$$p_t^{(y)} = \alpha + \beta\, p_t^{(x)} + \varepsilon_t, \tag{1}$$

$$\varepsilon_t = p_t^{(y)} - \alpha - \beta\, p_t^{(x)}, \tag{2}$$

where β is the hedge ratio and ε_t is the spread/residual series used for diagnostics and z-scoring.

A strict alignment contract is used throughout: the two series are aligned on timestamps, invalid values are removed, and minimum sample-size checks are enforced before any inference is attempted. This is deliberately conservative: it prevents "partial overlap" bugs where one leg carries missing values that silently distort regression and spread statistics.

6.2 Candidate screening: distance filter (Gatev-style)

To reduce the combinatorial search space, the notebook applies a Gatev-style distance filter (Module 6). For each formation window, log prices are normalized by the initial value:

$$p_t^{\text{norm}} \;=\; p_t - p_{t_0},$$

which is equivalent to cumulative log returns from the window start. Pair similarity is measured by the sum of squared deviations (SSD), implemented as squared Euclidean distance (`metric="sqeuclidean"`):

$$D(y,x) \;=\; \sum_t (p_t^{(y),\text{norm}} - p_t^{(x),\text{norm}})^2.$$

Rather than relying on a fragile absolute threshold, the notebook selects a fixed number of closest candidates per window (top-k). In the current configuration, `top_k=3000` candidates are retained per window (with checkpointing and caching utilities to avoid rerunning the expensive distance computation).

6.3 Validation tests and tradability gates (formation-only)

Candidate pairs are validated per window (Module 7) using a set of strict, formation-only gates designed to screen for statistical plausibility and practical tradability:

Cointegration and stationarity. The pipeline runs an Engle–Granger cointegration test and requires a small p-value (configured as `coint_alpha=0.01`). It also runs a stationarity test on the

residual (ADF) and requires `adf_alpha=0.05`. These are used as conservative gates rather than as guarantees of profitability.

OU/AR(1) mapping and half-life bounds. The residual series is mapped to an AR(1)-style dynamic, yielding a mean-reversion speed proxy and a half-life estimate. The notebook enforces:

$$\texttt{half\_life\_min} = 2 \leq \texttt{half\_life} \leq 40 = \texttt{half\_life\_max},$$

to avoid spreads that revert too quickly to trade reliably or too slowly to be useful in a 6-month trading window.

Zero-crossing activity. The notebook computes a zero-crossing count (spread crossing its mean) and annualizes it by 252 to form an activity proxy. The configured minimum is `zc_rate_min=8.0` crossings/year, filtering out "dead" spreads.

Scale gate. A minimum stationary sigma gate (`sigma_min=0.025`) ensures that z-scores are not generated from essentially flat residuals that would be dominated by noise and numerical artifacts.

Hedge ratio sign. The validation requires a positive hedge ratio (`require_beta_positive=True`) to avoid pathological cases and simplify interpretation under this notebook's long/short spread conventions.

6.4 Clone filtering (redundant exposure control)

A practical failure mode in large universes is that many "different" pairs are effectively clones of each other (highly correlated legs with tiny tracking error), leading to redundant bets and misleading backtest diversification. Module 7 includes an explicit clone filter (`drop_clones=True`) that flags pairs as clones when:

- return correlation exceeds `corr_clone=0.9990`,
- annualized tracking-error proxy is below `te_ann_clone=0.010`,
- spread standard deviation is below `spread_std_clone=0.0030`.

Clones are dropped and logged to a separate CSV for auditability.

6.5 Validated-pair output schema

After passing gates, each pair-window produces a validated record (Module 7) that includes identifiers and formation-window statistics needed downstream:

$(y, x, \texttt{window\_id}, \texttt{formation\_start}, \texttt{formation\_end}, \texttt{trading\_start}, \texttt{trading\_end}, \beta_{\text{ols}}, \mu_{\text{ols}}, \texttt{coint\_pval}, \texttt{adf\_pval}$

This `VALIDATED_DF` is the contract input to signal generation (Module 8) and the execution runner (Module 9).

7 Execution, Runner, and Consolidation (Module 9: Cells 9A–9E)

Module 9 is where the project transitions from research signals to realized accounting. It is also where most silent failure modes tend to hide: run mixing, partial writes, schema drift, corporate-action timing ambiguity, missing-price days, and inconsistent portfolio aggregation. For this reason, the notebook treats Module 9 as a contract-driven system with explicit policies and reproducibility guardrails. The module is intentionally split into five cells (9A–9E) that separate *run context*, *execution data preparation*, *pure simulation*, *streamed persistence*, and *consolidation/portfolio aggregation*.

7.1 Cell 9A: Run context and reproducible directories

Cell 9A defines the run boundary. A run is not simply "whatever files happen to exist"; it is a fully specified, isolated output root under which every artifact is written.

Run identifiers and directory layout. The notebook constructs:

- `RUN_ROOT`: a stable parent directory (Drive-first) under which runs are created.
- `RUN_ID`: a unique identifier for the run, typically including a timestamp and configuration tag (e.g., strategy mode, data end date, engine version).
- `RUN_DIR`: `os.path.join(RUN_ROOT, RUN_ID)`, the sole root for all files generated by that run.

Within `RUN_DIR`, the notebook creates a set of parts directories and metadata files. Although exact naming is implementation-specific, the run structure follows the same principle:

- $*_P ARTS\_DIR directories : append-only parquet parts for streamed writing (e.g., results parts, daily parts, trade$

- `progress/ and errors/: status logs and error reports suitable for resume debugging.`
- `snapshot.json (or equivalent): a compact record of config, universe hash, window settings, and engine version used for the run.`

Explicit resume-only rule and anti-mixing. Long Colab runs are frequently interrupted. A naive implementation that "just appends" to existing files can silently mix artifacts from different code versions or different parameter settings. Cell 9A therefore adopts an explicit rule:

> A run directory is either (i) new and empty, or (ii) resumed intentionally using its checkpoint state. The notebook never silently resumes into a directory whose state is ambiguous.

Operationally, this is enforced by:

- checking existence of `RUN_DIR` and refusing to write into it unless explicit resume is enabled;
- recording a run fingerprint (engine version/hash + config summary) and validating it on resume;
- writing commit markers (see §7.4) so consolidation reads only coherent, committed parts.

Why this matters scientifically. Run mixing is a stealthy form of "research misconduct by accident": it can create results that no longer correspond to any single codebase or configuration. By forcing run isolation and explicit resume, Cell 9A makes each reported result traceable to a single engine contract and configuration.

7.2 Cell 9B: ExecPack builder (alignment + flags)

Cell 9B converts raw execution panels into a compact per-pair execution bundle called an `ExecPack`. The key idea is that the backtest engine should not have to guess how to align data, how to treat missing opens/closes, or how to interpret corporate actions; those rules are made explicit upstream.

Inputs to `make_exec_pack`. For a given pair-window row (legs y and x, and a trading index `idx`), the builder reads:

$\texttt{EXEC\_OPEN}[y]$, $\texttt{EXEC\_CLOSE}[y]$, $\texttt{EXEC\_DIV}[y]$, $\texttt{EXEC\_SPLIT}[y]$ and $\texttt{EXEC\_OPEN}[x]$, $\texttt{EXEC\_CLOSE}[x]$, $\texttt{EXEC\_DIV}[x]$, $\texttt{EXE}$

restricted and aligned to the trading calendar `idx`.

Calendar alignment and dense arrays. A recurring source of bugs is partial overlap: one leg has a missing bar on a day where the other trades. The ExecPack builder therefore:

- reindexes each series to the master index `idx`,
- returns dense *numpy arrays* for speed and deterministic engine behavior,
- marks missingness explicitly rather than silently forward-filling execution prices.

This is deliberately conservative. Forward-filling execution prices can introduce subtle, unrealistic fills and can also embed lookahead in the presence of gaps.

Missingness flags. The builder creates boolean arrays:

$$\texttt{missing\_open\_y}[t],\ \texttt{missing\_close\_y}[t],\ \texttt{missing\_open\_x}[t],\ \texttt{missing\_close\_x}[t],$$

and also pair-level flags such as:

$\texttt{missing\_open}[t] = \texttt{missing\_open\_y}[t] \vee \texttt{missing\_open\_x}[t]$, $\texttt{missing\_close}[t] = \texttt{missing\_close\_y}[t] \vee \texttt{missing\_cl}$

The engine consumes these flags to decide whether to skip the day, force-flat, or carry state, depending on policy.

Corporate actions as data-quality events. Corporate actions are treated as *data-quality gates*, not alpha. The ExecPack builder surfaces two core event types:

- **Dividend event:** The builder defines a dividend-related corporate action flag using a conservative proxy that avoids lookahead:

$$\texttt{corp\_event}[t] \leftarrow \mathbf{1}\left\{\frac{\texttt{DIV}[t]}{\texttt{CLOSE}[t-1]} \geq \tau_{\text{div}}\right\}$$

  for either leg (with safe handling at $t = 0$). This proxies the magnitude of a dividend relative to the prior close and avoids relying on future information. The threshold τ_{div} is set conservatively to catch meaningful corporate-action distortions rather than to "game" results.

- **Split event:** The builder flags a split when the split ratio indicates a non-trivial split occurrence. In the notebook's philosophy, split events are particularly dangerous because free-data "split adjustment" may be inconsistent across tickers, and because a split can cause large apparent jumps in unadjusted series.

`split_event`, `dq_event`, and policy-driven enforcement. Cell 9B produces:

- `split_event`[t] for any detected split activity,
- `corp_event`[t] for dividend-style corporate actions,
- `dq_event`[t] as an aggregate guardrail:

$$\texttt{dq\_event}[t] = \texttt{split\_event}[t] \vee \texttt{corp\_event}[t] \vee \texttt{missing\_open}[t] \vee \texttt{missing\_close}[t]$$

  (or a closely related definition depending on the configured severity).

The key is that `dq_event` does not itself decide trading behavior; it informs the engine, which enforces a policy (e.g., force-flat at prior open, skip day, etc.). This separation prevents accidental changes in one place from silently changing behavior elsewhere.

ExecPack schema (conceptual). The builder returns an `ExecPack` with fields conceptually equivalent to:

- `idx` (DatetimeIndex for trading days),
- `open_y`, `close_y`, `open_x`, `close_x` arrays,
- `div_y`, `div_x`, `split_y`, `split_x` arrays,

- flags: `missing_open`, `missing_close`, `corp_event`, `split_event`, `dq_event`,
- `price_adjustment` metadata (e.g., `"split_adjusted"`), used only for auditability and to prevent double-counting.

7.3 Cell 9C: Backtest engine (pure, no file I/O)

Cell 9C is the simulation core. Its most important property is **purity**: it takes inputs (signal pack, exec pack, config) and returns outputs (rows/dataframes) without performing file I/O. This design makes correctness easier to test and prevents partial writes from being confused with partial simulation.

Inputs and contracts. The engine consumes:

- a `SignalPack` produced by Module 8 containing (at minimum) the trading index, z-score series, lagged position targets `pos_sig`, and exit codes,
- an `ExecPack` produced by Cell 9B with aligned open/close series and data-quality flags,
- `EXEC_CFG` (cost model, missing-data policy, corporate-action policy, slippage model parameters).

State variables. At each time t, the engine maintains:

- **position state**: whether the pair is currently in a long-spread, short-spread, or flat position,
- **holdings/exposure**: gross deployed notional on each leg (used for portfolio return),
- **entry information**: entry date, entry spread/z, and any counters for time-stop.

Timing diagram (daily). The engine adheres to the following daily timing convention:

1. Read `pos_sig[t]` and `exit_code_sig[t]` (already lagged by construction).
2. Read execution prices `open_y[t]`, `open_x[t]` and `close_y[t]`, `close_x[t]`.
3. Apply data-quality gating policy if `dq_event[t]` or missingness flags are triggered.
4. Execute entry/exit/flip at **open** according to policy and signal state.
5. Mark-to-market and realize daily PnL using **open-to-close** movement and any costs/slippage.
6. Emit daily records and event logs.

This explicit sequencing avoids the common mistake of using same-day close information to trigger a trade at the same day's open.

Transaction cost and slippage model (explicit and simple). The engine includes a parameterized cost model. Even if simplified, it is explicit and consistently applied:

- per-trade cost proportional to notional traded (e.g., basis points),
- optional slippage applied to fills (e.g., widen entry/exit prices),
- costs applied symmetrically on both legs and on both entry and exit.

The guiding principle is: *do not hide costs in ad-hoc adjustments*. If the cost model is simplistic, it should be documented as such and treated as a limitation, not as an implicit free lunch.

Corporate-action gating policy. When `corp_event[t]` or `split_event[t]` is triggered, the engine applies a policy selected by `EXEC_CFG["corp_event_policy"]`. The default described in the project snapshot is a conservative `"force_flat_prev_open"`:

- If a corporate-action event is detected for day t, the engine ensures the position is flat starting from the *previous open* (or equivalently does not carry exposure into day t).
- The policy is intended as a data-quality guardrail, not a dividend-avoidance alpha.

This policy is deliberately conservative: it may reduce measured performance, but it reduces the risk that free-data conventions around dividends/splits manufacture unrealistic PnL.

Engine outputs. The engine returns three primary outputs for each pair-window:

- `summary_row`: one row with pair-window identifiers and aggregated metrics (trading days, gross exposure stats, realized PnL, return-based metrics).
- `daily_df`: a time series with daily accounting fields such as PnL, gross exposure, return contributions, and state flags.
- `events_df` and/or `fills_df`: event logs including entries/exits/forced-flats and fill-level records at open/close.

By design, these outputs can be streamed to disk without requiring the engine to know anything about directories or file naming.

7.4 Cell 9D: Runner + checkpoint

Cell 9D is the orchestration layer. It loops over validated pair-windows, calls the signal builder (Module 8), the exec pack builder (Cell 9B), and the pure engine (Cell 9C), and then persists results as append-only parquet parts with checkpoint/resume safety.

Why streamed parts instead of monolithic outputs. The validated set can contain thousands of pair-windows across many months. Writing a single monolithic file for each artifact type is fragile: a single crash can corrupt the whole output. The runner therefore uses:

- **buffered writing**: accumulate a limited batch in memory, then flush to disk,
- **parquet parts**: write each flush as a separate part file,
- **atomic writes**: flush via a temporary path and `os.replace`.

Checkpoint state. To support resume, the runner writes a checkpoint that includes at minimum:

- the index of the next `VALIDATED_DF` row to process,
- the list/count of written part files per artifact stream,
- an error ledger for rows that failed (so the run can complete while preserving failure information),
- the run fingerprint (engine version/hash + key config).

On resume, the runner reads the checkpoint, verifies fingerprint compatibility, and continues from the last committed state.

Commit markers (anti-mixing contract). A subtle but critical design is the *commit marker*. The runner writes parts in multiple streams (results/daily/trades/events). If a crash occurs after writing some parts but before writing others, the run becomes inconsistent. To prevent consolidation from reading a half-committed tail, the runner writes a commit record only after a coherent batch is fully flushed. The consolidator (Cell 9E) then reads parts *only up to the last committed point*. This ensures that:

> Consolidated outputs always correspond to a coherent prefix of the run, never to a partially written suffix.

Error handling philosophy. The runner is designed to keep going. If an individual pair-window fails (e.g., due to pathological data, singular regression, or unexpected missingness), the failure is logged to `errors/` with the window identifiers and stack trace, and the run continues. This is important for research honesty: a "perfect" backtest that quietly drops hard cases without logging is not defensible.

7.5 Cell 9E: Consolidation + portfolio daily

Cell 9E reads the streamed parquet parts and produces consolidated outputs. Consolidation is not a cosmetic step; it is a correctness step that: (i) enforces a stable schema, (ii) deduplicates rows in case of resume overlaps, (iii) constructs portfolio-level time series, and (iv) generates derived artifacts (episodes) used for diagnostics.

Inputs and cap-by-commit. The consolidator reads:

- results parts: per pair-window summary rows,
- daily parts: per pair-window daily series,
- trades parts: fill/trade rows,
- events parts: entry/exit/force-flat logs,

but critically caps reading to the last committed batch as recorded by the runner. This cap prevents run mixing and prevents consolidation from reading a partially written tail.

Concatenation and deduplication. Because resume can cause overlapping writes near the checkpoint boundary, consolidation performs deduplication keyed by stable identifiers (e.g., `run_window_id` + date + pair). Any duplicates are dropped deterministically, and the resulting consolidated frames are sorted by (window id, date) to ensure stable downstream analysis.

Episodes construction. Trade rows and event logs are used to construct *episodes*: contiguous holding periods from entry to exit for each pair-window. Episodes are useful for reporting because they enable:

- distribution of holding periods,
- win-rate by episode rather than by day,
- exit-reason counts (mean-cross vs stop vs time-stop vs force-flat),
- PnL attribution by episode type.

Portfolio aggregation . A core design decision in this project is to define portfolio returns in a capital-consistent way that is compatible with a multi-window, multi-pair system. Let $\mathrm{PnL}_{i,t}$ denote the daily PnL of pair-window i on day t, and let $\mathrm{Gross}_{i,t}$ denote the gross deployed exposure for that pair-window on that day (as defined by the engine's accounting). Portfolio aggregates are:

$$\mathrm{PnL}_t^{\mathrm{port}} = \sum_i \mathrm{PnL}_{i,t}, \tag{3}$$

$$\mathrm{Gross}_t^{\mathrm{port}} = \sum_i \mathrm{Gross}_{i,t}, \tag{4}$$

and the portfolio daily return is:

$$r_t^{\mathrm{port}} = \begin{cases} \mathrm{PnL}_t^{\mathrm{port}} / \mathrm{Gross}_t^{\mathrm{port}}, & \mathrm{Gross}_t^{\mathrm{port}} > 0, \\ 0, & \mathrm{Gross}_t^{\mathrm{port}} = 0. \end{cases} \tag{5}$$

This rule handles the no-exposure days explicitly. The consolidator applies a safe-divide guardrail (epsilon or explicit conditional) so that days with zero gross exposure do not produce NaNs that later inflate Sharpe by dropping zero-activity days.

Portfolio equity curve. The portfolio equity curve is constructed from the return series by compounding:

$$E_t = E_{t-1}\,(1 + r_t^{\text{port}}),$$

with $E_0 = 1$ by default. If the engine directly outputs an equity series (e.g., `portfolio_equity`), the consolidator preserves it and ensures it matches the compounded return series within numerical tolerance; otherwise it reconstructs the series deterministically from returns.

8 Performance Metrics (Return-Based and Defensible)

The notebook distinguishes between *cash PnL* (useful for accounting) and *returns* (required for defensible risk-adjusted metrics). In earlier iterations of the project, a common mistake appeared: using mean(PnL)/std(PnL) and labeling it "Sharpe." This report adopts a strict definition: Sharpe is computed from *returns* and annualized in a consistent way [4].

8.1 Daily return series and annualization

Let r_t denote the daily portfolio return series produced by (Equation (5)). Define the sample mean and sample standard deviation over the evaluation days:

$$\bar{r} = \frac{1}{T}\sum_{t=1}^{T} r_t, \qquad s_r = \sqrt{\frac{1}{T-1}\sum_{t=1}^{T}(r_t - \bar{r})^2}.$$

The annualized Sharpe ratio (with zero risk-free rate in the base report) is:

$$\text{SR}_{\text{ann}} = \sqrt{252} \cdot \frac{\bar{r}}{s_r}. \tag{6}$$

The report fixes the annualization factor at 252 for simplicity and comparability across results tables. If an alternative factor (e.g., computed from trading days in the index) is used, it must be declared explicitly and applied consistently across all plots and tables.

Why return-based Sharpe matters here. Because the portfolio combines many pair-windows with varying gross exposure over time, raw PnL is not on a stable scale. A PnL-based "Sharpe-like" ratio can be inflated simply by varying exposure or by dropping zero-exposure days. returns explicitly normalize by gross exposure and define $r_t = 0$ when exposure is zero, making the return series stable and preventing silent inflation.

8.2 Additional metrics

In addition to Sharpe, the report plans to compute and present the following metrics from the daily return series and equity curve:

- **Annualized return and volatility:**

$$\mu_{\text{ann}} = 252\,\bar{r}, \qquad \sigma_{\text{ann}} = \sqrt{252}\,s_r.$$

- **Maximum drawdown (MaxDD):** computed from the running peak of the equity curve E_t:

$$\text{DD}_t = 1 - \frac{E_t}{\max_{u \le t} E_u}, \qquad \text{MaxDD} = \max_t \text{DD}_t.$$

- **Calmar ratio:** $\text{Calmar} = \mu_{\text{ann}}/\text{MaxDD}$ (when $\text{MaxDD} > 0$).
- **Turnover proxy:** a daily or episode-based measure derived from gross traded notional (from fills/trade rows) divided by gross exposure, reported as average daily turnover.
- **Win-rate:** reported at the *episode* level (fraction of episodes with positive total PnL), and optionally at the day level for additional context.
- **Holding period distribution:** mean/median and percentile bands of episode durations in trading days.
- **Exit reason breakdown:** fraction of exits due to mean reversion, stop, time-stop, and data-quality gating.

The emphasis is not to maximize the number of metrics but to ensure that each metric is computed from an unambiguous series with a clear definition and consistent annualization.

9 Reproducibility Engineering (Drive-first, Atomic Writes, Run Isolation)

Reproducibility in this project is treated as an engineering requirement, not as an afterthought. The notebook is designed to run in Google Colab, where session resets, runtime disconnects, and transient I/O errors are common. Without explicit guardrails, long backtests can silently corrupt outputs or mix artifacts across runs, making results unrecoverable and un-auditable.

9.1 Why Drive-first in Colab

Colab's ephemeral filesystem is not reliable for multi-hour experiments. The notebook therefore prefers to write to Google Drive whenever available. This has three practical benefits:

- **Persistence across resets:** partial progress and artifacts survive runtime restarts.

- **Checkpoint/resume becomes meaningful:** resumes can pick up from the last checkpoint without rerunning earlier windows.
- **A stable "paper artifact bundle":** the entire `RUN_DIR` can be zipped and archived or shared with an advisor.

Drive-first is a pragmatic choice: it trades some I/O latency for a large increase in experimental reliability.

9.2 Atomic write protocol

A key rule is that any file that is intended to be read later (parquet parts, consolidated parquet, checkpoints, and manifests) is written atomically:

- write to a temporary file path (e.g., `file.tmp`),
- flush and close the file handle,
- replace the target path via `os.replace(tmp, target)`.

This prevents a class of failures where a process crashes mid-write and leaves a corrupted file that looks valid by filename but is unreadable or partially written.

9.3 Run mixing prevention

Run mixing is prevented at two layers:

- **Run isolation (Cell 9A):** each run has a unique `RUN_DIR` and explicit resume-only behavior.
- **Commit-capped consolidation (Cells 9D–9E):** the runner writes commit markers only after coherent batches are fully flushed; the consolidator reads parts only up to the last committed point.

Together, these rules ensure consolidated outputs correspond to a coherent prefix of a single run, never a partially written suffix and never a mixture of incompatible experiments.

9.4 Artifact schema and manifest

Cell 9E produces a stable consolidated artifact set under `RUN_DIR`. These files are the interface consumed by diagnostics (Module 10) and optimization analysis (Module 11).

Table 1: Consolidated outputs under `RUN_DIR` (produced by Cell 9E).

File	Purpose
`portfolio_daily.parquet`	portfolio-level equity/returns
`pair_window_results.parquet`	per pair-window summary rows
`pair_daily.parquet`	per pair-window daily series
`pair_trade_rows.parquet`	trade-level records (fills/rows)
`pair_trade_events.parquet`	entry/exit/gating events
`pair_trade_episodes.parquet`	grouped episodes for diagnostics
`window_summary.parquet`	window-level aggregation summaries

In addition to the consolidated parquets, the run directory contains:

- checkpoint and progress logs used to resume and audit long runs,
- error ledgers for failed pair-windows,
- configuration snapshots and fingerprints to map artifacts back to code and settings.

10 Failure Modes and Debugging Postmortems

This section documents the major failure modes encountered during development and the specific guardrails introduced to prevent them. Each postmortem is written in a structured format:

$$\text{Symptom} \rightarrow \text{Root cause} \rightarrow \text{Fix} \rightarrow \text{Preventative rule.}$$

The goal is not to dramatize development but to provide an auditable record of how the system became defensible.

10.1 Vibecoding failure: speed exceeded understanding

Symptom. Early in the winter research period, the codebase expanded rapidly due to heavy use of AI coding agents (Claude Code, GitHub Copilot, and other assistants). Large blocks of code were accepted and integrated faster than they could be fully understood and tested. This produced a system that ran, but whose correctness was difficult to verify, and whose failure cases were not well characterized.

Root cause. The feedback loop was dominated by short-term "it runs" validation rather than invariant-based reasoning. Approving code without being able to explain every transformation and timing decision created a knowledge gap between the implemented system and the researcher's mental model.

Fix. The project was rebuilt in a single Colab notebook with explicit module boundaries, minimal contracts (SignalPack/ExecPack/Engine outputs), and incremental patching. The rebuild prioritized transparency and traceability over feature count.

Preventative rule. No code is merged into the core engine path unless the author can explain: (i) the timing convention, (ii) the exact data inputs and outputs, (iii) the invariants being preserved (e.g., "no silent alignment," "no silent resume," "return-based Sharpe").

10.2 Lookahead + survivorship incident (global integrity filtering)

Symptom. A major early data-handling bug produced an unexpectedly small tradable universe and suspiciously stable backtest behavior. Many assets that should have existed in later windows were missing entirely.

Root cause. Data integrity checks were performed globally across a 15-year horizon instead of within each window. Any ticker without full-sample coverage (including IPOs and later-launched ETFs) was removed, effectively conditioning on future existence and creating a survivorship/lookahead-style distortion.

Fix. Eligibility was redefined per formation window (Module 4), using minimum coverage thresholds computed on the window slice rather than on the full sample. Universe audit tables were added to make removals explicit and inspectable.

Preventative rule. Any filter that can remove assets must be evaluated under the question: *could this filter depend on future information?* If yes, it must be computed within the window or replaced with a historically valid rule.

10.3 Formula drift and metric misuse

Symptom. Different cells reported inconsistent "Sharpe" values, and some versions used a PnL-to-vol ratio that was not a return-based Sharpe ratio. This created confusion and made results hard to interpret.

Root cause. Metric definitions drifted as the pipeline evolved: PnL outputs, return outputs, and portfolio aggregation changed, but downstream diagnostics were not always updated to match the new schema.

Fix. The notebook standardized on a return-based daily series and defined annualized Sharpe strictly as Equation (6). Diagnostics were updated to read the correct return column (preferring gross-weighted returns), and safe-divide rules ensured zero-exposure days did not silently disappear.

Preventative rule. Metrics are treated as part of the interface contract. Any schema change to `portfolio_daily` requires updating and re-validating all downstream readers.

10.4 Engine complexity explosion (9A–9E)

Symptom. As the backtest engine expanded to handle real-world issues (corporate actions, missing data, checkpointing, consolidation), multiple uncontrolled edge cases emerged: run mixing, partial writes, inconsistent parts across streams, and ambiguity around corporate-action timing.

Root cause. The engine path initially combined too many responsibilities (simulation + I/O + recovery), which makes correctness and failure recovery difficult. Corporate actions from free data added ambiguity that could not be fully eliminated, only managed.

Fix. Responsibilities were separated across 9A–9E:

- 9C made pure (no file I/O),
- 9D responsible for checkpoint and commit markers,
- 9E responsible for cap-by-commit consolidation and stable schemas,
- corporate actions re-framed as data-quality gates with conservative policies.

Preventative rule. Any new edge-case handling must be implemented as:

$$\text{flag upstream} \rightarrow \text{policy in engine} \rightarrow \text{logged event} \rightarrow \text{diagnosable artifact.}$$

This ensures that complexity remains auditable rather than implicit.

10.5 What still remains "UNCERTAIN"

Despite conservative handling, several uncertainties remain intrinsic to free-data backtests:

- **Signal split adjustment:** whether the `CLOSE` series used for `LOGP_SIGNAL` is consistently split-adjusted across all tickers.
- **Dividend timing conventions:** the exact mapping between Yahoo's dividend date conventions and economically relevant ex-date price effects.
- **Implicit calendars and timezones:** the pipeline uses the observed index as the trading calendar without explicit exchange-specific calendars.

These are treated as limitations with conservative mitigations (data-quality gating, explicit flags, and cautious claims), and they define clear directions for future work (e.g., survivorship-free data, exchange calendars, and institution-grade corporate action feeds).

11 Diagnostics & Reporting (Module 10)

Module 10 converts consolidated artifacts into paper-ready figures and tables. A strict discipline in this project is that diagnostics are *read-only*: they do not modify trades, positions, or accounting. All plots and tables in this section are computed from the consolidated portfolio series (and consolidated pair-level summaries) produced by Module 9D–9E.

11.1 Figure directory (one-line editable)

11.2 Backtest coverage and headline outcomes

The consolidated portfolio backtest spans **2011-01-05 to 2025-12-05** (**3,753 trading days**). No missing daily returns were imputed as zero (`n_nan_returns_filled_as_zero` = 0). The strategy is active on most days (`active_day_frac` = 0.9723).

Table 2: Headline portfolio metrics (consolidated run). Annualized statistics are computed from daily portfolio returns assuming 252 trading days/year and zero risk-free rate.

Metric	Value
Start date	2011-01-05
End date	2025-12-05
Trading days (n)	3753
Active day fraction	0.972289
Geometric annualized return (CAGR)	0.029638
Annualized volatility	0.079652
Annualized Sharpe	0.406536
Annualized Sortino	0.573292
Max drawdown	-0.320485
Calmar ratio	0.092478
Skewness (daily)	-0.122175
Raw kurtosis (daily)	7.341810

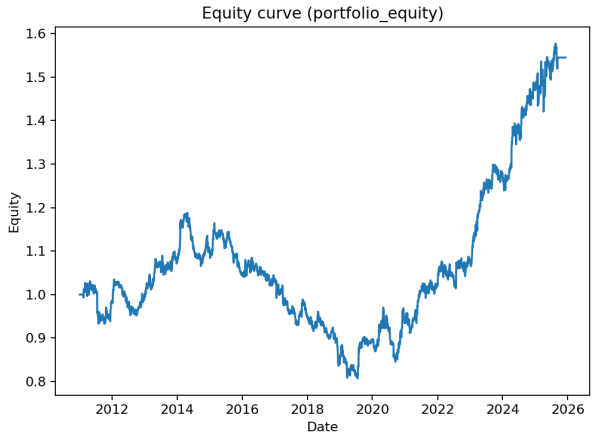
A notable regime dependence appears in a simple split: **2011–2019** is weak/negative, while **2020–end** is materially stronger (Table 3). The equity trough occurs on **2019-07-18** with equity **0.807**, ending at **1.545** (a **1.91**× multiple from trough).

Table 3: Subperiod metrics (descriptive regime split).

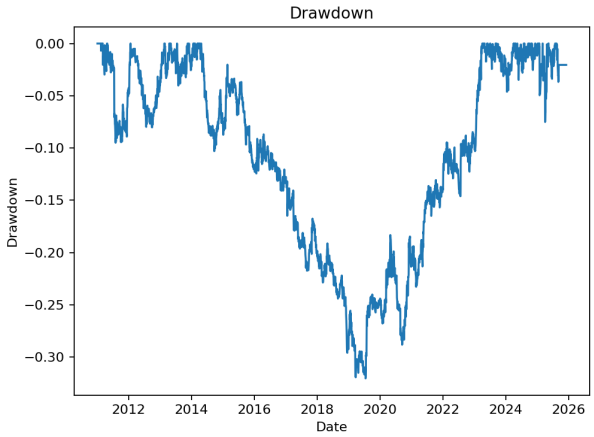
Period	n_{days}	Ann. return (geo)	Ann. vol	Ann. Sharpe
2011–2019	2262	-0.012357	0.071578	-0.137894
2020–end	1491	0.096782	0.090414	1.067116

11.3 Planned/produced figures (9-figure set)

This paper includes a compact 9-figure diagnostic set, focused on portfolio performance, distributional shape, and activity/exposure proxies.

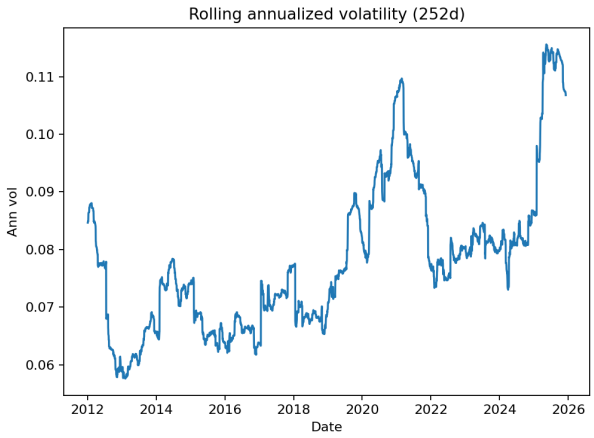


(a) Equity curve (portfolio equity).

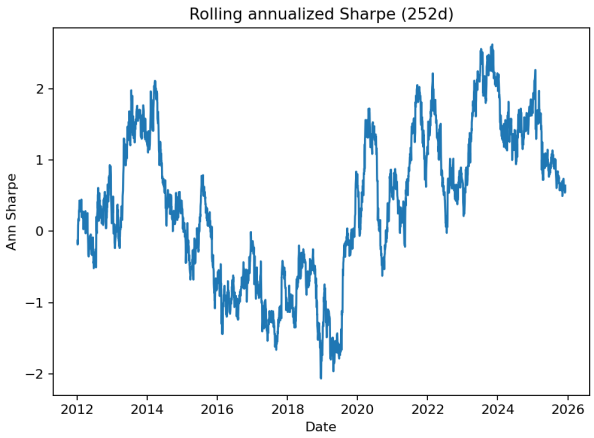


(b) Drawdown series (peak-to-trough).

Figure 1: Portfolio path diagnostics. The realized max drawdown is -0.3205 (Table 2).



(a) Rolling annualized volatility (252d).



(b) Rolling annualized Sharpe (252d).

Figure 2: Rolling risk and risk-adjusted performance. The post-2020 regime displays visibly improved rolling Sharpe compared to 2011–2019, consistent with Table 3.

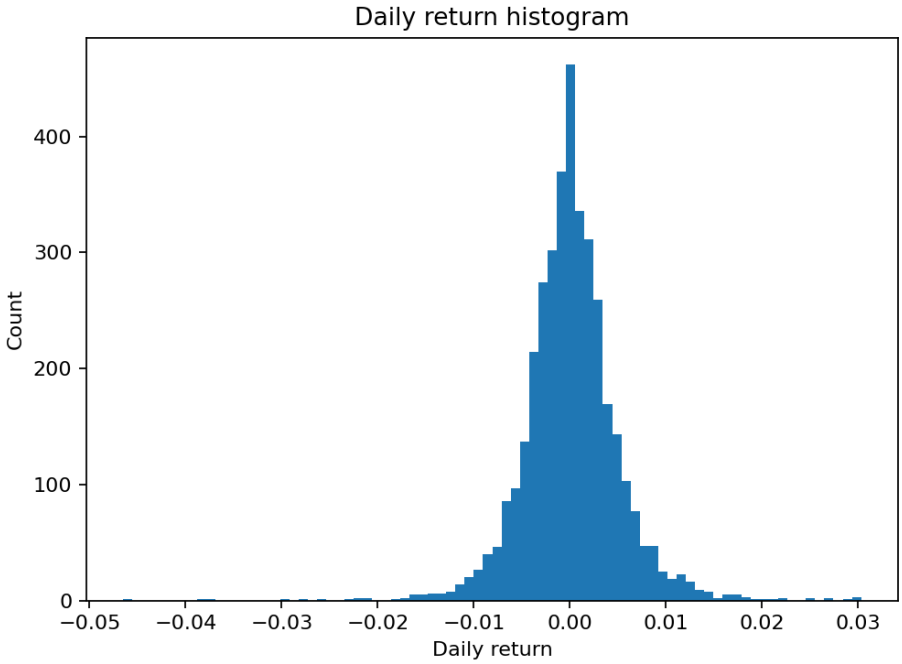
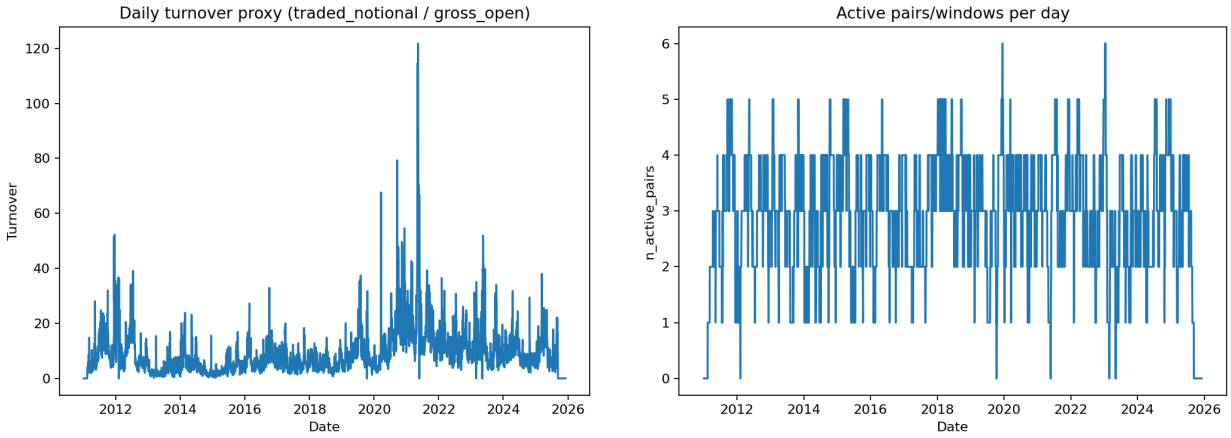


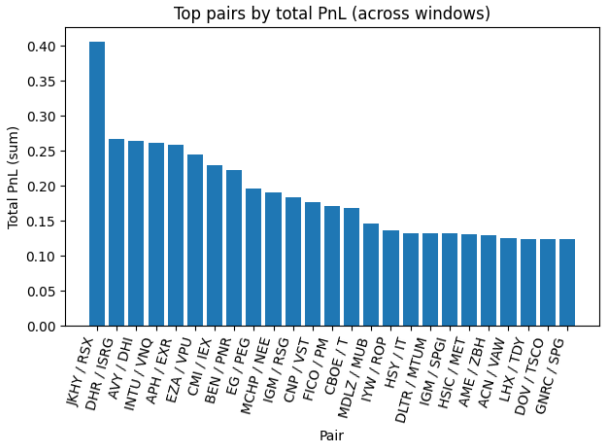
Figure 3: Histogram of daily portfolio returns. The distribution exhibits mild negative skew and heavy tails (raw kurtosis ≈ 7.34; Table 2), motivating conservative interpretations of Sharpe/normality-based summaries.



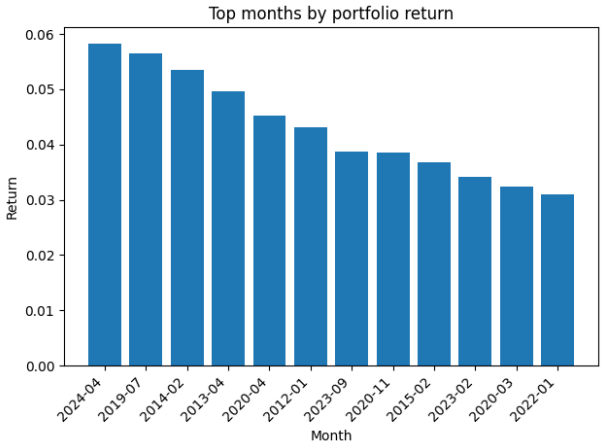
(a) Daily turnover proxy (traded_notional / gross_open).

(b) Number of active pair-windows per day.

Figure 4: Activity and trading intensity diagnostics. Spikes in the turnover proxy indicate periods of higher rebalancing/position churn, which should be interpreted jointly with transaction cost assumptions.



(a) Top pairs by total PnL (across windows).



(b) Top months by portfolio return.

Figure 5: Attribution-style summaries: best-performing pairs and best months (descriptive, not causal).

11.4 Calendar breakdown tables

Table 4: Year-by-year returns (portfolio).

Year	Year return
2011	-0.016740
2012	0.010930
2013	0.081364
2014	0.021064
2015	-0.045045
2016	-0.011672
2017	-0.084000
2018	-0.114416
2019	0.064403
2020	0.051713
2021	0.086918
2022	0.056081
2023	0.183338
2024	0.152929
2025	0.048761

Table 5: Best and worst 12 months (portfolio).

Top 12 months		Bottom 12 months	
2024-04	0.058275	2011-07	-0.059825
2019-07	0.056443	2018-12	-0.056241
2014-02	0.053465	2020-07	-0.040241
2013-04	0.049677	2019-03	-0.036349
2020-04	0.045158	2020-05	-0.036094
2012-01	0.043167	2021-02	-0.033612
2023-09	0.038649	2014-06	-0.033398
2020-11	0.038607	2017-12	-0.033148
2015-02	0.036827	2015-09	-0.030632
2023-02	0.034172	2020-08	-0.029704
2020-03	0.032452	2017-01	-0.029206
2022-01	0.030989	2024-01	-0.028859

11.5 Pair-level outcomes (descriptive attribution)

To avoid over-claiming, pair-level tables are interpreted as *descriptive attribution* rather than causal explanation. The results below summarize total PnL per pair across windows for the consolidated run.

Table 6: Top 25 pairs by total PnL (across windows).

y	x	windows	pnl_total	cost_total	trades	days	sharpe_ann_mean	sharpe_ann_median
JKHY	RSX	1	0.405895	0.002887	144	130	6.594260	6.594260
DHR	ISRG	1	0.267472	0.002765	124	127	3.762891	3.762891
AVY	DHI	1	0.264638	0.001916	184	125	3.178705	3.178705
INTU	VNQ	1	0.260848	0.002708	197	126	2.740575	2.740575
APH	EXR	1	0.258605	0.001826	132	129	3.435893	3.435893
EZA	VPU	1	0.244300	0.001844	98	123	4.622604	4.622604
CMI	IEX	1	0.229549	0.002626	96	128	4.090860	4.090860
BEN	PNR	1	0.223131	0.002000	183	128	2.488951	2.488951
EG	PEG	1	0.196201	0.001921	173	127	2.227528	2.227528
MCHP	NEE	1	0.190639	0.002020	173	125	1.990608	1.990608
IGM	RSG	1	0.184270	0.001829	116	128	2.443900	2.443900
CNP	VST	1	0.176332	0.001813	82	124	2.882601	2.882601
FICO	PM	1	0.171138	0.000978	92	126	4.030826	4.030826
CBOE	T	1	0.168304	0.002622	128	125	2.258245	2.258245
MDLZ	MUB	1	0.146917	0.001755	118	129	0.178761	0.178761
IYW	ROP	1	0.136228	0.001960	124	124	2.000133	2.000133
HSY	IT	1	0.133033	0.002629	130	126	2.262010	2.262010
DLTR	MTUM	1	0.132799	0.001873	172	126	1.936746	1.936746
IGM	SPGI	1	0.132387	0.002802	162	126	2.095836	2.095836
HSIC	MET	1	0.131602	0.006457	180	127	0.382002	0.382002
AME	ZBH	1	0.129153	0.002615	142	129	1.932259	1.932259
ACN	VAW	1	0.125842	0.001874	184	130	2.097846	2.097846
LHX	TDY	1	0.124662	0.001880	100	126	1.533456	1.533456
DOV	TSCO	1	0.124501	0.002661	145	123	1.946838	1.946838
GNRC	SPG	1	0.124414	0.002528	88	126	2.534682	2.534682

Table 7: Bottom 25 pairs by total PnL (across windows).

y	x	windows	pnl_total	cost_total	trades	days	sharpe_ann_mean	sharpe_ann_median
CSX	IWM	1	-0.318890	0.001904	144	125	-2.451391	-2.451391
LRCX	NUE	1	-0.277904	0.017752	168	130	-5.745244	-5.745244
CDW	NVDA	1	-0.257315	0.004513	168	126	-1.971757	-1.971757
AVY	IYK	1	-0.233709	0.004974	111	128	-3.365229	-3.365229
BF-B	IDU	1	-0.224512	0.001776	131	128	-4.046853	-4.046853
CME	FITB	1	-0.210842	0.001904	179	130	-2.514961	-2.514961
AME	HBAN	1	-0.199409	0.002697	144	123	-2.872644	-2.872644
CAH	MA	1	-0.197331	0.001913	130	125	-2.706806	-2.706806
JBHT	JKHY	1	-0.194881	0.004203	108	129	-5.038496	-5.038496
AMT	INVH	1	-0.182796	0.001851	148	127	-2.329298	-2.329298
CNP	VEA	1	-0.182096	0.005010	155	126	-2.692766	-2.692766
GDDY	IGV	1	-0.175719	0.001918	144	125	-2.485474	-2.485474
NDAQ	SCHW	1	-0.174959	0.002750	159	129	-2.377128	-2.377128
EWY	MCO	1	-0.164434	0.002785	180	126	-2.051754	-2.051754
SLB	UPS	1	-0.159241	0.000941	90	126	-4.313003	-4.313003
EL	IGN	1	-0.152979	0.001893	130	127	-2.922491	-2.922491
MKC	USMV	1	-0.152948	0.001967	183	128	-2.288363	-2.288363
CTAS	IYW	1	-0.149284	0.002064	154	126	-2.158644	-2.158644
AIG	AMCR	1	-0.149158	0.009009	126	125	-1.346800	-1.346800
ATO	SPLV	1	-0.143740	0.001885	182	126	-2.585079	-2.585079
BR	SO	1	-0.141951	0.002757	172	127	-1.366790	-1.366790
ORLY	SCHD	1	-0.135873	0.001832	168	125	-2.295067	-2.295067
ED	IWD	1	-0.135816	0.001815	182	126	-1.841797	-1.841797
CMS	EZA	1	-0.135614	0.017887	223	125	-1.924667	-1.924667
CBOE	QUAL	1	-0.132897	0.003560	156	128	-1.823163	-1.823163

11.6 Feature contrasts (top vs. bottom decile pairs)

Finally, we compute a simple contrast in formation-time features between top and bottom decile pairs (by realized outcomes).

Table 8: Feature contrast: top vs. bottom decile pairs (descriptive).

Feature	Top 10% mean	Bottom 10% mean	Top minus bottom
zc_rate	32.665166	33.696863	-1.031697
half_life	5.602923	6.580951	-0.978027
sigma	0.031873	0.030454	0.001419
coint_pval	0.009619	0.009733	-0.000114
adf_pvalue	0.002029	0.002050	-0.000021

The feature differences are small in magnitude in this run, reinforcing that these summaries should be presented as *diagnostic descriptions* rather than as strong predictive claims.

12 Optimizer and Overfitting Control (Module 11)

Module 11 answers one pragmatic question: *if we tune parameters, how likely are we to be fooling ourselves?* In this project, optimization is treated as a *secondary* analysis layer that reuses the same engine contracts (signal pack → execution pack → pure engine) and writes auditable, run-isolated artifacts. The optimizer is not allowed to "patch" the backtest ex-post; it can only propose parameter settings and then re-run the same accounting.

12.1 Why optimization is dangerous in finance

Parameter tuning increases false discovery risk: when many configurations are tested, some will look best purely by noise. This risk is amplified here by (i) non-stationarity and regime shifts, (ii) heavy-tailed returns (kurtosis ≈ 7.34 in the final run), and (iii) interaction effects among parameters (bands, stops, formation/trading lengths, gating rules, etc.). Therefore, optimizer outputs are reported as *hypothesis generation* unless they demonstrate stability across time splits.

12.2 Method: CSCV and Probability of Backtest Overfitting (PBO)

Module 11 follows the cross-validation logic of Combinatorially Symmetric Cross-Validation (CSCV) and reports an estimate of the Probability of Backtest Overfitting (PBO) in the sense of Bailey et al. [1]. Operationally, the optimizer:

- evaluates each configuration on multiple train/test split combinations,
- selects "best" configurations on training folds,
- measures how often those selections underperform out-of-sample relative to alternatives.

The key output is not a single best Sharpe, but a *risk statement*: how frequently the optimization process selects a configuration that fails out-of-sample.

12.3 Executed status and results (final locked run)

Module 11 was executed on the final run-isolated consolidated artifacts under `RUN_DIR`. Across **620** parameter scenarios, CSCV-style evaluation produced an estimated **PBO** ≈ 0.281. We interpret this as a **moderate overfitting warning**: a non-trivial fraction of "best-looking" configurations are plausibly benefiting from noise and regime-specific luck rather than stable edge.

Table 9: Optimizer robustness summary (Module 11).

Quantity	Value
Number of scenarios evaluated	620
Overfitting risk estimate (PBO)	0.281
Interpretation	Moderate warning (not negligible)

12.4 Guardrails for reporting optimizer outputs

To keep the paper defensible, we follow these reporting rules:

- **No "best config" victory laps.** The best configuration is not treated as a discovery unless it is stable across splits and subperiods.
- **Distributions over point estimates.** We report fold/split distributions (or summary percentiles) rather than a single maximum metric.
- **Consistency with baseline accounting.** All metrics use the same return definition, annualization, and cost model as the baseline run.
- **Regime-awareness.** A single PBO number does not "solve" non-stationarity; we still require subperiod consistency (Section 13).

In short: the optimizer is useful, but it is not a truth machine. A PBO near 0.28 says: *"optimization can help, but it can also easily hallucinate edge."*

13 Results (Final Locked Run)

All results in this section are computed from the consolidated daily return series (`portfolio_daily.parquet`) under a single, run-isolated `RUN_DIR`. Daily Sharpe is computed from daily returns and annualized with $\sqrt{252}$ (zero risk-free rate). Figures referenced here are generated in Module 10 (Section 11).

13.1 Headline performance and risk

Over 2011-01-05 to 2025-12-05 ($n = 3753$ trading days), the strategy produces a modest long-run risk-adjusted profile: annualized Sharpe ≈ 0.41, CAGR $\approx 2.96\%$, and max drawdown $\approx -32.0\%$ (Table 10). The equity curve and drawdown series (Figure 1) reveal a long drawdown episode culminating around mid-2019, followed by a strong recovery through 2023–2025.

Table 10: Performance summary (2011-01-05 to 2025-12-05; $n = 3753$ trading days).

Metric	Value
Annualized geometric return	2.9638%
Annualized volatility	7.9652%
Annualized Sharpe	0.4065
Annualized Sortino	0.5733
Maximum drawdown	-32.0485%
Calmar ratio	0.0925
Skewness	-0.1222
Raw kurtosis	7.3418
Active day fraction	0.9723

Equity trough occurs on 2019-07-18 (equity ≈ 0.807) and ends at equity ≈ 1.545 (about $1.91\times$ from trough).

13.2 Regime dependence (the main story, not a footnote)

Performance is strongly regime-dependent. The pre-2020 period is weak/negative, while 2020–2025 is materially stronger. This is not subtle; it is the dominant empirical feature of the backtest and should be stated plainly.

Table 11: Subperiod metrics (annualized; daily returns).

Period	Days	AnnGeoRet	AnnVol	AnnSharpe
2011–2019	2262	-1.2357%	7.1578%	-0.1379
2020–2025	1491	9.6782%	9.0414%	1.0671

The rolling Sharpe plot (Figure 2) visually supports this split: the strategy spends long stretches near or below zero Sharpe before 2020, then improves substantially afterward. Any claim of "stable edge" must therefore be tempered: the strategy appears to be *regime-sensitive*.

13.3 Distributional diagnostics: fat tails, not Gaussian comfort

Daily returns exhibit heavy tails (raw kurtosis ≈ 7.34) and mild negative skew (Table 10). The histogram (Figure 3) reinforces that a Gaussian mental model is unsafe here. Consequently, headline metrics like Sharpe should be treated as *summaries*, not guarantees; tail risk is material, and drawdown behavior (Figure 1) is part of the result, not an inconvenience.

13.4 Activity and trading intensity

The strategy is active on most days (active day fraction ≈ 0.97), but the number of active pair-windows varies over time (Figure 4). The turnover proxy series (Figure 4a) shows pronounced spikes, indicating periods of higher churn. These spikes matter because transaction costs are an explicit modeling assumption: if real-world costs are higher than modeled, performance would be most vulnerable precisely in high-turnover episodes.

13.5 Attribution-style summaries (descriptive, not causal)

We report pair-level and calendar summaries (top pairs, top months) as descriptive attribution rather than causal proof. The "top pairs by total PnL" plot (Figure 5a) shows that outcomes can be concentrated in a relatively small set of pairs/windows, while the existence of bottom-performing pairs (reported in the diagnostics tables) reminds us that selection and regime interactions can dominate.

13.6 Takeaways

The final locked run supports three sober conclusions:

- **The strategy is not uniformly profitable across eras.** 2011–2019 underperforms; 2020–2025 performs strongly (Table 11).
- **Risk is real and non-Gaussian.** MaxDD $\approx 32\%$ and kurtosis ≈ 7.3 imply that tail behavior must be respected, not hand-waved.
- **Optimization must be treated as a risk-managed tool.** With PBO ≈ 0.281 across 620 scenarios (Section 12), "best config" results should be framed as hypotheses unless stability is demonstrated.

14 Limitations

This project prioritizes reproducibility and auditability over maximum realism. The backtest results should therefore be interpreted as evidence about a *model strategy under explicit assumptions*, not as a production-ready trading system. The main limitations are grouped below.

14.1 Free-data uncertainty (corporate actions and price adjustments)

Two sources of uncertainty are inherent to relying on free market data feeds.

Split/dividend labeling uncertainty (UNCERTAIN). Corporate-action events (dividends, splits) are detected from free data feeds and may be missing, delayed, or mis-dated (e.g., pay-date vs ex-date conventions, incomplete coverage for some tickers). To mitigate event-driven gaps without relying on precise cashflow accounting, the engine implements a conservative gating policy:

- When day t is flagged as a corporate-action day, the strategy forces positions to be flat starting at **open**$(t-1)$ (i.e., it avoids carrying exposure overnight into the flagged day).
- This reduces sensitivity to corporate-action day discontinuities, but it cannot guarantee correct alignment if the event flags themselves are incorrect.

Signal price adjustment ambiguity (UNCERTAIN). Signal construction relies on adjusted closing prices when available (e.g., Yahoo-style adjustment conventions), but adjustment correctness and consistency across tickers is not guaranteed. The pipeline partially addresses this by surfacing split-related flags and using the gating mechanism above, but a definitive resolution would require a higher-quality corporate-action and adjustment feed.

14.2 Execution and market-friction simplifications

The execution model is intentionally simple and daily-frequency:

- Entries and exits occur at the **daily open**; intraday execution is not modeled.
- Transaction costs and slippage are modeled via a parametric rule rather than microstructure simulation.
- Borrow availability, borrow fees, financing rates, margin requirements, and broker constraints are not explicitly modeled.

These assumptions are reasonable for a reproducible undergraduate research backtest, but they limit any claim of deployability without additional validation.

14.3 Capital allocation model and overlapping windows

The strategy trades many overlapping walk-forward windows (multi-cohort overlap). Portfolio performance is therefore only meaningful under an explicit capital allocation rule. This project defines daily portfolio returns as an equal allocation across active windows (as specified in the consolidation definition), which is clear and reproducible but not necessarily optimal.

In live trading, alternative policies (e.g., volatility targeting, risk parity across pairs, concentration limits, leverage constraints, and inventory-aware sizing) may materially change both returns and drawdowns. The results reported here should be interpreted under the stated allocation rule only.

14.4 Universe definition, survivorship, and investability

The universe mixes sector ETFs (generally stable, liquid) with an equity set whose historical membership may not be survivorship-free. Without a survivorship-free constituent history, historical equity results can be biased upward by implicitly excluding delisted or hard-to-trade names.

The pipeline reduces one class of bias via *window-specific* eligibility and data-integrity checks (avoiding global lookahead filtering across the full sample), but it cannot fully eliminate survivorship effects for equities without a survivorship-free dataset.

14.5 Multiple testing and optimization risk

Any parameter search increases the probability of false discovery. Module 11 reports an explicit selection-bias diagnostic (CSCV/PBO) to quantify this risk, but the diagnostic reduces rather than eliminates overfitting concerns.

Accordingly:

- optimizer-selected configurations are treated as *hypothesis generation*,
- the report avoids claiming a universally "best" configuration unless robustness is supported across splits and subperiods.

14.6 Regime dependence and external validity

The strategy exhibits strong regime dependence in the final locked run: earlier years are weak while the post-2020 subperiod is materially stronger. This raises the usual external-validity question: whether performance reflects a persistent structural effect or a regime-specific opportunity.

Therefore, claims are restricted to the observed sample and clearly labeled subperiods. Extending conclusions beyond those regimes requires further out-of-sample testing, alternative universes, and more realistic execution assumptions.

15 Lessons Learned

This project produced at least as much engineering education as quantitative output. The strongest takeaway is not a single Sharpe number, but a practical set of habits that prevent self-deception and keep research reproducible under real constraints (Colab resets, long runs, partial writes, and free-data quirks).

15.1 Rules for using AI agents in research

The following rules were adopted after early failures where "fast progress" was actually uncontrolled drift.

- **No unexplainable merges.** AI-generated code is not accepted into the core path unless we can explain (i) inputs/outputs, (ii) timing assumptions, (iii) invariants preserved, and (iv) failure modes. If we cannot explain it, we cannot defend it.
- **Small diffs, frequent checkpoints.** Prefer incremental changes that can be audited. A working baseline is preserved; new logic is introduced behind clearly named flags or as read-only diagnostics. This prevents "fixed one thing, broke three others" situations.
- **Contracts before cleverness.** Before adding features, we define what each stage must produce and consume (SignalPack $\rightarrow$ ExecPack $\rightarrow$ Engine outputs). When contracts are stable, the rest of the system becomes debuggable and reviewable.

- **Log every policy that changes interpretation.** Missingness handling, corporate-action gating, cost/slippage conventions, and portfolio return definitions must be explicit and written to disk. A backtest without a policy log is not a backtest; it is a story.
- **Results are artifacts, not prints.** Anything reported in the paper must come from saved parquets/CSVs/PNGs under a run directory, not from a transient notebook cell output. This allows re-plotting and re-checking without rerunning expensive modules.
- **Keep analysis read-only.** Diagnostics must not modify trades/positions/accounting. The moment analysis code mutates the trading record, the pipeline becomes impossible to trust. This is why Module 10/11 read consolidated outputs only.
- **Refuse ambiguous metrics.** If a metric name can be misread (e.g., a "Sharpe" computed from a non-return series), it must be renamed or removed. A clean metric definition is more important than a pretty table.

15.2 Research hygiene checklist you now follow

Before trusting any backtest output, we run the following checklist. It is short by design: long checklists are not executed.

- **Timing sanity:** signals are lagged relative to execution (no same-bar trading; no accidental lookahead through joins/merges).
- **Window sanity:** eligibility and filtering are computed inside each walk-forward window (no global future-dependent screens).
- **Metric sanity:** Sharpe is computed from the portfolio daily return series and annualized consistently (and the return series definition is stated).
- **Run sanity:** outputs are isolated under a single `RUN_DIR`; consolidation enforces commit-marker consistency to avoid mixed snapshots.
- **Deduplication sanity:** consolidation removes resume overlaps so trades and daily rows cannot be double-counted.
- **Edge-case audit:** inspect rates for corporate-action gates, missing opens, and force-flat exits; confirm these events are rare and explainable.
- **Regime sanity:** report subperiod results and do not claim "it works" without stating where it works.

15.3 Interpretation discipline: what we will and will not claim

Given the regime dependence observed in the final locked run, the correct posture is cautious:

- We **will** claim the pipeline is reproducible and defensible, and that the strategy exhibits strong performance in a clearly stated subperiod (post-2020) under a stated capital allocation rule.
- We **will not** claim universal stationarity, production readiness, or a guaranteed edge across all market regimes without stronger out-of-sample evidence and more realistic frictions.

16 Conclusion

This work delivers a complete, reproducible pairs-trading research pipeline: candidate generation and validation, walk-forward trading evaluation, run-isolated artifact production, and a diagnostics layer that is strictly read-only. The central engineering contribution is the separation of concerns: trading logic is implemented upstream, while consolidation and reporting enforce consistency, deduplication, and auditability.

On the final locked run, performance is meaningfully regime-dependent. The full-sample headline metrics are modest, while the post-2020 subperiod shows substantially stronger behavior, including a large recovery from the 2019 equity trough to the end of the sample. This supports a careful but defensible research statement: under the stated assumptions, the strategy exhibits evidence of edge in a specific market era, and the pipeline provides a reliable foundation for further controlled testing rather than ad-hoc experimentation.

Reproducibility Statement

All artifacts for a given experiment are written under a single run directory `RUN_DIR` created in Cell 9A. The run directory contains:

- streamed parquet parts generated during execution (Cell 9D),
- consolidated parquet files produced by Cell 9E (Table 1),
- checkpoints, progress logs, error ledgers, and run fingerprints.

A result presented in this report is reproducible if and only if it can be regenerated by rerunning Module 10/11 on the consolidated parquets in the same `RUN_DIR`. No result is considered final until it is backed by a completed run with coherent commit markers and a verified fingerprint.

References

[1] David H. Bailey, Jonathan M. Borwein, Marcos López de Prado, and Qiji Jim Zhu. *The Probability of Backtest Overfitting.* Wiley, 2015.

[2] Robert F. Engle and Clive W. J. Granger. Co-integration and error correction: Representation, estimation, and testing. *Econometrica*, 55(2):251–276, 1987.

[3] Evan Gatev, William N. Goetzmann, and K. Geert Rouwenhorst. Pairs trading: Performance of a relative-value arbitrage rule. *The Review of Financial Studies*, 19(3):797–827, 2006.

[4] William F. Sharpe. Mutual fund performance. *The Journal of Business*, 39(1):119–138, 1966.

[5] Ganapathy Vidyamurthy. *Pairs Trading: Quantitative Methods and Analysis*. Wiley, 2004. ISBN 9780471460671.