LUNA LABS

INSTRUMENTS FOR INVISIBLE WORK

The Programme

An Operational Playbook

Stages -1 to 4 and the market versions, step by step:
every action, every governing formula, every gate — no derivations



Prepared for Riddhish · companion to the Course Book

August 2026

How to Read This Playbook

This is the execution companion to *Transformers for Market Prediction: A First-Principles Course* (the "Course Book"). The division of labour is strict: the Course Book derives and proves; this playbook *instructs*. Every governing formula appears here exactly once, at the step that uses it, with its symbols defined in a single line and a gold chip — [Course Book §15.4] — pointing to the section of the Course Book where it is derived, proved, and exercised. Nothing is derived here, ever.

The unit of this book is the **step**. Each step carries up to four fields: *Do* (the concrete action — the query, the table, the script behaviour, the parameter values), *The mathematics* (the formula governing the step, stated once), *Artifact* (what now exists on disk when the step is complete), and *Pass / fail* (the concrete check that it worked). Steps are numbered per chapter and are meant to be executed in order; where an ordering is flexible the text says so.

Each stage chapter ends with three fixed blocks: the **Stage gate** (the formal test that decides whether the stage passed — metric, threshold, decision rule), the **Kill criterion** (the pre-committed wording of what a failure means and what happens next — written to be signed *before* the stage runs), and the **Artifact register** (everything the stage leaves behind, pass or fail). Part V compresses the whole programme into a single master sequence — the overall steps — for the days when you need the map rather than the territory.

Two standing rules govern everything in these pages. First, *no result is believed until deflated*: every backtest enters the trial ledger and faces the multiple-testing machinery of Course Book Part VII. Second, *the kill criteria are pre-commitments*: each stage's kill wording is agreed and signed before that stage's first run, because the Course Book's Chapter 15 is a proof that a judgement formed after seeing results cannot be trusted, even by its author.

Contents

I The System and the Rules **1**

1 The System Today, and the Rules of the Programme **2**
1.1 What This Playbook Governs . . . 2
1.2 Component Inventory . . . 4
1.3 The Verified Findings Register . . . 6
1.4 Roles and Instruments of Honesty . . . 7
1.5 Conventions . . . 8
1.6 The Dependency Map . . . 11

II Preconditions: Stage −1 **12**

2 Stage −1a: The Cost Surface **13**
2.1 The schema and the raw pull . . . 13
2.2 Conditioning the surface . . . 15
2.3 Aggregating, costing, and composing . . . 18
2.4 Validation and the freeze . . . 20

3 Stage −1b: Point-in-Time Feeds **23**
3.1 The truth schema and the as-of join . . . 23
3.2 COT: anatomy, calendar, and the net-positioning z-score . . . 24
3.3 FRED to ALFRED: vintage ingestion . . . 27
3.4 RORO and VolRegime: freezing the gauges . . . 29
3.5 Yield differentials . . . 30
3.6 The feed registry and staleness monitors . . . 31
3.7 The audit query . . . 32

4 Stage −1c: The Harness, the CI Suite, and the Ledger **34**

III The Kill-Gated Stages **43**

5 Stage 0: The Envelope Honesty Test **44**
5.1 Preconditions and scope . . . 44
5.2 The features and the declared class . . . 46
5.3 Benchmarks, the run, and the deflation . . . 49
5.4 The verdict . . . 52

6 Stage 1: The Adaptive Width Engine **58**
6.1 Preconditions and the two entry paths . . . 58
6.2 The volatility engine . . . 59
6.3 From variance to a band . . . 63

6.4 Live operation . . . 68
6.5 Acceptance . . . 71
6.6 The gate . . . 74

7 Stage 2: The Regime Router and the Covariate Probes 77
7.1 Preconditions and the observable vector . . . 78
7.2 Fitting and deploying the router . . . 78
7.3 The CUSUM bank: a cheaper alternative . . . 81
7.4 From regime to action: the router wiring . . . 82
7.5 The Chronos-2 covariate probe . . . 83
7.6 Incremental evaluation and the read-out . . . 84

8 Stage 3: The Bespoke Model 87
8.1 The dataset and the model . . . 87
8.2 The objective and the run . . . 92
8.3 The gauntlet, the freeze, and the live path . . . 95
8.4 After the ship . . . 99

9 Stage 4: Generative Tail Tooling 102

IV The Market Versions 111

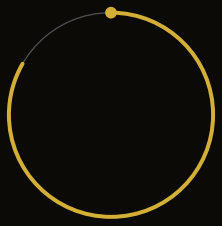
10 The Market Versions: v2 Options, v3 Pairs, v4 Commodities 112
10.1 v2 — Options . . . 112
10.2 v3 — Pairs . . . 116
10.3 v4 — Commodities . . . 120
10.4 The Shared Spine: Calibration, Cold Start, Rollout . . . 122

V The Overall Steps 125

11 The Overall Steps 126
11.1 The three standing rules . . . 126
11.2 The master sequence . . . 126
11.3 The programme at a glance . . . 129

PART I

The System and the Rules



LUNA LABS

1

The System Today, and the Rules of the Programme

This chapter is the map and the constitution. Section 1.1 states what the programme is and how this book divides labour with the Course Book. Section 1.2 inventories every live component of the system today, gap by gap. Section 1.3 fixes the four findings every later stage leans on. Section 1.4 names who owns what and the pre-commitments that bind every stage before it runs. Section 1.5 fixes units, timeframes, the instrument roster, the session clock and this playbook's own file conventions. Section 1.6 draws the whole programme as one diagram. Nothing below is derived; everything below is either inventory or rule.

1.1 What This Playbook Governs

The programme is five kill-gated stages, run once on the primary book and then reused, with per-market heads, on three further books. **Stage** -1 builds the instruments the rest of the programme trusts (a measured cost surface, point-in-time feeds, a harness that cannot lie to itself). **Stage 0** asks whether the system as it stands today — the live TimesFM envelope, nothing else — carries any net-of-cost tradeable information at all. **Stages 1–3** replace one component at a time, each only on the strength of a deflated, out-of-sample improvement over the stage before it: adaptive width, then a regime router and covariate probes, then, only if both survive, a bespoke small transformer. **Stage 4** is generative tooling for tail sizing and stress data, explicitly excluded from ever being an alpha source. Every stage carries a pre-committed metric, threshold and kill action; Table 1.1 summarises all six, and the Course Book's own tear-out version of the same table is [Course Book App. E].

The gating is sequential and load-bearing, not decorative. Stage $n+1$ may consume Stage n's output only after Stage n's gate has been cleared, because building on an unvalidated component multiplies unproven assumptions instead of testing one at a time, and because every stage that runs adds to the trial count the final Deflated Sharpe Ratio must survive [Course Book §15.4]. A stage that is killed is not a failure of execution — the execution is what produced the kill, correctly — and the correct response is the pre-committed one in Table 1.1's last column, never a second attempt at a lower threshold.

Once the primary FX book (**v1**) has cleared enough of that sequence, the same backbone grows three further per-market heads — **v2** (Australian and US options), **v3** (equity pairs) and **v4** (commodity CFDs) — sharing one trunk, one trial ledger, and strictly siloed calibration and cost models per market [Course Book §§19.5–19.6]. Table 1.2 gives each version's gate. Chapters 2–9 of this playbook (Parts II–III) build v1 stage by stage; Chapter 10 (Part IV) builds v2–v4 on top of it.

Every gold chip in this book, [Course Book §15.4] for instance, is a verified pointer: the exact Course Book section that derives the formula, proves the theorem, or states the result the preceding sentence used. This chapter's chips have all been checked against `/home/claude/book/chNN.tex` and `main.toc` at the section level; a wrong pointer is a defect, not a style choice. None of the six sections below re-derives anything — where a step needs the reasoning behind a number, the chip is the only place to find it, and the governing formula, where one exists, is stated once, at the step

Stage	This book	Course Book	Governing question	Default kill
−1	Ch. 2–4	[Course Book §24.2]	Can any claim below be trusted at all?	BLOCK: no Stage-0 result may be reported until the cost surface and point-in-time feeds exist.
0	Ch. 5	[Course Book §14.7]	Does the live TimesFM envelope carry any tradeable information, net of cost?	No policy clears the deflated bar: no model is built; effort moves to execution and portfolio construction.
1	Ch. 6	[Course Book §12.6]	Does a conditional width beat the frozen `width_mult`?	Keep the static multiplier; Stage 2 does not open.
2	Ch. 7	[Course Book §13.8]	Do the macro feeds carry conditional information the univariate engine cannot see?	Router dies; the feeds demote to ordinary Stage-3 covariate inputs.
3	Ch. 8	[Course Book §8.5]	Does a bespoke 1–10M-parameter model earn its complexity?	Discard it; keep the Stage-1/2 champion in production.
4	Ch. 9	[Course Book §23.7]	Measurement instrument, or alpha?	Offline scenario/stress tooling only; never a live sizing input.

Table 1.1. The six stage gates, one line each. Full metric and threshold in each stage's own chapter and gatebox; the Course Book's own version is Appendix E [Course Book App. E].

Version	This book	Gates on	Course Book
v1 — FX CFDs	Ch. 2–9	Stage 0 through Stage 3 in full; the only version with no upstream dependency.	[Course Book §22.5]
v2 — options	Ch. 10	v1's Stage 1 passing — the calibrated $\hat{\sigma}$ engine is the entire premise.	[Course Book §22.5]
v3 — equity pairs	Ch. 10	Nothing upstream; may run alongside v1 (edge is cointegration, not calibrated width). Its own kill bar is the pair-mining multiplicity correction.	[Course Book §22.5]
v4 — commodities	Ch. 10	Rides the shared harness; adds a roll- and seasonality-adjustment gate none of the others need.	[Course Book §22.5]

Table 1.2. The four market versions and what each waits on. All four report into the one trial ledger of Section 1.4.

that uses it, never twice and never proved.

This playbook and the Course Book are two documents built from the same material for the same reader, and Table 1.3 is the honest description of the difference every chip below leans on.

Dimension	Course Book	This playbook
Purpose	Derives, proves and teaches the machinery.	Instructs and executes it.
Formula treatment	Derived in full, sometimes more than one way, with worked examples.	Stated once per step, symbols defined in a single line, never derived.
Primary unit	The section, building one concept at a time.	The step (stepbox), producing one artifact.
Exercises and solutions	Yes, at the end of every chapter.	None.
Numerical verification	`verify_chNN.py`, every worked number in the chapter.	`verify_pbNN.py`, only this book's own arithmetic.
Read when	Before building anything, to understand why.	While building, to know exactly what to do next.
Cross-referencing	Self-contained; points its own chapters at each other.	Every formula or theorem carries a gold cross-reference chip back to the Course Book.

Table 1.3. What each book is for. The division of labour is strict in both directions — this playbook never derives, and it never omits a chip where a claim needs one.

1.2 Component Inventory

Table 1.4 is the master inventory: every live component of the system as it runs today, what it lacks, and which stage of Sections 1.1's two tables closes the gap. Read it before reading anything else in this book — every later chapter assumes exactly this state and no more, and every trap in this playbook is one row of this table, expanded.

Table 1.4. The component inventory: every live piece of the system, what it lacks, and the stage that closes the gap.

Component	What it is	What it lacks	Fixed by	CB
C# engine	Execution engine; polls the latest row per (instrument, granularity) from `timesfm.sqlite` and `book_ml_predictions`; no forecasting logic of its own.	Blind to how the row it just read was generated or how stale it is; a schema drift in either table is a silent failure, not a raised one.	Stage −1c formalises the read contract (Ch. 4)	—
`forecast_service.py`	TimesFM 200M zero-shot, batches all $21 \times 7 = 147$ series, context 1024–1536 bars, writes median, $q_{10}, q_{20}, q_{80}, q_{90}$ and a static `width_mult` to `timesfm.sqlite` per (instrument, granularity).	Median at-or-worse than a random walk on 6/7 timeframes; `width_mult` is frozen, calibrated once, never adapted.	Stage 0 tests envelope value (Ch. 5); Stage 1 replaces the width (Ch. 6)	[Course Book §9.2]

continued on next page

Table 1.4 (continued)

Component	What it is	What it lacks	Fixed by	CB
`book_ml_service.py`	A 20-minute polling loop writing `book_ml_predictions` into the SAME `timesfm.sqlite` database.	No documented accuracy or coverage claim; table drops are the only integration contract with the C# engine — the hand-off is a table name and a schema, not a versioned interface.	Stage −1c writes the contract down (Ch. 4)	—
`candles_3y`	Three years of OHLCV history, all seven timeframes, all 21 instruments: open, high, low, close, volume.	Mid-price only — no bid, no ask, ever stored; every historical spread is unrecoverable from this table.	Stage −1a builds the cost surface from a separate bid/ask re-pull (Ch. 2)	[Course Book §14.2]
`forecasts_hist.sqlite`	A history of stored TimesFM output pre-dating live production.	H1 and M30 only; 10 FX majors only, not the full 21; roughly one month stale; generated at context 1024 while live H1 runs context 1536.	Stage 0 must regenerate it at live settings before trusting it (Ch. 5)	[Course Book §9.2]
`cot.sqlite`	CFTC Commitments of Traders positioning, one row per `report_date`.	No release-timestamp column — the report describing Tuesday's positions is not public until the following Friday, and nothing in the schema records that.	Stage −1b adds `release_time` and rebuilds the join (Ch. 3)	[Course Book §18.1.5]
FRED key	Live access to FRED's current-mirror macro series.	No vintages — every query returns today's most-revised figure, never what was known as of a historical date.	Stage −1b requires ALFRED vintages instead (Ch. 3)	[Course Book §18.1.4]
RORO / VolRegime	Live gauges computed by the C# engine.	No historical as-of reconstruction — only the live, current value exists; any backtest "using" these gauges today recomputes them non-causally over history.	Stage −1b builds the causal, point-in-time reconstruction (Ch. 3)	[Course Book §18.2]
GPUs	One RTX 3060, pinned for ML via `CUDA_VISIBLE_DEVICES`, serving live inference today.	An idle RTX 5080 with no assigned role; contention between live inference and any asynchronous Stage-4 sampler is unresolved.	Resolve before Stage-3 scoping (Ch. 8); training cadence and MPS caps are Course Book economics	[Course Book §24.2]

Step 1.1 — Verify the component inventory

Do. Before trusting a single cell of Table 1.4, query the running system directly: row counts and `MAX(timestamp)` on `timesfm.sqlite`, `book_ml_predictions`, `candles_3y`, `forecasts_hist.sqlite` and `cot.sqlite`; `echo $CUDA_VISIBLE_DEVICES` on the training host; diff one FRED client call against a known ALFRED vintage for the same series and date. Every "what it lacks" cell is a claim about the system's state on the day this chapter was written, not a permanent property of it.

Artifact. A dated inventory-check note, one paragraph per row, pass or fail, filed alongside the trial ledger of Section 1.4.

Pass / fail. Every row of Table 1.4 matches what was just queried. Any mismatch is filed as a new trap before the next stage runs, not silently corrected in this book.

Trap — Mid-price candles price a fictional book

`candles_3y` and every table derived from it store open, high, low, close and volume — never bid, never ask. A backtest that fills on these prices is not approximating the real cost of trading; it is omitting the cost entirely, because the spread that closed every real trade was never recorded to begin with. Hit rate, expectancy and every Sharpe ratio computed against mid fills describe a book that has never existed and cannot be traded into. Nothing downstream of Stage −1a is meaningful until a bid/ask cost surface is re-pulled and wired into the P&L path. [Course Book §14.2]

Trap — `report_date` is not the date COT became knowable

`cot.sqlite` carries `report_date` and the positioning figures, and nothing that records when a row became public. The CFTC publishes Tuesday's positioning the following Friday, roughly 20:30 UTC — a three-calendar-day gap. A pipeline that joins on `report_date` treats Tuesday's positioning as knowable from Tuesday, when it is unknowable until Friday evening: every Wednesday and Thursday bar that used it traded on data that did not yet exist. Typing COT as known-future compounds the error into a look-ahead large enough to manufacture an entire backtest edge on its own. [Course Book §18.1.5]

1.3 The Verified Findings Register

Four facts, all measured rather than assumed, carry the weight of every decision in this book. Table 1.5 states each with its verified value and its caveat; none is re-derived here.

Finding	Verified value	Status / caveat	CB
Median forecast vs. random walk	MAE ratio 1.04–1.09 on 6 of 7 timeframes	At-or-worse than the trivial benchmark; not evidence of skill in the point forecast.	[Course Book §9.2]
80% band coverage	Aggregate 78–81%	Unconditional only — says nothing about calm- vs. violent-regime breach; the Christoffersen conditional test [Course Book §17.2] has not been run.	[Course Book §9.2]
Envelope net-of-cost P&L	Never validated	Not a negative result — an absent one; this is exactly the undecided hypothesis below, not a completed test.	[Course Book §14.7]
Hit-rate vs. expectancy	High hit-rate with negative expectancy is a live structural possibility on this cost surface	Hit rate is not the acceptance metric anywhere in the programme; net-of-cost expectancy is.	[Course Book §14.1]

Table 1.5. The verified findings register. Every later stage gate in Table 1.1 is a refinement of, or a response to, one of these four rows.

These four rows compress to one open question, stated formally exactly once because every stage gate in this book is an attempt to answer it.

Definition — The Stage-0 hypothesis

There exists a sizing policy $m(\cdot)$, a function only of envelope-derived features (band width, width change, position-in-band, regime), whose out-of-sample, cost-net performance across CPCV paths exceeds constant-fractional-risk and volatility-target sizing, with a Deflated Sharpe Ratio above 0.5 under the full trial ledger. [Course Book §24.1]
Until Chapter 5 demonstrates this, the envelope's profitability is an inference from the median's failure to beat a random walk — not a measured fact.

Each row of Table 1.5 points at exactly one piece of Stage −1 or Stage 0 work: the median's failure is why Stage 0 tests sizing policies rather than point accuracy; the unconditional coverage figure is why Stage 1 exists at all rather than keeping `width_mult` forever; the absent P&L validation is Stage 0's entire subject; and the hit-rate/expectancy gap is why Table 1.6's rules bind the Finder GA before it is allowed near real capital.

Trap — `forecasts_hist.sqlite` is not a Stage-0 shortcut

It is tempting to point Stage 0 at `forecasts_hist.sqlite` because it already exists and looks like exactly the stored-envelope history Chapter 5 wants. It is the wrong table as it stands: it covers 2 of 7 timeframes and 10 of 21 instruments, is roughly a month stale, and was generated at context 1024 against a live H1 context of 1536 — a different model input, not a snapshot of the same one. A verdict built on it would be a verdict about a system that no longer runs. Table 1.4's fix stands: regenerate at live settings before Stage 0 reads a single row.

1.4 Roles and Instruments of Honesty

Four rules bind the programme regardless of stage or version. None is a suggestion; each has a named failure mode in the Course Book that motivates it.

Rule	Statement	CB
Ledger ownership	One named owner, accountable for the trial ledger's completeness; on a one-person programme, one hat worn on a fixed schedule, never "whoever is free."	[Course Book §15.6]
Kill-wording sign-off	Every stage's kill wording is written, dated and signed before that stage's first run — never amended after seeing a result.	[Course Book §24.2]
Finder-GA quarantine	The Finder GA may not touch envelope outputs, or size a dollar of capital from them, until SPA/Romano–Wolf family-wise gating is live.	[Course Book §15.6]
Retrain-is-a-new-strategy	Every retrain of the Stage-3 model re-enters the pipeline as a new strategy: a new ledger row, a new trial count N, never a silent update to an existing one.	[Course Book §24.2]

Table 1.6. The four standing rules of honesty. All four are enforced mechanically by the harness of Chapter 4, not by memory.

Table 1.7 makes the separation of duties explicit. On a one-person programme every hat is worn by the same person on a fixed schedule [Course Book §15.6] — but the schedule, and the rule that no hat reviews its own work in the same sitting, is not optional.

Hat	Responsible for	Must not, in the same sitting
Researcher	Designing and running candidate configurations.	—
Ledger owner	Recording every trial — kept, abandoned or crashed — before its result can influence whether it gets recorded.	Choose which trials to omit after seeing their results.
Sign-off authority	Approving and dating each stage's kill wording before that stage's first run.	Amend a kill wording after that stage's first result exists.
Gate auditor	Confirming DSR, PBO and SPA are computed against the full ledger before any capital moves.	Be the researcher who built the champion under judgement.

Table 1.7. Separation of duties, one row per hat. A one-person programme wears all four; the table still applies, one hat at a time.

Key idea — Pre-commitment

No threshold, kill wording, or ledger status is ever chosen after seeing the result it would judge. Every stage's gate (Table 1.1) and every version's kill wording (Table 1.2) is written, dated and signed before that stage's or version's first run — because [Course Book §15.4]'s proof is that a judgement formed after seeing results cannot be trusted, even by its author, and no amount of goodwill repeals it.

Step 1.2 — Name the ledger owner and sign the kill-wording protocol

Do. Before Stage 0's first run: assign one named individual (or, on a one-person programme, one hat worn on a fixed schedule) as ledger owner per [Course Book §15.6]; draft Stage 0's kill wording from Table 1.1's question and default kill; date-stamp and sign it. Repeat for Stage 1's wording before Stage 1's first run, and so on for every subsequent stage and version — a kill wording is never drafted more than one stage ahead of the run it governs, because a stage's own result can change what its successor should measure.

Artifact. A signed, dated kill-wording document per stage (`killcriterion_stageNN_vYYYYMMDD.md` or equivalent), stored in version control and immutable after signing; a one-line ledger-owner record filed alongside the trial ledger.

Pass / fail. No stage's harness executes until that stage's kill wording exists, is dated strictly before the harness's first timestamped run, and names an owner. A harness run without a signed wording on file is void regardless of its result.

1.5 Conventions

Fixed once here, used without re-explanation everywhere else in this book.

Units

Numerical precision

Every computed number in this playbook — bar counts, coverage fractions, deviations — is produced by its chapter's `verify_pbNN.py` script and transcribed at four significant figures, the same convention the Course Book states and uses throughout [Course Book §1.1]. Parameter recommendations (a γ, a shrinkage constant, a lookback window) are judgements, not computations; this playbook states them as recommendations with a one-line reason, never as if they were derived. Every label in this book is prefixed `pb01:` for this chapter (tables `tab:pb01:...`, figures `fig:pb01:...`); the Course Book's own `chNN:`-prefixed labels are never imported directly, only pointed to via a `\cbref` chip.

Unit	Definition	Use in this book	CB
bp	0.01% = 0.0001 in decimal return terms	Universal; comparable across every instrument and timeframe.	[Course Book §1.1]
pip	Instrument-specific minimum quoted increment	NOT comparable across instruments — a pip of XAU/USD, USD/JPY and BTC/USD are different amounts of money and of risk; never pool pip P&L across the book.	[Course Book §14.1]
R-multiple	X/R, with R the pre-trade risk (entry-to-stop distance × position size)	The book's poolable unit — every cross-instrument, cross-timeframe expectancy figure in this playbook is stated in R.	[Course Book §14.1]

Table 1.8. Units used throughout this playbook.

Timeframes and bar counts

The platform runs seven timeframes, M1 through D, on every instrument. Table 1.9 gives bars per instrument and pooled across the 21-instrument book over three years, under the assumptions stated beneath it; reproduced from and verified identical to [Course Book §6.8]'s data-wall table, computed independently in `verify_pb01.py` § 1.

Timeframe	Bar interval	Bars / instrument / 3y	Pooled ×21
M1	1 min	1,104,480	23,194,080
M5	5 min	220,896	4,638,816
M15	15 min	73,632	1,546,272
M30	30 min	36,816	773,136
H1	60 min	18,408	386,568
H4	240 min	4,602	96,642
D	1 trading day	780	16,380
Sum, all seven timeframes			30,651,894

Table 1.9. Bars per instrument and pooled across 21 instruments, three years. Assumptions: 52 weeks/year (156 weeks over 3 years); FX trading week ≈ 118 hours for intraday timeframes (five 24-hour days less ≈ 2 hours of rollover/maintenance gap); 5 daily bars/week for D. The seven rows aggregate one tick stream — an H1 bar is a deterministic function of 60 M1 bars — so the row sum is not seven times the information of any one row [Course Book §6.8].

The instrument roster

Table 1.10 gives the 21-instrument book by class; verified in `verify_pb01.py` § 2. The FX majors/crosses count is derived, not independently stated in either book: [Course Book §22.4]'s master table gives "FX majors/crosses + BTC/ETH + 2 indices (18 of 21)," so FX majors/crosses $= 18 - 2 - 2 = 14$. Class matters beyond bookkeeping: cost structure, covariate set and evaluation quirks all change at the class boundary, never within one, so every stage's split by instrument (Chapter 4's harness, Chapter 16's CPCV grouping) may subdivide within a class but must never silently pool across one [Course Book §22.4].

The session clock

Table 1.11 gives nominal session windows; verified in `verify_pb01.py` § 6. These are standard-time approximations, not a derived quantity: Sydney (AEDT), London (BST) and New York (EDT)

Class	Count	Notes	CB
FX majors/crosses	14	Full roster is the live OANDA instrument config, not reproduced here; pairs cited in the Course Book include EURUSD, GBPUSD, USDJPY, AUDUSD, EURGBP, EURJPY, GBPJPY.	[Course Book §22.4]
Crypto	2	BTC/USD, ETH/USD.	[Course Book §22.4]
Indices	2	Two index CFDs.	[Course Book §22.4]
Metals	2	XAU/USD, XAG/USD.	[Course Book §22.4]
Energy	1	WTI.	[Course Book §22.4]
Total	21	$21 \times 7 = 147$ (instrument, timeframe) series.	[Course Book §16.5]

Table 1.10. The 21-instrument book by class.

each observe daylight saving on their own calendar, shifting every boundary below by up to one hour for part of the year. The platform's own trading-session-flag covariate [Course Book §8.4] must resolve against the exchange's calendar, never against this table directly. The clock matters beyond labelling a covariate: spread and liquidity vary by session at least as much as by instrument, so the cost surface Stage −1a builds must be sliced by session, not averaged across it, or a wide-session cost will mask a narrow-session one at exactly the hours a strategy is most likely to trade [Course Book §14.2].

Session	UTC (nominal)	AEST, UTC+10 (nominal)
Sydney	22:00–07:00	08:00 (+1d)–17:00
Tokyo	00:00–09:00	10:00–19:00
London	08:00–17:00	18:00–03:00 (+1d)
New York	13:00–22:00	23:00–08:00 (+1d)

Table 1.11. Nominal session windows, UTC and AEST (standard time, no daylight saving applied).

File and versioning conventions for this playbook's artifacts

Artifact	Convention	Where
Verify-script output	`verify_pbNN.py` console log; every printed value is transcribed into `pbNN.tex`, never hand-computed.	This book's source tree (build-time only).
Trial-ledger row	Content-hash-keyed, append-only; one row per trial; status is kept, abandoned or crashed — never deleted.	Platform SQLite store [Course Book §15.6].
Kill-wording document	`killcriterion_stageNN_vYYYYMMDD.md`, signed, immutable after signing.	Version control.
Frozen model artifact	`<name>_vN.onnx` plus a config hash; every retrain bumps N, never overwrites.	Model registry [Course Book §24.2].

Table 1.12. Naming and storage conventions for what this playbook produces.

Notation carried over from the Course Book

Table 1.13 is a lookup, not a teaching aid: the symbols and acronyms this playbook uses without re-explaining, each with the one section that defines it. The Course Book's own full symbol table is [Course Book App. A].

Symbol	Meaning	CB
$\hat{\sigma}$	Conditional volatility estimate (GARCH/HAR-RV) driving the Stage-1 width.	[Course Book §12.6]
$m(\cdot)$	Sizing policy: a function only of envelope-derived features.	[Course Book §24.1]
DSR	Deflated Sharpe Ratio — Sharpe corrected for the trial count N.	[Course Book §15.4]
PBO	Probability of Backtest Overfitting.	[Course Book §15.5]
CPCV	Combinatorial Purged Cross-Validation.	[Course Book §16.4]
R	Pre-trade risk unit (Table 1.8).	[Course Book §14.1]
$q_{10}, q_{20}, q_{80}, q_{90}$	TimesFM's quantile outputs, per (instrument, granularity).	[Course Book §9.2]
`width_mult`	Static per-pair band-width multiplier, $\in [0.95, 1.07]$.	[Course Book §9.2]
H	Forecast horizon: 50 bars, on every timeframe.	[Course Book §16.5]

Table 1.13. Notation used without re-explanation from Chapter 2 onward.

1.6 The Dependency Map

Figure 1.1 draws Tables 1.1 and 1.2 as one diagram: the spine is v1, gated stage by stage; the three later versions hang off the stage each one's edge thesis actually depends on, not off the calendar. Read the spine left to right for build order and each version node's arrow as its one upstream dependency: a box with no incoming arrow at all (Stage −1) has none, which is why Table 1.2 calls v1 the only version that waits on nothing else in this diagram.

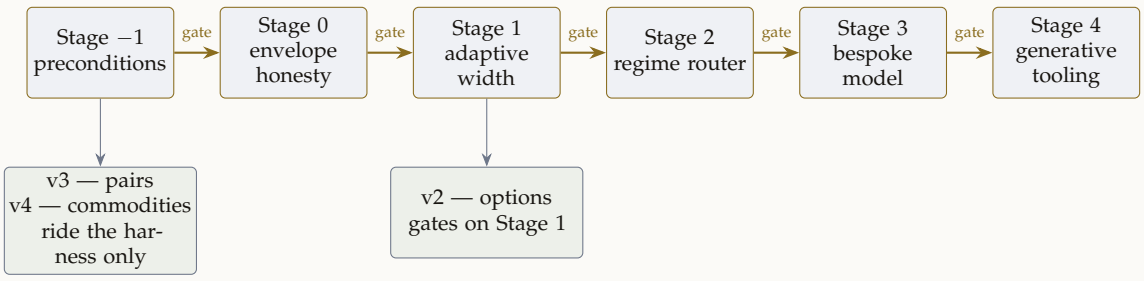
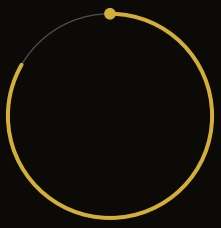


Figure 1.1. The programme's dependency map. The spine (Stage −1 through Stage 3) is v1 — FX CFDs, the only version with no upstream dependency (Table 1.2). Stage 4 is shared tail-risk tooling, gated on its own terms but not version-specific. Each "gate" arrow is one row of Table 1.1; each version node is one row of Table 1.2.

Chapters 2 through 10 execute everything promised above, one stepbox at a time. Nothing in them contradicts this chapter; where a later step needs a number stated here, it cites the table, not memory.

PART II

Preconditions: Stage -1



LUNA LABS

2

Stage −1a: The Cost Surface

Every P&L figure this programme will ever compute is conditional on one number it does not yet have: what a round trip actually costs, at the moment it is placed, not on average. `candles_3y` *stores open, high, low, close and volume — never bid, never ask — so every historical spread was discarded at the moment it was aggregated into a mid-price bar* [Course Book §14.2]. *This chapter rebuilds it from nothing: a fresh bid/ask pull, a conditioning scheme tying spread to the hour of day and to scheduled news, an honestly measured slippage term, and a frozen, versioned surface that every backtest from Chapter 5 onward looks up and never re-derives. Ten steps, in order. Nothing here is estimated twice, and nothing here is re-estimated inside a training loop —* `cost_surface` *is built once per vintage, by this chapter's machinery, and consumed everywhere else by lookup.*

2.1 The schema and the raw pull

Two artifacts before anything can be aggregated: the table every later chapter queries, and the bid/ask history it is computed from.

Step 2.1 — Define the surface schema

Do. Fix the grain every downstream stage already assumes before a single row exists: Chapter 5's execution engine and Chapter 8's training pipeline both look this cost up as $c(i, \text{cell}(t))$, with $\text{cell}(t) = (\text{session_bucket}(t), \text{news_flag}(i,t), \text{rollover_flag}(t))$ — so the grain is fixed here, once, and never renegotiated downstream.

Column	Type	Meaning
`instrument`	TEXT	OANDA native symbol, underscore form (`EUR_USD`, `XAU_USD`, ...)
`session_bucket`	TEXT	one of 8 values, Step 2.3
`news_flag`	INTEGER	0/1, Step 2.4
`rollover_flag`	INTEGER	0/1, Step 2.3 (narrow, independent of `session_bucket`)
`spread_med, spread_p75, spread_p95`	REAL	pips, empirical quantiles, Step 2.5
`n_obs`	INTEGER	row count backing the cell, Step 2.5
`vintage`	TEXT	build stamp, Step 2.10

`PRIMARY KEY (instrument, session_bucket, news_flag, rollover_flag, vintage)`. Five more columns — `fallback_level`, `slip`, `cost_base`, `cost_stress_15x`, `cost_stress_20x` — are added at the steps that populate them (2.5, 2.6, 2.8); a column is declared where it is first written, not speculatively here.

Five queries this schema must answer without a schema change, because they are the only five ways Stages 0–4 ever touch it:

Q1 *Per-bar lookup* (Stages 0–3): `SELECT spread_med, spread_p75, spread_p95,`

`cost_base, cost_stress_15x, cost_stress_20x FROM cost_surface WHERE instrument=:i AND session_bucket=:b AND news_flag=:n AND rollover_flag=:r AND vintage=:v`.

Q2 *Regime aggregation* (Stage 2): group cells by a router regime label, compute the `n_obs`-weighted mean of `spread_med` across member cells.

Q3 *Training-time materialisation* (Stage 3): a build-time join of every (instrument, decision timestamp) training example against its cell, written once into `kappa` of Stage 3's tensor table — never a per-batch lookup.

Q4 *Stress sweep* (Stage 0): Q1 selecting `cost_stress_15x/20x` in place of `cost_base`, equivalently the base column scaled by λ at read time (Step 2.8).

Q5 *Ledger audit* (Step 2.10): `SELECT DISTINCT vintage FROM cost_surface`, checked against every backtest's declared `cost_vintage` before its result is trusted.

Artifact. `cost_surface.sqlite`, table `cost_surface`, schema above, no rows yet.

Pass / fail. Every one of Q1–Q5 can be written against the schema with no join beyond `calendar` (Step 2.4) and `candles_ba` (Step 2.5). If a query needs a column this table does not have, the schema is wrong, not the query.

Step 2.2 — Re-pull bid/ask candles

Do. Pull OANDA `price=BA`, granularity `M1`, 3 years, all 21 instruments (Chapter 1's roster: 14 FX majors/crosses, 2 crypto, 2 indices, 2 metals, 1 energy) — all 21, not v1's 18, because this pull is shared infrastructure and v4 needs XAU/XAG/WTI later; re-pulling a second time is strictly more expensive than over-scoping once.

Column	Type	Meaning
`instrument`	TEXT	OANDA native symbol
`ts`	INTEGER	unix seconds UTC, bar-*open* instant (OANDA convention)
`bid_o, bid_h, bid_l, bid_c`	REAL	bid OHLC
`ask_o, ask_h, ask_l, ask_c`	REAL	ask OHLC
`volume`	INTEGER	ticks in the bar

`PRIMARY KEY (instrument, ts)`. `ts` is the bar's *open* instant, so the bar's close is the next row's `ts` when the pull is gap-free; Step 2.6's $t{+}1$-open fill is therefore literally the next row's `ask_o/bid_o`, no offset arithmetic, provided this step's gap audit is clean at that timestamp.

Chunked pull. OANDA's v20 candle endpoint caps a request at 5,000 candles. Expected M1 bars/instrument/3y = 1,104,480 (156 weeks $\times$118 trading hours/week $\times$60 [Course Book §6.8]), so requests/instrument = $\lceil 1{,}104{,}480/5{,}000 \rceil$ = 221, total requests across 21 instruments = 221×21 = 4,641. Page by `ts` in bar-index order and persist a resume cursor `(instrument, last_successful_ts)` so a failed pull resumes rather than restarts.

Pip size. Not hand-maintained: read `pipLocation` from OANDA's `GET /v3/accounts/{id}/instruments` response at pull time and store `pip_size` = $10^{\texttt{pipLocation}}$ per instrument in `instrument_meta(instrument, pip_location, pip_size, refreshed_at)`, refreshed at every cost-surface build — never inlined as a literal, because pip conventions differ across the platform's five instrument classes and a hand-maintained value silently goes stale if OANDA reclassifies an instrument's display precision.

Integrity checks, per instrument. (i) Row count against 1,104,480, tolerance $+0/-0.5\%$ ($-5{,}522$ bars, accept range $[1{,}098{,}958,\ 1{,}104{,}480]$) — one-sided, because a count *above* 1,104,480 means duplicate rows from a chunk-boundary bug, not a bigger pull. Any instrument outside the

band, or listed on the platform for less than the full 3 years, gets a full gap audit rather than a blind pass/fail: Chapter 18's own worked example of the survivorship trap is this exact platform's instrument list applied to years before some instruments were listed [Course Book §18.1.6] — a genuinely short history is not a pull defect and must not be forced to match. (ii) Gap audit: generate the expected M1 timestamp grid over the pull window excluding the weekly closure (Friday 22:00 UTC to Sunday 22:00 UTC, winter anchor), left-join actual rows, and bucket every missing timestamp by the session bucket of Step 2.3. A clean pull shows gaps concentrated in `rollover_hour` and `weekend_open`, summing close to the 120 bars/week (2 hours) the platform's own trading-week convention already prices in [Course Book §6.8]; any gap outside those two buckets is a pull defect and blocks Step 2.9's validation.

The mathematics. $s_t = A_t - B_t$ at each M1 close (A_t =`ask_c`, B_t =`bid_c`), converted to pips by s_t/`pip_size`. [Course Book §14.2] [Course Book §1.1]

Artifact. `candles_ba.sqlite` (table `candles_ba`, $\leq$ 23,194,080 rows) and `instrument_meta`.

Pass / fail. All 21 instruments present; row counts within tolerance or explained by a documented listing-date shortfall; gap audit shows no unexplained gap outside `rollover_hour/weekend_open`.

2.2 Conditioning the surface

Section 2.1's pull is unconditional — one row per instrument per minute. This section conditions it: which minutes belong together for cost purposes, and which minutes sit near scheduled news, because both change the distribution of the spread the next section aggregates.

Step 2.3 — Session bucketing

Do. Chapter 1's session-clock table gives four nominal, overlapping windows in standard (winter) UTC: Sydney 22:00–07:00, Tokyo 00:00–09:00, London 08:00–17:00, New York 13:00–22:00 — stated there as an orientation approximation, with the DST-aware, mutually exclusive resolution explicitly deferred to wherever the platform's session-flag covariate is actually built. This step is that construction.

DST rule. Two independent calendars matter: the US (New York) and the UK (London); Tokyo has no DST and Sydney's AEDT/AEST affects only Chapter 1's AEST column, not the UTC boundaries partitioned here. US DST runs 2nd Sunday of March to 1st Sunday of November; UK DST runs last Sunday of March to last Sunday of October. Under DST, New York is UTC−4 and London UTC+1; under standard time, UTC−5 and UTC+0.

Assignment, by priority (highest first; every M1 timestamp takes the first rule it matches).

- **P1** `rollover_hour` — the clock hour starting at 5pm New York local, Monday through Thursday's close in the continuous trading week.
- **P2** `weekend_open` — the same clock hour, Sunday only, *replacing* that day's `rollover_hour` slot (the week's reopen).
- **P3** `London_NY_ovl` — New York's open to London's close.
- **P4** `Tokyo_London_ovl` — London's open to Tokyo's close.
- **P5** `NY_late` — London's close to the next `rollover_hour`.
- **P6** `London` — `Tokyo_London_ovl`'s end to `London_NY_ovl`'s start.
- **P7** `Tokyo` — Tokyo's open to `Tokyo_London_ovl`'s start.
- **P8** `Sydney` — the residual: the clock hour between `rollover_hour`/ `weekend_open` and Tokyo's open.

Table 2.1 gives the resulting boundaries and widths for both DST regimes.

The continuous trading week (Sunday 22:00 UTC to Friday 22:00 UTC, winter anchor) is five daily rotations of this 24-hour cycle; `rollover_hour` fires at four of the five rotation boundaries (Monday, Tuesday, Wednesday, Thursday 22:00 UTC), `weekend_open` at the fifth (Sunday); Friday's rotation ends at 22:00 UTC with no further rollover slot, because that instant is the week's close, not an intraday transition. Wednesday's `rollover_hour` is the one a held position pays triple financing against (Step 2.7) — the standard CFD weekend value-date convention.

`rollover_flag` is *independent* of `session_bucket`, not a duplicate of it: `session_bucket=rollover_hour` is the fixed 60-minute bucket above, on the same hour-granular boundaries as the other seven; `rollover_flag`$=1$ is a narrower, separately-configurable $\pm$10-minute window (20 minutes wide, a strict subset) around the exact instant the platform's own swap-application log records — kept separate so Stage 1+ can isolate the effect at finer resolution without redefining the eight-way partition. `rollover_flag`$=1 \Rightarrow$`session_bucket`$\in$ {`rollover_hour`, `weekend_open`}, never the converse. Figure 2.1 draws the winter cycle schematically.

The mathematics. session_bucket(t) = the first rule P1–P8 that t matches — a piecewise assignment, not an equation to solve. The one arithmetic constraint it must satisfy: bucket widths sum to 24h in *every* DST regime (Table 2.1: both rows sum to 24).

Artifact. `session_windows(bucket, dst_regime, utc_start, utc_end)`, 16 rows (8 buckets $\times$ 2 regimes); the assignment function as code, keyed off this table plus the two DST calendars.

Pass / fail. For a full synthetic week of M1 timestamps in each DST regime, every timestamp maps to exactly one bucket (no gap, no double assignment); per-bucket nominal weekly bar counts match Table 2.1's 60$\times$width figures exactly (verified in `verify_pb02.py` §5); the nominal weekly total (7,200 bars) exceeds the platform's 118-hour/week convention (7,080 bars) by exactly 120 bars (2 hours), matching Chapter 1's stated rollover/maintenance gap [Course Book §6.8] rather than contradicting it.

Bucket	Winter UTC	h	Summer UTC	h
`rollover_hour`	22:00–23:00	1	21:00–22:00	1
`Sydney`	23:00–00:00	1	22:00–00:00	2
`Tokyo`	00:00–08:00	8	00:00–07:00	7
`Tokyo_London_ovl`	08:00–09:00	1	07:00–09:00	2
`London`	09:00–13:00	4	09:00–12:00	3
`London_NY_ovl`	13:00–17:00	4	12:00–16:00	4
`NY_late`	17:00–22:00	5	16:00–21:00	5
Sum		**24**		**24**

Table 2.1. Bucket boundaries, both DST regimes; `weekend_open` takes `rollover_hour`'s width (1h) on its single weekly occurrence. Verified in `verify_pb02.py` §4.

Trap — DST moves the rollover hour — twice a year, and twice inconsistently

5pm New York is 22:00 UTC under EST and 21:00 UTC under EDT: every boundary in Table 2.1 shifts by exactly one hour, twice a year, at the US transition dates. The sharper trap is that the US and UK do not change clocks on the same date. In 2026 the US moves to EDT on 8 March but the UK does not move to BST until 29 March — a 21-day window where New York is already on summer time and London is still on winter time, so neither this

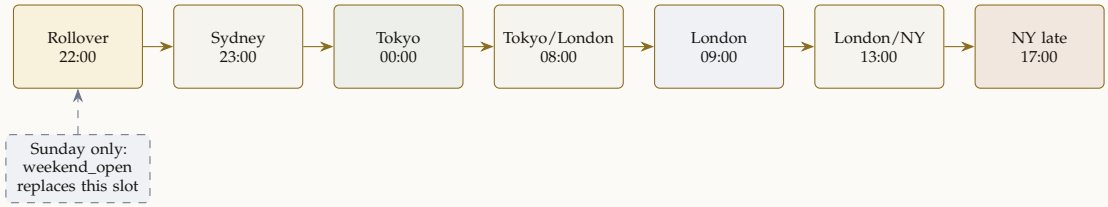


Figure 2.1. The eight-bucket daily cycle, winter UTC, schematic (not to scale). Each label gives the bucket's start time; the next node's start time is the previous bucket's end. The cycle repeats Monday–Friday; `weekend_open` substitutes for `rollover_hour` once, on Sunday.

step's "winter" nor "summer" row of Table 2.1 is exactly right: the `London_NY_ovl` boundary implied by New York's clock differs by an hour from the boundary implied by London's. The reverse mismatch is shorter but not absent: the UK reverts to GMT on 25 October 2026, a week before the US reverts to EST on 1 November — a 7-day window with the opposite skew. `rollover_hour`, anchored to New York alone, is unaffected by London's state in both windows; `London_NY_ovl` and `Tokyo_London_ovl` are not, and an assignment function that reads one global "is it DST season" flag rather than each city's own transition date will misclassify bars in both windows, every year, on a schedule computable in advance — it must not be discovered by inspecting a coverage report.

Step 2.4 — News windows

Do. Build `calendar`:

Column	Type	Meaning
`event_id`	INTEGER	
`currency`	TEXT	NULL iff `instrument_scope` is set
`instrument_scope`	TEXT	NULL iff `currency` is set (e.g. an instrument-only event)
`event_name`	TEXT	
`tier`	INTEGER	1 = flagged by this step; 2+ = ignored
`scheduled_ts_utc`	INTEGER	
`actual_ts_utc`	INTEGER	NULL until release
`snapshot_ts_utc`	INTEGER	when *this row* was captured

`PRIMARY KEY (event_id, snapshot_ts_utc)`.

Tiering rule. Tier 1 = scheduled central-bank rate decisions, headline CPI, and (US only) Non-Farm Payrolls, for the eight currencies the platform's instruments quote; everything else (secondary indicators, speeches, revisions) is tier 2+ and never sets `news_flag`. WTI carries one instrument-specific addition, the EIA Weekly Petroleum Status Report, because it moves that one instrument on a schedule no currency-level tier captures.

Currency	Tier-1 events
USD	FOMC rate decision; Non-Farm Payrolls; CPI (headline)
EUR	ECB rate decision; flash HICP
GBP	BoE rate decision; CPI
JPY	BoJ rate decision
AUD	RBA rate decision
CAD	BoC rate decision; Employment Change
NZD	RBNZ rate decision
CHF	SNB rate decision

Currency mapping ccy(i): an FX pair maps to both legs (e.g. `EUR_USD` → {EUR, USD}); `XAU_USD`, `XAG_USD`, `BTC_USD`, `ETH_USD` map to {USD}; the two index CFDs map to their home currency; WTI maps to {USD} plus its own `instrument_scope` row above.
Join. news_flag(i,t) = 1 iff ∃ a `calendar` row with tier= 1, (currency ∈ ccy(i) or `instrument_scope`= i), and $|t - \texttt{scheduled_ts_utc}| \leq 15$ minutes — evaluated against the snapshot that was current as of t's own backtest-time horizon, never a snapshot captured after t: Chapter 18's as-of predicate applied to the calendar itself, not only to macro feeds [Course Book §18.2].
Point-in-time requirement. A calendar vendor that silently corrects a historical event's scheduled time or tier after the fact (common when a vendor reclassifies importance from the market reaction it caused) reproduces exactly FRED's mutable-current-value problem [Course Book §18.2]. `calendar` is therefore append-only: a reschedule inserts a new row with a new `snapshot_ts_utc`, never an `UPDATE` to `scheduled_ts_utc` in place, and a historical t's `news_flag` always reconstructs from the snapshot nearest to, and not after, t.

The mathematics. news_flag(i,t) = $\mathbf{1}\{\exists$ tier-1 row, currency match, $|t - \texttt{scheduled_ts_utc}| \leq 15\text{min}\}$ — [Course Book §18.2]'s as-of predicate with the calendar as the feed.

Artifact. `calendar.sqlite` (table `calendar`).

Pass / fail. Five known Tier-1 releases (e.g. an FOMC date) checked by hand: the flag is 1 for exactly the 30 minutes surrounding each, 0 immediately outside it; no row has a NULL `scheduled_ts_utc`; a reschedule test (insert a corrected row for a past event) leaves the *original* snapshot's `news_flag` reconstruction unchanged — only queries run after the correction's own `snapshot_ts_utc` see the new time.

2.3 Aggregating, costing, and composing

Every cell now has a well-defined population of M1 rows. This section turns that population into numbers: the measured spread, the measured slippage, the contractual commission and financing, and the single composed figure Chapters 5 and 8 actually consume.

Step 2.5 — Aggregate the surface

Do. For each (instrument, session_bucket, news_flag, rollover_flag) at the candidate vintage, pull every `candles_ba` row whose `ts` maps to that cell (Steps 2.3– 2.4), convert each row's spread to pips (s_t/`pip_size`), sort, and take order statistics. `n_obs` is the row count. If `n_obs`< 100, fall back through coarser cells in this order, recording which level actually supplied the numbers in `fallback_level`:

Level	Grain	Drops
L0	instrument, session_bucket, news_flag, rollover_flag	(finest — nothing)
L1	instrument, session_bucket, news_flag	`rollover_flag`
L2	instrument, session_bucket	`news_flag` too
L3	instrument	`session_bucket` too (pooled over the full 3y)

A cell still below 100 at L3 is marked `fallback_level`= 4 (UNAVAILABLE) and excluded from Stage 0+ until more history accumulates; this should not occur for any of the 21 instruments over 3 years (Step 2.9c). Drop `rollover_flag` *before* `news_flag`: Step 2.4's own arithmetic shows the (`rollover_hour`, news= 1) and (`weekend_open`, news= 1) intersections

are the ones expected to be thin — Tier-1 releases are scheduled 12:30 or 14:00 UTC, never inside the 21:00–23:00 UTC rollover/weekend_open window — so collapsing `rollover_flag` first repairs exactly those cells while keeping the news conditioning Stage 0 most wants.

The mathematics. $\hat{q}_\tau = x_{(k)}$, $k = \lceil n\tau \rceil$, on the sorted per-cell pip-spread sample; `spread_med`$=\hat{q}_{0.5}$, `spread_p75`$=\hat{q}_{0.75}$, `spread_p95`$=\hat{q}_{0.95}$. [Course Book §2.3.3]

Artifact. `cost_surface` rows: `spread_med`, `spread_p75`, `spread_p95`, `n_obs`, `fallback_level`, for every combination reachable by the fallback (up to L3).

Pass / fail. Every combination for every instrument resolves at `fallback_level`≤ 3, or the exceptions are named and excluded before Step 2.9; L2-level cells never fail $n \geq 100$ — even the thinnest single bucket, `weekend_open`, carries 9,360 bars/3y unconditionally, 93.6× the floor (`verify_pb02.py` §5).

Step 2.6 — Slippage term

Do. For every `candles_ba` row t with a valid, gap-free next row $t{+}1$, compute the two one-sided adverse gaps and assign each to cell(t) exactly as Step 2.5's spread aggregation does (same fallback order if thin):

$$\text{adverse_gap_buy}(t) = \texttt{ask_o}(t{+}1) - \texttt{ask_c}(t),$$
$$\text{adverse_gap_sell}(t) = \texttt{bid_c}(t) - \texttt{bid_o}(t{+}1).$$

slip(cell) $= \left[\mathbb{E}[\text{adverse_gap_buy} \mid \text{cell}] + \mathbb{E}[\text{adverse_gap_sell} \mid \text{cell}]\right]/2$, converted to pips. Calibrate/sanity-check against realised `spread_cost` in the platform's historical fills table *with* the caveat below: those fills exist only where the *old* system fired, so they estimate $\mathbb{E}[\text{cost} \mid A_{\text{old}}]$, never $\mathbb{E}[\text{cost}]$ or $\mathbb{E}[\text{cost} \mid A_{\text{new}}]$ for any policy Stage 0 onward will size [Course Book §14.1]. Use them to bound plausibility (Step 2.9a); never let them overwrite the bid/ask-measured slip(cell).

The mathematics. adverse_gap_buy$(t) = A^{\text{open}}_{t+1} - A^{\text{close}}_t$, adverse_gap_sell$(t) = B^{\text{close}}_t - B^{\text{open}}_{t+1}$ — the same decision-close-to-$t{+}1$-open gap Chapter 18's phantom-edge arithmetic prices, measured per cell rather than averaged over the whole book. [Course Book §18.3]

Artifact. `cost_surface.slip`, populated per cell, same grain, same vintage.

Pass / fail. slip(cell) ≥ 0 on average for every cell (a negative mean would mean the gap is systematically favourable, i.e. the fill convention is inverted for that cell — investigate before accepting); every cell reuses `fallback_level` from Step 2.5 rather than computing its own — spread and slip must never disagree about which cell they describe.

Trap — The old system's fills are not a free calibration set

Realised fill costs are tempting to calibrate against directly: they are real, they exist, and they need no bid/ask re-pull. They are also conditioned on the old system's firing rule, and Chapter 14's own conditional-cost identity, $\mathbb{E}[C \mid A] = \mathbb{E}[C] + \text{Cov}(C, \mathbf{1}\{A\})/\mathbb{P}(A)$ [Course Book §14.1], is exactly the statement that this matters: if the old rule fired disproportionately in wide-spread conditions (or disproportionately avoided them), its realised fills estimate $\mathbb{E}[C \mid A_{\text{old}}]$, a number with no guaranteed relationship to $\mathbb{E}[C \mid A_{\text{new}}]$ for whatever policy Stage 0 sizes. Overwriting slip(cell) with the realised figure imports the old system's selection pattern into a surface meant to be policy-agnostic. Use the realised numbers as a bound (Step 2.9a's reconciliation, reviewed, wide tolerance) — not as the estimator.

Step 2.7 — Commission and financing

Do. `commission_per_side`, flat per instrument class, default $0 for all five of the platform's classes under OANDA's spread-inclusive Standard pricing — *confirm against the live account terms before assuming zero*, and store the confirmed value, not the assumption, in `cost_params(instrument_class, commission_per_side, source, checked_at)`. Financing (needed for v4 and any multi-day hold): `financing_rates(instrument, side, rate_pa, date)`, one row per (instrument, long/short, calendar date), refreshed daily from OANDA's published financing schedule; Wednesday's rate is charged at 3× to cover the weekend value-date roll (standard CFD convention — confirm against the live schedule before hard-coding the multiplier).

The mathematics. $\text{financing} = \text{notional} \times \text{rate}_{\text{p.a.}} \times (\text{hours held}/24)/365$. [Course Book §14.2]

Artifact. `cost_params`, `financing_rates`.

Pass / fail. Reproducing the formula on notional $100,000, rate 1.5%, 25 hours held gives $4.2808 = 0.4281 pips at $10/pip — matches the Course Book's own worked figure exactly (`verify_pb02.py` §7); `commission_per_side` is present and overridable per class, never silently assumed zero for a class never actually checked.

Step 2.8 — Compose round-trip cost

Do. Add three columns to `cost_surface`, at the same grain as Steps 2.5– 2.6:

$$\texttt{cost_base} = \texttt{spread_med}(\text{cell}) + 2 \times \texttt{slip}(\text{cell}) + \texttt{commission_per_side}(\text{class}(i)),$$
$$\texttt{cost_stress_15x} = 1.5 \times \texttt{cost_base}, \qquad \texttt{cost_stress_20x} = 2.0 \times \texttt{cost_base}.$$

Stressing the whole composed figure — not spread and slip alone, leaving commission untouched — is a deliberate simplification: commission is contractually known, not sampling error, so stressing it is slightly conservative, but it matches Chapter 5's own harness exactly, which multiplies a single $c(i, \text{cell}(t))$ by $\lambda \in \{1.0, 1.5, 2.0\}$ at read time [Course Book §18.3]. Storing the two stressed columns is a convenience and an audit trail — so a downstream reader need not re-derive $\lambda \times c$ and risk a transcription mismatch — mathematically identical to that runtime scaling.

The mathematics. $c(i,t) = \text{spread}(\text{cell}(t)) + 2\,\text{slip}(\text{cell}(t)) + \text{commission}$; stress variants at $1.5c$ and $2c$, the same multiples Chapter 14's own cost-stress worked example uses. [Course Book §14.2]

Artifact. `cost_surface`, fully populated: Step 2.1's base columns + `fallback_level` (2.5) + `slip` (2.6) + this step's three composed-cost columns.

Pass / fail. `cost_stress_15x/cost_base`= 1.5 and `cost_stress_20x/ cost_base`= 2.0 to within 10^{-9} on every row (Chapter 5's own cost-reconciliation identity depends on exactly this linearity); `cost_base`> 0 for every populated cell.

2.4 Validation and the freeze

Step 2.9 — Validate

Do. Three checks, per instrument, against the candidate vintage before it is frozen:
(a) Reconciliation vs. realised fill costs. Where the platform's fills table has entries, compare `cost_base`(cell) against the realised `spread_cost` on those fills, cell by cell; flag (not auto-

fail) any cell whose absolute relative difference exceeds 25% — a recommendation, not a computed threshold, kept wide because Step 2.6's trapbox means the comparison set is itself selection-biased and a tight band would penalise the surface for being *more* representative than the biased comparison.

(b) Sanity vs. indicative public spreads. Pull OANDA's publicly advertised typical/minimum spreads for a sample of instruments (labelled indicative — a best-case marketing figure, not a session-weighted median); expect `cost_surface`'s `London_NY_ovl`-bucket `spread_med` (the tightest, most liquid bucket) to sit at or above the indicative figure for every instrument. A violation in the *other* direction (measured below indicative) is not economically explicable and is a pull or aggregation bug, not a pass.

(c) Coverage. Fraction of historical M1 bars, per instrument, whose cell resolved at `fallback_level` ≤ 3 — target $\geq 99\%$. Step 2.3's own arithmetic already shows every bucket clears $n \geq 100$ unconditionally (even `weekend_open`'s 9,360 bars/3y is 93.6× the floor), so a coverage failure below 99% signals a pull gap (Step 2.2), not a genuinely sparse cell.

The `.json` twin's per-instrument record carries exactly these fields, so the gate check of the gatebox below is a machine query, not a manual read of the `.md`:

```
{"instrument": "EUR_USD", "vintage": "cost_v1_20260818",
 "coverage_pct": 99.42, "coverage_pass": true,
 "reconciliation_flags": [{"cell": "London_NY_ovl/0/0", "rel_diff_pct": 31.2}],
 "sanity_violations": [], "disposition": "accepted; see note #14"}
```

The mathematics. $\text{coverage} = \dfrac{\sum_{\text{cells, fallback_level}\leq 3} n_{\text{obs}}}{\sum_{\text{all cells}} n_{\text{obs}}}$, per instrument.

Artifact. `validation_report_<vintage>.md` (and a machine-readable `.json` twin), one section per instrument, checks (a)–(c).

Pass / fail. (c) $\geq 99\%$ for all 21 instruments is a hard gate (gatebox below); (a) and (b) are reviewed, not auto-failed — every flagged cell is read by the ledger owner before the vintage is frozen, and a documented decision (accept / re-pull / widen fallback) is attached to the report.

Step 2.10 — Freeze and version

Do. Once Step 2.9 passes, stamp the vintage: format `cost_v<N>_<YYYYMMDD>` (e.g. `cost_v1_20260818`), N incrementing per rebuild, dated at build completion. `cost_surface` rows are append-only, the same discipline Chapter 1 fixes for every artifact this playbook produces — deleting or overwriting a row destroys evidence of what the surface once said, which is exactly the failure Chapter 18 names a leak when it happens to a trial ledger [Course Book §18.1.10]: a new pull, a new fallback rule, or a new slip calibration is a *new* vintage with a new N, never an `UPDATE` to an existing vintage's rows. Every downstream artifact that consumes a cost figure records which vintage it used — Stage 0's `stage0_results` carries a `cost_vintage` column, Stage 3's per-shard metadata pins `cost_vintage` at build time — a result with no recorded vintage is void regardless of its number, the same rule Chapter 1 applies to an unsigned kill wording.

Artifact. The frozen `cost_surface` vintage (immutable row-set); one row in `cost_vintage_registry(vintage, built_at, validation_report_path, status)`, status $\in$ {frozen, superseded}.

Pass / fail. `SELECT DISTINCT vintage FROM cost_surface` includes the new vintage; a content hash of the new vintage's rows, taken before and after freezing, is identical — nothing wrote to it after the validation report was filed.

Key idea — Measure once, look up everywhere

Every number in `cost_surface` is measured from the bid/ask tape (Steps 2.2, 2.5, 2.6) or read from a contractual source (Step 2.7) — none of it is ever re-estimated inside a backtest or a training loop. Chapter 5 multiplies a stored `cost_base` by λ; Chapter 8 materialises `kappa` from a pinned vintage at build time and never queries `cost_surface` again once training starts. This is not a performance shortcut, it is a consistency requirement: two stages that each re-estimated cost from their own slice of history could disagree about what a spread was on a given day, and every downstream Sharpe ratio would become uncomparable across chapters. One measurement, one vintage, looked up by everyone — disagreements about cost become a reason to build a new vintage (Step 2.10), never a reason to let two stages compute two different numbers.

Stage gate — Stage −1a — cost surface

`cost_surface` populated for all 21 instruments at one frozen vintage; coverage (Step 2.9c) $\geq$ 99% for *every* instrument; checks 2.9(a) and (b) reviewed, with every flagged cell dispositioned in the validation report. All three hold *simultaneously*, for the *same* vintage, or the gate is red. Any one instrument below 99% coverage, or any undispositioned (a)/(b) flag, blocks the whole gate — Chapter 5's own precondition line reads "`cost_surface` populated for all 21 instruments," not per-instrument, so a partial surface does not open a partial Stage 0.

Kill criterion — Stage −1a has no kill — only a block

None: this chapter is infrastructure, not a hypothesis test, so nothing here is killed by a result. What *is* pre-committed is the block, stated once in Chapter 1's stage table: no Stage-0 result may be reported until the cost surface (this chapter) and the point-in-time feeds (Chapter 3) both exist. A validation failure here is not a verdict; it is re-measured — widen the pull window, re-pull a thin instrument's longer history, or accept a documented, reviewed exception at Step 2.9 — then the gate is re-tested. The one irreversible action in this chapter is Step 2.10's freeze; everything before it can be re-run at will.

Artifacts — Stage output register

- `candles_ba.sqlite` (`candles_ba`): instrument, `ts`, bid/ask OHLC, volume; up to 23,194,080 rows (Step 2.2).
- `instrument_meta`: instrument, pip_location, pip_size, refreshed_at (Step 2.2).
- `session_windows`: bucket, dst_regime, utc_start, utc_end, 16 rows (Step 2.3).
- `calendar.sqlite` (`calendar`): event/currency/instrument_scope, tier, scheduled/actual/snapshot timestamps (Step 2.4).
- `cost_surface.sqlite` (`cost_surface`): full schema, Steps 2.1, 2.5–2.8 — the artifact every later chapter reads.
- `cost_params`, `financing_rates` (Step 2.7).
- `validation_report_<vintage>.md/.json` (Step 2.9).
- `cost_vintage_registry` (Step 2.10).

3

Stage −1b: Point-in-Time Feeds

Five of the programme's covariates — COT positioning, FRED macro levels, the RORO gauge, the VolRegime gauge, and every yield differential — are, as stored today, not point-in-time. Each is joined by reference date or reporting period, or by whatever value happens to be currently on file: release-lag leakage and revision leakage respectively [Course Book §18.1.5][Course Book §18.1.4]. The first is verified fact, not a hypothetical, for the platform's sharpest instance: [Course Book §18.1.5]'s own audit found that `cot.sqlite` carries `report_date` and a value, nothing that records when a row became knowable. [Course Book §18.2] gives the general fix this chapter builds. This is not a data-quality nuisance to schedule alongside other work. Stage 2's regime router reads its emission vector from these five series [Course Book §13.8]; a Chronos-2 covariate probe run alongside the live TimesFM envelope, the cheapest available instrument for asking whether any of the five carries conditional information at all, reads the same five [Course Book §9.3]; and the platform's documented failure pattern — high hit rate, thin or negative expectancy — is exactly the signature a three-day macro peek produces once costs are applied honestly [Course Book §18.5]. Nothing downstream of this chapter can be believed until it is done.

The fix is one table and one predicate, applied five times. Stage −1a built the cost surface (spread, slippage, commission); this chapter builds the feeds; Stage −1c assembles the harness — CPCV, the trial ledger, the leak-injection CI suite — that both feed into, and that suite will not be trusted until this chapter's audit query (Step 3.12) is green and wired in ahead of it.

3.1 The truth schema and the as-of join

Step 3.1 — One table, one predicate

Do. Create table `feeds` in a new store `feeds.sqlite`:

Column	Type	Meaning
`series_id`	text	Identifier, e.g. `COT_EURUSD_NONCOMM_NET` or `FRED_NFP`.
`reference_period`	date	The period the value describes: COT's Tuesday, NFP's reporting month.
`release_ts`	timestamp (UTC)	The instant the value became usable under this platform's convention — the *only* column the as-of predicate reads.
`value`	real	The figure as published at that vintage.
`vintage_ts`	timestamp (UTC)	Ingestion bookkeeping: when this row landed in `feeds`. Never read by the join.

Primary key (`series_id`, `release_ts`). One row per vintage: a revised series carries several rows sharing one `reference_period` with distinct `release_ts` and `value`. Every vendor feed

— COT, FRED/ALFRED, RORO, VolRegime, yield differentials — writes into this table and no other; retire `cot.sqlite` and any ad hoc RORO/VolRegime dump once Sections 3.2–3.5 have backfilled it. Column mapping onto [Course Book §18.2]'s formal triple: `reference_period`$= \pi$, `release_ts`$= \rho$, `value`$= v$; `vintage_ts` is not part of that triple — like π, it plays no role in the predicate below.

The mathematics. [Course Book §18.2]'s as-of join, stated once:

$$v^*(s,t) \;=\; v_{j^*}, \qquad j^*(t) \;=\; \underset{j:\,\rho_j \le t-\ell}{\arg\max}\ \rho_j,$$

translated directly into the query every consumer runs:

```
SELECT value, release_ts, reference_period
FROM   feeds
WHERE  series_id  = :series_id
  AND  release_ts <= :t - :safety_lag
ORDER BY release_ts DESC
LIMIT  1;
```

Zero rows returned means s is undefined at `:t` — missing, per Step 3.11, never backfilled or interpolated. `:safety_lag` is ℓ, fixed per series in the registry (Step 3.11), which also closes a gap this query alone leaves open: no ceiling on how stale a returned vintage may be.
Add an index on (`series_id`, `release_ts` DESC) at creation time — the as-of query above is the single most frequently executed statement in the whole point-in-time pipeline, run once per (series, bar) at both backtest and live-inference time, and without it every call degenerates to a full-table scan of that series' entire vintage history.

Artifact. `feeds.sqlite` → table `feeds`, schema above plus the (`series_id`, `release_ts` DESC) index, empty pending the backfills in Sections 3.2–3.5.

Pass / fail. Every column present with the stated type and primary key; every loader built below writes only into this table; grep the codebase for any other table storing a COT, FRED, RORO, VolRegime or yield-differential value — zero hits.

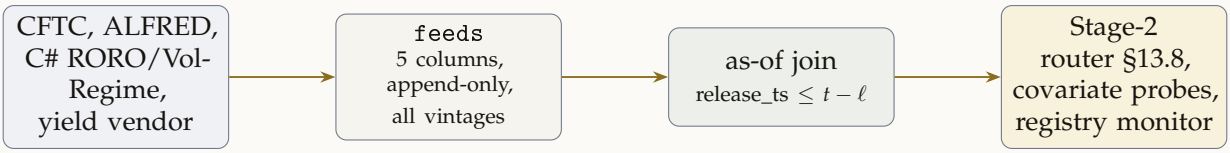


Figure 3.1. Every feed lands in one append-only table; every consumer reads through one predicate. Nothing downstream ever touches a vendor source directly.

3.2 COT: anatomy, calendar, and the net-positioning z-score

Step 3.2 — Report anatomy and series selection

Do. The CFTC publishes three weekly report variants: *Legacy* (Commercial / Non-Commercial / Non-Reportable, every futures market), *Disaggregated* (Producer/Merchant, Swap Dealer, Managed Money, Other Reportable — physical-commodity markets only), and *Traders in Financial Futures*, TFF (Dealer/Intermediary, Asset Manager, Leveraged Money, Other — financial futures, including FX, rates and equity index). Primary se-

ries: Legacy Non-Commercial net position (long minus short contracts), `series_id` pattern `COT_<INSTRUMENT>_NONCOMM_NET` — the exact naming [Course Book §18.2] already fixed. Coverage rule, applied per instrument:

Instrument class	Example	Rule
USD-leg major	EUR/USD, GBP/USD, USD/JPY, AUD/USD	Direct CME futures contract; map 1:1.
Cross, no USD leg	EUR/GBP, EUR/CHF, GBP/JPY	No CME contract; synthesise net = leg1_net − leg2_net (both vs. USD, both Legacy Non-Commercial); flag as an approximation.
Metals, energy	XAU, XAG, WTI	Direct COMEX/NYMEX contract; Legacy Non-Commercial by default.
Crypto	BTC, ETH	CME futures COT exists but is short (BTC from 2018, ETH from 2021) and thin; before contract inception, missing — never backfilled.
Equity index	the two platform indices	E-mini equivalent, Legacy Non-Commercial.

Recommendation: promote metals and WTI to Disaggregated Managed Money net, and FX/index majors to TFF Leveraged Money net, as a later refinement — one-line reason: both isolate the speculative bucket more cleanly than Legacy Non-Commercial, which mixes hedge-fund and non-hedge-fund speculative flow; not switching now keeps this build's naming aligned with [Course Book §18.2].

Artifact. `cot_series_map(instrument, series_id, rule, first_valid_period)`, rule ∈ {direct, synthesised, missing-before-date}.

Pass / fail. Every platform instrument has exactly one row; zero silent gaps.

Step 3.3 — The release calendar, and eligibility

Do. For `reference_period` π (a Tuesday), the scheduled release is $\pi + 3$ calendar days (a Friday), 15:30 ET. Convert to UTC through the US DST calendar — EDT, UTC−4, second Sunday of March to first Sunday of November; EST, UTC−5, otherwise — never a hardcoded clock time. *Holiday-shift rule:* if the scheduled Friday is a US federal holiday, the CFTC does not publish; walk forward one calendar day at a time, skipping weekends and federal holidays, to the next business day, and release lands there under the same 15:30 ET rule. *Eligibility:* [Course Book §18.2] pushes first usability to the opening of the following week's first H1 bar, not to any point within release Friday or the Sunday reopen; generalised, eligible_from(release_ts) = 00:00 UTC of the next FX-*trading* day strictly after release_ts's calendar date — weekend only, never the federal-holiday calendar of the paragraph above, since the platform's 21-instrument book trades through MLK Day, Presidents' Day, Columbus Day and Veterans Day. Store eligible_from *as* `feeds.release_ts` for COT rows, so the generic query (Step 3.1) reads a single, already-margined timestamp with `:safety_lag` = 0 for this series family; keep the true CFTC publish instant only in the backfill job's log, for Step 3.11's latency audit. Backfill every Tuesday from the TFF report's 2010-01-05 start (868 weekly reference periods to 2026-08-18, `verify_pb03.py` §5) to the present.

The mathematics. A definition, not a derivation: release_ts(π) = the earliest $t \in \{\pi + 3\text{d}, \pi + 4\text{d}, \dots\}$ with t a weekday and $t \notin$ federal holidays, evaluated at 15:30 ET→UTC by the DST offset in force on t; then eligible_from = 00:00 UTC of the earliest weekday strictly after release_ts's date.

Artifact. `us_federal_holidays(date, name)`; backfilled `feeds` rows for every series in

`cot_series_map`.

Pass / fail. Step 3.4's five-pair hand-check and the sanity sweep below.

Trap — Joining on `report_date` is a three-day peek

Take the ordinary case from `verify_pb03.py` §1: reference Tuesday 2026-08-11, release 2026-08-14 19:30 UTC. A join on `reference_period` (the current `cot.sqlite` schema's only option) admits that Tuesday's positioning from the Tuesday it describes — Wednesday and Thursday's bars trade on a number that will not exist for another two to three days, exactly the release-lag defect [Course Book §18.1.5] names. The as-of join (Step 3.1) closes it by construction, because `reference_period` never appears in the predicate at all; it is carried in `feeds` only to describe what a value is *about*, never when it may be used.

Trap — Two calendars, not one

The holiday-shift rule and the eligibility rule in Step 3.3 deliberately use *different* calendars, and collapsing them into one is the easiest mistake to make here. Whether the CFTC publishes on a given day depends on the federal calendar, because the CFTC is a federal agency. Whether an H1 bar exists depends on the FX/CFD market calendar, and this platform's 21 instruments trade through MLK Day, Presidents' Day, Memorial Day, Labor Day and Columbus Day — five of the eleven federal holidays fall on a Monday by construction, and any one of them can immediately follow an ordinary Friday release. Reusing the federal calendar for eligibility pushes usability a full extra day late on every such occurrence, for a reason — the CFTC's office hours — that has nothing to do with whether the platform could actually have traded on the datum.

Step 3.4 — Validate: five hand-checked pairs

Do. Run `verify_pb03.py` §1 against five pairs chosen to cross both DST seasons and to exercise the holiday-shift rule in its two forms — an actual-date holiday landing on a Friday (Juneteenth, Christmas), and a Saturday holiday observed the preceding Friday (Independence Day):

Case	Reference (Tue)	Scheduled (Fri)	Release_ts (UTC)	Eligible_from (UTC)	Margin
Ordinary, EST	2026-01-06	2026-01-09	2026-01-09 20:30	2026-01-12 00:00 (Mon)	51.5h
Ordinary, EDT	2026-08-11	2026-08-14	2026-08-14 19:30	2026-08-17 00:00 (Mon)	52.5h
Juneteenth (holiday Fri)	2026-06-16	2026-06-19 shifted→2026-06-22	2026-06-22 19:30	2026-06-23 00:00 (Tue)	4.5h
Independence Day (Sat obs. Fri)	2020-06-30	2020-07-03 shifted→2020-07-06	2020-07-06 19:30	2020-07-07 00:00 (Tue)	4.5h
Christmas (holiday Fri)	2026-12-22	2026-12-25 shifted→2026-12-28	2026-12-28 20:30	2026-12-29 00:00 (Tue)	3.5h

Artifact. None beyond the script; this step is pure validation.

Pass / fail. All five reproduce independently in `verify_pb03.py` §1 (weekday assertions on reference and scheduled dates, holiday lookup, DST offset, all asserted, not merely printed). The full 2026 calendar carries 52 reference Tuesdays and exactly 4 holiday-shifted releases —

Juneteenth, Independence Day, Christmas, and a year-boundary case where the reference period of 2026-12-29 schedules into New Year's Day 2027 (`verify_pb03.py` §2). Separately, §4 sweeps every ordinary (non-federal-holiday) Friday release from 2020 to 2031 — 607 dates — through the generalised eligibility rule of Step 3.3 and confirms every one reduces exactly to the following Monday 00:00 UTC, [Course Book §18.2]'s stated convention.

Step 3.5 — Backward-only net-positioning z-score

Do. For a COT series with weekly reports $k = 1, 2, \ldots$ in ascending `reference_period` order and raw net position x_k: standardise x_k against moments computed only from the W reports strictly preceding it, never including k itself, per [Course Book §13.8]'s causal-standardisation requirement and [Course Book §18.1.3]'s fix for normalisation leakage. Recommend $W = 156$ (three years of weekly reports) — one-line reason: long enough to span more than one positioning cycle, short enough that the standardisation adapts to a genuine regime shift rather than anchoring to a decade-old range. Undefined (missing) for $k \leq W$.

The mathematics.

$$\hat{\mu}_k = \frac{1}{W} \sum_{i=k-W}^{k-1} x_i, \qquad \hat{\sigma}_k = \sqrt{\frac{1}{W-1} \sum_{i=k-W}^{k-1} (x_i - \hat{\mu}_k)^2}, \qquad z_k = \frac{x_k - \hat{\mu}_k}{\hat{\sigma}_k}$$

[Course Book §13.8].

Artifact. `feeds` rows, `series_id` `COT_<INSTRUMENT>_NONCOMM_NET_Z`, same `reference_period` and `release_ts` as the raw report each derives from — z_k's ingredients are all released strictly before x_k itself, so the derived series inherits no extra lag [Course Book §18.2].

Pass / fail. Illustrative arithmetic only (a synthetic net-position path, not a real CFTC print), `verify_pb03.py` §3, $W = 4$:

k	window (reports $k-4..k-1$)	$\hat{\mu}_k$	$\hat{\sigma}_k$	z_k
5	120.0, 135.0, 128.0, 142.0	131.2500	9.4296	1.9884
6	135.0, 128.0, 142.0, 150.0	138.7500	9.4296	−0.0795
7	128.0, 142.0, 150.0, 138.0	139.5000	9.1469	1.6946
8	142.0, 150.0, 138.0, 155.0	146.2500	7.6757	1.7914

Operational check, on real series: recompute z_k for a held-out sample of reports using only data strictly before k and diff against the stored value — zero diffs, the recompute-and-diff detection method [Course Book §18.1.1] prescribes, applied to a normalisation statistic rather than a feature.

3.3 FRED to ALFRED: vintage ingestion

COT's defect is release-lag leakage; FRED's is the companion failure, revision leakage [Course Book §18.1.4] — the value itself is correct, but the wrong vintage of it is read. The fix lives in a different data source, not a correction factor [Course Book §18.2], and it writes into the same `feeds` table as everything else in this chapter.

Step 3.6 — ALFRED vintage ingestion

Do. Two macro series in active use: CPI, in [Course Book §8.4]'s covariate table, and NFP (non-farm payrolls), [Course Book §18.2]'s own `series_id` example and the series [Course Book §18.1.4]'s revision illustration is built on. Query ALFRED's vintage-aware observations endpoint per series with `realtime_start`/`realtime_end` spanning the full sample — never FRED's single-current-value endpoint, which cannot answer "what did this series say on this date" [Course Book §18.2]. ALFRED returns one row per (reference period, vintage); the mapping is direct: `reference_period` = the observation month, `release_ts` = that vintage's `realtime_start`, `value` = the figure first current as of that vintage. Write one `feeds` row per returned vintage — a twice-revised series contributes three rows sharing one `reference_period`. `series_id` pattern `FRED_<SERIES>`, e.g. `FRED_NFP`, [Course Book §18.2]'s own example. Unlike COT, the release instant itself is already the correct eligibility instant: [Course Book §8.4] states plainly that carrying a release's value "makes it observed-past with $\ell = 0$ from the release instant." Use a small fixed ingestion buffer (minutes) as `safety_lag` here, not a calendar rule — Step 3.3's weekend-fold machinery is a COT-specific correction, not a general one.

Artifact. `feeds` rows for `FRED_NFP` and `FRED_CPI`, every historical vintage.

Pass / fail. Row count per `reference_period` ≥ 1, and > 1 wherever the series has been revised; spot-check five reference periods against ALFRED's own vintage-dates listing.

Step 3.7 — Revision audit protocol

Do. For every FRED-sourced series, the standing audit compares the first-published value to the latest vintage on file, per `reference_period`, over a rolling sample window:

```
WITH first_release AS (
  SELECT series_id, reference_period, value AS first_value,
         release_ts AS first_release_ts
  FROM   feeds f
  WHERE  release_ts = (SELECT MIN(release_ts) FROM feeds g
                       WHERE g.series_id = f.series_id
                         AND g.reference_period = f.reference_period)
),
latest AS (
  SELECT series_id, reference_period, value AS latest_value,
         release_ts AS latest_release_ts
  FROM   feeds f
  WHERE  release_ts = (SELECT MAX(release_ts) FROM feeds g
                       WHERE g.series_id = f.series_id
                         AND g.reference_period = f.reference_period)
)
SELECT a.series_id, a.reference_period, a.first_value, b.latest_value,
       b.latest_value - a.first_value AS abs_revision,
       (b.latest_value - a.first_value) / a.first_value AS pct_revision
FROM   first_release a JOIN latest b USING (series_id, reference_period)
ORDER  BY ABS((b.latest_value - a.first_value) / a.first_value) DESC;
```

The rule this audit exists to enforce: every backtest join goes through Step 3.1's as-of predicate against `release_ts`, never against this table's `latest_value` column — the audit exists to *quantify* the exposure of a join done wrong, not to be joined into a live pipeline itself.

Artifact. `revision_audit`, a materialised view over the query above.

Pass / fail. Every `series_id` matching `FRED_%` has at least one row; any $|$`pct_revision`$| > 10\%$ is flagged for manual review before that series enters a Stage-2 covariate probe.

3.4 RORO and VolRegime: freezing the gauges

[Course Book §18.2]'s projbox names this section's requirement exactly: both gauges exist today only as live numbers, computed once and discarded, and reconstructing them correctly sits where release-lag leakage and smoothed-state leakage [Course Book §18.1.7] collide. Everything below applies identically to both gauges; run it twice.

Step 3.8 — Freeze the definition, then reconstruct backward-only

Do. *Freeze.* Pull the RORO gauge's current formula and full input list directly from the C# engine's live gauge-computation module — this is the one step in this chapter whose governing formula cannot be stated here, because it is presently defined only in running code, not in any specification; that absence is itself the defect this step fixes. A gauge of this kind typically combines a handful of market-observed series into one scalar or discrete state — for RORO, commonly some blend of equity-index returns, credit-spread or funding-stress proxies, an implied-volatility level, and safe-haven FX moves; for VolRegime, commonly a realised-over-implied or short-over-long realised-volatility ratio, thresholded or fed to a filter — so when reading the C# source, extract every one of those inputs individually, not only the combining formula: each input carries its own release lag, and Step 3.11's registry needs one row per input, not one row for the gauge as a whole. Transcribe the whole of it verbatim into a versioned spec document — formula, every input series, every parameter, extraction date, extracting engineer — before touching the historical reconstruction: freeze first, recompute second, or the parity check below compares against a moving target. *Reconstruct.* Recompute the gauge over the full history using only the frozen formula and backward-only windows at every historical point: a single full-sample pass whose normalisation constants are fit once on the entire history is smoothed-state leakage wearing a macro-gauge disguise [Course Book §18.1.7]. If the gauge consumes any release-lagged input (COT positioning, a scheduled macro print), the reconstructed row's `release_ts` at bar t cannot be earlier than the latest `release_ts` among its inputs at t — a derived series cannot know something before its slowest ingredient does [Course Book §18.2]. Persist under the same five-column schema as every vendor feed (Step 3.1); `series_id` `RORO_GAUGE` / `VOLREGIME_GAUGE`. Repeat both parts for VolRegime.

The mathematics. The one general statement available: `release_ts`$(\text{derived}, t) = \max_i$ `release_ts`(input_i, t) — the propagation rule, stated once, applied to any derived series in this chapter.

Artifact. `roro_gauge_spec_v<N>.md` and `volregime_gauge_spec_v<N>.md` (the frozen definitions); `feeds` rows for `RORO_GAUGE` and `VOLREGIME_GAUGE`, full history.

Pass / fail. *Freeze:* a second engineer, given only the spec document, reproduces today's live reading to floating-point tolerance from the same inputs. *Reconstruct:* rerun the reconstruction under a deliberately full-sample (non-causal) normalisation and confirm the two disagree materially in the early sample, [Course Book §18.1.1]'s recompute-and-diff method applied here — agreement would mean the "backward-only" implementation is not actually backward-only.

Step 3.9 — Parity check against the live gauge

Do. Over an overlap window where both the reconstructed as-of series and the live gauge exist (recent history), compare bar by bar. Recommended tolerance $|\text{reconstructed} - \text{live}| \leq \max(0.02,\ 0.05 \times |\text{live}|)$ per bar — one-line reason: wide enough to absorb float and rounding noise between the two code paths, tight enough to catch a genuine transcription error. Investigate, never silently accept, any divergence outside tolerance; the two likely causes are a transcription error in the frozen spec (Step 3.8) or a release-lagged input the reconstruction handled differently from the live engine.

Artifact. `parity_report_roro.csv` and `parity_report_volregime.csv` — per-bar reconstructed value, live value, diff, in/out of tolerance.

Pass / fail. At least 99% of overlap-window bars within tolerance; every excursion outside tolerance has a logged, investigated cause before Step 3.12's gate is evaluated.

3.5 Yield differentials

The fifth and last raw feed, and the simplest: no report-release calendar, no vintage history to speak of, only the discipline of picking a source that timestamps its own publication and never mutating a row once written.

Step 3.10 — Source, the differential, and vendor restatements

Do. The source must carry both an observation date and a publication timestamp. FRED's `DGS2` / `DGS10` (2-year / 10-year constant-maturity Treasury, the daily H.15 release, timestamped) satisfy this for the USD leg; each counterparty currency needs the equivalent sovereign-curve series from its own central bank or data vendor (Bundesbank/ECB for EUR, BoE for GBP, BoJ for JPY, and so on), held to the identical timestamp discipline before use. Per pair, compute both a 2-year differential (short-rate, carry proxy) and a 10-year differential (term-structure proxy) — extending [Course Book §8.4]'s two-year covariate to both tenors the platform's regime refinement uses [Course Book §14.7]. Fix the sign convention once: differential = base-currency yield − quote-currency yield, e.g. EUR/USD's 2-year differential = EUR 2y − USD 2y. Store each currency's own yield as a `feeds` series (`series_id` pattern `Y2_<CCY>` / `Y10_<CCY>`), not the pair-level differential directly — the differential is computed at query time from two independently as-of-joined legs, keeping the store normalised and avoiding a combinatorial explosion of pair-level series. *Vendor restatements:* yield levels are not revised in the NFP/CPI sense, but a vendor occasionally corrects an erroneous print (a bad tick, an outage backfilled wrong); treat every correction as a new vintage — an additional row with a later `release_ts` — never an in-place update to the existing row's value, which would destroy the as-of predicate's ability to reproduce what a historical query would have returned before the correction landed.

The mathematics.

$$d^{(2y)}_{\text{pair},t} = y^{(2y)}_{\text{base}}(t) - y^{(2y)}_{\text{quote}}(t), \qquad d^{(10y)}_{\text{pair},t} = y^{(10y)}_{\text{base}}(t) - y^{(10y)}_{\text{quote}}(t),$$

each leg read through Step 3.1's as-of predicate independently before differencing; differencing two values that were not each independently point-in-time reproduces the leak one leg at a time. `release_ts` of the differential is the later of its two legs', the same propagation rule as Step 3.8.

Artifact. `feeds` rows for `Y2_<CCY>` / `Y10_<CCY>` per currency in the platform's book; a derived view `yield_diff(pair, tenor, t)`; an ingestion role with `UPDATE` revoked on `feeds`.

Pass / fail. Every currency in the instrument list has both tenors covered, or is explicitly flagged missing; attempt an `UPDATE` against an existing `feeds` row under the restricted ingestion role and confirm it is rejected.

3.6 The feed registry and staleness monitors

Five feeds now write into `feeds` on five different cadences and five different safety-lag policies. Nothing so far tracks whether any one of them is still arriving on schedule; this section closes that gap and, in the same step, closes the staleness hole left open in Step 3.1's original query.

Step 3.11 — The registry, and missing rather than carried forward

Do. One row per `series_id`: `feed_registry(series_id, cadence, expected_latency, safety_lag, last_release_ts, last_checked_ts, status)`. Populate from Sections 3.2–3.5's builds — COT weekly, safety_lag = 0 against an already-margined `release_ts` (Step 3.3); FRED per series (NFP and CPI, both monthly), safety_lag a few minutes; RO-RO/VolRegime continuous, safety_lag propagated from their slowest input; yield differentials daily, safety_lag near zero. A scheduled job updates `last_release_ts` and `last_checked_ts` on every ingestion run and raises an alert on transition into STALE. Then close a gap Step 3.1's query leaves open on its own — it has no upper bound on how old a returned vintage may be, so a feed that silently stops updating is reused forever. Extend the predicate with a staleness ceiling drawn from this same registry:

```
SELECT value, release_ts, reference_period
FROM   feeds
WHERE  series_id  = :series_id
  AND  release_ts <= :t - :safety_lag
  AND  release_ts >= :t - :max_staleness
ORDER BY release_ts DESC
LIMIT  1;
```

with `:max_staleness` = cadence + expected_latency, the same quantity the monitor job alerts on. Zero rows returned — whether from the missing-vintage case or the new staleness ceiling — means the covariate is missing at `:t`, exactly [Course Book §18.2]'s own rule for an undefined vintage: "never backfilled, interpolated, or substituted from a later vintage." The consuming model must handle that explicitly — a dedicated missing-indicator input, or an imputation policy fixed at Stage-2 design time and logged as such — rather than receive a value silently reused past its cadence; forward-filling the last good value is the one behaviour this rule exists to forbid, because it reintroduces precisely the staleness the registry was built to catch.

The mathematics. Staleness predicate: series s is STALE at t iff $t - \texttt{last_release_ts}(s) > \texttt{cadence}(s) + \texttt{expected_latency}(s)$.

Artifact. `feed_registry` table; the extended as-of query above, replacing Step 3.1's version in every consumer; a monitor job evaluating the staleness predicate on a fixed interval.

Pass / fail. Every `series_id` ever written to `feeds` has exactly one registry row; a deliberately delayed test ingestion trips the alert within one monitor interval; a synthetic series with a gap longer than its cadence returns zero rows (missing) from the extended query, never the last pre-gap value.

3.7 The audit query

Step 3.12 — The standing no-future-value-reachable test

Do. Two halves, run together as one indivisible test, never separately. *Positive half:* for a large sample of historical (`series_id`, t) pairs — every H1 bar timestamp over the backfilled window is the recommended sample, not a thin spot-check — run Step 3.11's extended as-of query and assert the returned row's `release_ts` $\leq t -$ `safety_lag`; independently re-derive the maximum admissible `release_ts` by direct aggregation over `feeds` and diff against what the query returned, so the test checks the query's logic against an independent computation rather than its own self-consistency. *Negative half:* rerun the identical sample against a deliberately naive join on `reference_period` instead of `release_ts` — the first trapbox's exact defect — and confirm it admits rows the as-of version rejects. A negative half that finds nothing is not a passing audit; it is a broken negative control, in exactly the sense [Course Book §18.4]'s four-control suite requires each control to prove it has power, not merely that the correct query looks correct in isolation.

The mathematics. None beyond Steps 3.1 and 3.11's predicates, restated as an assertion over a sample rather than a single call.

Artifact. `audit_pit.py` — the two-part test above, emitting a CI-consumable report.

Pass / fail. Zero violations on the positive half, for every series in `feed_registry`; strictly more than zero admitted future-dated rows on the negative half, for at least the COT series — proof the audit can detect the defect it exists to prevent.

`audit_pit.py` is the prerequisite gate ahead of Stage −1c's own leak-injection CI suite: that suite's feature-timestamp-scramble control [Course Book §18.4] is, concretely, a scramble of this chapter's `feeds.release_ts` column, and a scramble control proves nothing if the join it is scrambling was never enforcing the predicate to begin with. Wire `audit_pit.py` into CI ahead of — not alongside — Stage −1c's suite; a red `audit_pit.py` blocks that suite from running at all, the same freeze discipline [Course Book §18.4]'s keybox states for its own four controls.

Stage gate — Stage −1b gate

Metric: `audit_pit.py` (Step 3.12) returns zero violations across the full sampled history for every series in `feed_registry`; the RORO and VolRegime parity checks (Step 3.9) clear tolerance on at least 99% of overlap-window bars, with every excursion logged and explained; the five-pair calendar hand-check and the 2026 holiday-shift count reproduce exactly in `verify_pb03.py`. Threshold: all of the above, simultaneously — one feed failing the audit is infrastructure failing, not a soft warning on that one feed. Decision rule: pass → Stage 2's regime router [Course Book §13.8] and any covariate probe [Course Book §9.3] may read from `feeds`; every earlier result that touched COT, FRED, RORO, VolRegime or a yield differential before this gate was green is void, per [Course Book §18.1.5]'s projbox, and must be re-run, not relabelled, once it is.

Kill criterion — No trading hypothesis to kill

Stage −1b is infrastructure: there is no strategy here to kill. What exists instead is a freeze. Any result — Stage 0's envelope-honesty test, Stage 2's regime router, a covariate probe against TimesFM or Chronos-2 — computed by joining a bar to COT, FRED, RORO, VolRegime or a yield differential before this stage's gate is green is not a disappointing result; by [Course Book §18.5]'s own audit walkthrough, a claimed Sharpe ratio that fails the macro-join

step of that eight-step checklist is reported as FAIL, with no corrected number offered, until the join is fixed and the leak-injection suite is rerun end to end. Apply the identical discipline here, pre-committed: no Stage-2-or-later number computed against an un-gated feed is reported, discussed, or used to justify a decision, full stop, until Step 3.12 is green.

Artifacts — Stage −1b output register

- `feeds.sqlite` — table `feeds(series_id, reference_period, release_ts, value, vintage_ts)`, the single store for every point-in-time series.
- `us_federal_holidays` — the calendar table backing the COT holiday-shift rule.
- `cot_series_map` — instrument → COT series_id / synthesis rule / missing-before-date.
- `roro_gauge_spec_v<N>.md`, `volregime_gauge_spec_v<N>.md` — the frozen gauge definitions extracted from the C# engine.
- `parity_report_roro.csv`, `parity_report_volregime.csv`.
- `revision_audit` — the FRED first-release-vs-latest-vintage view.
- `feed_registry` — cadence, expected latency, safety lag, last release, status, per series; the monitor job that reads it.
- `audit_pit.py` — the standing no-future-value-reachable test, wired into CI [Course Book §18.4], gating Stage −1c's leak-injection suite.

Every version reads the same table through the same predicate: v1's regime router is the immediate consumer [Course Book §13.8], but v4's real-yield and RORO covariates [Course Book §22.4] route through this identical `feeds`/as-of pair, not a second implementation.

4

Stage −1c: The Harness, the CI Suite, and the Ledger

Stage −1a built the cost surface; Stage −1b built point-in-time feeds. Neither is worth anything without an instrument capable of reporting, honestly, whether a candidate built on top of them works. This chapter builds that instrument: the backtest engine that turns a policy into a P&L series, the combinatorial purged splitter that turns a P&L series into a distribution, the metrics module that turns a distribution into a deflated verdict, the trial ledger that makes the deflation honest, and the leak-injection suite that is the only thing entitled to certify any of the other four are wired correctly. Every stage from 0 to 4 calls into this chapter by number rather than re-implementing it — "the pb04 harness" recurs throughout the rest of this book as a fixed reference, not a figure of speech.

The five instruments are inseparable. A splitter without a leak-injection suite is a plausible number generator; a ledger without a write-before-inspection discipline is a place selection bias hides; a metrics module that cannot reject its own null is not a metrics module. Build all five before Stage 0's first run, not some of them.

The build order below follows a dependency chain, not a convenience one. The engine (Step 4.1) must exist before the splitter (Step 4.2) has anything to purge and embargo around; the splitter must exist before the metrics module (Step 4.3) has paths to compute a Sharpe distribution over; the metrics module must exist before the ledger (Step 4.4) has a schema to store its output in; and all four must exist, wired together, before the leak-injection suite (Steps 4.5–4.9) has anything to inject a leak into. Building the suite first against a stub engine tests the suite, not the platform.

Step 4.1 — The backtest engine contract

Do. Fix the bar-loop invariants as structural constraints in the engine, not as options any strategy author can relax.

- *Information.* The decision at bar t is a function of $\mathcal{F}_t$ only — everything known up to and including bar t's close. No feature, forecast row, or label may be read from $t+1$ or later; the engine must refuse to compile a policy that indexes forward.
- *Fills.* A signal computed from bar t's close fills no earlier than bar $t{+}1$'s open ([Course Book §18.3]); this is enforced by the engine's execution simulator, never left to a strategy's own code path.
- *Latency.* A latency budget $\Lambda_{\max}$ bounds the wall-clock time from bar close to order submission ([Course Book §18.3]); any signal whose compute path would exceed it is inadmissible at that timeframe rather than silently treated as instantaneous. Recommend $\Lambda_{\max} = 5{,}000$ ms: generous enough to cover a full TimesFM forward pass on the pinned 3060 plus the C# engine's SQLite poll, small enough to stay under 10% of the shortest timeframe (M1, 60,000 ms).

- *Cost lookup.* Read $c(\text{instrument}, \text{cell}(t))$ from Stage −1a's cost surface with $\text{cell}(t) = (\text{session_bucket}(t), \text{news_flag}(t), \text{rollover_flag}(t))$, at the run's *pinned* vintage: the vintage is fixed once in the run config and never re-resolved to "latest" mid-run, so a rerun of an old config reproduces the same cost, not today's. A bar whose cell resolves to $n < 100$ observations (Stage −1a's minimum-cell-count rule) falls back through the same coarser-cell order the cost surface itself defines — the engine never invents a cost for an under-populated cell.
- *Position state machine.* Three states, transitions driven by the policy's target direction $z_t \in \{-1, 0, +1\}$:

state	z_t seen	next state / action
FLAT	$z_t = 0$	stay FLAT, no order
FLAT	$z_t \neq 0$	open at $t{+}1$ open, size per the policy's exposure rule; go LONG/SHORT
LONG/SHORT	z_t unchanged in sign	hold, no order
LONG/SHORT	$z_t = 0$ or sign flips	close at $t{+}1$ open (and reopen opposite if flipped); a round trip completes

- *Currency.* Every P&L figure is stated in account currency. Gross P&L and cost are computed in the instrument's quote currency and converted at the prevailing quote/account rate *at the trade's own close timestamp* — never at open, never at a period average, and never looking ahead of that timestamp for the rate itself.

The mathematics. Net P&L on round trip i: $X_i = G_i - C_i$, with G_i the signed gross P&L (account currency, per the conversion rule above) and $C_i \geq 0$ the round-trip cost read from the pinned cost surface. Stated once here; every downstream sum, mean, and Sharpe ratio in this chapter is built from this quantity. [Course Book §14.1]

Artifact. `backtest_engine` module (versioned); run-config schema (`latency_ms`, `cost_vintage`, `split_config_ref`, `universe`, `tf`); a golden-fixture regression test fixing one hand-computed X_i.

Pass / fail. (a) replay determinism: identical config plus identical data produces a byte-identical per-trade X_i series on rerun; (b) the fill-timestamp audit reports 100% of fills at a $t{+}1$ open and 0% at a signal bar's own close; (c) the hand-computed fixture of (Artifact) reproduces to floating-point tolerance.

Step 4.2 — The CPCV splitter

Do. Group by wall clock, across everything, per [Course Book §16.5]: cut the three-year history into N contiguous intervals with boundaries at the 5pm New York close (already a seam in the cost surface); every row of every one of the 21 instruments whose bar closes inside an interval belongs to that interval's group. Where a model is fit per timeframe, use the *same* group boundaries across all 7 timeframes so that path p names the same stretch of history everywhere. Fix the platform default at $N = 8$, $k = 2$ test groups per split, uniformly across timeframes (recommendation): more paths and training fraction than $N = 6$ at 62% of the fit count $N = 10$ demands (verify_pb04.py §1) — the fit-count/training-fraction trade-off itself is [Course Book §16.4]'s. Purge width is $h = 50$ bars each side of every test block's label span, fixed across timeframes because the horizon is fixed in bars ([Course Book §16.2]). Embargo δ is set per timeframe by the rule $\delta \geq \max(0, w_{\max} - 1 - h)$ ([Course Book §16.2]), $w_{\max}$ the longest feature look-back *in bars at that timeframe's own cadence*: a fixed calendar-time memory (a weekly COT refresh, a monthly macro print) converts to far more bars at M1 than at H4, so

the fast timeframes carry the wider embargo, not the slow ones.

TF	purge h (bars)	embargo δ (bars)	reason
M1	50	30	fastest cadence: weekly/monthly-refresh feeds span the most bars here
M5	50	25	cadence mismatch narrows but remains material
M15	50	18	
M30	50	12	platform default; Stage 0's primary timeframe
H1	50	12	
H4	50	12	purge alone already removes a non-trivial slice of the sample
D	50	12	purge alone removes the dominant share — see trap below

Unit tests, run on every commit to the splitter (bridges to Step 4.9):

- `assert` no training sample's label span Λ_i intersects any test group's label span, for every split;
- `assert` group boundaries are identical across all 21 instruments and (for a jointly-fit model) across all timeframes in the run;
- `assert` a single-bar test block at the very first or last group purges and embargoes correctly rather than indexing out of range;
- `assert` the 5pm New York boundary resolves to the correct UTC instant across a US daylight-saving transition (twice yearly the UTC offset changes and a naive fixed-offset boundary function silently mis-groups a week either side of the transition).

The mathematics. Splits: $\binom{N}{k}$. Paths: $\varphi = \binom{N-1}{k-1} = \frac{k}{N}\binom{N}{k}$. [Course Book §16.4] At the platform default $N = 8, k = 2$: 28 splits, 7 paths, training fraction 0.75, 4 fits per path (verify_pb04.py §1).

Artifact. `cpcv_splitter` module; the per-timeframe config table above, checked into the harness config; the splitter's unit-test suite.

Pass / fail. All three unit tests above pass; a manual audit of one split's group assignment matches the config table by eye.

Trap — The daily timeframe cannot be CPCV-evaluated at $h = 50$

At a 50-bar horizon the daily timeframe contains $780/50 = 15.6$ independent labels over three years ([Course Book §16.5]); no combinatorial scheme manufactures a credible Sharpe distribution from fifteen units of information. The splitter above still accepts a D-timeframe config — it must, since D is a live platform timeframe — but no stage gate in this book may read a D-timeframe CPCV verdict as evidence. Use D only as a conditioning feed for faster timeframes, or shorten its horizon before gating on it: $h = 5$ on D gives $780/5 = 156$ independent labels, enough to be worth CPCV-splitting, at the cost of evaluating a materially shorter-horizon claim than the platform's standard 50-bar decision. [Course Book §16.5]

Step 4.3 — The metrics module

Do. Build one module, called by every stage from this chapter on, that turns a set of per-path net P&L series into a deflated verdict and a calibration verdict. Its two entry points:

```
deflated_sharpe(sr_hat, T, gamma3, gamma4, ledger_srs)  →  {V, N, SR0, DSR}
```

`christoffersen_battery(hits, p, regime=None)` → `{LR_uc, LR_ind, LR_cc, p_uc, p_ind, p_cc}`

`pit_calibration(y, q_lo, q_hi, cdf_fn=None)` → `{coverage_hat, hits, pit_values, ks_stat, ks_pvalue}`

`ledger_srs` is a live query against the trial ledger (Step 4.4) — for example `SELECT metrics->>'sr_hat' FROM trial_ledger WHERE tf=:tf AND status IN ('run','superseded')` — for every $\widehat{\mathrm{SR}}_n$ at the same bar frequency as `sr_hat`: V and N are read off the ledger, never off a researcher's recollection of how many configurations were tried.

The mathematics. Per-path Sharpe ratio at the timeframe's own bar frequency, $\widehat{\mathrm{SR}}^{(p)} = \hat{\mu}^{(p)}/\hat{\sigma}^{(p)}$, from the path's net-of-cost P&L series of Step 4.1. [Course Book §15.1] Trial variance V: the sample variance of every $\widehat{\mathrm{SR}}_n$ on the ledger at matched frequency. Expected-maximum Sharpe ratio under the null of no skill, with $\gamma = 0.5772156649$ the Euler–Mascheroni constant:

$$\mathrm{SR}_0 \approx \sqrt{V}\left[(1-\gamma)\,\Phi^{-1}\!\left(1-\tfrac{1}{N}\right) + \gamma\,\Phi^{-1}\!\left(1-\tfrac{1}{Ne}\right)\right].$$

[Course Book §15.4] Mertens standard-error term, evaluated at the selected strategy's own $\widehat{\mathrm{SR}}$, skewness γ_3 and kurtosis γ_4 (non-excess): $\sqrt{1-\gamma_3\widehat{\mathrm{SR}}+\frac{\gamma_4-1}{4}\widehat{\mathrm{SR}}^2}$. [Course Book §15.2] Deflated Sharpe Ratio:

$$\mathrm{DSR} = \mathrm{PSR}(\mathrm{SR}_0) = \Phi\left(\frac{(\widehat{\mathrm{SR}}-\mathrm{SR}_0)\sqrt{T-1}}{\sqrt{1-\gamma_3\widehat{\mathrm{SR}}+\frac{\gamma_4-1}{4}\widehat{\mathrm{SR}}^2}}\right).$$

[Course Book §15.3–15.4] Coverage and PIT, per path: breach indicator $I_t = \mathbf{1}\{y_t \notin [\hat{q}_{\mathrm{lo},t}, \hat{q}_{\mathrm{hi},t}]\}$, empirical coverage $\widehat{\mathrm{cov}} = 1-\bar{I}$; probability-integral-transform value $u_t = \hat{F}_t(y_t)$ against the forecast's own predictive CDF, tested for Uniform$(0,1)$ via a Kolmogorov–Smirnov statistic on $\{u_t\}$. [Course Book §17.3] Christoffersen battery on the same breach indicator I_t, claimed coverage p: unconditional coverage $\mathrm{LR}_{uc} \sim \chi^2(1)$, independence $\mathrm{LR}_{ind} \sim \chi^2(1)$, joint conditional coverage $\mathrm{LR}_{cc} = \mathrm{LR}_{uc} + \mathrm{LR}_{ind} \sim \chi^2(2)$; 5% critical values 3.8415 and 5.9915. [Course Book §17.2] Per-regime: pair counting uses the calendar-consecutive convention (only pairs adjacent in the original series count), and p is never re-centred on a bucket's own frequency. [Course Book §17.4]

Artifact. `metrics` module exposing the three call signatures above; a regression fixture tying the module to the course book directly: $N = 200$, $V = 0.0225$, $\widehat{\mathrm{SR}} = 0.31$, $T = 500$, $\gamma_3 = -0.4$, $\gamma_4 = 5$ must return $\mathrm{SR}_0 = 0.4148$ and $\mathrm{DSR} = 0.01700$ (4 s.f., verify_pb04.py §6, matching [Course Book §15.4]'s own worked example exactly).

Pass / fail. The regression fixture reproduces the cited figures to 4 s.f.; a second fixture at $N = 20$ on the same inputs returns $\mathrm{SR}_0 = 0.2851$, $\mathrm{DSR} = 0.6927$ (the course book's own "honest, smaller search" comparison, [Course Book §15.4]) — both must pass before the module may be called from any stage chapter.

Step 4.4 — The trial ledger

Do. One append-only SQLite table, written by the harness itself, never by hand.

column	type	contents
`trial_id`	text, PK	content hash of the row at write time
`created_ts`	timestamp	UTC, set by the harness, not the researcher
`author`	text	the human or process that launched the run
`config_hash`	text	SHA-256 of the trial config as canonical JSON (keys sorted, fixed float precision, no whitespace)
`universe`	text	instrument set evaluated
`tf`	text	timeframe
`cost_vintage`	text	the pinned cost-surface vintage of Step 4.1
`split_config`	text	N, k, purge, embargo used (Step 4.2)
`metrics`	JSON	T, $\widehat{\mathrm{SR}}$, γ_3, γ_4, per-path Sharpe array, turnover, DSR, PBO if computed
`status`	enum	`run` / `abandoned` / `superseded`
`notes`	text	free text; never a substitute for a status change

Index `(tf, status, cost_vintage)`: it is the predicate Step 4.3's `ledger_srs` query filters on for every DSR computed, and an unindexed scan of a multi-year trial history is the difference between a deflation that runs inside a research loop and one that does not.

Artifact. `trial_ledger` table; one owner named in the run config, accountable for its completeness on a fixed schedule.

Pass / fail. The write-discipline test: instrument the harness so that *every* evaluated configuration inserts its row *before* the calling code path can read the result back — write, then return, never the reverse. Abandoning a trial changes `status` to `abandoned`; nothing is ever `DELETE`d or `DROP`ped. [Course Book §15.6]

Trap — A leak-injection run is not a trial

The four controls of Steps 4.5–4.8 evaluate the *harness*, not a candidate strategy competing for capital. They must never write a `trial_ledger` row: doing so would silently inflate N in every deflation computed afterwards, understating every future DSR for a reason that has nothing to do with how many strategies were actually tried. They write to the CI report of Step 4.9 instead — a different table, with a different purpose, read by a different query.

The four controls, run as one indivisible unit

Key idea — Four legs, one protocol

Steps 4.5–4.8 are not four independent checks to run when convenient; they are one protocol with four legs, executed in this order, every time. Each targets a different failure mode — the engine's information discipline, the splitter's purge, a feed's timing, the cost model's wiring — and none of the other three can detect a failure specific to the fourth. Passing three of four proves nothing about the fourth: run all four, every time, or trust none of the results this harness has ever produced. [Course Book §18.4]

Step 4.5 — Control 1 — the sign oracle

Do. Append $x_{\text{leak},t} = \operatorname{sgn}(y_{t+1})$ to the feature matrix as an ordinary input. Rerun the *full* pipeline unchanged — Step 4.1's engine, Step 4.2's splitter, Step 4.3's metrics, the same pinned cost vintage — and record every per-path Sharpe ratio and the run's own realised annualised trade count $n^{(p)}$ per path. Then remove x_{leak} and rerun on fresh state (no cached feature matrix, no warm-started model, no unrefit scaler).

The mathematics. Oracle floor, i.i.d. Gaussian per-trade returns, zero cost: $1.324\sqrt{n}$ per annum, n the annualised trade count. [Course Book §18.4] Worked anchors (verify_pb04.py §4): $n = 240$ (a typical M30 book) $\Rightarrow$ 20.51; $n = 60$ (a typical H4 book) $\Rightarrow$ 10.25. These are illustrative, not universal — the operational rule is to compute $n^{(p)}$ from *this run's own* measured turnover on each path, never to assume a fixed constant per timeframe, since turnover is a property of the policy, not the timeframe.

Artifact. CI report row (Step 4.9) — not a ledger row (see the trap above Step 4.3).

Pass / fail. *Injection*: $\min_p \widehat{\mathrm{SR}}^{(p)} \geq 1.324\sqrt{n^{(p)}}$ on every path; clearing the floor by less than a factor of two on any path is a pass but must additionally be flagged for a manual purge audit, since a modest clearance is what a partially broken purge produces ([Course Book §18.4]) — verify_pb04.py §4 gives the audit lines (41.01 at $n = 240$, 20.51 at $n = 60$). *Removal*: $|\widehat{\mathrm{SR}}^{(p)}_{\text{removed}} - \widehat{\mathrm{SR}}^{(p)}_{\text{base}}| \leq \tau$ on every path, $\tau = 0.05$ (annualised Sharpe; recommendation, not a computation — two runs of identical code and data should reproduce to floating-point precision, so τ only absorbs incidental nondeterminism such as non-deterministic GPU reduction order, and is therefore set roughly two orders of magnitude below the sampling-noise band of Step 4.6, not calibrated to sampling noise at all).

Step 4.6 — Control 2 — regime-stratified label shuffle

Do. Permute y_t within each regime stratum, preserving the marginal distribution of outcomes per regime while destroying time-ordering. Rerun the *unmodified* pipeline (no injected feature) at the same pinned cost vintage and split config. [Course Book §18.4]

The mathematics. Under the null $\widehat{\mathrm{SR}} = 0$, the Mertens formula's skewness and kurtosis terms vanish identically — both scale with SR or SR^2 — leaving an exact, distribution-free null standard error

$$\mathrm{SE}_0(T) \;=\; \frac{1}{\sqrt{T-1}}.$$

[Course Book §15.2] T the path's bar count. The shuffled and scrambled nulls are i.i.d. by construction, so [Course Book §15.1]'s $\sqrt{q}$ annualisation scaling applies exactly (no autocorrelation term to carry): annualising over Y years of history at n bars/year ($T = nY$),

$$\mathrm{band}(Y, n) \;=\; \frac{z_c}{\sqrt{Y - 1/n}} \;\longrightarrow\; \frac{z_c}{\sqrt{Y}} \quad \text{as } n \to \infty.$$

Recommend $z_c = 3$: the suite reruns on every commit (Step 4.9), so a conventional two-sided 5% band ($z_c \approx 1.96$) would trip the freeze rule on close to one commit in twenty even with a perfectly clean harness — too high a nuisance rate for a gate that halts all result-reading. At the platform's standard three-year window ($Y = 3$) this gives band $\approx z_c/\sqrt{3} = 1.732$, *essentially the same value on M30, H1 and H4* (verify_pb04.py §3: 1.732074, 1.732097, 1.732236) — annualising cancels almost exactly against the extra precision finer bars would otherwise buy.

Artifact. CI report row: per-path shuffled Sharpe ratio against the band.

Pass / fail. $|\widehat{\mathrm{SR}}^{(p)}_{\text{shuf}}| \leq \mathrm{band}(3, n^{(p)})$ on *every* path — a single path outside the band fails the control, since a leak localised to one stretch of history is exactly the failure mode this control exists to catch, and averaging across paths would hide it.

Trap — Finer bars do not narrow the annualised noise band

It is tempting to assume M1's tens of thousands of bars make its null Sharpe estimate far more precise than H4's few thousand. At the *per-bar* level this is true (verify_pb04.py §3: the per-bar band $z_c\,\mathrm{SE}_0$ ranges from 0.01550 at M30 to 0.04386 at H4). At the *annualised* level it is almost exactly false: annualising multiplies the per-bar band by $\sqrt{\text{bars/year}}$, which grows precisely as fast as SE_0 shrinks. What is left depends on the *years* of history, not the bar frequency. Do not expect a faster timeframe's shuffle control to be a tighter test; it is not.

Step 4.7 — Control 3 — feature-timestamp scramble

Do. Choose a target point-in-time feature series and randomly permute its release timestamps while leaving the values themselves intact, then rerun the *unmodified* pipeline. COT is the platform's canonical target for this control by default, given its verified point-in-time gap flagged at Stage −1b; repeat the control against every other feed the policy grid under test actually consumes once Stage −1b's registry is populated. [Course Book §18.4]

The mathematics. The same null noise band as Step 4.6; not restated.

Artifact. CI report row: per-path scrambled Sharpe ratio against the band.

Pass / fail. $|\widehat{\mathrm{SR}}^{(p)}_{\mathrm{scr}}| \le \mathrm{band}(3, n^{(p)})$ on every path, for any performance attributable to the scrambled series. A pass here that accompanies a fail elsewhere means whatever the pipeline was using was not the series' timing — it was leaking through some other channel.

Step 4.8 — Control 4 — cost-doubling sensitivity

Do. For this run only, switch the cost lookup of Step 4.1 from the pinned vintage's 1× column to its pre-stored 2× column — Stage −1a stores 1.5× and 2× stress variants alongside the measured cost, so this control reads a column rather than recomputing one. Rerun the *unmodified* pipeline; record per-path Sharpe, per-path realised round-trip count $n^{(p)}_{RT}$, and per-path net P&L at both vintages.

The mathematics. A reconciliation identity, following directly from Step 4.1's net-P&L identity summed over a path's round trips:

$$\Delta(\text{net P\&L}) \;=\; -\,n_{RT}\,\Delta\bar{c},$$

n_{RT} the path's realised round-trip count, $\Delta\bar{c}$ the added round-trip cost (here, $\Delta\bar{c} = \bar{c}$, since cost exactly doubles). Worked check reusing [Course Book §14.2]'s own illustrative cost levels (gross edge 3.0 pips, $c : 1.2 \to 2.4$ pips, $n_{RT} = 240$/yr, the platform's M30 anchor): predicted $\Delta = -288.0$ pips/yr; ±10% reconciliation band $[-316.8, -259.2]$ pips/yr (verify_pb04.py §5).

Artifact. CI report row: per-path Δ(net P&L) against the reconciliation band, per-path Sharpe at both vintages.

Pass / fail. Two independent conditions, both required. (a) *Direction*: per-path Sharpe strictly lower at 2× cost than at 1×, on every path. [Course Book §18.4] (b) *Magnitude*: observed Δ(net P&L) within ±10% (recommendation) of $-n^{(p)}_{RT}\Delta\bar{c}$ computed from that path's *own* realised turnover. Direction without magnitude is not a pass: it means the cost model is wired in the right direction at the wrong scale, itself a bug worth catching before it reaches Step 4.3's deflation.

Step 4.9 — CI wiring and the freeze rule

Do. Package Steps 4.5–4.8 into a single automated job. Trigger it on every commit touching `backtest_engine/` (Step 4.1), `cpcv_splitter/` (Step 4.2), `metrics/` (Step 4.3), the cost-surface vintage pin, or Stage −1b's feed registry — and on every such trigger, also rerun Stage −1b's own as-of-join audit query as a precondition, since a feed regression is exactly the kind of change this suite exists to catch. On completion, write one CI report row per commit.

column	contents
`run_id, commit_hash, ts`	identity of the run
`control1_status, control1_value`	oracle: pass/fail, floor vs. observed
`control2_status, control2_value`	shuffle: pass/fail, band vs. observed
`control3_status, control3_value`	scramble: pass/fail, band vs. observed
`control4_status, control4_value`	cost: pass/fail, direction & magnitude
`overall_status`	`GREEN` iff all four `GREEN`, else `RED`

Artifact. The CI report table; the automated job definition.

Pass / fail. `overall_status` is the single bit every stage chapter from here on checks before reading anything else out of this harness: `RED` freezes all result-reading downstream of that commit — no DSR, PSR, or gate decision computed by this harness may be reported until a fresh full rerun turns it `GREEN` again, the licence rule of [Course Book §18.4].

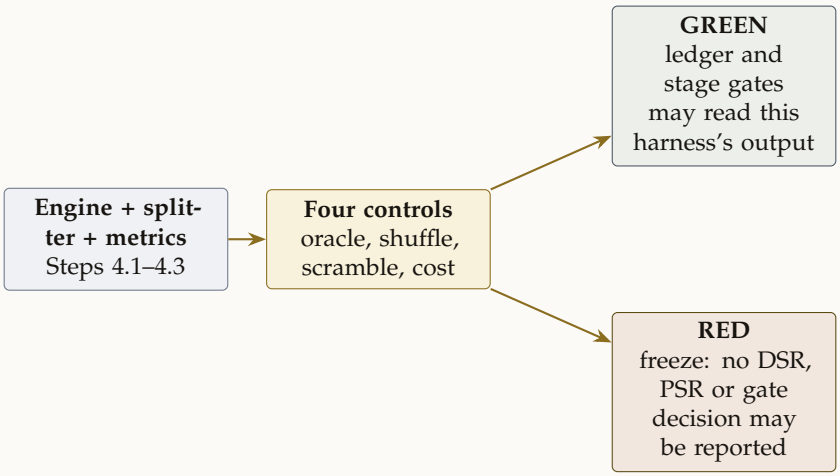


Figure 4.1. The freeze rule. All four controls must be green simultaneously before any Stage 0 or later result computed by this harness may be reported; a single red control freezes the whole suite, not just the affected metric.

Stage gate — Stage −1c

The gate is binary and has no partial-credit reading: all four controls of Steps 4.5–4.8 green, simultaneously, on the assembled harness (engine, splitter, metrics, cost lookup, all as they will actually run in Stage 0). Passing three of four is not a partial pass; it is a fail with one leg still untested. [Course Book §18.4]

Kill criterion — Stage −1c

There is no strategy to kill here — the harness is infrastructure, not a candidate competing

for capital, so no capital or research programme is terminated by a red suite. Its failure mode is procedural, not financial: a red suite freezes all result-reading downstream of the failing commit. No DSR, no PSR, no coverage statistic, no gate decision computed by this harness — for Stage 0 or any later stage — may be read, reported, or used to advance or kill a stage until the suite is rerun green on the *exact* commit that will produce the reported numbers. The freeze notice names three things: the commit hash that failed, the ledger owner (notified immediately), and which of the four controls failed and by how much. It lifts only on a fresh green run of all four controls together — never on a partial rerun of the failing control alone, since a fix to one leg can silently break another. [Course Book §18.4]

Artifacts — Stage −1c output register

- `backtest_engine` module and its run-config schema (Step 4.1).
- `cpcv_splitter` module, the per-timeframe purge/embargo config table, and its unit-test suite (Step 4.2).
- `metrics` module (`deflated_sharpe`, `christoffersen_battery`) and its course-book-anchored regression fixtures (Step 4.3).
- `trial_ledger` table, append-only, with a named owner (Step 4.4).
- The leak-injection CI suite: four control implementations and their thresholds — oracle floor $1.324\sqrt{n}$, removal tolerance $\tau = 0.05$, shuffle/scramble noise band $z_c/\sqrt{Y-1/n}$ at $z_c = 3$, cost-reconciliation band $\pm 10\%$ (Steps 4.5–4.8).
- CI report table and the automated job definition wiring the suite to every commit touching engine, splitter, metrics, or feeds (Step 4.9).

PART III

The Kill-Gated Stages



LUNA LABS

5

Stage 0: The Envelope Honesty Test

This is the stage that can end the programme. The platform's TimesFM envelope has been checked for median accuracy and for band coverage; it has never been checked for money. Stage 0 asks one question and pre-commits to its answer: does any *function of the existing envelope, used as a size multiplier, beat naive volatility-target sizing on out-of-sample net-of-cost P&L, deflated for the number of things tried? Everything in this chapter exists to make that question answerable exactly once. Regenerate the forecast history at live settings, because a verdict computed on other settings is a verdict about a different model. Declare the policy grid before the run, because its size is the deflation input. Normalise exposure on train splits only, because otherwise the benchmark comparison is a leverage comparison. Write the memo, get it signed, then run. The order is not negotiable, and neither is the kill wording once results exist.*

5.1 Preconditions and scope

Stage 0 does not begin until three gates are green and stay green for the duration of the run. Any amber gate freezes result-reading, not just result-generation [Course Book §18.4].

Gate	Owner	Condition that must hold on the Stage-0 commit
Chapter 2 (Stage −1a)	data	`cost_surface` populated for all 21 instruments; coverage of historical bars by cells with $n \geq 100$ at or above 99 per cent; vintage frozen and recorded.
Chapter 3 (Stage −1b)	data	every feed answers the as-of join with `release_ts` $\leq t -$ safety lag; audit query green [Course Book §18.2].
Chapter 4 (Stage −1c)	research	all four leak-injection controls green on the exact harness commit used for the run; CI report attached to the memo [Course Book §18.4].

Scope is declared before anything is computed, because scope is a trial dimension. The Stage-0 universe is all 21 instruments on three timeframes — M30, H1, H4 — giving 63 (instrument, timeframe) series. The finer timeframes are excluded because the 50-bar horizon collapses to 12.5 hours at M15 and less below it, so round-trip cost dominates the move the envelope is describing; D is excluded because 780 daily bars over three years cannot be cut into cross-validation groups that are long relative to a 50-bar horizon (Step 5.5). Adding a timeframe later is not a free extension: it multiplies the trial count.

Step 5.1 — Regenerate the forecast history at live settings

Do. Treat `forecasts_hist.sqlite` as unusable evidence and demote it to a smoke-test fixture. It holds only H1 and M30, only the 10 FX majors, is roughly a month stale, and was generated at context 1024 while live H1 inference runs at 1536. Regenerate from scratch:

(1) **Read the live generation configuration out of the running service, not out of documentation.** Capture, per (instrument, granularity): checkpoint identity and SHA-256, context length L, input and output patch sizes, the quantile list, the static `width_mult` value, dtype, batch shape, PyTorch and CUDA versions, driver version and device name. Write one `gen_config` row; the entire regeneration carries that single `gen_id`.

(2) **Batch by (granularity, bar close).** One batch is all 21 instruments at one granularity at one bar close: 21 contexts of length L, one forward pass, 21 output rows. Iterate bar closes forward in time. Slice every context strictly from bars with `close_ts` $\leq$ `bar_ts` and assert that bound in the loader rather than trusting the query.

(3) **Estimate the runtime before committing the GPU.** Warm up with 20 batches and discard them; time 200 consecutive batches at each distinct (granularity, L) shape; take the p95 per-batch latency; multiply by the batch count for that granularity; sum; add a 25 per cent margin for SQLite writes and checkpointing. Abort and re-plan if the total exceeds the declared budget rather than discovering it overnight.

(4) **Checkpoint every 5,000 batches** with the last completed (granularity, `bar_ts`) so an interrupted run resumes without regenerating rows, and never resumes under a different `gen_id`.

(5) **Pin the GPU.** The RTX 3060 is the live device; run on it. The idle 5080 is unresolved and is not a Stage-0 device.

Counts, from `verify_pb05.py`: with a 120-hour trading week and 52 weeks per year over three years, the universe is 37,440 M30 bars, 18,720 H1 bars and 4,680 H4 bars per instrument, so 60,840 batches and $21 \times 60{,}840 = 1{,}277{,}640$ forecast rows.

TF	live L	tokens	batches	p95 ms/batch	GPU hours
M30	1024	32	37,440	145.0	1.508
H1	1536	48	18,720	180.0	0.936
H4	1536	48	4,680	180.0	0.234
total	—	—	**60,840**	—	**2.678**
with the 25 per cent margin, wall clock					**3.348**

The latencies above are the illustrative measured inputs the procedure consumes; substitute the measured p95 from step (3) and re-run the estimator. Token counts follow from the input patch size of 32 [Course Book §9.2]. Running M30 at 1536 as well — the natural mistake — costs 13.6 per cent more wall clock *and* invalidates the run.

The mathematics. The stored band is the model's raw band rescaled by the live static multiplier: $q_{\tau,t} = \hat{m}_t + \kappa_i \left(\hat{q}^{\text{raw}}_{\tau,t} - \hat{m}_t\right)$ for $\tau \in \{0.10, 0.20, 0.80, 0.90\}$. Here $\hat{m}_t$ is the forecast median at bar t, $\hat{q}^{\text{raw}}_{\tau,t}$ the model's raw τ-quantile, $\kappa_i \in [0.95, 1.07]$ the static per-instrument `width_mult`, and $q_{\tau,t}$ what the C# engine actually polls. [Course Book §9.2]

Artifact. `forecasts_s0.sqlite`, two tables.

Table	Column	Type and meaning
forecasts_s0	instrument	TEXT, OANDA symbol
	granularity	TEXT, one of M30, H1, H4
	bar_ts	INTEGER, epoch seconds of the bar close the forecast was made at
	median	REAL, $\hat{m}_t$ in price units
	q10, q20, q80, q90	REAL, post-width_mult quantiles
	gen_id	INTEGER, FK to gen_config
	PK	(instrument, granularity, bar_ts)
gen_config	gen_id	INTEGER PK
	checkpoint_sha256	TEXT, model file digest
	context_len	INTEGER, per granularity, copied from the live service
	patch_in, patch_out	INTEGER
	width_mult_json	TEXT, the per-instrument κ_i map as generated
	code_sha256	TEXT, digest of the generation script
	torch_ver, cuda_ver	TEXT, the exact inference stack
	gpu_name, dtype	TEXT, device and precision
	run_start_ts, run_end_ts	INTEGER

At 104 bytes per row including the primary-key index the database is about 126.7 MiB — small enough to keep every regeneration rather than overwrite one.

Pass / fail. Three checks, all scripted. (a) *Completeness*: row count equals 1,277,640 minus the documented candle gaps, and every gap is reconciled against the candle table by (instrument, granularity, bar_ts). (b) *Parity*: pick 500 bars for which the live timesfm.sqlite already holds a row, regenerate them under gen_id, and require agreement on all five columns — bitwise with deterministic kernels enabled, otherwise maximum relative deviation below 10^{-4}. Any larger divergence means the captured configuration is not the live configuration; stop. (c) *Parity of intent*: assert gen_config.context_len equals the value read live for every (instrument, granularity), and that exactly one gen_id appears in forecasts_s0.

Trap — Context parity is the whole point

A Stage-0 verdict computed on forecasts generated at context 1024 when live H1 runs at 1536 is a verdict about a different model. The band width, the quantile spacing and therefore every feature in Step 5.2 all move with context length; a policy tuned on the short-context band is not the policy that would run. The same applies to checkpoint, dtype and patch sizes. If the live configuration changes during Stage 0, the run is void: regenerate, re-run, and add the discarded run to the trial ledger — it was evaluated, so it counts.

5.2 The features and the declared class

The envelope carries five numbers per bar. Everything Stage 0 may use is a backward-only function of those five and of price. Define the features once, freeze them, and never add one after seeing a result — an added feature is a new grid, and a new grid is a new N.

Step 5.2 — Build the feature table

Do. Compute, per (instrument, timeframe, bar t), the columns below into features_s0, keyed identically to forecasts_s0 and joined on (instrument, granularity, bar_ts). Every window is strictly backward-looking and excludes bar t's own future. Drop the warm-up prefix; do not forward-fill it.

Column	Symbol	Definition
`width`	W_t	$q_{0.90,t} - q_{0.10,t}$, in price units
`dwidth`	ΔW_t	$\ln W_t - \ln W_{t-1}$
`dwidth_s`	$\Delta W_t^{(n)}$	$\frac{1}{n}\sum_{j=0}^{n-1} \Delta W_{t-j}$, $n = 8$
`band_pos`	b_t	$(P_t - q_{0.10,t})/(q_{0.90,t} - q_{0.10,t})$, P_t the bar close
`band_pos_c`	β_t	$(P_t - \hat{m}_t)/W_t$, the centred variant [Course Book §14.7]
`asym`	A_t	$[(q_{0.90,t} - \hat{m}_t) - (\hat{m}_t - q_{0.10,t})]/W_t$
`width_pct`	u_t	$\frac{1}{L}\sum_{j=1}^{L} \mathbf{1}\{W_{t-j} \le W_t\}$, L per timeframe
`band_pos_a`	β_t^a	$d_t\,(2b_t - 1)$, band position aligned to the base direction d_t of Step 5.3
`regime`	g_t	INTEGER, written as 0 at Stage 0; the Stage-2 router populates it [Course Book §13.8]

Declare the percentile window in *wall clock*, not in bars, and convert per timeframe. The recommendation is 1,000 trading hours — 8.3 trading weeks, long enough to span a volatility cycle and short enough not to average across a policy regime — giving $L = 2000$ bars on M30, 1000 on H1 and 250 on H4. Each costs the same 5.34 per cent of its timeframe's history as warm-up, and each buys the same embargo bill in Step 5.5, which a bar-denominated L would not: a flat $L = 2000$ bars applied to H4 would consume 42.7 per cent of that timeframe's history before a single feature was usable. The smoothing length $n = 8$ is a recommendation: one shift's worth of bars on M30 and H1, enough to damp single-bar band jitter without outrunning the 50-bar horizon.

The mathematics. Under a Gaussian working model the width is a scale statistic, $W_t = 2\,\Phi^{-1}(0.9)\,\hat{\sigma}_t = 2.5631\,\hat{\sigma}_t$, so $\hat{\sigma}_t = W_t/2.5631$. Here $\hat{\sigma}_t$ is the envelope's implied volatility over the 50-bar horizon and $\Phi^{-1}(0.9) = 1.281552$. This is the only conversion Stage 0 needs, and it is what makes the volatility-target benchmark computable from the band alone. [Course Book §10.5]

Artifact. `features_s0` table, one row per (instrument, granularity, `bar_ts`), plus a `feature_config` row recording L per timeframe, n, and the code digest.

Pass / fail. (a) *Causality*: for 1,000 randomly sampled rows, recompute every feature from a copy of the source tables truncated at `bar_ts` and require exact equality — this is the feature-causality control from Chapter 4's CI suite pointed at Stage-0 columns. (b) *Warm-up*: the first L bars per (instrument, granularity) are NULL, and the count of NULLs matches L exactly. (c) *Range*: $W_t > 0$ everywhere; $u_t \in [0,1]$; flag and investigate any bar with $b_t \notin [-3,4]$ before it reaches a policy.

Step 5.3 — Declare the policy grid — its size is the deflation input

Do. Fix the base signal first. Stage 0 tests *sizing*, not direction, so a single declared direction is shared by every policy and every benchmark: $d_t \in \{-1, 0, +1\}$, the live C# engine's entry/exit state replayed deterministically over history from $\mathcal{F}_t$ information only. Holding d_t common makes every comparison a paired difference, so the engine's own edge (or lack of it) cancels out of the contrast that Stage 0 measures. Changing the base signal is not a tweak: it doubles the ledger.

Then enumerate the three families below into `policy_grid.json`, one JSON object per cell with a canonical-form `config_hash`. All multipliers $m_t \ge 0$; the position is $w_t = d_t\, m_t\, \kappa_\pi\, U$

with U the base unit and κ_π the exposure normaliser of Step 5.4.

Family A — inverse-width linear. $m_t^A = \min\big(a + b/\max(u_t, 0.05),\ m_{\max}\big)$, the width-percentile floor at 0.05 keeping the map finite. Grid: $a \in \{0.00, 0.25, 0.50\}$, $b \in \{0.10, 0.20, 0.40\}$, $m_{\max} \in \{2.0, 3.0\}$ — $3 \times 3 \times 2 = 18$ cells.

Family B — monotone bucket ladders. Cut a feature v_t into K equal-count buckets over the same trailing window as u_t; assign bucket k (counting from the low end) the multiplier $m_t^B = m_{\text{top}} + (m_{\text{bot}} - m_{\text{top}})\, k/(K-1)$, a monotone ladder by construction so the family cannot fit a non-monotone shape. Grid: $K \in \{3, 5\}$, $m_{\text{top}} \in \{1.5, 2.0, 2.5\}$, $m_{\text{bot}} \in \{0.00, 0.25, 0.50\}$, $v \in \{u_t,\ \Delta W_t^{(8)},\ \beta_t^a\}$ — $2 \times 3 \times 3 \times 3 = 54$ raw cells.

Family C — threshold gates on the volatility-target base. $m_t^C = \mathbf{1}\{u_t < x\} \cdot \mathbf{1}\{G(\Delta W_t^{(8)})\} \cdot \min\big(1/\hat{\sigma}_t,\ w_{\max}\big)$, with $\hat{\sigma}_t = W_t/2.5631$. Grid: $x \in \{0.3, 0.5, 0.7, 0.9\}$, $G \in \{\text{none},\ \Delta W^{(8)} \le 0,\ \Delta W^{(8)} \ge 0\}$, $w_{\max} \in \{2.0, 3.0\}$ — $4 \times 3 \times 2 = 24$ cells. The target volatility σ^* is absent by design: it is a pure scalar and would be normalised away, so putting it on the grid would buy nothing but three times the trials.

Prune scalar duplicates before evaluating anything. Matched exposure (Step 5.4) makes two policies whose multiplier paths are proportional *identical*, so proportional cells must be collapsed to one representative before the run. In Family B the three ladders with $m_{\text{bot}} = 0$ are mutually proportional for every (K, v) pair, removing $2 \times 6 = 12$ cells. Cells removed this way were never evaluated and therefore never entered the ledger [Course Book §15.4].

Family	raw cells	pruned	distinct
A — inverse-width linear	18	0	18
B — monotone bucket ladders	54	12	42
C — gate × vol-target	24	0	24
declared grid	**96**	**12**	**84**
run on 3 timeframes, so ledger rows written			**252**
benchmarks (cannot be selected, so not trials)			2
configurations run per timeframe			86

$N = 252$ is the deflation input, not 84. The timeframe is a searched dimension: all three are run and the best is nominated, so every (policy, timeframe) pair could have been selected and every one bought a ticket [Course Book §15.4].

The mathematics. Per-bar net P&L, the quantity every policy is scored on, is the engine contract of Chapter 4's Step 4.1 evaluated at cost stress λ:

$$\Pi_t \;=\; w_{t-1}\, r_t \;-\; \lambda\, c\big(i, \text{cell}(t)\big)\, |w_t - w_{t-1}|.$$

Here w_t is the signed position after bar t, r_t the bar return of instrument i, $c(i, \text{cell}(t))$ the round-trip cost from the pinned `cost_surface` vintage, and $\lambda \in \{1.0, 1.5, 2.0\}$ the stress multiplier. [Course Book §14.1] [Course Book §14.2]

Artifact. `policy_grid.json` (84 objects, each with `family`, parameters, `config_hash`), its SHA-256 recorded in the sign-off memo, and 252 rows in `trial_ledger` with `status = 'declared'` written before the first result exists.

Pass / fail. (a) `len(policy_grid)` $= 84$ and all `config_hash` values distinct. (b) A pairwise proportionality scan over the 84 multiplier paths, evaluated on a held-out sample of 10,000 bars, returns no proportional pair — the pruning was complete. (c) `SELECT COUNT(*) FROM trial_ledger WHERE stage='0'` returns 252, and max(`created_ts`) over those rows is earlier than min(`created_ts`) over any Stage-0 result row.

Trap — N is not the size of the grid you kept

The deflation input is the number of configurations *evaluated*, including every cell run and abandoned, every re-run after a bug fix that changed returns, and every timeframe the grid was swept over. With $\sqrt{V} = 0.01$ per bar the bar that luck alone sets moves from 0.0247 per bar at the flattering count $N = 84$ to 0.0284 at the honest $N = 252$, and to 0.0370 if per-instrument slices are mined for a winner ($N = 84 \times 3 \times 21 = 5{,}292$). A champion at $\widehat{\mathrm{SR}} = 0.0330$ per bar reads DSR $= 0.87$ against the wrong count, 0.73 against the honest one, and 0.29 if it was found by slicing — the same backtest, three different verdicts, and only one of them admissible.

5.3 Benchmarks, the run, and the deflation

Step 5.4 — Build the two benchmarks and match exposure

Do. Two benchmarks, both parameter-free once the base signal is fixed, so neither can be accused of having been tuned.
B1, constant fractional risk. $w_t^{B1} = d_t\, U$. The null of "the envelope contains nothing": it reads no forecast column at all.
B2, volatility targeting. $w_t^{B2} = d_t \min\left(\sigma^*/\hat{\sigma}_t,\ w_{\max}\right) U$ with $\hat{\sigma}_t = W_t/2.5631$ and $w_{\max} = 3$. The null of "the envelope contains scale and nothing else". This is the benchmark the programme's kill gate is written against, and it is not a straw man: any policy that is a decreasing function of width alone is a reparametrisation of it [Course Book §14.4].
Then normalise exposure. Compute, for every (policy π, timeframe, CPCV path), the scalar κ_π on the *train* bars of that path only, freeze it, and apply it unchanged to the test bars. Use B1 as the single yardstick so that all 86 configurations — including B2 — are expressed at one common average exposure and the comparison is of shape, not leverage. Record κ_π per (policy, timeframe, path) in the results table; a κ_π outside $[0.1, 10]$ is a defect in the policy definition, not a scaling to accept.

The mathematics. The volatility-target rule and the exposure normaliser:

$$w_t = \min\left(\frac{\sigma^*}{\hat{\sigma}_t},\ w_{\max}\right), \qquad \kappa_\pi = \frac{\bar{E}_{B1}}{\bar{E}_\pi}, \qquad \bar{E}_\pi = \frac{1}{|\mathcal{T}_{\mathrm{tr}}|} \sum_{t \in \mathcal{T}_{\mathrm{tr}}} |w_t^\pi|.$$

Here σ^* is the target volatility, $\hat{\sigma}_t$ the envelope-implied volatility, $w_{\max}$ the leverage cap, $\mathcal{T}_{\mathrm{tr}}$ the train bars of the path in hand, $\bar{E}_\pi$ the policy's mean absolute exposure on them and $\bar{E}_{B1}$ B1's mean absolute exposure on the same bars. [Course Book §14.4]
Two consequences follow and both are load-bearing. First, σ^* cancels: scaling every w_t^{B2} by a constant scales $\bar{E}_{B2}$ by the same constant and leaves $\kappa_{B2} w_t^{B2}$ unchanged wherever the cap is slack, which is why σ^* is not on the grid. Second, any two policies with proportional multiplier paths become the same policy after normalisation — the pruning rule of Step 5.3.

Artifact. `benchmarks_s0` (two rows, hashed like grid cells) and the `exposure_norm` table: (policy_id, granularity, path_id, kappa, mean_exposure_train, mean_exposure_test).

Pass / fail. (a) For every (policy, timeframe, path): $|\bar{E}^{\mathrm{train}}_{\kappa\pi} / \bar{E}^{\mathrm{train}}_{B1} - 1| < 10^{-9}$. (b) Test-split mean exposure is *not* forced to match; report the ratio as a diagnostic and investigate any policy whose test exposure drifts beyond ± 25 per cent, since that is a policy whose behaviour is regime-dependent in a way the train split did not see. (c) κ is constant within a (policy, timeframe, path) triple — assert it, because a per-fold recomputation on test bars is exactly the leak this step exists to prevent.

Trap — Where the exposure normaliser leaks

Computing κ_π on the full sample uses test bars to set the leverage a test-bar P&L is measured at. Recomputing it per test split is worse: it hands each fold its own hindsight. Both produce a benchmark comparison that flatters whichever policy has the more volatile exposure profile, and neither is detectable in the headline Sharpe. Normalise on train bars, freeze, apply.

Step 5.5 — Run the grid under CPCV at the pinned cost vintage

Do. Run the 86 configurations per timeframe through the Chapter 4 harness on the splitter of Step 4.2, unchanged, in the assembled order: split, then fit, then score, then log [Course Book §16.6]. Groups are contiguous wall-clock slices cut at 5 pm New York daily closes and span all 21 instruments and all timeframes simultaneously [Course Book §16.5]; purge 50 bars — one horizon — on each side of every train/test boundary and apply the per-timeframe embargo [Course Book §16.2]. Group counts differ per timeframe because a group must stay long relative to the horizon; the declared rule is group length $\geq 10\times$ horizon.

TF	N_g	k	splits	paths	group bars	group / horizon
M30	12	2	66	11	3,120	62.4×
H1	10	2	45	9	1,872	37.4×
H4	8	2	28	7	585	11.7×
total	—	—	**139**	**27**	—	—

D would give 97-bar groups at $N_g = 8$, a mere 1.94× the horizon, which is why it is out of scope. Splits and paths follow the counting identities of Step 4.2 [Course Book §16.4].

Purge and embargo are applied in wall-clock units at each timeframe's own bar width. The purge is one horizon, 50 bars, on each side of every train/test boundary. The embargo follows the Step 4.2 rule — $\delta \geq$ (widest feature window) − (horizon) — and at Stage 0 the widest feature window is the rolling width percentile L, so $\delta = L - 50$. Set it to that minimum and price it:

TF	L (bars)	purge (bars)	δ (bars)	δ (hours)	train bars lost
M30	2,000	50	1,950	975	13.1%
H1	1,000	50	950	950	14.0%
H4	250	50	200	800	17.1%

The loss is computed against that split's training bars, $(N_g - k)/N_g$ of the sample, with $k = 2$ embargo regions and up to $2k$ purged boundaries. This is the real price of a long-memory feature and it is why L was declared in wall clock: a bar-denominated $L = 2000$ on H4 would embargo 1,950 of 3,510 training bars and leave nothing to fit κ_π on. If a future feature needs a longer window, its embargo cost is computed and declared *in the memo*, before the run [Course Book §16.2].

Execution follows the engine contract without exception: decisions from $\mathcal{F}_t$, fills at the $t{+}1$ open, cost looked up per (instrument, cell(t)) from the *pinned* vintage [Course Book §18.3]. Sweep $\lambda \in \{1.0, 1.5, 2.0\}$ inside the innermost loop so the three stress runs share one set of fills and one set of κ_π values; re-running the whole harness per λ invites drift between the stress variants.

Volume, from `verify_pb05.py`: $86 \times 27 = 2{,}322$ path evaluations per cost vintage, 6,966 across the three; $86 \times 139 = 11{,}954$ train-split exposure normalisations. Persist, per (config,

timeframe, path, λ): the net P&L series, its per-bar Sharpe, total net P&L, turnover, gross cost paid, exposure statistics, and the per-instrument decomposition of net P&L. Write the decomposition to its own table and seal it: no query against `stage0_by_instrument` until the verdict row of Step 5.7 exists. Sealing is enforced by permissions, not by intention.

Artifact. `stage0_results` (config_id, granularity, path_id, lambda, cost_vintage, sharpe_bar, pnl_net, turnover, cost_paid, kappa, n_bars) plus `stage0_pnl_series` (the per-bar series, needed for the paired differences of Step 5.6) and `stage0_by_instrument`.

Pass / fail. (a) The leak-injection suite is green on the exact commit that produced these rows; attach the CI report hash. (b) *Cost reconciliation*: because cost enters linearly, $\Pi(\lambda) = \Pi(0) - \lambda C$ with $C = \sum_t c_t |w_t - w_{t-1}|$; assert $\Pi(1.0) - \Pi(2.0) = \Pi(0) - \Pi(1.0) = C$ to within 10^{-6} on every row. A failure here means the cost lookup is not vintage-pinned or the turnover accounting is wrong. (c) Every path covers every bar of the sample exactly once, and no test bar's label window intersects its own train split — the splitter unit tests from Step 4.2, re-run on this universe. (d) Per-instrument P&L sums to the portfolio P&L for every row.

Step 5.6 — Deflate against the ledger

Do. Score the contrast, not the level. For each policy π, benchmark $B \in \{B1, B2\}$, timeframe and path, form the paired difference series $\Delta_t^{\pi,B} = \Pi_t^{\pi} - \Pi_t^{B}$ over the identical bars, and compute its per-bar Sharpe. This is the excess-over-benchmark performance that the multiple-testing machinery is built for [Course Book §15.6], and pairing removes the base signal's own contribution from both sides.

Then, per (benchmark, timeframe):

(1) Take $\widehat{\mathrm{SR}}_n$ for $n = 1, \ldots, 84$ as the median-over-paths per-bar Sharpe of $\Delta^{\pi_n,B}$ at $\lambda = 1.0$, and set V to their sample variance. Compute V at the same bar frequency as the champion's own Sharpe; mixing a per-bar Sharpe with an annualised cross-trial variance inflates the bar by $\sqrt{q}$ and rejects everything [Course Book §15.4].

(2) Read N from the ledger, not from the grid file: `SELECT COUNT(*) FROM trial_ledger WHERE stage='0'`. That query *is* the definition of N. Declared cells, abandoned cells, superseded re-runs and every timeframe sweep are all in it.

(3) Compute SR_0 and DSR with the metrics module of Step 4.3, using the champion's own T, skewness and kurtosis. Do not restate the formulas here; call the module. [Course Book §15.4]

(4) Compute the worst-decile path statistic on the champion *policy*'s full path set — all 27 paths, 11 M30 plus 9 H1 plus 7 H4, since the policy is the object promoted and its behaviour on every path of every timeframe is evidence about it. Take net P&L in account currency at $\lambda = 1.0$, sort ascending, and interpolate at position $1 + 0.10\,(27 - 1) = 3.6$: 60 per cent of the way from the third-lowest path to the fourth. Report the nominated timeframe's own minimum path alongside it as a diagnostic.

(5) Compute PBO from the path rankings as a secondary read: for each split of the path set into complementary in-sample and out-of-sample halves, rank the 84 policies in-sample, take the in-sample winner, and record its out-of-sample rank; PBO is the fraction of splits in which that winner lands in the bottom half out-of-sample [Course Book §15.5]. It does not enter the gate; it enters the memo, and a PBO above one half beside a passing DSR is a contradiction to be resolved before promotion, not an inconvenience to be omitted.

Four things count as trials at Stage 0 besides the 252 declared cells, and each is a ledger row that is written, never deleted: a declared cell whose run crashed and was not retried

(`status = 'abandoned'`); an exploratory policy evaluated on Stage-0 data while debugging the harness (`'abandoned'`, with a note naming the debugging session); a cell re-run after a fix that changed its returns (the original marked `'superseded'`, the new row `'run'`, both counted); and a void run under Step 5.8's void conditions (all its rows kept). The count query filters on `stage` and on nothing else — a `WHERE status = 'run'` in the deflation input is the single most consequential one-line bug available in this chapter [Course Book §15.4].

The champion is nominated once, before any of this: the (policy, timeframe) pair with the highest median-over-paths $\Delta^{\pi,B2}$ Sharpe at $\lambda = 1.0$. Nominating after inspecting the deflation is selection on the test statistic.

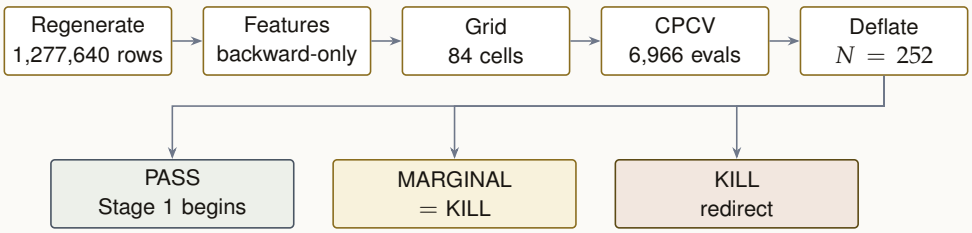
How much the count matters, at $\sqrt{V} = 0.01$ per bar (H1 clock, annualiser $\sqrt{6{,}240} = 78.99$):

N	SR_0/bar	annualised	**what that N means**
84	0.0247	1.9501	the grid only — the wrong count
252	0.0284	2.2436	grid $\times$ 3 timeframes — honest
504	0.0305	2.4132	plus one re-run after a bug fix
5,292	0.0370	2.9245	plus mining the per-instrument slices

Artifact. `stage0_deflation`: (benchmark, granularity, champion_config_id, n_trials, V, sr0_bar, sr_hat_bar, T, skew, kurt, dsr, pbo, q10_path_pnl, lambda) — one row per (benchmark, timeframe, λ) — plus the ledger itself, exported and hashed.

Pass / fail. (a) An independent second implementation recomputes SR_0 and DSR from the exported ledger and agrees to 10^{-9}. (b) The ledger count equals declared cells plus abandoned cells plus re-runs, reconciled line by line against the run log; an unexplained discrepancy voids the deflation. (c) DSR > 0.5 if and only if $\widehat{SR} > SR_0$ — assert the identity as a unit test on the returned pair, since it is the cheapest possible detector of a frequency mismatch.

5.4 The verdict



Step 5.7 — Read the results against the pre-committed protocol

Do. Evaluate the four conditions below in order and stop at the first failure. Read them off `stage0_deflation`; do not compute anything new at this point, and do not open the per-instrument breakdown until the verdict is recorded.

	Condition	**at $\lambda = 1.0$**	**at $\lambda = 1.5$**
C1	DSR of $\Delta^{\pi^\dagger,B1}$	> 0.60	> 0.50
C2	DSR of $\Delta^{\pi^\dagger,B2}$	> 0.60	> 0.50
C3	median-over-paths Sharpe	$>$ both benchmarks	$>$ both benchmarks
C4	q10 path net P&L	> 0 by ≥ 1 standard error	> 0

$\lambda = 2.0$ is reported but does not gate: it is the diagnostic that says how far the result is from

the cliff. The 1 standard error in C4 is the standard error of the path mean across the 27 paths, computed at $\lambda = 1.0$.

Three verdicts exist, and only three.

Verdict	Definition, fixed before the run
Clear pass	C1–C4 all hold at $\lambda = 1.0$ *and* at $\lambda = 1.5$.
Marginal	Any of: DSR $\in (0.50, 0.60]$ against either benchmark; q10 path P&L positive but within one standard error of zero; C1–C4 holding at $\lambda = 1.0$ but any of them failing at $\lambda = 1.5$.
Clear kill	Any of C1–C4 fails at $\lambda = 1.0$.

Marginal is kill. Under $N = 252$ trials the distribution of the best result is shifted upward by selection alone, so a champion sitting a fraction of a standard error above the bar that luck sets is precisely the observation the null predicts [Course Book §15.4]; and the probability of backtest overfitting is at its highest exactly where the ranking of policies is unstable across paths, which is what a marginal spread between champion and runner-up means [Course Book §15.5]. There is no third option in which more data or a longer look resolves the ambiguity, because the ambiguity is the answer.

The mathematics. The arithmetic of the three verdicts, on the H1 clock with the champion's path length $T = 18{,}720$ bars, $\sqrt{V} = 0.01$ per bar, $N = 252$ so $\mathrm{SR}_0 = 0.0284$ per bar, skewness -0.5 and kurtosis 6:

$\widehat{\mathrm{SR}}$/bar	annualised	surplus	DSR	**verdict**
0.0270	2.1328	−0.00140	0.4245	clear kill: below the bar luck sets
0.0286	2.2592	+0.00020	0.5107	marginal — kill by pre-commitment
0.0330	2.6068	+0.00460	0.7335	clear pass, if C3 and C4 agree

Note how little separates the rows in Sharpe terms and how much in verdict terms: at $T = 18{,}720$ the DSR is a steep function of the surplus, which is exactly why C3 and C4 exist. A gate that rested on DSR alone would be a knife-edge.

Artifact. `stage0_verdict`: (verdict, champion_config_id, granularity, condition-by-condition pass/fail at each λ, deflation row hashes, ledger hash, decided_ts, decided_by).

Pass / fail. The verdict row is written and hashed *before* any breakdown table is opened, and the recorded `decided_ts` precedes the first access timestamp on `stage0_by_instrument`. This is auditable and should be audited.

Trap — The breakdowns are diagnostics, never selectors

Per-instrument and per-timeframe P&L exist to explain a verdict, not to find one. Selecting the champion on a slice — "it works on the majors", "drop XAG and it passes" — multiplies the trial count by the number of slices examined: $84 \times 3 \times 21 = 5{,}292$, moving the bar from 0.0284 to 0.0370 per bar and turning the illustrative DSR = 0.7335 champion into 0.2928. If a slice genuinely looks different, that is a Stage-2 hypothesis with its own memo and its own N, not a Stage-0 rescue.

Step 5.8 — Write the pre-commitment memo, sign it, then run

Do. The memo is written and signed *before* the harness runs, with the fields below in this order. Angle-bracketed slots are filled from artifacts; the quoted sentences are fixed wording and are copied verbatim.

1. **Data.** "Regenerated TimesFM bands at live generation settings, `gen_id` <id>, checkpoint <sha256>, context <L per granularity>; all 21 instruments; timeframes M30, H1, H4; <n_rows> forecast rows over <start>-<end>. Candles <source>. Fills at the $t{+}1$ open under the measured session, news and rollover spread surface, cost vintage <vintage_id>, stressed at 1.5× and 2.0×. The leak-injection suite (oracle, label shuffle, timestamp scramble, cost doubling) is green on harness commit <sha>, CI report <hash>."
2. **Policy class.** "Size multipliers on a declared base direction d_t (<engine replay spec>), over a pre-registered finite grid of <84> cells in three families — inverse-width linear, monotone bucket ladders, and threshold gates composed with the volatility-target base — enumerated in `policy_grid.json`, SHA-256 <hash>. Each cell is a ledger entry. Scalar duplicates were collapsed before evaluation and were never run."
3. **Benchmarks.** "Constant fractional risk and volatility targeting at $\hat{\sigma}_t = W_t/2.5631$, $w_{\max} = 3$, both at matched average exposure: each policy is scaled by $\kappa_\pi = \bar{E}_{B1}/\bar{E}_\pi$ computed on train bars only and frozen for the test bars."
4. **Splits.** "CPCV with groups cut on 5 pm New York daily closes spanning all 21 instruments and all timeframes; $k = 2$; $N_g = 12/10/8$ for M30/H1/H4 giving 11/9/7 paths; purge one 50-bar horizon each side; embargo <delta per TF>."
5. **Metric.** "Out-of-sample net-of-cost P&L per path, summarised as the per-bar Sharpe ratio of the paired difference against each benchmark."
6. **Thresholds.** "Accept only if, against *both* benchmarks, DSR > 0.60 at $\lambda = 1.0$ and > 0.50 at $\lambda = 1.5$; the median path Sharpe exceeds both benchmarks at both λ; and the q10 CPCV path net P&L is positive by at least one standard error at $\lambda = 1.0$ and positive at $\lambda = 1.5$."
7. **Marginal rule.** "A marginal result is a kill. Marginal means: DSR in $(0.50, 0.60]$ against either benchmark, or a q10 path within one standard error of zero, or any condition that holds at $\lambda = 1.0$ and fails at $\lambda = 1.5$."
8. **Ledger rule.** "Every configuration ever evaluated counts, including abandoned ones, superseded re-runs after bug fixes that changed returns, and every timeframe the grid was swept over; N is `SELECT COUNT(*) FROM trial_ledger WHERE stage='0'` and nothing else. Declared N at sign-off: <252>. Owner: <name>. The Finder GA may not touch envelope outputs until SPA / Romano–Wolf gating is live."
9. **Void conditions.** "The run is void, and must be repeated with a new memo, if the live generation configuration changes, if the cost vintage changes, or if the leak-injection suite goes red at any point between sign-off and verdict. A void run's trials remain in the ledger."
10. **Kill wording.** "If no policy in the declared class clears the deflated bar against both benchmarks, we conclude the envelope is not demonstrably tradeable as sized here; no forecasting model will be built on it, and the programme redirects to execution and portfolio construction. This paragraph may not be renegotiated after results are seen."
11. **Consequences, stated now.** "On a kill, Stages 1 to 3 are not begun; the next work items are the cost surface and execution, in that order. Stage 0 may be re-run only after a change that alters the band itself — a new cost vintage, or a Stage-1 conditional-width model — and that re-run requires a fresh memo and carries the old ledger forward into its N."
12. **Signatures.** Research owner <name>, <date>. Data owner <name>, <date>. Independent

reviewer (did not build the harness) `<name>`, `<date>`. Memo SHA-256 `<hash>`, committed at `<commit>` before the first result row exists.

Artifact. `memo_stage0.md`, hashed and committed; its hash referenced by every row of `stage0_deflation` and by the verdict row.

Pass / fail. The memo's commit timestamp precedes the earliest `created_ts` in `stage0_results`; all three signatures present; the declared N in field 8 equals the ledger count at sign-off; the grid hash in field 2 equals the SHA-256 of the `policy_grid.json` actually loaded by the run.

Step 5.9 — If it passes: promotion, shadow, and the new bar for Stage 1

Do. A pass promotes exactly one policy — the nominated champion at its nominated timeframe — and promotes it as a *size multiplier behind the existing engine*, not as a new engine.

(1) **Integrate by table drop.** The multiplier is computed by a Python sidecar on the same 20-minute cadence as `book_ml_service.py` and written to a `stage0_multiplier` table in `timesfm.sqlite`: (instrument, granularity, bar_ts, multiplier, policy_id, gen_id, computed_ts). The C# engine polls the latest row and multiplies its own size by it, clipped to $[0, m_{\max}]$. Table drops are the only integration contract; no C# change beyond the poll and the clip.

(2) **Fail safe.** If the latest row is older than two bar intervals, or `policy_id` does not match the promoted policy, or `gen_id` does not match the live generation config, the engine uses multiplier 1.0 and raises a monitor alert. Staleness must degrade to the benchmark, never to zero and never to the last value.

(3) **Shadow first.** Run 60 trading days in shadow: the sidecar writes multipliers, the engine logs what it *would* have sized but trades its existing size. Compare realised shadow P&L against the CPCV distribution, and compare realised round-trip cost against the cost-surface prediction per cell. Promote to live only if shadow net P&L falls above the 25th percentile of the CPCV path distribution and realised cost is within 20 per cent of predicted.

(4) **Re-state the bar.** Record, in the promotion note, the champion's median path Sharpe, q10 path P&L and DSR at each λ. Stage 1 — HAR/GARCH conditional width with adaptive conformal calibration [Course Book §12.6] — must beat *this* policy, under this harness, this cost vintage and this deflation, not the raw envelope. The new benchmark set for Stage 1 is $\{B1, B2, \pi^{\dagger}\}$, and the Stage-1 ledger starts at $N = 0$ with its own memo.

Artifact. `stage0_multiplier` table and its writer; `promotion_note.md` carrying the new benchmark definition; shadow-period log and its reconciliation report.

Pass / fail. (a) The sidecar reproduces, bar for bar, the multipliers the backtest used on an overlapping historical window — maximum absolute deviation below 10^{-9}. (b) A forced-staleness test drives the engine to multiplier 1.0 and fires the alert. (c) The shadow reconciliation runs before, not after, the live switch.

Trap — One SQLite file, several writers

`timesfm.sqlite` already carries two producers: the TimesFM writer the C# engine polls, and `book_ml_service.py`'s 20-minute loop writing `book_ml_predictions`. Stage-0 regeneration must not write into it — use `forecasts_s0.sqlite`, a separate file, so that a multi-hour GPU

job cannot hold a write lock the live engine is waiting behind. On promotion the multiplier table *does* go into `timesfm.sqlite`, because table drops are the integration contract, but as a new table with its own writer, its own cadence and its own staleness rule; it never shares a table, a transaction or a lock with `book_ml_predictions`. Enable WAL mode and set a busy timeout before the first write, not after the first corruption.

Key idea — What Stage 0 does and does not decide

Stage 0 decides one thing: whether the *existing* envelope, at its *live* generation settings, under the *measured* cost surface, sized by a *declared* finite class of multipliers, beats free sizing by more than selection across 252 trials can explain. It does not decide whether the envelope is well calibrated — coverage was measured before this chapter and stands. It does not decide whether a better forecaster exists; that is what Stages 1 to 3 would have been for, and a kill says only that they are not yet licensed. And it does not decide whether the platform is worth running: a kill redirects effort to cost and execution, where the measured evidence already points [Course Book §24.2]. The one thing a kill forecloses is building a bigger model on an unproven signal, which is the failure mode this entire programme is arranged to prevent.

Stage gate — Stage 0

Metric. Per-bar Sharpe ratio of the paired difference series $\Delta_t^{\pi,B} = \Pi_t^{\pi} - \Pi_t^{B}$, computed on out-of-sample CPCV paths at the pinned cost vintage, deflated at $N =$ the Stage-0 ledger count (declared: 252).

Threshold. DSR > 0.5 against *both* benchmarks — constant fractional risk and volatility targeting — **and** q10 CPCV path net P&L > 0 **and** both still holding at $1.5\times$ the measured cost.

Decision rule. Clear pass (all conditions, at $\lambda = 1.0$ and $\lambda = 1.5$, with DSR > 0.60 at $\lambda = 1.0$) $\rightarrow$ promote per Step 5.9 and open Stage 1. Anything else — including every marginal result — $\rightarrow$ kill. The verdict is recorded before the breakdowns are opened.

Kill criterion — Stage 0 — pre-committed wording

"If no policy in the declared class clears the deflated bar against both benchmarks, we conclude the envelope is not demonstrably tradeable as sized here; no forecasting model will be built on it, and the programme redirects to execution and portfolio construction. This paragraph may not be renegotiated after results are seen."

A marginal result is a kill. Marginal means DSR $\in (0.50, 0.60]$ against either benchmark, or a q10 path within one standard error of zero, or any condition that holds at $\lambda = 1.0$ and fails at $\lambda = 1.5$.

What happens next. Stages 1 to 3 are not begun. The work redirects, in order, to: the cost surface (measured spreads, slippage, and whether the no-trade band can be narrowed [Course Book §14.2]); execution realism (fill modelling, order placement [Course Book §18.3]); and portfolio construction across the existing engine's signals. Stage 0 may be re-opened only by a change that alters the band itself — a new cost vintage, or a Stage-1 conditional-width model — and only under a fresh memo whose N carries the existing ledger forward. The envelope's coverage result stands; it was never a P&L claim [Course Book §24.2].

Artifacts — Stage output register

- `forecasts_s0.sqlite` — 1,277,640 regenerated forecast rows across 63 (instrument, timeframe) series under a single `gen_id`, plus the `gen_config` row stamping checkpoint, context length, patch sizes, `width_mult` map, code digest and device (about 126.7 MiB).
- `features_s0` + `feature_config` — the nine backward-only feature columns and the per-timeframe window declarations.
- `policy_grid.json` — 84 declared cells with canonical `config_hash` values, plus `benchmarks_s0` (2 rows) and the pruning log recording the 12 scalar duplicates that were never evaluated.
- `trial_ledger` rows for Stage 0 — 252 declared, plus every abandoned and superseded row; exported and hashed as the deflation input.
- `exposure_norm` — 11,954 train-split κ_π values with train and test mean exposures.
- `stage0_results`, `stage0_pnl_series`, `stage0_by_instrument` — 6,966 path evaluations across three cost multipliers, with per-bar series, turnover, cost paid and per-instrument decomposition.
- `stage0_deflation` — V, SR_0, DSR, PBO and the q10 path statistic per (benchmark, timeframe, λ).
- `memo_stage0.md` — the signed pre-commitment memo, hashed and committed before the first result row.
- `stage0_verdict` — the condition-by-condition decision record with its timestamp, hashes and signatories.
- On a pass only: `stage0_multiplier` table, sidecar writer, `promotion_note.md`, and the 60-day shadow log with its cost reconciliation.

6

Stage 1: The Adaptive Width Engine

Stage 1 removes one number from the platform and replaces it with a live estimate. That number is `width_mult`: a per-pair scalar in $[0.95, 1.07]$, fitted once against roughly three years of aggregate coverage and frozen since. It is a conformal threshold in every formal respect [Course Book §11.3], and the freezing is the only defect — a threshold estimated on a calibration sample carries a guarantee for exactly as long as that sample stays exchangeable with today's bar, which for a three-year FX window is no time at all [Course Book §11.4].

The replacement has three components and each does one job [Course Book §12.6]. A conditional variance model supplies the *level* and *dynamics* of width. A quantile map imposes a *shape* on the standardised innovation. A conformal wrapper with an adaptive level supplies *coverage*, correcting whatever the first two got wrong. None of the three substitutes for another, and the frozen multiplier is a degenerate case of the third with the first two absent.

The engineering is cheap: pure Python sidecar, ordinary least squares and a one-dimensional likelihood, no GPU beyond what TimesFM already occupies, a score pool measured in megabytes. The evaluation is not. Stage 1 must clear two bars: conditional coverage must recover inside stress buckets, and the sizing policy driven by the new band must add deflated, cost-net P&L over the best baseline available. A better-calibrated band that does not change the trade, or changes it in a way the spread eats, is a research result and not a deployment.

6.1 Preconditions and the two entry paths

Do not begin until every row of Table 6.1 is signed. The Stage -1 gates licence the harness; without them there is no instrument capable of reading Stage 1's result.

Table 6.1. Stage-1 preconditions. Each is a signed artifact from an earlier chapter, not a judgement.

Precondition	Evidence required	Source
Cost surface built	`cost_surface` populated from a bid/ask re-pull, vintage-pinned	Chapter 2 gate
Feeds point-in-time	COT release-lag join, ALFRED vintages for FRED, RORO/VolRegime as-of series	Chapter 3 gate
Harness green	four leak-injection controls passing on the assembled engine	Chapter 4 gate
Trial ledger live	every evaluated configuration writes a row before results are read	Chapter 4, [Course Book §15.6]
Forecast history regenerated	stored forecasts at *live* context settings, Stage-0 universe	Chapter 5, Step 5.1
Stage-0 verdict recorded	signed memo, pass or kill, with its ledger N	Chapter 5, Step 5.8

The last row admits two values and they lead to different chapters of the same stage. Record which path you are on *before* any fitting, in the ledger, as a field on every Stage-1 trial row.

Table 6.2. The two entry paths. The calibration half of the Stage-1 gate is identical on both; the P&L half differs only in its baseline.

Stage-0 verdict	Stage-1 status	P&L baseline for the gate
Pass	Calibration upgrade *and* P&L candidate	The Stage-0 winning sizing policy, rerun on the new band at the same cost vintage
Kill	Calibration upgrade first, P&L candidate second	The better of the two Stage-0 benchmarks — constant fractional risk and volatility targeting — at matched average exposure

If Stage 0 killed the sizing thesis, Stage 1 still proceeds, and it proceeds honestly only if the following three sentences are written into the ledger and the Stage-1 memo verbatim.

(i) Stage 1 is being run as a *calibration* upgrade. A conditional band is risk information worth having whether or not it is tradeable, and the platform currently reports an 80% envelope whose conditional behaviour has never been measured.

(ii) The Stage-0 kill is not renegotiated by anything in this chapter. Stage 1 does not reopen the policy class, the cost vintage, the splits or the thresholds that produced it.

(iii) The P&L half of the Stage-1 gate is now a test against benchmarks, not against a champion. Passing it revives the P&L thesis; failing it leaves the thesis dead and ships the calibration anyway.

Trap — The kill path is not a second attempt at Stage 0

A killed Stage 0 creates pressure to re-run the same policy grid on a slightly different band and call the result new evidence. It is not new evidence: it is the same class, the same instruments and substantially the same information, evaluated a second time, and the only correct response is that the trial ledger's N grows and the deflation bar rises accordingly [Course Book §15.4]. The Stage-1 P&L test earns its status because the *band* changed conditionally, not because the search resumed. Any Stage-1 configuration that also perturbs the sizing policy belongs to Stage 0's class and must be scored against Stage 0's already-spent multiplicity.

Two conventions hold for the whole chapter and appear in every table without further comment. The trading calendar is OANDA's 24×5: one trading day is 24 hours, one week is 5 days, one month is 22 days, three months is 65 days, one year is 260 days. The forecast horizon is $h = 50$ bars on every timeframe, so a score stamped at bar t becomes observable only at $t + 50$, and B denotes bars per trading day — 1440, 288, 96, 48, 24, 6, 1 for M1 through D.

6.2 The volatility engine

Three steps: measure volatility, fit two models to it, pick one per cell out of sample. A cell is an (instrument, granularity) pair; there are $21 \times 7 = 147$ of them.

Step 6.1 — Volatility proxy per timeframe

Do. Build one realised-variance series per cell in the sidecar, written to `vol_proxy`. Subsample each bar of timeframe T using the finest *stored* bar that yields at least $m = 12$ sub-returns at M5; drop to M1 only when M5 cannot; fall back to the squared bar return when no finer bar exists. Table 6.3 fixes the choice per timeframe and is not a per-instrument decision. Compute sub-returns from stored mid closes, log differences, no overnight gap

special-casing (FX trades through). Discard any bar whose sub-bar count falls below $0.8m$ (weekend edges, feed outages) and record the discard in `vol_proxy.status` rather than interpolating.

Before trusting any M1-sourced series, measure the microstructure floor: compute the mean realised variance of a fixed sample as a function of sampling interval from M1 to M30 and locate the interval below which it rises rather than flattens. Candles are mid-only, so the bid/ask bounce that normally dominates this floor is partly suppressed and partly replaced by stale-quote bias — the direction is not predictable and must be measured, not assumed [Course Book §12.1]. If the floor sits above M1, promote M15/M30 to the next admissible sampling interval and re-run this step.

The mathematics. Partition bar t into m sub-intervals with log returns $r_{t,i}$; the realised variance and the fit-quality loss are

$$\mathrm{RV}_t = \sum_{i=1}^{m} r_{t,i}^2, \qquad \mathrm{QLIKE}(\hat{\sigma}^2; \mathrm{RV}) = \frac{\mathrm{RV}}{\hat{\sigma}^2} - \ln \frac{\mathrm{RV}}{\hat{\sigma}^2} - 1 \;\geq\; 0,$$

where $\hat{\sigma}^2$ is any $\mathcal{F}_{t-1}$-measurable variance forecast and equality holds only at $\hat{\sigma}^2 = \mathrm{RV}$. QLIKE is the loss for every model comparison in this chapter: it is scale-free, and it diverges as $\hat{\sigma}^2 \to 0$, so under-forecast width is punished without limit. [Course Book §12.1]

Artifact. `vol_proxy(instrument, granularity, bar_ts, rv, m_used, source_tf, status)`, one row per bar per cell, plus `proxy_floor_report.json` recording the measured sampling floor per instrument.

Pass / fail. Three checks. (a) The relative standard error column of Table 6.3 is reproduced empirically: regress $\ln \mathrm{RV}_t$ on its own lag and confirm the residual scale falls as m rises across timeframes. (b) No cell has more than 2% of bars in a non-`ok` status. (c) The floor report shows a flat or falling curve at the chosen sampling interval for every instrument; any instrument that fails is demoted to M5 sourcing and re-checked.

Table 6.3. Volatility proxy per timeframe. m is the number of sub-returns per bar; the relative standard error of RV under independent Gaussian sub-returns is $\sqrt{2/m}$, and the final column expresses it as a fraction of the squared return's $\sqrt{2} = 1.4142$.

TF	m at M5	m at M1	Source	m used	$\sqrt{2/m}$	Status
M1	0	1	r_t^2	1	1.4142	no finer bar stored
M5	1	5	M1	5	0.6325	floor-test required
M15	3	15	M1	15	0.3651	floor-test required
M30	6	30	M1	30	0.2582	floor-test required
H1	12	60	M5	12	0.4082	clean
H4	48	240	M5	48	0.2041	clean
D	288	1440	M5	288	0.0833	clean

The $m \geq 12$ rule is a recommendation with one reason: $\sqrt{2/12} = 0.4082$ is under a third of the squared return's noise, which is where the proxy stops dominating a QLIKE comparison, while sampling finer than M5 buys microstructure contamination the platform cannot yet quantify. M1 has no proxy at all — its width model is fitted against a statistic whose standard deviation exceeds its mean, so every M1 result in this chapter is provisional and carries no gate weight.

Step 6.2 — HAR fit protocol

Do. Fit one *direct* log-HAR per cell, targeting the average realised variance over the next 50 bars — the quantity the envelope actually needs — rather than the next single bar. The horizon is fixed and known, so iteration buys nothing and compounds 49 model errors [Course Book §12.5]. Scale the three component windows to the timeframe by Table 6.4: daily $= B$ bars, weekly $= 5B$, monthly $= 22B$.

Estimation is ordinary least squares on logs, with three non-negotiable details. Regress on $\ln \mathrm{RV}^{(d)}, \ln \mathrm{RV}^{(w)}, \ln \mathrm{RV}^{(m)}$, not on levels, because levels admit negative fitted variance and badly heteroscedastic residuals. Use Newey–West standard errors with Bartlett bandwidth $L = h - 1 = 49$; the direct target overlaps 49 bars, so ordinary-least-squares standard errors understate by roughly $\sqrt{50}$ and every coefficient looks significant. And apply the back-transform correction below before the variance leaves the fitting code — never after, where it will be applied twice by somebody else.

Refit weekly, every Saturday, on a rolling window ending at the last completed bar. Reason: the $22B$-bar monthly regressor already dominates the fit's memory, so a refit more frequent than weekly re-estimates the same coefficients on almost the same data, while a refit less frequent than weekly leaves coefficient staleness longer than the monthly window it feeds. Cost is 147 regressions of four parameters — seconds.

The mathematics. With $\mathrm{RV}_t^{(d)} = \mathrm{RV}_t$, $\mathrm{RV}_t^{(w)} = (5B)^{-1}\sum_{i=0}^{5B-1}\mathrm{RV}_{t-i}$ and $\mathrm{RV}_t^{(m)} = (22B)^{-1}\sum_{i=0}^{22B-1}\mathrm{RV}_{t-i}$, the direct log-HAR is

$$\ln \overline{\mathrm{RV}}_{t+1:t+h} = \beta_0 + \beta_d \ln \mathrm{RV}_t^{(d)} + \beta_w \ln \mathrm{RV}_t^{(w)} + \beta_m \ln \mathrm{RV}_t^{(m)} + \varepsilon_{t+h}, \qquad \widehat{\overline{\mathrm{RV}}} = \exp\!\left(\hat{\mu} + \tfrac{1}{2}s^2\right),$$

where $\overline{\mathrm{RV}}_{t+1:t+h}$ is the mean realised variance over the h bars ahead, $\hat{\mu}$ the fitted log-mean, and s^2 the residual variance in logs; $\exp(\hat{\mu})$ alone is the conditional *median* and understates the mean by $e^{s^2/2}$. [Course Book §12.5]

Artifact. `vol_models` (Table 6.5), one vintaged row per cell per refit, family `HAR`.

Pass / fail. (a) $\hat{\beta}_d + \hat{\beta}_w + \hat{\beta}_m < 1$ in every cell — this sum is HAR's persistence and a value at or above one is a non-stationary fit, reject and fall back to the previous vintage. (b) The back-transform factor $e^{s^2/2}$ lies in $[1.00, 1.40]$; above 1.40 means $s > 0.83$ in logs, which is a fit explaining almost nothing and must be escalated, not shipped. (c) In-sample QLIKE against a constant-variance benchmark improves in at least six of the seven timeframes for at least 18 of 21 instruments. (d) Refit determinism: re-running the Saturday job on the same window reproduces every coefficient bit-for-bit.

Table 6.4. HAR component windows scaled to each timeframe, and the warm-up before the first usable training row ($22B$ bars for the monthly regressor plus $h = 50$ for the target). The last two columns give the number of training rows available in three years and the effective count after dividing by the h-bar target overlap.

TF	B	Daily	Weekly	Monthly	Warm-up	Rows / 3 yr	$N_{\mathrm{eff}} = N/h$
M1	1440	1440	7200	31680	31730	1 091 470	21 829
M5	288	288	1440	6336	6386	218 254	4 365
M15	96	96	480	2112	2162	72 718	1 454
M30	48	48	240	1056	1106	36 334	727
H1	24	24	120	528	578	18 142	363
H4	6	6	30	132	182	4 498	90
D	1	1	5	22	72	708	14

Read the last column before reading anything else. HAR has four parameters. At D the direct fit has 14 effective observations, three and a half per parameter, on three years of history — the weakest cell in the platform by an order of magnitude and the one whose coefficient standard errors must be reported beside every use. At H4 it is 90, about 22 per parameter, which is thin but workable. Everything at H1 and faster is comfortable. Two consequences are operational, not philosophical: the D and H4 cells are the first candidates for pooled estimation across instruments if their standalone fits are unstable across vintages, and no gate decision in this chapter rests on a D-only result.

Table 6.5. `vol_models` store. One row per (cell, family, vintage). The vintage is what makes a backtest reproducible: every band ever emitted names the exact coefficient row that produced it.

Column	Type	Contents
`vintage_id`	TEXT	ULID, primary key with (instrument, granularity, family)
`instrument`	TEXT	one of the 21
`granularity`	TEXT	M1 . . . D
`family`	TEXT	`HAR` or `GARCH`
`fit_ts`	INT	UTC epoch of the refit job
`train_start_ts`	INT	first bar in the estimation window
`train_end_ts`	INT	last bar; must precede `fit_ts` by at least h bars
`n_rows`	INT	usable training rows after warm-up
`params`	TEXT	canonical JSON: HAR $\{\beta_0, \beta_d, \beta_w, \beta_m\}$, GARCH $\{\omega, \alpha_1, \beta_1\}$
`s2_log`	REAL	residual variance in logs (HAR); null for GARCH
`bt_factor`	REAL	$e^{s^2/2}$, the back-transform correction (HAR)
`persistence`	REAL	$\beta_d + \beta_w + \beta_m$ (HAR) or $\alpha_1 + \beta_1$ (GARCH)
`val_qlike`	REAL	out-of-sample QLIKE on the validation block
`proxy_source`	TEXT	sourcing timeframe from Table 6.3
`status`	TEXT	`active`, `rejected`, `superseded`
`config_hash`	TEXT	hash of the full fit configuration

Step 6.3 — GARCH alternative and per-cell model selection

Do. Fit GARCH(1,1) on bar log returns for the same 147 cells and write it to the same store with `family = GARCH`. Initialise σ_1^2 at the sample second moment, hand the negative log-likelihood to a numerical optimiser, and remove the inequality constraints by reparameterisation rather than imposing them — $\omega = e^a$ with (α_1, β_1) mapped from two unconstrained reals onto the simplex — so the search is unbounded and admissibility holds by construction. Use variance targeting (ω pinned by the sample variance, two free shape parameters) on the D and H4 cells only; reason: those are the cells short of data, and fixing the level halves the parameter count where the effective sample is thinnest.

Reject any fit with $\alpha_1 + \beta_1 \geq 0.999$ and fall back to the previous admissible vintage, logging the rejection. Reason: at $\phi = \alpha_1 + \beta_1 = 0.999$ a variance shock has a half-life of 692.8 bars, nearly fourteen forecast horizons, so the model is near-integrated, the long-run level $\omega/(1-\phi)$ is estimated with an exploding standard error, and the horizon variance of Step 6.4 becomes a lever arm on one bar's surprise. The admissible band and its half-lives are $\phi = 0.95 \to 13.5$ bars, $0.98 \to 34.3$, $0.99 \to 69.0$, $0.995 \to 138.3$.

Select HAR against GARCH per cell by out-of-sample QLIKE. Protocol: within each CPCV training fold, split the fold chronologically into an estimation block and a validation block, separated by a purge of $h = 50$ bars plus the fold's embargo; fit both families on the estimation block; score both by mean QLIKE on the validation block against the same proxy series; take the lower. Repeat in every fold. The shipped live choice per cell is the *modal* winner across folds, recorded with the fold-level vote counts.

The mathematics. The recursion and its Gaussian conditional log-likelihood are

$$\sigma_t^2 = \omega + \alpha_1 r_{t-1}^2 + \beta_1 \sigma_{t-1}^2, \qquad \ell(\omega, \alpha_1, \beta_1) = -\tfrac{1}{2} \sum_{t=1}^{T} \Big[\ln(2\pi) + \ln \sigma_t^2 + r_t^2 / \sigma_t^2 \Big],$$

with $\omega > 0$, $\alpha_1, \beta_1 \geq 0$, $\alpha_1 + \beta_1 < 1$, every σ_t^2 produced by running the recursion forward from σ_1^2, and r_t the bar log return. [Course Book §12.3]

Artifact. `vol_models` rows with `family = GARCH`; `model_select` (instrument, granularity, fold_id, qlike_har, qlike_garch, winner) and its per-cell modal summary.

Pass / fail. (a) Every shipped GARCH row satisfies $\omega > 0$, $\alpha_1, \beta_1 \geq 0$, $\alpha_1 + \beta_1 < 0.999$. (b) The optimiser reaches the same optimum from three distinct starting points in at least 95% of cells; cells failing this are marked `unstable` and forced to HAR. (c) The modal winner commands at least 60% of fold votes in each cell, or the cell is recorded as *undecided* and defaults to HAR, which has fewer parameters. (d) Selection is computed strictly inside training folds — an audit query confirms no validation block used in selection overlaps any test block of its own fold.

Trap — Selecting the family once, globally, is a leak

Choosing HAR or GARCH per cell on the full sample and then running CPCV on the winner leaks the test blocks into the model class. It is cheap to get right — the selection is one more thing refitted inside each training fold — and expensive to get wrong, because the leak is invisible in every diagnostic and inflates exactly the out-of-sample statistic the gate reads [Course Book §16.2]. The same rule binds the innovation parameter ν of Step 6.4 and the buffer scale of Step 6.5.

6.3 From variance to a band

The vol engine emits a variance for the 50-bar move. Three steps turn it into the four quantiles the platform's table already carries, with a coverage guarantee attached and the frozen multiplier removed.

Step 6.4 — Width manufacture

Do. For each cell and each bar, produce $\hat{\sigma}_h^2$, the conditional variance of the cumulative 50-bar move, then map it to raw quantiles at $\tau \in \{0.10, 0.20, 0.80, 0.90\}$.

The two families reach $\hat{\sigma}_h^2$ differently and the difference must be coded once, not inferred. The direct log-HAR of Step 6.2 already predicts the *mean* realised variance per bar over the window, so $\hat{\sigma}_h^2 = h \cdot \widehat{\mathrm{RV}}$ with the back-transform correction already applied. GARCH predicts one bar at a time and must be summed by the closed form below; do *not* substitute $\sqrt{h}\,\hat{\sigma}_{t+1}$ for the square root of that sum. At $\phi = 0.95$ the $\sqrt{h}$ shortcut overstates half-width by 20.9% when the one-step variance is twice its long-run level and understates it by 21.7% when it is half — precisely the wrong sign in both directions, wide after the storm and narrow before it.

Take the location $\hat{\mu}$ from TimesFM's median head as stored. Stage 1 does not re-estimate the point forecast; that is Stage 3's job, and mixing the two makes the gate unreadable.

Select the innovation shape ν per cell by validation pinball, on the same fold-internal split used for family selection in Step 6.3. Grid: $\nu \in \{4, 5, 6, 8, 10, 20, \infty\}$ with ∞ denoting the Gaussian. Score the four raw quantiles by mean pinball summed over $\tau \in \{0.10, 0.20, 0.80, 0.90\}$ on the validation block; take the minimiser. Expect the choice to matter little and record it

anyway: at the 80% band the t_5 half-width is $1.1432\hat{\sigma}$ against the Gaussian $1.2816\hat{\sigma}$, a ratio of 0.8921, so the entire admissible range of ν spans about 15% of width — the same order as the legacy multiplier's $[0.95, 1.07]$ range and well inside what the conformal layer will absorb [Course Book §10.5].

Enforce monotonicity before writing: sort $\tilde{q}_{0.10} \le \tilde{q}_{0.20} \le \hat{\mu} \le \tilde{q}_{0.80} \le \tilde{q}_{0.90}$. Sorting a crossed pair strictly reduces pinball loss, so the guard is free [Course Book §10.2].

The mathematics. Under the martingale-difference structure the cross terms vanish and

$$\hat{\sigma}_h^2 \;=\; \mathrm{Var}\Big(\sum_{k=1}^{h} r_{t+k} \;\Big|\; \mathcal{F}_t\Big) = \sum_{k=1}^{h} \sigma^2_{t+k|t} = h\,\bar{\sigma}^2 + \big(\sigma^2_{t+1|t} - \bar{\sigma}^2\big)\frac{1-\phi^h}{1-\phi}, \qquad \phi = \alpha_1 + \beta_1,$$

with $\bar{\sigma}^2 = \omega/(1-\phi)$ the long-run variance and $\sigma^2_{t+1|t}$ the one-step forecast known at t. [Course Book §12.3] The quantile map and the selection loss are

$$\tilde{q}_\tau = \hat{\mu} + \hat{\sigma}_h F_z^{-1}(\tau), \quad F_z^{-1}(\tau) = \Phi^{-1}(\tau) \;\text{ or }\; \sqrt{\tfrac{\nu-2}{\nu}}\, t_{\nu,\tau}, \qquad \rho_\tau(u) = \big(\tau - \mathbf{1}\{u<0\}\big)u,$$

where F_z is the distribution of the standardised innovation (Var $= 1$ in both variants), $t_{\nu,\tau}$ the raw Student-t quantile, and ρ_τ the pinball loss of the residual $u = y - \tilde{q}_\tau$. [Course Book §10.5] [Course Book §10.1]

Artifact. `raw_bands(instrument, granularity, forecast_ts, mu, sigma_h, nu, family, vintage_id, q10_raw, q20_raw, q80_raw, q90_raw)`, one row per forecast, joinable to `vol_models` by `vintage_id`.

Pass / fail. (a) $\hat{\sigma}_h^2 > 0$ and finite in every row; any non-finite row aborts the cycle rather than emitting. (b) Reconstructing $\hat{\sigma}_h$ from the emitted quantiles by $(\tilde{q}_{0.90} - \tilde{q}_{0.10})/(2F_z^{-1}(0.90))$ returns the stored `sigma_h` to 10^{-9} relative — the round-trip that catches a mismatched ν between writer and reader. (c) Monotonicity holds in 100% of rows after the sort. (d) The ν chosen per cell is stable across at least 60% of folds, or the cell is forced to Gaussian.

Table 6.6. Left: the geometric multiplier $(1-\phi^{50})/(1-\phi)$ that carries the current variance deviation across the horizon, and its size relative to $h = 50$. Right: unit-variance quantile constants $F_z^{-1}(\tau)$ for the ν grid.

ϕ	$(1-\phi^{50})/(1-\phi)$	$\div h$	ν	$F_z^{-1}(0.80)$	$F_z^{-1}(0.90)$
0.900	9.9485	0.1990	4	0.6654	1.0841
0.950	18.4611	0.3692	5	0.7123	1.1432
0.980	31.7915	0.6358	6	0.7395	1.1756
0.990	39.4994	0.7900	8	0.7698	1.2097
0.995	44.3375	0.8867	10	0.7863	1.2273
0.999	48.7944	0.9759	20	0.8158	1.2573
1.000	50.0000	1.0000	∞	0.8416	1.2816

The left half of Table 6.6 is the operational content of the persistence bound in Step 6.3. As $\phi \to 1$ the multiplier approaches h itself: the whole 50-bar band becomes a linear function of one bar's squared surprise, with no mean reversion inside the horizon to damp it. That is why a fit at $\phi = 0.999$ is rejected rather than merely flagged.

Step 6.5 — CQR score buffer

Do. Maintain, per cell, *two* rolling score buffers: one for the 80% pair $(\tilde{q}_{0.10}, \tilde{q}_{0.90})$ at $\alpha = 0.20$ and one for the 60% pair $(\tilde{q}_{0.20}, \tilde{q}_{0.80})$ at $\alpha = 0.40$. Two buffers, because one threshold

calibrated at 80% says nothing about the 60% band; $147 \times 2 = 294$ buffers in total, 3.47 MiB resident at eight bytes per score.

Use the *multiplicative* score as the shipped default and carry the additive CQR score as the declared alternative in the grid. Reason: the multiplicative form reproduces the arithmetic the C# engine already performs — median plus a multiple of each half-band — so the replacement is a change of one number's provenance rather than a change of the engine's formula, and the correction scales with the model's own uncertainty instead of adding a fixed pip count to a calm hour and a violent one alike [Course Book §11.3].

Push a score onto the buffer only when its realisation exists. At a 50-bar horizon the score for bar t becomes available at $t + 50$ and not before; back-filling the buffer with scores stamped earlier than that is a look-ahead leak of the plainest kind [Course Book §18.2]. Implement the buffer as a ring of fixed capacity n_T from Table 6.7, indexed by *realisation* time, with a pending queue holding the 50 open bands.

Size the buffer by $n_T = \min\big(\max(500,\ B \cdot 65),\ 5000\big)$. The floor of 500 is a level-resolution requirement: the achievable coverage levels are $k/(n+1)$, so a buffer of 500 resolves 0.1996 percentage points and anything shallower makes the adaptive level of Step 6.6 move without moving the band. The cap of 5000 is a staleness limit: beyond it the resolution gain is under 0.02 percentage points and the pool reaches into a different volatility regime, which is the exchangeability failure this whole stage exists to fix.

The mathematics. With m the median head and $\tilde{q}$ the raw quantiles, the two score forms and the lookup rule are

$$M_i = \max\left\{\frac{m_i - y_i}{m_i - \tilde{q}_{\mathrm{lo},i}},\ \frac{y_i - m_i}{\tilde{q}_{\mathrm{hi},i} - m_i}\right\}, \quad E_i = \max\left\{\tilde{q}_{\mathrm{lo},i} - y_i,\ y_i - \tilde{q}_{\mathrm{hi},i}\right\}, \quad k = \left\lceil (n+1)(1-\alpha) \right\rceil,$$

$\hat{c} = M_{(k)}$ or $\hat{Q} = E_{(k)}$ being the k-th smallest of the n buffered scores, with y_i the realised 50-bar move; the $+1$ is the slot the future observation will occupy and dropping it is the most common bug in the subject. If $k > n$ the threshold is $+\infty$ and the band is unbounded — the honest answer when the buffer cannot certify the level requested. [Course Book §11.3] [Course Book §11.2]

Artifact. `score_buffers(instrument, granularity, alpha, realisation_ts, score_m, score_e, band_ts)` persisted every cycle so the sidecar restarts warm; plus `pending_bands` holding the open 50-bar windows.

Pass / fail. (a) For every buffered score, `realisation_ts` $\geq$ `band_ts` $+ h$ bars — one SQL predicate, run in CI, and a violation is a hard stop. (b) Buffer occupancy reaches n_T within the expected warm-up and never exceeds it. (c) Replaying a saved buffer through the lookup reproduces the emitted threshold exactly. (d) At $\alpha = 0.20$ the realised index is $k = 4001$ for $n = 5000$, 2497 for $n = 3120$, 1249 for $n = 1560$ and 401 for $n = 500$, certifying 80.00%, 80.01%, 80.01% and 80.04% respectively.

Trap — The two ends of the buffer table are not equivalent

At M1 and M5 the cap binds and the pool spans days, not months — acceptable, because a few days at M1 is thousands of observations across many sessions. At D the floor binds and the pool spans 550 trading days, better than two years: those scores are not exchangeable with today's under any reading, so at D the finite-sample conformal guarantee is decorative and the only live guarantee is the long-run one from Step 6.6 [Course Book §11.4]. Report the D band as long-run valid and conditionally unverified until its Christoffersen bucket has 250 observations, which takes a quarter of pooled shadow output.

Table 6.7. Score-buffer size per timeframe. "Binds" names which of the three terms of $\min(\max(500, 65B), 5000)$ is active. Span is $n_T + h$ bars, the calendar reach of the pool once the horizon lag is included.

TF	Bars / 3 mo	n_T	Binds	$100/(n_T{+}1)$ pp	Span (bars)	Span (days)
M1	93 600	5000	cap	0.0200	5050	3.51
M5	18 720	5000	cap	0.0200	5050	17.53
M15	6 240	5000	cap	0.0200	5050	52.60
M30	3 120	3120	3 mo	0.0320	3170	66.04
H1	1 560	1560	3 mo	0.0641	1610	67.08
H4	390	500	floor	0.1996	550	91.67
D	65	500	floor	0.1996	550	550.00

Step 6.6 — ACI level adaptation

Do. Run the miscoverage level as a feedback variable, one controller per buffer, so 294 controllers in total. Initialise $\alpha_1 = \alpha$ (the target: 0.20 for the 80% pair, 0.40 for the 60% pair). At each bar look the threshold up at the *clipped* level $\beta_t = \min\{1, \max\{0, \alpha_t\}\}$; after the realisation arrives 50 bars later, update the *unclipped* state.

Clip the level you look up, never the state you carry. The excursions of α_t outside $[0, 1]$ are how the controller remembers how far into deficit it has run; clipping the state discards exactly the memory that the telescoping identity converts into the guarantee, and the long-run bound is then void [Course Book §11.5]. Store α_t as a REAL with no constraint, and write the clipped β_t into the emitted row as a separate column so the two are never confused downstream.

Set γ per timeframe from Table 6.8. Faster timeframes take the smaller step because they receive more updates per unit of calendar time, so the same effective memory of $1/\gamma$ observations is bought at a lower per-update jitter — and jitter is a running cost, not a statistical one: every level move resizes positions across the spread.

Treat the PID generalisation as a declared upgrade path, not as v1. Wire it at block level only: feedforward from a scorecaster fitted to the score history, an unbounded integrator carrying the accumulated coverage error, and a small proportional gain for speed. The long-run guarantee rests on the integrator alone, so a useless scorecaster costs width and never validity [Course Book §11.6]. Ship it only after Stage 1 has passed its gate on the pure-integral law, and count it in the ledger as a separate configuration when it is evaluated.

The mathematics. With $\mathrm{err}_t = \mathbf{1}\{y_t \notin C_t(\beta_t)\}$ the breach indicator and $\gamma > 0$ the step size, the update, its guarantee, and the PID generalisation are

$$\alpha_{t+1} = \alpha_t + \gamma(\alpha - \mathrm{err}_t), \qquad \left|\tfrac{1}{T}\textstyle\sum_{t \le T} \mathrm{err}_t - \alpha\right| \le \tfrac{\max\{\alpha_1, 1-\alpha_1\}+\gamma}{\gamma T},$$

$$q_{t+1} = \underbrace{\hat{q}_{t+1}}_{\text{feedforward}} + \underbrace{r_t\Big(\textstyle\sum_{s \le t} e_s\Big)}_{\text{integral}} + \underbrace{k_P e_t}_{\text{proportional}},$$

where $C_t(\beta)$ is the band emitted at nominal miscoverage β, $e_t = \mathrm{err}_t - \alpha$ the instantaneous coverage error, $\hat{q}_{t+1}$ a scorecast, r_t an increasing unbounded saturation function with $r_t(0) = 0$, and k_P a small proportional gain. The bound is deterministic and assumes nothing about the data. [Course Book §11.5] [Course Book §11.6]

Artifact. `aci_state(instrument, granularity, alpha_target, alpha_state, beta_emitted, n_buffer, k_index, threshold, updated_ts, reset_count)`, one row per controller, rewritten every cycle and journalled on every reset.

Pass / fail. (a) The telescoping identity holds exactly on any replayed window: $\alpha_{T+1} - \alpha_1 = \gamma \sum_t (\alpha - \text{err}_t)$ to floating-point tolerance. Any mismatch means the state was clipped, an update was skipped, or a reset went unjournalled. (b) The down-step $\gamma(1-\alpha)$ exceeds the buffer's level resolution $1/(n_T+1)$ in every cell — the ratio column of Table 6.8 is never below 8. (c) No controller has $|\alpha_t| > 2$ at any point in the shadow window; an excursion that large means the band family is not saturating and the $k > n$ branch is mis-implemented.

Table 6.8. ACI step size per timeframe. "Width jump" is the band-width change caused by a single breach, using the local sensitivity of 2.2231% of width per percentage point of coverage near the 80% band [Course Book §11.4]. "Step / res." is the down-step divided by the buffer's level resolution and must exceed one for the band to move at all.

TF	γ	Mem. (bars)	Mem. (days)	Width jump	Step / res.	Reason
M1	0.0050	200	0.14	0.89%	20.0	most updates per regime; least jitter
M5	0.0050	200	0.69	0.89%	20.0	as M1; buffer cap-bound, not stale
M15	0.0075	133	1.39	1.33%	30.0	memory of one trading day
M30	0.0100	100	2.08	1.78%	25.0	memory of two trading days
H1	0.0100	100	4.17	1.78%	12.5	the course book's anchor value
H4	0.0200	50	8.33	3.56%	8.0	few updates; needs faster response
D	0.0500	20	20.00	8.89%	20.0	one update per day; top of range

The D row is the uncomfortable one and is stated rather than hidden: at $\gamma = 0.05$ a single breach widens the daily band by 8.9%, which on a book paying the spread at every resize is a real cost, and the alternative — a smaller γ — buys a controller that needs 1700 daily bars to certify a one-percentage-point bound. The D cell is therefore the first candidate for the PID upgrade, whose proportional and feedforward blocks supply speed without buying it entirely from step size. Until then, D is the cell where the sizing layer should be least aggressive.

Pseudocode — The `width_mult` replacement, per bar, per cell, per level

1: **function** EMIT(cell, bar t, level α)
2: $\hat{\sigma}_h^2 \leftarrow h \cdot \widehat{\text{RV}}$ if HAR else the geometric sum ▷ Step 6.4, active `vintage_id`
3: $\tilde{q}_\tau \leftarrow \hat{\mu} + \hat{\sigma}_h F_z^{-1}(\tau)$ for $\tau \in \{0.10, 0.20, 0.80, 0.90\}$; sort
4: $\beta \leftarrow \min\{1, \max\{0, \alpha_{\text{state}}\}\}$ ▷ clip the LEVEL, never the state
5: $k \leftarrow \lceil (|\mathcal{B}|+1)(1-\beta) \rceil$
6: $\hat{c} \leftarrow +\infty$ if $k > |\mathcal{B}|$, else $-\infty$ if $k \le 0$, else the k-th smallest of $\mathcal{B}$ ▷ $\hat{c}$ IS the old `width_mult`
7: emit $\left[\, \hat{\mu} - \hat{c}(\hat{\mu} - \tilde{q}_{\text{lo}}),\ \ \hat{\mu} + \hat{c}(\tilde{q}_{\text{hi}} - \hat{\mu}) \,\right]$ with β, $|\mathcal{B}|$, $\hat{c}$, `vintage_id`
8: push $(t, \hat{\mu}, \tilde{q}_{\text{lo}}, \tilde{q}_{\text{hi}}, \hat{c})$ onto $\mathcal{P}$ ▷ pending, resolves at $t+h$
9: **end function**
10: **function** RESOLVE(bar t)
11: pop the entry stamped $t-h$ from $\mathcal{P}$; if absent, return
12: $y \leftarrow$ realised cumulative move over bars $t-h+1 \ldots t$
13: $M \leftarrow \max\{(\hat{\mu} - y)/(\hat{\mu} - \tilde{q}_{\text{lo}}),\ (y - \hat{\mu})/(\tilde{q}_{\text{hi}} - \hat{\mu})\}$
14: err $\leftarrow \mathbf{1}\{M > \hat{c}\}$; push M onto $\mathcal{B}$; push err onto $\mathcal{E}$
15: $\alpha_{\text{state}} \leftarrow \alpha_{\text{state}} + \gamma(\alpha - \text{err})$ ▷ state stored UNCLIPPED
16: $S \leftarrow \max\{0,\ S + \text{err} - \alpha - \kappa\}$; evaluate the alarm ladder of Step 6.7
17: **end function**

Line 6 is the whole replacement. The legacy engine computes $\left[\, m - c(m - q_{10}),\ m + c(q_{90} - m) \,\right]$

with c a frozen per-pair constant; the adaptive engine computes the identical expression with $c = \hat{c}_t$, the k-th order statistic of a live score pool taken at a level the controller is steering. Same arithmetic, same units, same monotone structure — one number's provenance changed from a three-year average to a conditional estimate with a long-run guarantee attached.

6.4 Live operation

The engine now emits a band. Three steps make it observable, make it readable by the C# engine without breaking the integration contract, and run it beside the incumbent for long enough to compare them honestly.

Step 6.7 — Live monitoring and the alarm ladder

Do. Compute three instruments every bar, per cell and per level. Rolling coverage over a window W from Table 6.9, with its standard error printed beside it so no one reacts to noise. A rolling PIT histogram over the same window — coverage is one functional of the PIT, the PIT is the whole object, and it is the only one of the three that says *which* tail is wrong. And a one-sided CUSUM on the coverage error as the alarm.
Set the CUSUM reference value to detect a drift from $\alpha = 0.20$ to 0.30: half the shift, $\kappa = 0.05$. Reason: $\kappa = \delta/2$ is the point at which the likelihood ratio between the two breach rates changes sign, so the increment has mean $-\kappa$ under the null and $+\kappa$ under the alternative [Course Book §13.7].
Use two thresholds, not one. h_{notify} raises a flag and pages the owner; h_{reset} licenses the destructive actions. Reason: purging the score buffer and resetting the ACI state throw away the controller's accumulated deficit, which is exactly the memory the long-run guarantee is built from, and must not fire on a noisy fortnight. Simulated in-control average run lengths under an i.i.d. Bernoulli(0.2) null at $\kappa = 0.05$ are 155.6 bars at $h = 3$, 328.2 at $h = 4$, 654.4 at $h = 5$, 1243.0 at $h = 6$, 4117.8 at $h = 8$ and 13 063.9 at $h = 10$; detection delays under a true rate of 0.30 are 38.7, 56.1, 74.7, 94.1, 133.0 and 173.0 bars. Breaches cluster, so these are ceilings on the true in-control run length: recalibrate h against a block bootstrap of the platform's own breach series before the shadow window opens, never from a table computed under independence [Course Book §13.7].
The three alarm actions are not interchangeable and Table 6.10 assigns each to a trigger. Log every reset with its trigger, its timestamp and the buffer state before and after; an unjournalled reset breaks the audit in Step 6.6's pass check.

The mathematics. With err_s the breach indicators, the three instruments are

$$\widehat{\text{cov}}_t = 1 - \frac{1}{W}\sum_{s=t-W+1}^{t} \text{err}_s, \qquad \text{SE} = \sqrt{\frac{\alpha(1-\alpha)}{W}},$$

$$S_t = \max\{0,\ S_{t-1} + \text{err}_t - \alpha - \kappa\}, \qquad \text{alarm at } S_t > h,$$

where W is the rolling window, κ the reference value and h the threshold; the standard error treats breaches as independent and is therefore a floor on the noise, not an estimate of it. [Course Book §11.8] [Course Book §13.7]

Artifact. `coverage_monitor(instrument, granularity, alpha, bar_ts, cov_w, se_w, pit_hist_json, cusum_s, alarm_level)`; `alarm_journal` recording every trigger and action.

Pass / fail. (a) Rolling coverage recomputed from `score_buffers` offline matches the live column exactly. (b) The PIT histogram integrates to one and its χ^2 against uniform is reported, not acted on — the formal test lives in Step 6.10. (c) The CUSUM fires at least once in a

deliberately injected drift test (force α to 0.30 in a replay) within the tabulated delay. (d) No cell logs more than one h_{reset} event per calendar quarter during shadow; more than that means h_{reset} is too low or the vol model is broken.

Table 6.9. Monitoring parameters per timeframe. W is chosen to hold the coverage standard error at or below 2.53 percentage points while keeping the window's calendar span readable. h_{notify} and h_{reset} are the two CUSUM thresholds with their approximate detection delays $h/(\delta - \kappa) = 20h$ bars.

TF	W	SE (pp)	W (days)	h_{n}	delay (d)	h_{r}	delay (d)	CUSUM role
M1	1000	1.26	0.69	5	0.07	10	0.14	primary
M5	1000	1.26	3.47	5	0.35	10	0.69	primary
M15	500	1.79	5.21	5	1.04	10	2.08	primary
M30	500	1.79	10.42	5	2.08	10	4.17	primary
H1	500	1.79	20.83	5	4.17	10	8.33	primary
H4	250	2.53	41.67	4	13.33	8	26.67	primary
D	250	2.53	250.00	3	60.00	6	120.00	secondary

At D the CUSUM needs sixty trading days to detect a fifty-per-cent rise in the breach rate, so it is demoted to a secondary instrument and the rolling coverage z-score leads: with $W = 250$ the standard error is 2.53 percentage points, so a rolling coverage of 70% sits 3.95 standard errors below target and is evidence, while 75% sits 1.98 below and is not.

Table 6.10. Alarm ladder. The three actions address different failures and firing the wrong one makes the calibration worse, not better.

Trigger	Action	Why this action, and its cost
$\widehat{\text{cov}}_t$ below target by $>$ 2 SE, CUSUM below h_{n}	Raise a regime flag to the router; no change to the band	Costs nothing statistically; tells the sizing layer to stand down where losses concentrate
$S_t > h_{\text{n}}$	Flag, page the owner, freeze the weekly refit at the current vintage	Prevents a refit from absorbing a break into its coefficients before it is understood
$S_t > h_{\text{r}}$ and a break time is identifiable	Purge $\mathcal{B}$ of scores predating the estimated break; then reset $\alpha_{\text{state}} \leftarrow \alpha$ and $S \leftarrow 0$	Fixes a level shift in the score distribution; discards the controller's deficit, so it is correct only *after* a purge and must be journalled
$S_t > h_{\text{r}}$ with no identifiable break	Purge nothing; widen by holding β at its current value and escalate	A CUSUM alarm with no locatable break is usually a broken vol model, not a regime change

Step 6.8 — Integration with the C# engine

Do. Write the adaptive output into `timesfm.sqlite` as a table the sidecar *owns*: `adaptive_bands`, dropped and recreated wholesale on every cycle, exactly as `book_ml_service.py` does with `book_ml_predictions`. Table drops are the only integration contract on this platform and this step does not amend it. Do not `ALTER TABLE` the TimesFM forecast table to add columns: that table is owned by the forecast writer, a shared schema is a two-writer coordination problem nobody has solved here, and the contract explicitly does not cover it.

The C# engine reads by a left join on `(instrument, granularity, forecast_ts)`, so the adaptive columns appear on the row it already polls. Apply the compatibility rule in

Table 6.11 in order; the first failing condition selects the legacy path for that row and increments a counter. Fallback is per row, never per session — one stale cell must not disable the engine for twenty others.

Artifact. `adaptive_bands(instrument, granularity, forecast_ts, adaptive_q10, adaptive_q20, adaptive_q80, adaptive_q90, engine_version, coverage_stat, beta_level, c_multiplier, sigma_h, vol_family, vintage_id, buffer_n, written_ts)`, plus `fallback_counter` by reason code.

Pass / fail. (a) A cold-start test with `adaptive_bands` absent runs the engine to completion on the legacy path with zero exceptions. (b) A poisoned-row test (null quantile, crossed quantiles, stale `written_ts`, `coverage_stat` out of range) falls back on exactly those rows and no others. (c) The drop-and-recreate cycle never leaves the table absent for more than one polling interval — write to a temporary table, then swap in a single transaction. (d) Reconciliation: for any historical bar, the band reconstructed from `c_multiplier`, `sigma_h` and `vintage_id` equals the stored `adaptive_q10..q90` to 10^{-9}.

Table 6.11. Compatibility rule, evaluated in order per row. Any failure selects the legacy median-plus-`width_mult` band for that row.

#	Condition	Reason
1	`engine_version` is non-null and appears in the engine's accepted-version list	An unknown writer version is not a contract
2	All four `adaptive_q*` are non-null, finite and monotone	A crossed or missing quantile is a broken band, not a wide one
3	`written_ts` is within two polling intervals of now	A stale row is a dead sidecar
4	`buffer_n` ≥ 0.9 of the cell's target n_T	A warming buffer certifies a level it cannot support
5	`coverage_stat` (rolling $\widehat{\text{cov}}$) within 4 SE of target	The engine refuses a band its own monitor disowns
6	`c_multiplier` is finite	$\hat{c} = +\infty$ means the buffer cannot certify the level

Step 6.9 — Shadow mode

Do. Run both bands live, side by side, with the legacy band authoritative for execution throughout. Set the engine flag `use_adaptive = 0` for the whole window; flipping it is a single configuration change gated on Step 6.10, and it is the only change that flip makes. Figure 6.1 is the wiring.

Every bar, write one row to `shadow_ledger` carrying *both* bands, the state that produced the adaptive one, and the sizing decision each would have produced. Resolve the row 50 bars later with the realised move, both breach indicators and both realised P&L contributions. The sizing decision is computed by the same policy object the Stage-0 harness uses — not a reimplementation — so the would-have-been comparison is a difference in inputs and nothing else.

Run for at least six weeks, and for thirteen weeks (one quarter) before treating H4 or D as gate-decisive. Reason: at six weeks a per-cell H4 stress bucket holds about 36 observations and a per-cell D bucket about 6, both far below the floor at which a Christoffersen verdict means anything; pooled across the 21 instruments the six-week H4 stress bucket reaches 756 but the D bucket only 126, and thirteen weeks is what lifts pooled D to 273.

Freeze everything for the duration. No change to the proxy sourcing, the model families, ν, γ, the buffer sizes, the score form or the alarm thresholds. Any change restarts the clock from zero. Reason: a shadow window containing a mid-flight parameter change is not a sample of anything, and the temptation to make one arrives precisely when the early numbers look bad.

Artifact. `shadow_ledger` (Table 6.12) and the comparison dashboard of Table 6.13.

Pass / fail. (a) `use_adaptive` is 0 in every configuration snapshot taken during the window — checked, not assumed. (b) Row coverage: the ledger has one resolved row per cell per bar, with gaps only where the feed gapped, reconciled against `vol_proxy.status`. (c) The frozen-configuration hash is identical at the first and last bar of the window. (d) The legacy columns of `shadow_ledger` reconcile bit-for-bit with what the engine actually executed.

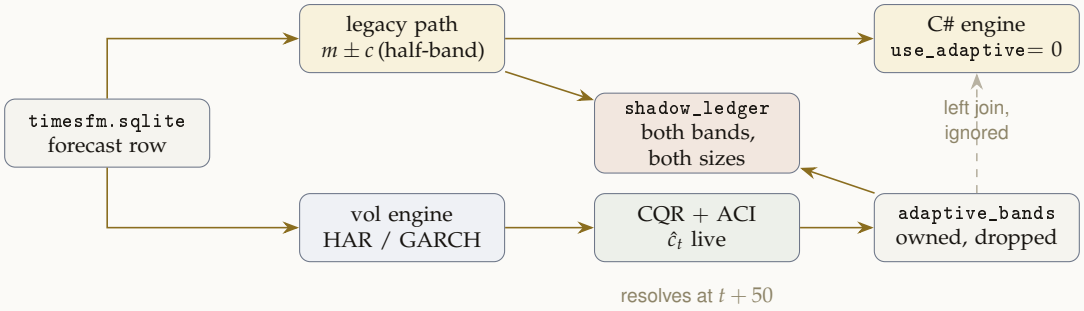


Figure 6.1. Shadow-mode wiring. Solid arrows carry live data; the dashed arrow is the join the C# engine performs and then discards while `use_adaptive` is zero. Acceptance flips exactly one flag; nothing else in the diagram moves.

Table 6.12. `shadow_ledger` columns. One row per cell per bar, written at emit and completed at resolution.

Group	Columns
Keys	`instrument, granularity, forecast_ts, resolution_ts`
Legacy	`legacy_q10, legacy_q90, width_mult, legacy_size`
Adaptive	`adaptive_q10, adaptive_q20, adaptive_q80, adaptive_q90, adaptive_size`
State	`sigma_h, vol_family, vintage_id, nu, beta_level, alpha_state, c_multiplier, buffer_n`
Context	`width_pct, regime_label_asof, vol_bucket, cost_cell, cost_vintage`
Outcome	`y_realised, err_legacy, err_adaptive, pit_legacy, pit_adaptive, pnl_legacy, pnl_adaptive`

The third panel is the one that decides the stage, and it can be read weeks before the gate. If the adaptive band moves size a great deal and the priced turnover of that movement is the same order as the P&L difference it produces, Stage 1 is heading for the calibration-only kill and the team should be preparing that memo rather than hoping.

6.5 Acceptance

One step, two independent verdicts, and a ledger that decides how much either is worth.

Table 6.13. Comparison dashboard. Three panels, refreshed daily, each answering one question the gate will ask.

Panel	Contents	What a failure looks like
Coverage by regime	Breach rate for both bands within each `vol_bucket` quintile and each as-of regime label, with bucket n printed beside every figure	Adaptive coverage flat near 80% across quintiles is the target; a residual slope means the vol model, not the conformal layer, is short
Width-ratio distribution	Histogram and deciles of $w_{\text{adaptive}}/w_{\text{legacy}}$ per cell, plus its autocorrelation and the mean absolute bar-to-bar change	A ratio distribution centred near 1 with no dispersion means the engine is reproducing the frozen multiplier and adding only churn
Would-have-been sizing	Distribution of $\lvert s_{\text{adaptive}} - s_{\text{legacy}}\rvert$, turnover implied by the difference, and that turnover priced at the cost vintage	Sizing differences whose priced turnover exceeds the P&L difference: calibration bought at a cost the book cannot recover

Step 6.10 — Acceptance runs

Do. Three runs, in this order, on the frozen shadow output.

Run A — per-regime Christoffersen battery. Partition the resolved shadow rows by `vol_bucket` (quintiles of the width percentile, assigned from information available at $t-1$) and by the as-of regime label. Bucketing on anything known only at t or later — realised volatility over the bar, a smoothed label — manufactures rejections out of look-ahead and voids the whole run [Course Book §17.4]. Run the battery on both bands, at $p = \alpha$, once for each bucket and once pooled. Count transition pairs *calendar-consecutive*: retain only pairs adjacent in the original series, so the pair total falls below $n-1$ by the number of regime entries, and report the pair total beside every n. Read no bucket with $n < 100$; treat $100 \le n < 250$ as underpowered, where a rejection is informative and a pass is not; treat $n \ge 250$ as decisive. Table 6.14 is the arithmetic behind those two thresholds. Run the battery three times per bucket — upper tail at $p = 0.1$, lower tail at $p = 0.1$, two-sided at $p = 0.2$ — because a two-sided indicator pools the tails and a band too narrow above and too wide below passes. The number of buckets is itself a multiplicity problem; apply the correction of [Course Book §17.5] and report both raw and adjusted p-values.

Run B — sizing policy on the new band. Take the Stage-0 policy object unchanged, feed it `adaptive_q10..q90` in place of the legacy band, and put it through the Chapter 4 harness in full: CPCV with the same group count, the same 50-bar purge, the same per-timeframe embargo, the same cost surface at the same vintage, the same per-path net P&L. Nothing about the harness is re-tuned for Stage 1. Rerun at 1.5× and 2× cost as Stage 0 did. If Stage 0 killed, run the two benchmarks — constant fractional risk and volatility targeting — through the identical pipeline at matched average exposure, normalised on the training split only, and use the better of them as the baseline.

Run C — deflation. Write a ledger row for every one of the 420 declared Stage-1 configurations before any result is inspected, abandoned ones included (Table 6.16); compute the trial variance V from the ledger and the Deflated Sharpe Ratio with the Chapter 4 metrics module. At $N = 420$ the expected-maximum multiplier is 2.9997: the champion must clear almost exactly three trial standard deviations before deflation credits it with anything [Course Book §15.4]. The relevant N is cumulative — Stage-0 trials do not leave the ledger because Stage 1 started.

The mathematics. With I_t the breach indicator, n_0, n_1 the non-breach and breach counts, $\hat{\pi} = n_1/T$, transition counts n_{ij}, $\hat{\pi}_{01} = n_{01}/(n_{00}+n_{01})$, $\hat{\pi}_{11} = n_{11}/(n_{10}+n_{11})$ and $\hat{\pi}_2 =$

$(n_{01} + n_{11})/(T-1)$,

$$\mathrm{LR}_{\mathrm{uc}} = -2\big[n_0 \ln(1-p) + n_1 \ln p - n_0 \ln(1-\hat{\pi}) - n_1 \ln \hat{\pi}\big] \sim \chi^2(1),$$
$$\mathrm{LR}_{\mathrm{ind}} = -2\big[(n_{00}+n_{10}) \ln(1-\hat{\pi}_2) + (n_{01}+n_{11}) \ln \hat{\pi}_2 - n_{00} \ln(1-\hat{\pi}_{01}) - n_{01} \ln \hat{\pi}_{01} - n_{10} \ln(1-\hat{\pi}_{11}) - n_{11} \ln \hat{\pi}_{11}\big] \sim \chi^2(1), \qquad \mathrm{LR}_{\mathrm{cc}} = \mathrm{LR}_{\mathrm{uc}} + \mathrm{LR}_{\mathrm{ind}} \sim \chi^2(2),$$

with p the claimed breach rate, identical in every bucket. Critical values at 5%: 3.8415 on one degree of freedom, 5.9915 on two. [Course Book §17.2]

Artifact. `christoffersen_report` (one row per band per bucket per tail: n, pair total, n_1, $\hat{\pi}$, the four transition counts, the three statistics, raw and adjusted p); `stage1_cpcv_paths`; 420 `trial_ledger` rows; `stage1_decision_memo`.

Pass / fail. (a) The identity $n_{00} + n_{01} + n_{10} + n_{11} =$ pair total holds in every report row — the cheapest guard against a counting bug [Course Book §17.2]. (b) Every bucket is defined by information available at $t-1$; an audit query confirms no bucket predicate reads a column stamped at t or later. (c) The ledger contains exactly 420 Stage-1 rows and every one was written before its metrics were read; the CI check compares row creation timestamps against metric write timestamps. (d) The harness CI suite is green on the day of the run; a red suite freezes all result-reading.

Table 6.14. Christoffersen bucket sizing at $p = 0.2$. "Accept" is the breach count and coverage range the unconditional-coverage test does not reject at 5%; "size" is the actual rejection rate under the null, above 5% by the usual lattice discreteness. The $n = 30$ row reproduces the course book's worked rejection region.

n	Accept n_1	Accept coverage	Size	Power @ 0.30	Power @ 0.35	Gate status
30	[3, 10]	[66.7%, 90.0%]	0.0698	0.2717	0.4926	not read
100	[13, 28]	[72.0%, 87.0%]	0.0453	0.6232	0.9152	underpowered
250	[39, 62]	[75.2%, 84.4%]	0.0579	0.9596	0.9997	decisive
500	[83, 117]	[76.6%, 83.4%]	0.0502	0.9994	1.0000	decisive
1000	[176, 225]	[77.5%, 82.4%]	0.0481	1.0000	1.0000	decisive

A bucket of 30 accepts any coverage between 66.7% and 90.0%: a band delivering two-thirds coverage where it promised four-fifths passes without comment. That is why the gate reads nothing below 100 and treats a pass below 250 as silence rather than evidence. Table 6.15 says which cells reach those counts and when.

Table 6.15. Stress-bucket sample sizes during shadow, taking the stress bucket as the top quintile of the width percentile. Per-cell counts are one instrument; pooled counts are all 21.

	6 weeks (30 days)			13 weeks (65 days)		
TF	per cell	pooled	verdict	per cell	pooled	verdict
M1	8640	181 440	per-cell decisive	18 720	393 120	per-cell decisive
M5	1728	36 288	per-cell decisive	3 744	78 624	per-cell decisive
M15	576	12 096	per-cell decisive	1 248	26 208	per-cell decisive
M30	288	6 048	per-cell decisive	624	13 104	per-cell decisive
H1	144	3 024	per-cell underpwr	312	6 552	per-cell decisive
H4	36	756	pooled only	78	1 638	pooled only
D	6	126	neither	13	273	pooled only

Three consequences follow and all three are pre-committed, not negotiated after the numbers arrive. H4 and D are evaluated on instrument-pooled buckets only, and the pooling is declared before the window opens. A six-week window cannot deliver a D verdict at all, which is the

operational reason the recommended shadow period is thirteen weeks and not six. And M1, whose vol proxy is a squared return with noise-to-signal above one, is reported but carries no gate weight — abundant observations do not repair a proxy that cannot measure the thing being modelled.

Table 6.16. The declared Stage-1 configuration grid. Every cell of the product is a ledger row, written before results are read, whether or not it is ever run to completion.

Axis	Count	Values
Volatility family	2	HAR (direct log), GARCH(1,1)
Innovation shape ν	7	4, 5, 6, 8, 10, 20, ∞
ACI step γ	5	0.005, 0.0075, 0.01, 0.02, 0.05
Buffer scale	3	$0.5\times$, $1\times$, $2\times$ the n_T of Table 6.7
Score form	2	multiplicative, additive CQR
Total N	**420**	expected-maximum multiplier $2.9997\sqrt{V}$

6.6 The gate

Two limbs, evaluated independently and combined by a rule fixed before the shadow window opened. The calibration limb asks whether the band is now conditionally honest. The P&L limb asks whether that honesty is worth anything after costs and after multiplicity. Neither substitutes for the other.

Stage gate — Stage 1: adaptive width

Limb 1 — conditional coverage recovers. On the frozen shadow output, for the adaptive band, in *every* stress bucket with $n \geq 100$: $\mathrm{LR}_{\mathrm{cc}} < 5.9915$ and the multiplicity-adjusted p-value exceeds 0.05. In addition, at least one bucket per gate-weighted timeframe has $n \geq 250$ — the gate-weighted set is M5, M15, M30, H1, and H4 and D on pooled buckets; M1 is reported and excluded, its proxy being a squared return. And there must be something to recover: the legacy band must be rejected at 5% in at least one stress bucket where the adaptive band is not. Coverage that was already conditionally correct is not a Stage-1 achievement.

Limb 2 — the band pays. The Stage-0 sizing policy driven by the adaptive band, run through the Chapter 4 harness unchanged, must satisfy all three: DSR > 0.5 against the baseline at the *cumulative* ledger N (Stage-0 rows included); worst-decile CPCV path net P&L strictly positive; and the ranking against the baseline preserved at $1.5\times$ cost. The baseline is the Stage-0 winning policy if Stage 0 passed, and the better of constant fractional risk and volatility targeting at matched average exposure if Stage 0 killed.

Decision rule. Both limbs pass $\Rightarrow$ set `use_adaptive` $= 1$, retire `width_mult` to a fallback constant, proceed to Stage 2 with the adaptive envelope as the new baseline. Limb 1 only $\Rightarrow$ the kill criterion below. Limb 2 only $\Rightarrow$ *do not ship*. A P&L gain from a band whose conditional coverage did not recover is a sizing artefact of changed width dynamics, not a calibration result; hold the shadow for a further quarter and re-run, and record the extra configurations in the ledger. Neither $\Rightarrow$ revert to legacy, keep the vol-model store as a monitoring asset, and record the outcome as a completed negative trial.

Kill criterion — Calibration-only: the pre-committed wording

Sign this verbatim when Limb 1 passes and Limb 2 fails. Do not edit the thresholds; they were fixed before the window opened.

"Stage 1 delivered a conditionally calibrated 80% envelope. Per-regime Christoffersen tests on the shadow output show the adaptive band's conditional coverage is not rejected in any stress bucket with $n \geq 100$, where the frozen `width_mult` band was rejected. This is a genuine improvement in risk information and it ships: the adaptive band becomes the platform's reported envelope and the basis of every displayed uncertainty figure.
It did not pay. The Stage-0 sizing policy driven by the adaptive band failed the deflated net-of-cost test against _____ at cumulative ledger $N =$ _____: DSR $=$ _____ against a threshold of 0.50, worst-decile path P&L $=$ _____. The improvement in calibration was therefore not tradeable at this platform's cost surface and horizon.
Consequently: Stage 2 is not entered on the profit-and-loss thesis. The regime router and covariate probes may be built only as inputs to the calibration layer, evaluated on coverage, and any sizing use of them requires a new stage with its own pre-registered gate. This decision is not renegotiated by a subsequent re-run of the same policy class on the same instruments; a re-run increases the ledger's N and raises the deflation bar it must clear."

Signed, owner _____ reviewer _____ date _____

What happens next. The adaptive engine stays live for *reporting* and off for sizing, a configuration Step 6.8's compatibility rule already supports. Monitoring, refits and the alarm ladder continue unchanged: a calibration layer not fed the sizing decision still needs to stay calibrated. Vol-model store and score buffers become permanent assets of the signatory above.

Artifacts — Stage output register

Everything Stage 1 leaves on disk; a stage not reconstructible from this list did not happen.

- `vol_proxy` — realised variance per cell with `m_used`, `source_tf`, `status`; `proxy_floor_report.json`, the measured microstructure floor per instrument.
- `vol_models` — vintaged HAR and GARCH coefficients per cell (Table 6.5), rejected and superseded vintages included. Never pruned: every band emitted names a row here.
- `model_select` — per-fold QLIKE for both families, fold votes, per-cell modal winner.
- `raw_bands` — pre-conformal quantiles with `sigma_h`, ν, `vintage_id`.
- `score_buffers`, `pending_bands` — the 294 live score pools, persisted each cycle so a restart is warm; realisation-time predicate enforced in CI.
- `aci_state` — 294 unclipped controller states with reset counts; `alarm_journal` records every purge and reset with before/after buffer state.
- `coverage_monitor` — rolling coverage, standard error, PIT histogram, CUSUM per cell per level; Table 6.13's dashboard reads from here.
- `adaptive_bands` — the owned, drop-and-recreate integration table (Step 6.8), plus `fallback_counter` by reason code.
- `shadow_ledger` — both bands, sizings and outcomes, one row per cell per bar (Table 6.12), with the frozen-configuration hash.
- `christoffersen_report` — per band, bucket and tail: n, pair total, breach and transition counts, the three statistics, raw and adjusted p.
- `stage1_cpcv_paths` — per-path net P&L at base, 1.5× and 2× cost.
- `trial_ledger` — 420 rows, run and abandoned, each written before its metrics were read, each carrying the entry-path field of Table 6.2.
- `stage1_decision_memo` — the signed gate outcome, or the killbox wording with its

> blanks filled.

7

Stage 2: The Regime Router and the Covariate Probes

Stage 2 asks a narrower and cheaper question than Stage 1's. Stage 1 built a calibrated, adaptive band from each instrument's own price history and nothing else. Stage 2 asks whether anything outside *that history — positioning, risk sentiment, a slower macro clock than the return series itself carries — adds conditional information the univariate engine cannot see. Two independent, cheap probes answer it: a hidden Markov regime router built on four point-in-time macro gauges, filtered causally and wired into three concrete gates (Steps 7.1–7.5); and a zero-training, in-context probe of Chronos-2's covariate channel, run as a batch job over the regenerated history (Step 7.6). Neither probe is deployed on a pass. Both exist to produce one incremental verdict against the Stage-1 baseline, on the same cumulative, deflated ledger the whole programme shares (Steps 7.7–7.8). A pass licenses Stage 3's bespoke, covariate-typed model; nothing less does, and marginal is kill, exactly as Step 5.7 pre-committed for Stage 0.*

Two gates must already be green. First, Chapter 3's point-in-time feed registry: every gauge this chapter consumes answers the as-of join with `release_ts` $\leq t -$ safety lag, and Step 3.8's audit query is green on the current commit [Course Book §18.2]. Second, Stage 1's shadow period (Step 6.9) must be running or complete, because the baseline this whole chapter measures against is the Stage-1 *adaptive* band, not the Stage-0 static multiplier — comparing a router to a weaker baseline than the one actually in shadow would understate the bar and manufacture a false pass.

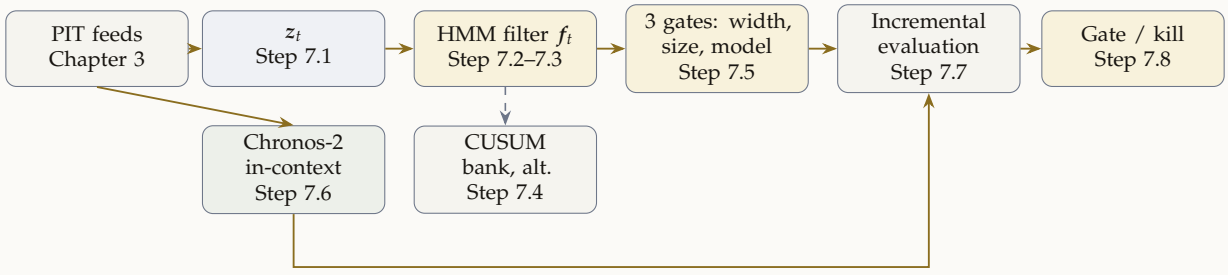


Figure 7.1. The Stage-2 pipeline: two independent probes feeding one incremental evaluation. The HMM router (top path) is causal from Step 7.1 onward — filtered probabilities only, never smoothed. The CUSUM bank is a lighter-weight *alternative* to the HMM box, not an addition to it (Step 7.4). The Chronos-2 probe (bottom path) never touches the router; it is scored standalone against the same Stage-1 baseline.

7.1 Preconditions and the observable vector

Step 7.1 — Define and standardise the observable vector

Do. Build one row per (instrument, t) at H1: $z_t = (\text{VolRegime}_t,\ \text{RORO}_t,\ \text{COT-}z_t,\ \text{RVratio}_t)^\top$, mirroring [Course Book §13.8]'s emission vector exactly. Sources: VolRegime_t and RORO_t from the reconstructed as-of gauge series (Step 3.4, Step 3.5); COT-z_t from the feeds table's as-of join (Step 3.1–3.2), which already enforces the release lag; $\text{RVratio}_t = \text{RV}_{\text{fast},t}/\text{RV}_{\text{slow},t}$, the fast/slow realised-volatility ratio built from the same proxy Step 6.1 defines. All four are point-in-time by construction of their upstream chapters — this step does not re-derive that property, it consumes it.

Standardise causally: a rolling or expanding window strictly before t, never the full sample (Step 3's own convention, generalised here to all four gauges). Missing data: diagonal covariance (Step 7.2) makes each gauge's emission density a separate factor, so a bar with one gauge flagged `MISSING` by Step 3.7's staleness rule drops only that factor from the live emission product — the filter still emits a probability update from the remaining gauges rather than stalling. *Fitting* is stricter: any bar with any gauge missing is excluded from the Baum–Welch training sample outright (Step 7.2), since a partial-likelihood EM step is not worth the complexity a single stale feed would buy.

Zero-mean pinning does **not** apply here. [Course Book §13.1] pins the return emission's mean at zero because Chapter 1's signal-to-noise arithmetic rules out a tradeable conditional mean in *returns*; z_t is not a return, and the entire point of a regime label is that its mean genuinely shifts across states — a stress regime should show an elevated VolRegime reading, a negative RORO reading, an extreme COT z-score and an RVratio above one. Fit both μ_i and σ_i per gauge per state.

The mathematics. Causal z-scoring, one gauge at a time:

$$\hat{z}_{t,k} = \frac{x_{t,k} - \hat{\mu}_{t-1,k}}{\hat{\sigma}_{t-1,k}},$$

with $\hat{\mu}_{t-1,k}$, $\hat{\sigma}_{t-1,k}$ the rolling or expanding mean and standard deviation of raw gauge k computed from bars strictly before t. [Course Book §13.8]

Artifact. Table `regime_obs(instrument, t, volregime_z, roro_z, cot_z, rv_ratio_z, any_missing, vintage)` at H1, 21 instruments.

Pass / fail. (a) *Point-in-time*: Step 3.8's audit query, re-run against `regime_obs`, returns zero rows where any component's underlying `release_ts` exceeds t. (b) *Coverage*: the fraction of H1 bars with all four components present is at least 95 per cent (recommended threshold; once Chapter 3's registries are live, a lower figure means a feed is stalling, not that the target is wrong). (c) Every persisted row's `any_missing` flag matches the four raw columns (schema self-check).

7.2 Fitting and deploying the router

Key idea — Fit smoothed, deploy filtered

Baum–Welch's E-step responsibilities γ_t, ξ_t (Step 7.2) condition on the whole training split; that is correct and required for estimation, not a defect to work around. What is not legitimate is carrying that smoothing into application. The instant θ is frozen, deployment uses only the forward filter of Step 7.3 on bars strictly after the train/test boundary, with Chapter 4's purge and embargo (Step 4.2) applied around it, and periodic refits (Step 7.3) are a walk-forward

scheme, never a re-smooth of published history. [Course Book §13.6]

Step 7.2 — Fit the router: Baum–Welch, seed grid, BIC, and label stability

Do. Per instrument, per $K \in \{2, 3\}$, fit a diagonal-covariance Gaussian HMM on z_t by Baum–Welch, on the CPCV train split only (Step 4.2) — never on bars this stage's own incremental evaluation (Step 7.7) will later test. Re-estimation is EM: the E-step responsibilities and the M-step closed forms are not restated here [Course Book §13.6].

Run 20 seeds per K: 15 random restarts plus 5 informed starts, one per realised-vol-ratio quantile in $\{0.20, 0.35, 0.50, 0.65, 0.80\}$ (generalising [Course Book §13.6]'s single-quantile informed start to a small grid) — a recommended count, not a derived one; `verify_pb07.py` confirms the resulting bookkeeping: 20 seeds $\times$ 2 values of $K = 40$ fits per instrument, $\times$ 21 instruments $= 840$ fits for the whole universe.

After each fit converges, impose an ordering constraint before anything downstream reads the state index. [Course Book §13.3]'s convention orders states by $\sigma_1 < \cdots < \sigma_K$ for a univariate emission; here, with $m = 4$ gauges, order instead by the fitted RVratio-component mean $\hat{\mu}_{i,\text{RVratio}}$, ascending — the one gauge whose economic direction (higher means more stressed) is unambiguous.

Select, per K, the seed with the lowest BIC. Among seeds within 2 log-likelihood units of that best fit (a standard "substantially indistinguishable" cutoff), compute the label-stability check: run the causal filter of Step 7.3 for each such seed over a held-out validation slice, take the bar-by-bar arg max state, and report the pairwise agreement rate across seeds. Require agreement ≥ 0.90 before deploying that K (recommended threshold: below it, the fit is not identified by the data, and BIC's own regularity conditions already fail for mixtures, so a shaky numerical win should not be trusted [Course Book §13.6]). If BIC prefers $K = 3$ but its stability fails while $K = 2$'s clears 0.90, deploy $K = 2$ and log the conflict in the fit report; do not average or otherwise blend the two.

The mathematics. $\text{BIC}(K) = -2 \ln \hat{L}_K + p_K \ln T$, with $p_K = K^2 + 2mK - 1$ the free-parameter count for m-gauge diagonal emissions ($m = 4$ here; [Course Book §13.6] states the criterion, [Course Book §13.8] the $2mK$ emission count it generalises). At $T = 18{,}720$ H1 bars (3 years, `verify_pb07.py`), $p_2 = 19$, $p_3 = 32$, $\ln T = 9.8373$: moving from $K = 2$ to $K = 3$ costs $13 \times 9.8373 = 127.89$ in BIC and must buy more than 63.94 in log-likelihood to be worth it on fit quality alone — a bar Step 7.2's stability check enforces independently of whether BIC clears it, because BIC is a tiebreaker here, not a test.

Artifact. Table `hmm_fit(instrument, K, seed_id, loglik, bic, converged, order_ok, is_selected, fit_ts)`, one row per seed (all 840, for the audit trail) plus a `hmm_params` table holding the full parameter blob for the *selected* seed only, per (instrument, K).

Pass / fail. Selected K's label-stability ≥ 0.90; `order_ok` true (RVratio-mean ordering holds); episode-count sanity: only about 64 calm/stress cycles are available per instrument over 3 years of H1 data at the durations Step 7.1's slow gauges target (`verify_pb07.py`), so a fit reporting materially shorter durations than that has fit noise, not regimes, and fails regardless of BIC or stability.

Trap — Label switching across refits inverts every gate at once

The ordering convention is enforced at fit time, but a walk-forward refit (Step 7.3) reruns Baum–Welch from a new seed grid on a shifted window, and nothing stops the EM optimum from landing on a permutation of the states relative to the previous vintage. A router that silently swaps which index means "stress" does not fail loudly — it inverts the width blend,

the size throttle and the model gate all at once, in the same refit, and every one of Step 7.5's three gates goes wrong together. Enforce the RVratio-mean ordering (Step 7.2) on *every* refit as a CI assertion, not a one-time check: reject any refit whose ordering does not hold, keep the previous vintage's filter running, and alert. [Course Book §13.3]

Step 7.3 — Deploy the filter and persist an as-of series

Do. Freeze θ at the train/test boundary (Step 4.2's purge and embargo applied around it, as for every other fitted component on this platform). Run the causal scaled forward filter — the router loop of [Course Book §13.8], restated once below — bar by bar over the evaluation and live window; emit $\boldsymbol{f}_t$ (a K-vector summing to one) at every bar and persist it as an as-of series, exactly like every other feed this platform stores: `regime_filtered(instrument, t, k, state_idx, prob, params_vintage, computed_at)`.
Refit on a walk-forward schedule matching Step 6.2's cadence (weekly; reused for the same reason: cheap enough to run often, slow enough that intraday noise does not dominate the retrained coefficients). On each refit: rerun Step 7.2's full seed grid and stability check on the new expanding window, apply the ordering-constraint CI assertion (the trapbox above), and only then resume filtering forward from the new θ. Never re-filter the historical window under the new θ: each persisted `regime_filtered` row keeps the `params_vintage` it was *actually* computed under, so a refit appends new rows going forward and never rewrites a published one — retroactively updating history under a better-fitting θ is itself a quiet form of the leak below.

The mathematics. The causal router loop, one bar: with predicted state distribution $\boldsymbol{p}$ carried from the previous bar, $u_j = p_j\, b_j(\tilde{\boldsymbol{z}}_t)$ for $j = 1, \ldots, K$; $c = \sum_j u_j$; $f_j = u_j / c$ (the filtered posterior, the only regime quantity this platform may consume); $\boldsymbol{p} \leftarrow \boldsymbol{f}\boldsymbol{A}$ for the next bar. [Course Book §13.4]

Artifact. `regime_filtered` as-of series (above); refit log (`instrument, refit_ts, prev_vintage, new_vintage, ordering_ok, stability_agreement`).

Pass / fail. For every persisted row, the fit's `train_end_ts` $\leq t$ (no smoothing, no look-ahead in the parameters); $\sum_k f_{t,k} = 1$ within float tolerance at every row; the refit log shows zero `ordering_ok = false` rows reaching `regime_filtered` (the trapbox's CI assertion actually blocked what it was built to block).

Trap — Smoothed regime labels are prophecy — and here is the query that catches it

A smoothed label $\gamma_t = \mathbb{P}(S_t \mid y_{1:T})$ conditions on the whole sample; used as a signal at bar t it hands the strategy the likelihood ratio of every future observation [Course Book §13.5]. Three enforcement points, each with a runnable check.
(i) Structural. `regime_filtered` must never contain a row computed from a fit or a bar beyond t. Audit query, run on every commit exactly as Step 3.8's feed audit is:

```
SELECT r.instrument, r.t, r.params_vintage
FROM regime_filtered r JOIN hmm_params v ON v.vintage_id =
r.params_vintage
WHERE v.train_end_ts > r.t OR r.computed_from_ts > r.t;
-- expected result:  empty set
```

(ii) Usage. No sizing or gating decision may reference a smoothed or Viterbi table. Flag any trial whose configuration references one:

```
SELECT trial_id, config_hash FROM trial_ledger
WHERE config_json::text ILIKE '%regime_smoothed%'
```

```
    OR config_json::text ILIKE '%viterbi%';
-- expected result:  empty set outside the offline-analysis tag
```

(iii) Statistical. Chapter 4's leak-injection suite (Step 4.5–4.8) carries a fifth, Stage-2-specific control: rerun Step 7.7's winning sizing policy with `regime_filtered` swapped for a smoothed `regime_smoothed` table on the same fold, and assert the reported Sharpe *increases*. If it does not, the regime signal was already contributing nothing and Step 7.7 will already show it; if it jumps materially, that jump is the size of the trap avoided, and it must never appear in a passing ledger row [Course Book §13.5].

Viterbi paths and smoothed plots remain legitimate for exactly one purpose: dating historical stress episodes offline, e.g. to build the regime-specific width fits of Step 7.5(a). They are never joined to a live decision or a backtested P&L series.

7.3 The CUSUM bank: a cheaper alternative

Step 7.4 — Run a per-gauge CUSUM bank instead of, or alongside, the HMM

Do. Run the two-sided CUSUM of Step 6.7 — unchanged — on each of z_t's four causally z-scored components independently: in-control mean $\mu_0 = 0$ by construction of the z-score (Step 7.1), rather than on a conformal breach indicator. This is the same recursion applied to a different input series; nothing about the statistic itself changes [Course Book §13.7].

Per gauge, choose the allowance k from the shift you want to catch: recommend $k = 0.75$ (detects a 1.5σ-equivalent shift in a standardised gauge, the Gaussian rule of thumb $k = \delta/2$ with $\delta = 1.5$), and calibrate h by the same ARL_0-simulation protocol as Step 6.7, reused verbatim — simulate the false-alarm rate under the platform's own three years of gauge history and target one false alarm per instrument-year, exactly as there.

When to prefer the bank over the HMM. Prefer it when the question is narrower than the router needs: *has this gauge moved*, not *which of K named states are we in, with what probability, blended into a continuous width and size*. The bank needs no transition matrix, no joint covariance, no label-switching convention and no seed grid — four independent one-line recursions against Step 7.2's episode budget, rather than one router competing for it, which matters most when a pair's own COT or RORO history is short or gapped. The cost is what it does not produce: a CUSUM alarm is a hard 0/1 flag with no persistence machinery, so it cannot feed [Course Book §13.8]'s smooth width blend (Step 7.5(a)) or a continuous size throttle (Step 7.5(b)) without an ad hoc wrapper that reintroduces the flicker a hard label produces. Its natural role on this platform is the discrete model gate (Step 7.5(c)) alone, or a second, independent corroboration signal alongside the HMM's filtered stress probability — not a full replacement for the router's three-gate design.

The mathematics. Unchanged from Step 6.7: $S_0 = 0$, $S_t = \max(0,\ S_{t-1} + x_t - \mu_0 - k)$, alarm at the first $S_t > h$, then reset. Only the input series and μ_0 change here. [Course Book §13.7]

Artifact. `cusum_bank(instrument, gauge, t, s_upper, s_lower, alarm_flag, params_vintage)`.

Pass / fail. Simulated ARL_0 on the platform's own history clears the same one-false-alarm-per-instrument-year target Step 6.7 sets, per gauge; reused, not re-derived.

7.4 From regime to action: the router wiring

Step 7.5 — Wire the filtered probabilities into three gates

Do. Everything downstream consumes f_t (Step 7.3) or, for the model gate only, optionally the CUSUM bank's alarm flag (Step 7.4) — never a hard arg max label, which flickers at $f \approx 0.5$ and turns every flicker into a discrete jump in width and size [Course Book §13.8]. Three actions, each a fixed wiring rule reviewed and versioned before any live use; none is itself searched in Step 7.7's grid except where stated.

(a) Width-model select. Blend a Chapter 6 engine per regime by the law of total variance, weighted by f_t:

Regime (K=2)	Width model	Rationale
calm	Step 6.3's per-instrument HAR-vs-GARCH selection, unmodified	ample calm-regime data; no reason to specialise
stress	dedicated GARCH(1,1) refit on Viterbi-dated stress bars only (offline dating, the permitted use of [Course Book §13.5])	stress dynamics differ enough to earn a dedicated fit once enough stress episodes exist

K=3 adds a transitional row between them: width model = the calm-fit engine, with Step 6.6's ACI level adaptation left to absorb the difference. No separate transitional fit — a third dedicated fit would spend episode budget the ∼64-cycle count (Step 7.2) does not have.
Blend variances, never quantiles directly: form $\hat{\sigma}_t^2$ below, map it to quantiles through a single innovation shape (or invert the mixture CDF numerically), and let Step 6.5's conformal layer absorb the remainder, exactly as [Course Book §13.8] requires — averaging the per-regime *quantiles* instead delivers a band with the wrong coverage.

(b) Size throttle. A continuous throttle on the position multiplier, never a switch [Course Book §13.8]. Recommended piecewise-linear cap:

f_{stress} range	Multiplier cap
$[0, 0.30)$	$1.00\times$ (no throttle)
$[0.30, 0.60)$	linear taper, $1.00\times \to 0.35\times$
$[0.60, 1.00]$	floor at $0.35\times$

One throttle curve, fixed, not searched — searching its breakpoints would add a fourth multiplicity axis (Step 7.7) the already-charged K and τ_{gate} dimensions should not also have to absorb. The breakpoints are a judgement call, not a computation: 0.30/0.60 place the taper where a filter can actually resolve a posterior (a decisive single bar routinely moves f_{stress} past 0.60 in one step, per Worked Example 13.2), and the $0.35\times$ floor matches the platform's existing `width_mult` floor, so a stress throttle never sizes more aggressively than the frozen multiplier it replaces. This multiplies into Chapter 14's position formula; no new sizing formula is stated here.

(c) Model gate. Route the forecaster by regime, the Complexity Router pattern (arXiv 2605.24381, as cited in the programme brief), inverted from the intuitive direction on purpose [Course Book §13.8]:

f_{stress}	Forecaster consulted	Reason
$< \tau_{\text{gate}}$	expensive (the Stage-1 adaptive band; the Stage-3 candidate once one exists)	cost floor low enough that a small edge clears it
$\geq \tau_{\text{gate}}$	cheap (frozen `width_mult` or a naive band)	spreads are widest exactly where the expensive model's marginal accuracy is least likely to survive cost

$\tau_{\text{gate}} \in \{0.55, 0.65\}$ is the one dimension of this wiring Step 7.7 actually searches, because it is the one dimension [Course Book §13.8]'s own kill-criterion discussion names as trial-worthy; the throttle curve (b) and the width assignment (a) are fixed by design so the ledger's multiplicity bill stays at Step 7.7's stated six rows.

The mathematics.

$$\hat{\sigma}_t^2 = \sum_{i=1}^{K} f_{t,i}\, \hat{\sigma}_{i,t}^2,$$

the law of total variance with the cross term dropped because each per-regime width engine is itself a mean-near-zero fit (Chapter 6's own convention), so $\mathrm{Var}(\mathbb{E}[r \mid S]) \to 0$. [Course Book §13.8]

Artifact. `router_config(config_id, K, tau_gate, throttle_curve_id, width_table_id, is_active, reviewed_by, valid_from)`.

Pass / fail. Reviewed against Step 7.7's grid before any row is marked `is_active`; a configuration never goes live purely because it is the default — every activation is itself a row in the trial ledger (Step 4.4's write-before-inspection discipline).

7.5 The Chronos-2 covariate probe

Step 7.6 — Probe the covariates with Chronos-2, in context

Do. *Licence and runtime check, before anything else runs.* Confirm the exact checkpoint's `LICENSE` file — not the model card, not a summary — is Apache-2.0, and that inference runs locally, offline, on the pinned 3060, with zero network egress during the run [Course Book §9.4]. Chronos-2 clears this by canon (Apache-2.0, in-context covariates), but the check is per checkpoint and per download, not per model family: re-verify on every checkpoint refresh, and log the SPDX identifier and the file's SHA-256 alongside the run, exactly as [Course Book §9.4]'s three-count screen (licence, delivery mechanism, audit trail) demands before a single accuracy number is read.

Batch inference, offline: universe = Step 5.1's regenerated-history universe restricted to {M30, H1, H4} across all 21 instruments = 63 series (`verify_pb07.py`); context length matches the live per-(instrument, granularity) setting Step 5.1 already captured in `gen_config` — reuse that stamp, do not re-derive context parity here. Supply the covariates *in context*, as additional token streams alongside the target series: the same z_t vector Step 7.1 built, already point-in-time by construction, so the probe inherits PIT safety rather than re-earning it [Course Book §9.3]. Sample $S = 100$ paths per series — up from a naive $S = 20$: Monte Carlo error in a sample quantile scales as $S^{-1/2}$, and 100 paths buys a 2.24× reduction over 20 for a probe whose whole point is a clean read on the 80% band (`verify_pb07.py` carries the arithmetic; the count itself is a recommendation, not a derivation). No training run: zero gradient steps, in-context only, exactly as the probe promises [Course Book §9.3].

Run two covariate-set variants, matching Step 7.7's family-2 grid: a univariate control (target

series only, no covariate tokens) and the full z_t covariate set. The control isolates "does Chronos-2 covariate-conditioning help" from "is a second foundation model's opinion worth anything on its own" — without it, a positive result cannot be attributed to the covariates at all.

The mathematics. None new. Chronos-2's quantiles are empirical sample quantiles of the S decoded paths, exactly as [Course Book §9.3] describes; this step supplies covariates as extra context and reads off the same sampling procedure, nothing about which is restated or altered here.

Artifact. `chronos2_probe(instrument, granularity, t, median, q10, q20, q80, q90, covariate_set_id, context_len, n_paths, generation_config_hash)` — schema parallel to `timesfm.sqlite`, per canon.

Pass / fail. Licence SPDX identifier recorded as Apache-2.0 and file hash matches the shipped checkpoint; zero outbound network connections logged for the run's duration; row count equals the expected 63-series × bar count, reconciled against Step 5.1's own candle-gap ledger for the same universe.

7.6 Incremental evaluation and the read-out

Step 7.7 — Run the incremental evaluation against the Stage-1 baseline

Do. Baseline: the Stage-1 adaptive band as accepted out of shadow (Step 6.9–6.10), run through Chapter 4's harness unmodified — a comparator with its own existing ledger rows, not a new trial.

Candidate family 1, router throttle: Step 7.5's wiring (width blend (a) + size throttle (b) + model gate (c)) applied on top of the baseline band, one candidate per (K, τ_{gate}) pair, throttle curve (b) held fixed. Grid $= \{2,3\} \times \{0.55, 0.65\} = 4$ configurations (`verify_pb07.py`). Gauge subsets are not searched: $2^4 - 1 = 15$ non-empty subsets of z_t's four gauges is a combinatorial dimension the ~64-episode budget (Step 7.2) already cannot support at the full vector, so the vector is fixed at all four gauges by design, not tuned [Course Book §13.8].

Candidate family 2, probe quantiles: the `chronos2_probe` band evaluated standalone — not blended with the baseline, since there is no regime weight to blend it by — normalised to the baseline's mean |exposure| over the train split only, reusing Step 5.4's exposure-matching convention by reference. Grid $=$ {univariate control, full z_t} $= 2$ configurations (`verify_pb07.py`). The three timeframes of Step 7.6's probe universe are pooled into one combined CPCV deflation pass per covariate-set variant, rather than charged as three separate per-timeframe trials — a deliberate pooling choice recorded in the ledger, kept small because these are meant to stay cheap probes, not the Stage-3 gauntlet.

Total new ledger rows this stage: $4 + 2 = 6$ (`verify_pb07.py`). Note explicitly: BIC (Step 7.2) picks the better-*fitting* K per instrument; this evaluation nonetheless tests both $K \in \{2,3\}$ for P&L regardless of BIC's preference, because a higher-likelihood fit is not guaranteed to be the higher-P&L one, and [Course Book §13.8]'s own kill-criterion discussion already prices every K tried into the ledger.

Run: Chapter 4's harness in full — engine contract (Step 4.1), CPCV splitter (Step 4.2), metrics module (Step 4.3) — referenced, not restated. Every one of the six configurations writes its ledger row before results are inspected (Step 4.4), abandoned or not.

Deflate: trial variance V over the *cumulative* ledger to date — Stage 0's grid, Stage 1's shadow-acceptance runs, and these six, not this stage's six in isolation — and SR_0, DSR by Step 4.3's already-stated formulas, referenced, not restated.

The mathematics. None new: this step consumes Chapter 4's engine, splitter and DSR formulas exactly as declared there.

Artifact. Six new trial-ledger rows and their metrics JSON; the deflated DSR table for the cumulative ledger including these six.

Pass / fail. None at this step; it produces the numbers Step 7.8 reads.

Step 7.8 — Read the results and record the decision

Do. Apply Step 5.7's three-verdict table to each of the two winning candidates (the best of family 1's four configurations, the best of family 2's two), each read against the Stage-1 baseline on the *same* cumulative deflated ledger:

Verdict	Condition	Action
Clear pass	deflated cost-net P&L strictly exceeds the baseline **and** per-regime coverage not degraded	proceed to the gatebox; Stage 3 opens on this evidence
Marginal	either condition holds only within noise, or one family passes weakly while the other does not	treated as kill, below
Clear kill	neither family clears the baseline, or coverage degrades in any qualifying ($n \geq 100$) bucket	Stage 3 cancelled, killbox below

Marginal is kill, restated, not re-argued: marginal results under multiplicity are the expected face of noise, not a near-miss [Course Book §15.4-15.5]; and a marginal Stage-2 result additionally carries Stage 3's entire premise on its back, which raises rather than lowers the bar for calling anything short of a clear pass a pass.
Kill-memo template: reuse Step 5.8's template verbatim, substituting data = `regime_filtered` + `chronos2_probe`, class = the six-row grid of Step 7.7, benchmark = the Stage-1 baseline, and the kill wording of the killbox below.

Artifact. Signed decision memo.

Pass / fail. Not applicable — this step produces the stage's pass/fail, consumed by the gatebox below.

Stage 2's gate, kill wording and artifact register close the chapter; fixed now, before the six rows above are read, not after.

Stage gate — Deflated P&L beats the Stage-1 baseline, coverage holds

Stage 2 passes iff at least one of the two candidate families (Step 7.7) shows a clear pass on Step 7.8's table: deflated, cost-net P&L strictly exceeds the Stage-1 adaptive baseline on the *same* cumulative ledger, **and** per-regime coverage (the Christoffersen battery, Step 6.10's stress-bucket $n \geq 100$ rule reused) is not degraded relative to that baseline in any qualifying bucket. Both conditions must hold on the *same* winning configuration: a configuration that wins on P&L while degrading coverage in a qualifying bucket does not pass. Comparison is against the Stage-1 baseline only — never against the Stage-0 static `width_mult`, too weak a comparator to license anything [Course Book §13.8].

Kill criterion — Stage 3 is cancelled — its premise is this gate

Marginal or clear-kill on both families: the macro covariates and the fitted regime state carry no tradeable conditional signal at these horizons, net of cost, at the multiplicity this ledger has now priced in. Stage 3 is cancelled — its premise, that a covariate-typed bespoke transformer will find what a two-probe screen could not, is exactly the gate this stage exists to test, and a kill here is a kill of that premise, not of the covariates' existence in principle. Wording to sign: "[date], [owner]: Stage 2's incremental evaluation shows no configuration in the pre-registered six-row grid (Step 7.7) clears the Stage-1 baseline on deflated cost-net P&L with non-degraded per-regime coverage. Per the pre-committed rule, marginal results are treated as kills. Stage 3, the bespoke covariate-typed model, is cancelled on this evidence. The covariate feeds and the filtered-regime series remain in the artifact register for any future re-test, which re-enters the ledger as new trials, not a continuation of this one." No renegotiation of this wording after results are seen.

Artifacts — Stage output register

- `regime_obs` — the causal, point-in-time observable vector (Step 7.1).
- `hmm_fit` (840 rows, the full seed grid) and `hmm_params` — BIC and the selected fit per instrument and K (Step 7.2).
- `regime_filtered` as-of series and its refit log — the only regime quantity any downstream decision may read (Step 7.3).
- `cusum_bank` — the per-gauge alternative and corroboration detectors (Step 7.4).
- `router_config` — the versioned width/throttle/gate wiring (Step 7.5).
- `chronos2_probe` and its licence and runtime attestation (Step 7.6).
- Six new trial-ledger rows and their metrics JSON, plus the updated cumulative DSR table (Step 7.7).
- The signed decision memo (Step 7.8).

8

Stage 3: The Bespoke Model

Stage 3 builds the one to ten million parameter distributional transformer the programme has been pointing at since the first page: a bespoke quantile model over the platform's own instruments, trained through a decision-focused objective with the measured cost surface inside it, and shipped as a frozen ONNX artifact that the C# engine consumes through a table drop. Nine steps take it from raw candles to a rollback protocol. Execute them in order.

Stage 3 has no premise of its own. Its premise is Stage 2's verdict — that at least one covariate carries conditional information, point-in-time, surviving CPCV and deflation — because the variate-attention layer is precisely the machinery for exploiting such information and, with nothing to exploit, it is pure overfitting surface [Course Book §8.5]. Do not open the editor until the Stage-2 memo is signed and its ledger rows are written.

Trap — Stage 3 is not authorised by enthusiasm

Three gates stand between here and the first line of `stage3_dataset.py`: Stage 0's envelope-P&L verdict, Stage 1's conditional-width acceptance, Stage 2's covariate probe. A Stage-2 kill cancels Stage 3 outright — that kill's wording says so. Building the model anyway, "to see", produces a result that no deflation can rescue, because the trial that produced it was never in the ledger.

8.1 The dataset and the model

The first two steps are construction, not research. Everything they produce is hashed, versioned, and referenced by every ledger row that follows; a change to either is a new configuration and a new trial, not an edit.

Step 8.1 — Data pipeline: tensors, normalisation, typed covariates

Do. Write `stage3_dataset.py`, a single builder producing sharded, immutable tensors. It reads `candles_3y` (mid), the `feeds` table and registry from pb03, and `cost_surface` from pb02; it writes nothing back. Per training example — one (instrument, granularity, decision timestamp) triple — emit the layout of Table 8.1. Normalise each context window by its own first two moments (RevIN), learnable affine on, before patching; the statistics come from the context window and nothing else, which is what makes the normalisation causal. Patch the normalised window with length $P = 32$ and stride $S = 32$ over context $L = 1536$, giving $N = 48$ non-overlapping patch tokens per series. Append $C = 8$ covariate tokens, one per slot in Table 8.2, for a token stream of $T = 56$. Every observed-past covariate is read through the pb03 as-of join and carries a live/stale mask bit; a stale feed is presented as missing with its mask cleared, never carried forward. The label is the realised forward log-return path

over the $H = 50$ bar horizon in RevIN units, plus three path summaries (terminal cumulative return, maximum favourable excursion, maximum adverse excursion) that the decision term of Step 8.3 consumes and the pinball term ignores. Materialise the round-trip cost κ for each example at build time by looking up `cost_surface` at the pinned vintage by (instrument, session bucket, news flag, rollover flag): no cost lookup happens inside the training loop, so no future cell can be reached from it. Attach a sample-uniqueness weight to every example. Split by contiguous time into the pb04 CPCV groups before writing, and shard within split so that no shard straddles a group boundary.

The mathematics. RevIN, per variate v over its own context window $\boldsymbol{x}^{(v)} \in \mathbb{R}^L$, with learnable $\gamma_v \neq 0$, β_v and guard $\varepsilon > 0$:

$$\mu_v = \frac{1}{L}\sum_{i=1}^{L} x_i^{(v)}, \qquad \sigma_v^2 = \frac{1}{L}\sum_{i=1}^{L}\left(x_i^{(v)} - \mu_v\right)^2, \qquad x_i'^{(v)} = \gamma_v \frac{x_i^{(v)} - \mu_v}{\sqrt{\sigma_v^2 + \varepsilon}} + \beta_v, \tag{8.1}$$

inverted after the head by $\hat{y}_j^{(v)} = \sqrt{\sigma_v^2 + \varepsilon}\,(\hat{y}_j'^{(v)} - \beta_v)/\gamma_v + \mu_v$. [Course Book §8.1] Patch count, with context L, patch length P, stride S:

$$N = \left\lfloor \frac{L - P}{S} \right\rfloor + 1 \;=\; \left\lfloor \frac{1536 - 32}{32} \right\rfloor + 1 = 48. \tag{8.2}$$

[Course Book §8.2] Sample uniqueness, with Λ_i the label span of example i and c_t the number of label spans covering bar t:

$$c_t = \sum_i \mathbf{1}\{t \in \Lambda_i\}, \qquad u_i = \frac{1}{|\Lambda_i|}\sum_{t \in \Lambda_i} \frac{1}{c_t} \in (0, 1], \qquad \tilde{u}_i = \frac{u_i}{\sum_j u_j}. \tag{8.3}$$

$\tilde{u}_i$ is the training weight. [Course Book §16.5] The as-of predicate every observed-past covariate must satisfy by construction: a value is usable at t if and only if `release_ts` $\leq t -$ safety lag. [Course Book §18.2]

Artifact. `/data/stage3/{train,val,test}/shard-*.npz` plus `manifest.json` recording builder git SHA, cost vintage, feed vintages, split configuration, per-shard row counts and the `config_hash` of the canonical JSON of all of it.

Pass / fail. Four checks, all required. (a) Shape and dtype assertions on every shard against Table 8.1. (b) The pb03 S8 audit query run over 10,000 sampled examples returns zero rows in which any covariate's `release_ts` exceeds its decision timestamp minus the safety lag. (c) Twenty randomly chosen windows reconstructed from raw candles reproduce the stored tensor to 10^{-12} absolute, including the RevIN statistics. (d) No training label span intersects any test group after purge and embargo — the pb04 S2 splitter unit test, re-run against these shards rather than against synthetic data.

The dataset's size is the first hard fact of this stage, and it is smaller than it looks. Over three years, across the 21 instruments at M30, H1 and H4, stride-1 sampling yields 1,177,785 windows: 752,955 at M30, 359,835 at H1, 64,995 at H4 (H4 loses most of its history to the 1536-bar context). Because every label spans $H = 50$ bars and consecutive windows overlap in 49 of them, (8.3) gives $u_i = 1/50$ for interior examples and the effective count of independent labels is $1{,}177{,}785/50 = 23{,}555.7$. Against the 3,765,754 parameters of Step 8.2 that is 159.87 parameters per effectively independent observation. Every regularisation decision below follows from that ratio, and so does the curriculum in Step 8.3.

Table 8.1. Stage-3 training example. B is the batch axis; all other axes are static.

Tensor	Shape	Dtype	Contents
`x`	$(B, 48, 32)$	float32	RevIN-normalised log mid-close, patched
`stat`	$(B, 2)$	int64	instrument id $\in [0, 21)$, granularity id $\in [0, 7)$
`known`	$(B, 3)$	float32	known-future covariates, Table 8.2
`obs`	$(B, 5)$	float32	observed-past covariates, as-of joined
`mask`	$(B, 8)$	bool	1 where the covariate is live at the decision bar
`y`	$(B, 50)$	float32	realised forward log-return path, RevIN units
`ysum`	$(B, 3)$	float32	terminal return, max favourable, max adverse excursion
`kappa`	$(B,)$	float32	round-trip cost at the pinned vintage, price units
`w`	$(B,)$	float32	normalised uniqueness weight $\tilde{u}_i$
`meta`	$(B, 4)$	int64	instrument, granularity, decision ts, label-end ts

Table 8.2. Covariate typing. The type declares which connections the graph may contain; a misdeclaration is look-ahead expressed as architecture. [Course Book §8.4]

Type	Field	Source	Availability rule
static	instrument id	canon, 21 CFDs	embedding, no time index
static	granularity id	canon, 7 timeframes	embedding, no time index
known-future	session bucket	pb02 S3	deterministic calendar
known-future	day-of-week, day-of-month	calendar	deterministic calendar
known-future	minutes to next tier-1 release	pb02 S4	schedule known; value not
observed-past	VolRegime, as-of	pb03 S5	`release_ts` $\leq t - \ell$
observed-past	RORO, as-of	pb03 S4	`release_ts` $\leq t - \ell$
observed-past	COT net-position z	pb03 S2	Friday 15:30 ET release
observed-past	2y/10y differential	pb03 S6	vendor release timestamp
observed-past	realised-vol fast/slow ratio	pb06 S1	bar-close derived

Step 8.2 — Architecture: one config file, one parameter budget

Do. Write `stage3.yaml`, reproduced in full below, and make it the only source of architectural truth: the builder, the trainer, the exporter and the ledger all read it, and its canonical-JSON hash is the `config_hash` on every ledger row. Width $d = 256$, depth $n_l = 4$, heads $n_h = 4$, pre-LN blocks, feed-forward expansion 4. Patch embedding is one shared linear map applied to every patch of every instrument — channel independence above the variate layer, so the trunk cannot learn instrument-specific shortcuts. Exactly *one* variate-attention block, positioned after patch and covariate embedding and before the shared trunk: it is the only place in the graph where cross-sectional information may move, and its position is a fixed architectural decision, not a hyperparameter. Moving it is a different architecture and a different ledger row [Course Book §8.3]. The head emits five quantiles at each of $H = 50$ horizon steps through a monotone softplus-increment construction, so crossing is impossible by construction rather than repaired by sorting afterwards. Instantiate instrument and granularity embeddings; leave asset-class embeddings out of v1 and add them at v2 as the cold-start hook [Course Book §19.4].

The mathematics. Monotone quantile head, for levels $\tau_1 < \cdots < \tau_K$, base output $b(\boldsymbol{x})$ and increment logits $\delta_j(\boldsymbol{x})$:

$$\hat{q}_{\tau_1}(\boldsymbol{x}) = b(\boldsymbol{x}), \qquad \hat{q}_{\tau_{j+1}}(\boldsymbol{x}) = \hat{q}_{\tau_j}(\boldsymbol{x}) + \varsigma\big(\delta_j(\boldsymbol{x})\big), \qquad \varsigma(z) = \ln\big(1 + e^z\big), \tag{8.4}$$

ς the softplus, strictly positive, so $\hat{q}_{\tau_1} \le \cdots \le \hat{q}_{\tau_K}$ holds for every input. [Course Book §10.2] Parameter budget, with $12d^2$ per pre-LN block (four $d \times d$ attention projections plus a feed-forward of expansion four) and $13d$ of biases and LayerNorm gains:

$$N_{\text{block}} = 12d^2 + 13d, \qquad N_{\text{core}} = n_l\big(12d^2 + 13d\big) \;\approx\; 12\,d^2 n_l. \tag{8.5}$$

[Course Book §6.5]

Artifact. `stage3.yaml`; `config_hash` written to the ledger before the first optimiser step.

Pass / fail. The trainer prints `sum(p.numel() for p in model.parameters())` at construction and asserts it equals 3,765,754; any mismatch means the config file and the code have diverged and the run aborts. A forward pass on one batch asserts monotone outputs at every one of the $B \times 50$ (example, horizon) pairs.

```
# stage3.yaml -- the only source of architectural truth
model:
  d_model: 256
  n_layers: 4
  n_heads: 4
  d_head: 64
  ffn_expansion: 4
  norm: pre_LN
  dropout: 0.10
  patch:      {length: 32, stride: 32, context: 1536, n_tokens: 48}
  covariates: {n_known_future: 3, n_observed_past: 5, n_tokens: 8}
  variate_attention: {n_blocks: 1, position: after_embed_before_trunk}
  variable_selection: {n_grn: 2, hidden: 256}
  revin: {learnable_affine: true, eps: 1.0e-5}
  head:
    type: monotone_quantile_softplus
    levels: [0.10, 0.20, 0.50, 0.80, 0.90]
```

```
    horizon: 50
  embeddings: {instrument: 21, granularity: 7}
data:
  universe:      [all_21]
  granularities: [M30, H1, H4]
  history:       3y
  cost_vintage:  PINNED_AT_RUN_TIME
loss:
  pinball_weight: 1.0
  lambda_pnl:     0.25
  lambda_cvar:    0.20
  cvar_alpha:     0.95
  tanh_k:         2.0
  w_max:          1.0
  smooth_abs_eps: 0.01
  curriculum: {warmup_epochs: 20, ramp_epochs: 15, blend_epochs: 25}
optim:
  name: adamw
  lr: 3.0e-4
  betas: [0.9, 0.95]
  weight_decay: 0.05
  no_decay: [layernorm, bias, embedding]
  warmup_steps: 2000
  schedule: cosine_to_3.0e-5
  grad_clip_norm: 1.0
  batch_size: 256
  seeds: [11, 23, 37]
```

Table 8.3. Parameter budget at $d = 256$, $n_l = 4$, $P = 32$, $K = 5$, $H = 50$. Counts are exact; the rule $12d^2 n_l = 3{,}145{,}728$ understates the core by the $13dn_l = 13{,}312$ bias-and-gain term.

Module	Count	Share
Multi-head attention per block, $4d^2 + 4d$	263,168	
Feed-forward per block, $8d^2 + 5d$	525,568	
Two LayerNorms per block, $4d$	1,024	
One block, $12d^2 + 13d$	789,760	
Four blocks (core)	3,159,040	83.89%
Patch embedding, $Pd + d$	8,448	
Instrument embedding, $21d$	5,376	
Granularity embedding, $7d$	1,792	
Covariate slot embeddings, $8d$	2,048	
Two variable-selection GRNs, $2 \cdot 4d^2$	524,288	
Quantile head, $d \cdot KH + KH$	64,250	
Final LayerNorm, $2d$	512	
Extras	606,714	16.11%
Total as configured	**3,765,754**	100%

Two reconciliations against the course book, both required before the number is trusted. Strip the covariate machinery — the two variable-selection networks and the instrument, granularity and covariate embeddings — and the remainder is 3,232,250, exactly the trunk-only budget [Course Book §6.5]. Add the machinery back and the total exceeds the blueprint's 3,747,834 [Course Book

§8.5] by 17,920 parameters, or 0.478%: 13,312 from the $13dn_l$ term the approximation rule drops, 4,096 because this configuration patches at $P = 32$ rather than 16, and 512 for a final LayerNorm the blueprint did not count. Both agreements are exact where they should be exact, and the discrepancy is fully attributed. At 3.77M the model sits inside the declared one-to-ten-million envelope and is 53.1 times smaller than the 200M TimesFM it must beat.

Memory is not the constraint and should be recorded once so that nobody re-litigates it. In fp32 the weights are 15.06 MB; with AdamW's two moment buffers, 45.19 MB; with gradients as well, 60.25 MB, which is 0.468% of the 3060's 12 GiB. Add activations at batch 256 — sixteen tensors of shape (B, T, d) retained per block, 939.52 MB, plus 51.38 MB of attention logits — and the whole training footprint is 1,051.16 MB, or 8.16% of the card. What binds is time, and Step 8.4 prices it.

8.2 The objective and the run

Step 8.3 — The blended objective, with the cost surface inside it

Do. Implement one loss function with three terms and a curriculum that switches them on. Term one is the uniqueness-weighted average pinball over all five levels and all fifty horizon steps — dense supervision, 294,446,250 scalar targets over the training set. Term two is the negative net P&L of a smooth sizing rule applied to the model's own band, simulated bar by bar with the materialised cost surface charged on every change of position. Term three is a CVaR penalty on the same P&L, in Rockafellar–Uryasev sample form, which adds exactly one scalar parameter to the optimisation. The signal fed to the sizing rule is the model's own output and nothing else: $s_t = \hat{q}^{\text{cum}}_{50} / \left[\tfrac{1}{2}(\hat{q}^{\text{cum}}_{80} - \hat{q}^{\text{cum}}_{20})\right]$, the cumulative-horizon median divided by half the 20–80 band width, unit-free by construction. Charge turnover with the smoothed absolute value so the gradient survives at the no-trade boundary. Curriculum: epochs 1–20 pinball only at stride 1; epochs 21–35 ramp λ_1 linearly from zero to target at stride 50; epochs 36–60 blended at stride 50, λ_1, λ_2 constant.

The mathematics. The objective, minimised jointly over parameters θ and the CVaR auxiliary c, on a batch $\mathcal{B}$ of examples with weights $\tilde{u}_i$, quantile levels $\tau_1 < \cdots < \tau_K$, horizon steps $j = 1, \ldots, H$, realised path values y_{ij}, predictions $\hat{q}_{ijk}$, simulated bar P&L $\mathrm{PnL}_i(\theta)$ and tail level a:

$$\begin{aligned}\mathcal{L}(\theta, c) \;=\; & \underbrace{\frac{\sum_i \tilde{u}_i \sum_{j,k} \rho_{\tau_k}\!\left(y_{ij} - \hat{q}_{ijk}\right)}{KH \sum_i \tilde{u}_i}}_{\text{pinball}} \;-\; \lambda_1 \frac{1}{|\mathcal{B}|} \sum_i \mathrm{PnL}_i(\theta) \\ & +\; \lambda_2 \left(c + \frac{1}{|\mathcal{B}|(1-a)} \sum_i \left(-\mathrm{PnL}_i(\theta) - c\right)_+ \right),\end{aligned} \tag{8.6}$$

with the pinball loss $\rho_\tau(u) = (\tau - \mathbf{1}\{u < 0\})u$ [Course Book §10.1] and the third term the Rockafellar–Uryasev sample representation of $\mathrm{CVaR}_a(-\mathrm{PnL})$, whose minimising c is the corresponding VaR_a [Course Book §14.5]. The differentiable decision inside PnL, with $w_{\max}$ the exposure cap, $k > 0$ the ramp sharpness, κ the round-trip cost of the example's cost cell and r the realised return:

$$w_t = w_{\max} \tanh\!\left(k\, s_t\right), \qquad \mathrm{PnL}_t = w_{t-1} r_t - \kappa\, \psi_\varepsilon\!\left(w_t - w_{t-1}\right), \qquad \psi_\varepsilon(x) = \sqrt{x^2 + \varepsilon^2} - \varepsilon. \tag{8.7}$$

[Course Book §14.6]

Artifact. `stage3_loss.py`; a per-epoch diagnostics CSV recording the two gradient norms $\|\nabla \mathcal{L}_{\text{pin}}\|$ and $\|\nabla \mathcal{L}_{\text{dec}}\|$, the fitted c, realised turnover and the fraction of examples in the saturated region $|k s_t| > 3$.

Pass / fail. (a) With $\lambda_1 = \lambda_2 = 0$ the loss reproduces the standalone pinball value to 10^{-9}. (b) With costs set to zero and the ramp made hard, the simulated P&L reproduces the pb04 backtest engine's P&L on the same 1,000 windows to 10^{-6} — the training simulator and the evaluation engine must be the same accountant. (c) The fitted c converges to within 2% of the empirical $\mathrm{VaR}_{0.95}$ of the batch loss distribution. (d) No term in PnL_i reads a field with a timestamp after the example's decision bar; assert it in code, not in review.

Five parameter recommendations, each a judgement rather than a computation. $\lambda_1 \in [0.1, 0.5]$, chosen at the end of the ramp so the two gradient norms sit within one order of magnitude — larger and the decision term drowns the dense supervision it depends on. $\lambda_2 \in [0.1, 0.3]$: enough to suppress the tail-seeking behaviour a pure mean objective rewards, not enough to collapse the policy to flat. $a = 0.95$, matching the tail level the sizing layer already reports. $k = 2$: $\tanh(1) = 0.7616$ is too soft to resemble the deployed threshold and $\tanh(3) = 0.9951$ is too near a step to pass gradient, so 2 is the default and $k \in \{1, 2, 4\}$ is the search axis. $\varepsilon = 0.01$, which understates a turnover of 0.50 as 0.4901 — well below the smallest rebalance worth charging for.

Trap — The decision term has four orders of magnitude less supervision

The pinball term sees 294,446,250 scalar targets. A rule firing on 2.5% of bars generates 29,444.6 trades, a ratio of 10,000 to 1. That is why the curriculum is not optional: the P&L objective must adjust a distribution the dense loss has already fixed, not discover one from scratch, or a 3.77M-parameter model will memorise thirty thousand trades outright and produce a superb backtest attached to a losing book [Course Book §14.6]. It is also why a P&L loss is an invitation to leakage of the worst kind: the model does not merely benefit from a leak, it is trained to find and exploit one. Gauntlet run 6 exists for this.

Step 8.4 — Training protocol, and which card it runs on

Do. AdamW, learning rate 3×10^{-4}, $\beta = (0.9, 0.95)$, weight decay 0.05 with no decay on LayerNorm gains, biases or embeddings, gradient-norm clip 1.0, linear warmup over 2,000 steps then cosine decay to 3×10^{-5}; dropout 0.10. The formulas are not restated here [Course Book §3.6], [Course Book §3.8]. Batch 256 windows drawn as a *cross-sectional block*: sample one contiguous temporal block inside one CPCV training group, then take all 21 instruments within it, so every batch contains the cross-section the variate-attention layer exists to read. A block never straddles a group boundary and never crosses a purge or embargo band. Early-stop on validation uniqueness-weighted pinball with patience 5 epochs and relative minimum delta 10^{-4}, restoring the best checkpoint; never early-stop on the decision term, whose effective sample is thirty thousand noisy scalars. Pin `PYTHONHASHSEED`, `numpy`, `torch` and CUDA seeds to $\{11, 23, 37\}$, enable deterministic kernels, and run three seeds per configuration. Metrics are *averaged within* a configuration: seeds are not a selection axis, so the ledger's trial count counts configurations, not runs. Record the environment — CUDA driver and runtime, cuDNN, torch, onnxruntime, numpy, Python, GPU UUID and the `CUDA_VISIBLE_DEVICES` pin — into `env.lock` and hash it into `config_hash`.

The mathematics. Cost per run, from the forward FLOP count of Step 8.7 and a backward pass costing twice the forward:

$$F_{\text{step}} = 3\, F_{\text{series}}\, B, \qquad T_{\text{run}} = \frac{F_{\text{step}}}{\Phi}\Big(E_{\text{warm}} S_1 + E_{\text{blend}} S_{50}\Big), \tag{8.8}$$

F_{series} the forward FLOPs of one series, B the batch, Φ the sustained throughput, E epoch counts and S steps per epoch at the stated stride. Search feasibility is the same identity read

the other way: with n configurations, m seeds and W usable window-hours per night, the search costs $n\,m\,T_{\text{run}}/W$ nights.

Artifact. `checkpoints/<config_hash>/seed-{11,23,37}/best.pt`, `env.lock`, per-run training and validation curves, and the GPU contention log.

Pass / fail. `env.lock` present and hashed into the ledger row; three seeds completed; the three seeds' best validation pinball within a declared spread (recommend 2% relative — wider means the configuration is unstable and must not be selected on); and the contention log showing an empty inference queue at every training step. A single non-empty sample fails the run, because a run that stole the live card is a run whose latency evidence is worthless.

At $B = 256$ a step costs 281.15 GFLOP; at an assumed sustained 4.0 TFLOP/s on the pinned 3060 — an assumption to be replaced by a measurement in the first hour — that is 70.29 ms. The warmup phase runs stride-1 over all 1,177,785 windows, 4,601 steps per epoch, 92,020 steps in twenty epochs, 107.80 minutes. The blended phase runs stride-50 over 23,556 windows, 93 steps per epoch, 3,720 steps in forty epochs, 4.36 minutes — cheap precisely because the decision-focused sample is small. One full run is 1.869 hours, or 112.2 minutes.

That number, multiplied out, decides the shape of the search, and the two GPU paths give two different answers.

Table 8.4. Where Stage 3 trains. Path A is the plan of record. Path B changes the trial count by a factor of three and a half and must be recorded in the ledger as a search-budget field, not discovered afterwards.

	Path A: pinned RTX 3060	Path B: RTX 5080 (unresolved)
Availability	maintenance window only, 6 h/night	dedicated, continuous
Contention rule	trainer acquires the GPU only when the inference queue is empty; MPS priority cap; killed, not throttled	none: live inference stays on the 3060
Sustained Φ	4.0 TFLOP/s (measure)	3.5× the 3060 (measure)
One run	1.869 h	0.534 h
Unconstrained grid	216 configs = 1,211.3 h = 201.9 nights	346.1 h = 14.42 days
Declared search	32 configs = 179.4 h = 29.9 nights	51.3 h continuous
Configs in 30 nights	32	112
Retrain cadence	quarterly	monthly feasible

The unconstrained grid — $d \in \{192, 256, 384\}$, $n_l \in \{3, 4, 6\}$, $\lambda_1 \in \{0, 0.25, 0.5, 1.0\}$, $k \in \{1, 2, 4\}$, learning rate $\in \{10^{-4}, 3 \times 10^{-4}\}$, so $3 \cdot 3 \cdot 4 \cdot 3 \cdot 2 = 216$ configurations at three seeds each — is 648 runs and 1,211.3 GPU-hours. On Path A that is 201.9 maintenance nights. Refuse it. A thirty-night budget at six hours a night buys 32 configurations, so the declared search is staged: a coarse tier of 24 configurations varying d, n_l and λ_1, then a fine tier of 8 around the coarse winner varying k and the learning rate. 32 is then the trial count N that the deflation of Step 8.5 must absorb, and it is written into the ledger before the first run, not counted afterwards. On Path B the same search costs 51.3 continuous hours and the same thirty nights would buy 112 configurations — a larger, better search whose deflation bar is correspondingly higher. Neither path is free; the 5080 buys resolution, not absolution.

Trap — Do not split a search across two cards

A coarse tier on the 5080 and a fine tier on the 3060 produces two populations with different numerics, and the in-sample/out-of-sample rank pairs that PBO consumes are then not comparable. If the 5080 is resolved mid-programme, finish the current search on the card it started on and record the switch as a new search-budget field on the next one. Before any

switch, run the Step 8.6 parity test across both cards and require the same tolerances.

8.3 The gauntlet, the freeze, and the live path

Nothing below is optional and nothing below is reordered. The gauntlet is six named runs executed in sequence; each writes its ledger row before the next begins, and a failure at any run stops the sequence there. Reading the runs out of order — in practice, reading the P&L run first and the baselines afterwards — is how a model that loses to a linear layer acquires a champion's paperwork.

Step 8.5 — The six-run evaluation gauntlet

Do. Execute Table 8.5 top to bottom against the frozen candidate configuration, through the pb04 harness, at the pinned cost vintage, on the CPCV configuration of pb04 S2: $N_g = 8$ groups, $k = 2$ test groups, purge 50 bars each side, per-granularity embargo. Every run writes a ledger row — `trial_id`, `config_hash`, `cost_vintage`, `split_config`, metrics JSON, status — *before* its numbers are read. Run 1 is a forecast-quality screen and can only disqualify; runs 2–4 are the economic and multiplicity tests; run 5 is calibration; run 6 is the licence. Run 6 sits last deliberately: it is the run that invalidates the other five if it fails, so it must execute against the exact artifact that produced them, not against an earlier build of the pipeline.

The mathematics. CPCV enumeration, stated once: with N_g groups and k test groups per split, the number of splits is $\binom{N_g}{k}$ and the number of complete backtest paths is

$$\frac{k\binom{N_g}{k}}{N_g} = \frac{2\binom{8}{2}}{8} = \frac{2 \cdot 28}{8} = 7. \tag{8.9}$$

[Course Book §16.4] DSR, PBO and the Christoffersen statistics are computed by the pb04 S3 metrics module and are not restated here [Course Book §15.4], [Course Book §15.5], [Course Book §17.2].

Artifact. `gauntlet/g{1..6}-<config_hash>/report.html` plus the raw per-path P&L series, and six ledger rows.

Pass / fail. Six rows in the ledger, in order, each carrying the same `config_hash` and `cost_vintage`. The pass condition of each run is in Table 8.5; a run that fails ends the stage and routes to the kill wording. A run whose report exists without a preceding ledger row is void, and the stage restarts.

Run 4 deserves one sentence of enforcement. PBO computed over survivors is not PBO; it is a tautology. The population it consumes is every configuration the ledger holds for this strategy family, including the ones abandoned after three epochs because the loss curve looked wrong. If those rows were never written, run 4 cannot be executed and the stage cannot pass — which is the entire reason pb04 S4 demands the row before the result is inspected.

Step 8.6 — Freeze: ONNX export, parity, hash

Do. Export the selected checkpoint to ONNX, opset 17, with a dynamic batch axis and every other axis static ($T = 56$, $d = 256$, $K = 5$, $H = 50$). Put the RevIN statistic computation *inside* the graph, so the consumer cannot compute μ_v, σ_v differently from the trainer; fold the RevIN affine and all inference-time constants; disable dropout; run ONNX Runtime graph optimisation at extended level and save the optimised graph as the artifact, not the raw

Table 8.5. The gauntlet, in execution order. Each row is a named run in the trial ledger.

#	Run	What it asks	Pass
1	`g1-baselines`	Does the model beat random walk, seasonal-naive, DLinear, the Stage-1 HAR/GARCH width and the incumbent TimesFM band on weighted pinball, CRPS and 80% coverage, per (instrument, granularity)? [Course Book §7.3] [Course Book §7.5]	Wins on a majority of cells, and beats DLinear
2	`g2-cpcv-pnl`	What is the cost-net policy P&L against the Stage-2 ensemble across the 7 CPCV paths, at the pinned cost vintage, with 1.5× and 2× cost stress reruns?	Median path beats the ensemble; q_{10} path > 0
3	`g3-dsr`	Does the excess survive the search? DSR computed with the ledger's trial count $N = 32$ and the realised skew and kurtosis of the path returns [Course Book §15.4]	DSR > 0.5 vs the Stage-2 ensemble
4	`g4-pbo`	Over the full search population — every configuration, run *and* abandoned — how often does the in-sample winner underperform out of sample? CSCV over $S = 16$ subsets [Course Book §15.5]	PBO ≤ 0.5
5	`g5-coverage`	Per-regime Christoffersen battery on the Stage-2 filtered regime labels, stress buckets with $n \geq 100$ [Course Book §17.2] [Course Book §17.4]	No LR_{cc} rejection at 5% where $n \geq 100$
6	`g6-leak`	All four pb04 controls — oracle injection, label shuffle, timestamp scramble, cost doubling — re-run against the Stage-3 builder, loss and exporter [Course Book §18.4]	All four green; oracle explodes, both nulls inside the noise band, Sharpe falls under doubled cost

export. Disable TF32 on the CUDA execution provider for the parity run. Build the parity set: 10,000 windows spanning all 21 instruments (476 each), covering every session bucket and both news-flag states, drawn from the test split only. Run torch (eval, fp32, CPU and CUDA) and ONNX Runtime (CUDA EP, fp32) over it and compare on all four tolerances of Table 8.6 — 2,500,000 quantile scalars. Compute `sha256` of the optimised `.onnx` bytes and stamp `model_version = s3-<YYYYMMDD>-<first 8 hex>`. Write the registry row. The file is immutable: a re-export is a new hash and a new version even if the weights are bit-identical.

Artifact. `models/s3-<YYYYMMDD>-<hash8>/model.onnx`, `parity_report.json`, `registry.sqlite` row carrying `model_version`, `config_hash`, `env.lock` hash, `cost_vintage` and the six gauntlet `trial_ids`.

Pass / fail. All four tolerances hold with zero exceptions. A breach is never fixed by widening the tolerance — find the operator. The three usual causes, in order of frequency: LayerNorm epsilon placed inside rather than outside the square root by a fused kernel; TF32 left enabled on the CUDA provider; and reduction order in the RevIN mean over the 1536-element window. At load time the serving sidecar recomputes the hash from the file on disk and refuses to serve on mismatch with the registry.

Step 8.7 — Integration: the table drop, the latency budget, the fallback chain

Do. Add a *new* table to `timesfm.sqlite`. Do not migrate an existing one: the table-drop contract is the only integration contract the platform has, and it means the C# engine's current `SELECT` statements are untouched and a rollback is a `DROP TABLE`. Schema `bespoke_predictions: instrument TEXT, granularity TEXT, ts INTEGER, model_version TEXT, q10 q20 q50 q80 q90 REAL, regime INTEGER, regime_prob REAL, size_mult REAL,`

Table 8.6. ONNX parity tolerances. All four are required; the third is the only one with economic meaning, and the fourth is the one that silently breaks a monotone head.

Quantity compared	Domain	Tolerance
Quantile outputs, RevIN units	2,500,000 scalars	$\max \lvert\Delta\rvert \le 10^{-5}$
Quantile outputs, after inverse	price units	$\max \lvert\Delta\rvert \le 10^{-3}$ pip
Size multiplier through the live rule	10,000 decisions	$\max \lvert\Delta\rvert \le 10^{-4}$
Monotonicity $q_{10} \le \cdots \le q_{90}$	500,000 (window, step) pairs	zero violations, both runtimes

`coverage_stat REAL, latency_ms REAL, cost_vintage TEXT, written_ts INTEGER`, primary key (`instrument, granularity, ts`). The C# engine selects the latest row per (instrument, granularity) and accepts it only if `model_version` equals the pinned version, `written_ts` is inside the granularity's staleness bound, the five quantiles are monotone and `coverage_stat` is inside its band; otherwise it descends the fallback chain of Figure 8.1. Refresh all 21 instruments of a granularity in one batched forward pass, never in a loop: at batch one the kernel is memory-bandwidth-bound so instruments 2–21 are nearly free, and a staggered refresh puts a timing skew between the signal and the cost surface it is traded against [Course Book §6.7]. Measure the budget of Table 8.7 in the first week and replace every allocation with its measured $p50$ and $p99$.

The mathematics. Forward cost of one (instrument, granularity) series, with T tokens, width d, depth n_l, patch length P, N patch tokens, K levels and horizon H:

$$F_{\text{series}} = \underbrace{2T\,(12d^2)\,n_l}_{\text{projections}} + \underbrace{4T^2 d\,n_l}_{\text{attention}} + \underbrace{2NPd}_{\text{patch embed}} + \underbrace{2dKH}_{\text{head}} = 366{,}081{,}024 \text{ FLOP}, \tag{8.10}$$

so a 21-series refresh is 7.6877 GFLOP and a 147-series refresh 53.8139 GFLOP; wall-clock is F/Φ plus fixed overheads, and at these sizes the overheads dominate.

Artifact. `bespoke_predictions` table; `engine_events` table logging every fallback transition; a latency histogram per granularity written nightly.

Pass / fail. A rehearsed failure drill, not a review. Kill the Stage-3 sidecar mid-session and assert the engine serves from level 2 within one bar; kill the Stage-1 writer as well and assert level 3 within one bar; assert `engine_events` recorded both transitions; assert no bar was ever served from a row failing the staleness, monotonicity or version check. Separately, assert the measured $p99$ is inside the 10 ms cap and the measured $p100$ inside the 35 ms cap over a full week including at least five daily coincident closes.

The arithmetic floor is 1.9219 ms for the ordinary close and 13.4535 ms for the coincident one, at an assumed sustained 4.0 TFLOP/s. The single-digit-millisecond target therefore holds comfortably for every ordinary bar close and is *exceeded* exactly once per day, when the daily bar closes and drags the other six granularities with it: 27.90 ms, 5.29 times the ordinary close. Declare that spike in the monitoring configuration before go-live. A latency alarm that fires every day at the same minute for a month teaches the operator to ignore latency alarms.

Table 8.7. Inference latency budget on the pinned 3060, milliseconds. Component figures are allocations to be replaced by measurement; the totals, caps and bar fractions are arithmetic. The right-hand column is the once-per-day event in which all seven granularities close together and $21 \times 7 = 147$ series must refresh inside a single M1 bar.

Component	Ordinary close (21 series)	Coincident close (147 series)
Feature assembly and as-of joins	1.20	8.40
Host to device copy	0.15	1.05
ONNX Runtime forward, one fused batch	1.92	13.45
Device to host copy	0.05	0.35
RevIN inverse, monotone check, conformal offsets	0.35	2.45
SQLite write, one transaction	1.60	2.20
Total	**5.27**	**27.90**
Declared cap	10.00 (p99)	35.00 (p100)
Headroom	4.73 (47.3%)	7.10
Fraction of an M1 bar	0.00878%	0.0465%

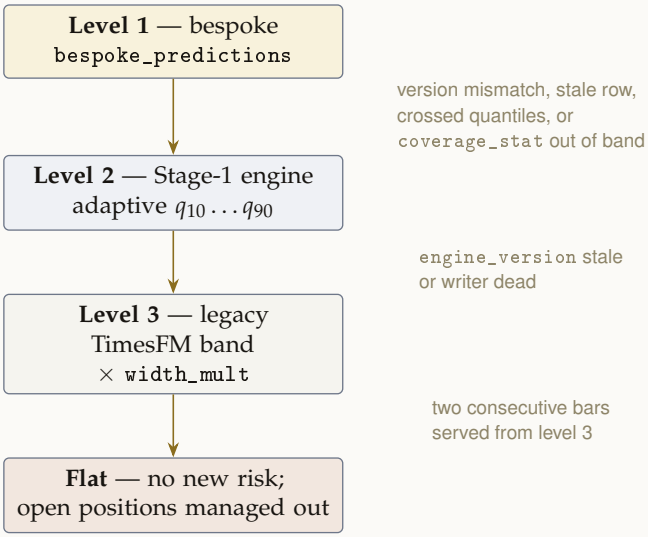


Figure 8.1. The fallback chain, evaluated on every read. Each descent writes an `engine_events` row; levels 2 and 3 raise a degradation alarm. The chain is checked before the model is trusted, not after it has misbehaved.

8.4 After the ship

Step 8.8 — Retrain policy: a retrain is a new strategy

Do. Treat every retrain as a fresh hypothesis. It re-enters at Step 8.1, rebuilds the dataset at the current feed and cost vintages, re-runs the full six-run gauntlet, writes six new ledger rows, takes a new `model_version` and a new ONNX hash, and shadows before it serves. There is no in-place weight refresh, no "same model, new data", and no partial gauntlet. The reason is one sentence: a retrain re-selects over the same search space with overlapping data, so its Sharpe carries the accumulated selection bias of every prior attempt, and the deflation must be recomputed with the *cumulative* trial count. Recommend a quarterly cadence on Path A — a 32-configuration search is 29.9 maintenance nights, which is exactly what one shared card sustains per quarter — and monthly on Path B if the 5080 is resolved. All retraining runs inside the declared maintenance window; a retrain that would overrun is truncated at the window boundary and resumed the next night, never extended into live hours. Any of the four drift triggers below forces early re-entry.

The mathematics. The cumulative trial count entering the deflation after r retrains of an n-configuration search:

$$N_{\text{cum}} \;=\; n\,(r+1), \qquad \text{so at } n=32,\ r=3:\ N_{\text{cum}}=128. \tag{8.11}$$

N_{cum}, not n, is the number the DSR of run 3 must use [Course Book §15.4].

Artifact. `drift_events` table; a retrain register mapping each `model_version` to its predecessor, its trigger, its cumulative trial count and its gauntlet `trial_ids`.

Pass / fail. No `model_version` is ever served whose registry row lacks six gauntlet rows at the cumulative trial count. Audit quarterly by joining the registry to the ledger and asserting the join is total.

The four drift triggers, any one of which forces early re-entry. First, a CUSUM alarm on the coverage error sequence $(\text{err}_t - \alpha)$ for the 80% band, per (instrument, granularity), with the pb06 S7 constants, sustained over the declared window [Course Book §13.7]. Second, coverage decay: trailing empirical coverage of the 80% band outside $[0.74, 0.86]$ in three or more traded cells [Course Book §11.8]. Third, a cost-vintage change moving the median round-trip cost by more than 20% in any cell the strategy trades — the decision term of (8.6) was trained against a cost surface that no longer exists, and the sizing it learned is calibrated to the wrong no-trade band. Fourth, a feed break: any pb03 registry feed exceeding its expected latency for a full cadence, because the covariate the variate-attention layer was trained to read is now permanently masked.

Trap — "Just refresh the weights on new data"

This is the single most common route by which a deflated result becomes an undeflated one. The refresh is a new selection; the new selection is a new trial; the new trial raises N_{cum}, and a DSR computed at the old N is arithmetic about a search that no longer exists. A drift trigger authorises a retrain. A retrain is a new strategy. It re-enters at Step 8.1.

Step 8.9 — Rollback protocol

Do. Evaluate the five rollback criteria daily against the live and shadow record. *One*: trailing realised cost-net P&L below the q_{10} CPCV path of the pinned version, over a window the length of one CPCV path segment. *Two*: aggregate 80%-band coverage outside $[0.70, 0.90]$,

or a Christoffersen LR_{cc} rejection at 1% in any regime with $n \geq 100$. *Three*: an ONNX hash or `model_version` mismatch at load. *Four*: latency $p99$ above 10 ms or $p100$ above 35 ms on three consecutive sessions. *Five*: any pb04 CI control red — the licence rule freezes result-reading offline and freezes the model live. On any trigger, execute the mechanics in order: (i) flip the version pin in the engine configuration to the previous `model_version`, or to null to fall straight to level 2 of Figure 8.1; this is a configuration change, not a deployment, and no binary moves. (ii) Stop the Stage-3 sidecar. Do *not* drop `bespoke_predictions`: it is the evidence. (iii) Restart the rolled-back version writing to `bespoke_predictions_shadow`, which nothing reads, for a minimum of six weeks, so the failure acquires an out-of-sample record instead of disappearing. (iv) Write the rollback memo: trigger, timestamp, the trailing metric values that fired it, the gauntlet `trial_ids` that licensed the version, and the decision on whether the family re-enters at Step 8.1 or retires. (v) Never re-pin a rolled-back version. It returns only as a retrain, which is a new strategy.

Artifact. `rollback_memo_<model_version>.md`, the shadow table, and the amended registry row marking the version `rolled_back` with its reason.

Pass / fail. Rehearse the rollback once, in staging, before the first live bar: flip the pin, confirm level-2 service inside one bar, confirm the memo template renders with real values, confirm the shadow writer starts. An unrehearsed rollback is not a rollback; it is a hope.

Stage gate — Stage 3

Three conditions, all required, evaluated on the frozen artifact of Step 8.6 at the pinned cost vintage.

1. Deflated economics. The bespoke policy's Deflated Sharpe Ratio against the Stage-2 ensemble exceeds 0.5, computed with the cumulative trial count N_{cum} of (8.11) — 32 at first entry — and the realised skew and kurtosis of the per-path returns [Course Book §15.4]. The q_{10} CPCV path is positive and survives the 1.5× cost stress.

2. Overfitting probability. PBO ≤ 0.5 over the *full* search population, abandoned configurations included [Course Book §15.5].

3. Sequential confirmation. A strictly sequential walk-forward — run after the CPCV runs, never used for selection, never re-run with a changed configuration — produces cumulative cost-net P&L inside the CPCV path band, between the q_{10} and q_{90} paths [Course Book §16.3]. CPCV recombines the past; the walk-forward is the only run whose ordering matches the order in which the money is actually made. A walk-forward below the q_{10} path means the CPCV distribution is optimistic, and the gate fails regardless of conditions 1 and 2.

Decision rule. All three, or the stage fails. Marginal is a fail: under a declared search of 32 configurations, a marginal result is the expected face of noise [Course Book §15.4]. There is no partial pass, no "pass on FX majors only" and no re-run with a widened universe — either of those is a new trial that raises N_{cum} and lowers the bar it must clear.

Kill criterion — Stage 3 — wording to sign before the first run

We tested the hypothesis that a bespoke _____-parameter quantile transformer, trained with a decision-focused objective against the measured cost surface, produces a deflated, cost-net improvement over the Stage-2 ensemble. Universe: _____. Period: _____. Cost vintage: _____. Declared search: _____ configurations, cumulative trial count _____. Result: DSR _____, PBO _____, walk-forward _____ the CPCV path band.
It did not. We ship the Stage-2 ensemble. The Stage-3 model is retired, its ONNX artifact archived against its registry row, and its version pin removed; the live configuration reverts

to the Stage-2 ensemble over the Stage-1 adaptive envelope. No re-run of this hypothesis is authorised on this data. A future re-entry requires new data covering at least _____, a pre-registered search space written before the data is seen, and a trial count that includes this attempt. A Stage-3 kill does not authorise Stage 4 as a substitute: Stage 4 is scenario tooling and was never an alpha thesis.

Ledger owner: _____ Date: _____

Artifacts — Stage output register

Code and configuration. `stage3_dataset.py` (builder); `stage3.yaml` (architecture, loss, optimiser — the `config_hash` source); `stage3_loss.py`; `env.lock`.

Data. `/data/stage3/{train,val,test}/shard-*.npz` with `manifest.json`: 1,177,785 stride-1 windows over 21 instruments at M30, H1 and H4, effective independent label count 23,555.7.

Models. `checkpoints/<config_hash>/seed-{11,23,37}/best.pt` for all 32 declared configurations; `models/s3-<date>-<hash8>/model.onnx` (frozen, 3,765,754 parameters); `parity_report.json`; the `registry.sqlite` row binding version, config hash, environment hash, cost vintage and gauntlet trial ids.

Evaluation. Six gauntlet reports `g1-baselines` ...`g6-leak` with raw per-path P&L; the walk-forward confirmation run; the deflation worksheet carrying N_{cum}.

Ledger. Six rows per evaluated configuration where the gauntlet was completed, one row per abandoned configuration, the search-budget field recording which GPU path was used, and the cost-vintage pin on every row.

Live. `bespoke_predictions` and `bespoke_predictions_shadow` in `timesfm.sqlite`; `engine_events`; `drift_events`; the retrain register; nightly latency histograms; the rehearsed rollback memo template.

9

Stage 4: Generative Tail Tooling

Stage 4 builds a joint path sampler and turns it into three objects the platform's per-bar quantile head cannot produce: drawdown distributions, first-passage probabilities, and a path-conditional CVaR column for Stage 3's sizing layer [Course Book §23.1]. None of it is alpha, and none of it may become alpha by the back door: every output is measurement or stress, consumed by a sizing calculation or an evaluation harness, never by a direction decision. Three controls make that promise enforceable rather than aspirational, and this chapter is built around them: the scope guard (S1), the memorisation guard (S4), and the consistency check against the live quantile head (S5). Latency (S3) and incremental value (the stage gate) are arithmetic and evaluation exercises built on top of those three, not substitutes for them.

Key idea — Tooling, not alpha

A generative sampler re-expresses the conditional information Stages 1–3 already extracted as a joint law over paths; it does not forecast direction better than the model it trains alongside [Course Book §23.1]. Its licence to be used at all is conditional: its per-bar marginals must reproduce the live calibrated quantile head to within Monte Carlo error before any path functional built from it may be reported anywhere [Course Book §23.5]. Treat every number S5 produces as provisional until S4's memorisation guard and S5's own divergence check both clear.

Step 9.1 — Scope guard

Do. Stage 4 is async-only, by construction and by enforcement. Every Stage-4 process is a separate Python sidecar — its own OS process, its own CUDA context — triggered after bar close, never imported by and never called synchronously from the C# engine's bar loop or the live TimesFM/Stage 1–3 inference path [Course Book §23.1]. Its outputs feed exactly two consumers: Stage 3's sizing layer (a `path_cvar` column) and the Chapter 16 evaluation harness (a stress-scenario library); neither consumer may use a Stage-4 output to decide direction or entry. State the circularity ban as an enforceable rule, not a guideline: synthetic paths may never train or select the live quantile head, the bespoke model (pb08), or any router (pb07) without every derived ledger row carrying a `synthetic_input` flag, and a metric measured only on synthetic data may never stand alone as evidence [Course Book §23.5]. Two prohibitions follow directly: (i) generated windows may not be appended as extra training rows to any live forecaster without the flag and a matching real-data result reported alongside it; (ii) a model or hyperparameter may not be selected by a metric measured on synthetic data alone. Wire a CI check on both: the import graph of the live inference entrypoint must contain zero edges into the Stage-4 sampler module, and the build fails the moment one appears.

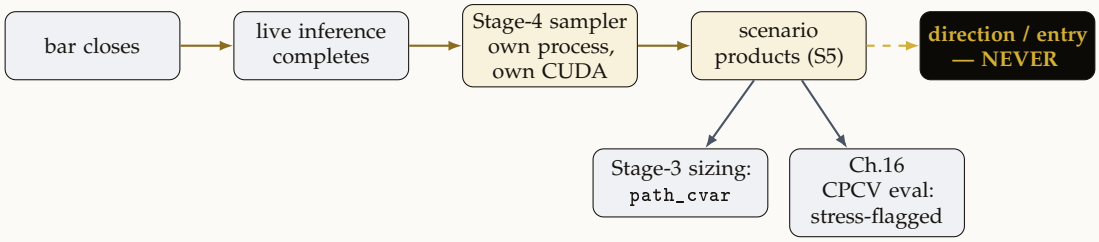
Artifact. `stage4_scope_guard` policy file (the two prohibitions, stated so each is machine-checkable where possible); the CI import-graph assertion; a migration adding `synthetic_input BOOLEAN` and `synthetic_source_run_id` to the trial ledger (pb04 S4).

Pass / fail. Pass: the CI import-graph check is green, and a query over the full trial ledger finds every row touching any Stage-4 output flagged. Fail: one import edge from the live entrypoint into the sampler module, or one unflagged row — either blocks every later step in this chapter.

Trap — Circularity is a ledger defect, not a modelling error

Synthetic paths contain no information the sampler was not trained on. Training a forecaster on generated data, or selecting a model by its performance on generated data, validates the generator's smoothing of history and reports it as out-of-sample evidence [Course Book §23.5]. The two prohibitions above are the whole of the defence: no unflagged synthetic row in the ledger, no selection decision made on synthetic-only evidence. A strategy that looks good only on its own generator's output has discovered nothing.

The pipeline S1 enforces, end to end:



Step 9.2 — Sampler build

Do. A CSDI-style conditional diffusion transformer [Course Book §23.4], denoising only the target coordinates of a joint window spanning the full H-bar horizon across all m instruments at one timeframe simultaneously — correlated multi-asset paths are the entire point of a joint sampler, and joint modelling only makes sense within one timeframe's bar grid, so build one checkpoint per timeframe. Mask conditioning: partition the window into an observed/context set, pinned to its true value at every forward and reverse step, and a target set, noised forward and denoised in reverse; signal the split to the network with a binary mask channel $\boldsymbol{m} \in \{0,1\}^n$ carried alongside the pinned values themselves [Course Book §23.4]. Context reuses the pb08 dataset builder's output unchanged: the same 1536-bar, patch-32 tokenisation (pb08 S1), 48 tokens per instrument, fed as pinned sequence elements — do not build a second context pipeline. Only the target branch, the next $H = 50$ bars raw (unpatched) per bar per instrument, is diffused: $n_{\text{tgt}} = H \cdot m$. Recommended configuration:

Quantity	Recommended value	Reason
H, m	50, 21 (canon, fixed)	horizon and universe are platform constants
$n_{\text{tgt}} = Hm$	1050	diffused target coordinates
context tokens	1008 (48×21, pb08 S1 reused)	pinned, not diffused
denoiser	d=128, n_l=3, n_h=4, temporal + variate attention	smaller than Stage 3's 256/4/4: denoising, not quantile regression; must stay batchable at $B \leq 256$ (S3)
T_d^{train}	1000, linear $\beta : 10^{-4} \to 0.02$	[Course Book §23.2]'s realistic schedule; training cost is independent of T_d
T_d^{sample}	20–50 (DDIM)	[Course Book §23.2]; S3 shows wide latency headroom even at 50, so no pressure to cut below 20

Training data: the pb08 dataset builder's output, unmodified, so the sampler's splits align exactly with Stage 3's and no extra leakage surface is introduced. Masking for training is random over the whole joint window (any subset hidden and restored), not only the forecasting mask, so one checkpoint also serves gap-filling and denoising diagnostics with no retraining [Course Book §23.4]. Train in the Stage-3 maintenance window on the 3060, or on the 5080 pending the owner's decision — pb08 S4's two paths, unchanged here.

The mathematics. One-shot forward identity, restricted to the target coordinates, context pinned at every step:

$$\boldsymbol{x}_t^{\text{tgt}} = \sqrt{\bar{\alpha}_t}\,\boldsymbol{x}_0^{\text{tgt}} + \sqrt{1-\bar{\alpha}_t}\,\boldsymbol{\varepsilon}, \quad \boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}_{n_{\text{tgt}}}), \qquad \boldsymbol{x}_t^{\text{ctx}} \equiv \boldsymbol{x}_0^{\text{ctx}} \;\; \forall t.$$

$\bar{\alpha}_t$ is the cumulative signal retention at diffusion step t under the fixed training schedule; the context block is never noised, only carried alongside the mask channel so the network can tell a pinned value from a denoised one [Course Book §23.2, §23.4].

Artifact. `csdi_sampler_v{n}_{tf}.pt` checkpoint per timeframe + config (window/-mask/network/schedule) + training log (loss curve, epochs, seed) + a parity note confirming the sampler's splits match pb08's exactly.

Pass / fail. Pass: validation denoising loss (target coordinates only) plateaus within the final 10% of scheduled epochs without diverging, and a smoke-test batch sampled at $T_d^{\text{sample}} = 50$ contains no NaN/Inf with every target coordinate's standardised variance in $[0.5, 2]$ — a sanity bound, not a fidelity claim; fidelity is S4–S6. Fail: retrain before S3 measures latency on a model that fails the smoke test.

Step 9.3 — Latency measurement

Do. Protocol, on the RTX 3060, model fixed at S2's config: discard 10 warm-up calls, then time ≥ 50 calls of one denoising forward pass at each batch size $B \in \{64, 128, 256\}$; record median and p95 ms/pass and the VRAM high-water mark per B. Run in isolation first (no contending load); S7 repeats the measurement under simulated contention. Compute the scenario-set wall clock from the measured $t_{\text{pass}}(B)$ by

$$\text{wall}(N, T_d, B) = \left\lceil \frac{N}{B} \right\rceil \times T_d \times t_{\text{pass}}(B),$$

with $N = 1000$ paths (the scenario-set size a path-CVaR column needs, [Course Book §23.4]), T_d the sampling step count from S2's DDIM range, and $\lceil N/B \rceil$ the batch count. Always state which reading of "t_{pass} costs X ms" a quoted number used — see the trap below. Illustrative figures below reuse [Course Book §23.4]'s own stated benchmark input for this hardware and job,

$t_{\text{pass}} = 4$ ms at every tested B; S3's protocol above replaces this with the platform's measured value on first run and the tables are regenerated from it, unchanged in structure.

B	batches $\lceil N/B \rceil$	wall clock, $T_d = 20$	wall clock, $T_d = 50$
64	16	1.2800 s	3.2000 s
128	8	0.6400 s	1.6000 s
256	4	0.3200 s	0.8000 s

Recommend an async-budget fraction $f = 10\%$ of one bar interval: it leaves the remaining 90%-plus of the inter-bar gap for engine housekeeping, live inference and scheduling jitter, since Stage 4 must yield to live inference (S7) and cannot assume it owns the whole gap. Against the seven canon timeframes (D taken as 24h = 86,400 s, continuous FX/CFD trading, not an equity session):

TF	bar length	best case (B=256, T_d=20)	worst tested (B=64, T_d=50)	verdict
M1	60 s	0.5333%	5.3333%	PASS (both)
M5	300 s	0.1067%	1.0667%	PASS (both)
M15	900 s	0.0356%	0.3556%	PASS (both)
M30	1800 s	0.0178%	0.1778%	PASS (both)
H1	3600 s	0.0089%	0.0889%	PASS (both)
H4	14400 s	0.0022%	0.0222%	PASS (both)
D	86400 s	0.0004%	0.0037%	PASS (both)

Every tested configuration clears f on every timeframe with wide margin: the arithmetic gate is easy. S7's contention test is the one that is not.

The mathematics. wall(N, T_d, B) above; cite [Course Book §23.4].

Artifact. `stage4_latency_report_{date}.csv` (`batch_size, T_d, median_ms, p95_ms, vram_peak_MiB`) and the two tables above, regenerated from it.

Pass / fail. Pass: the chosen production (B^*, T_d^*) clears f on every timeframe Stage 4 serves. Fail: falls back to the nightly offline job (Kill criterion, below).

Trap — Two readings of "one pass costs X ms"

A per-path (unbatched) benchmark and a batched benchmark differ by two orders of magnitude, and quoting the wrong one silently is the most common latency-budget error [Course Book §23.4]. On the same 4 ms/pass assumption, the unbatched reading fails four of the seven timeframes outright (M1 through M30) even before any GPU contention is considered, at up to 333.3333% of the M1 bar interval it should fit inside at $T_d = 50$. Log the batch size next to every measured t_{pass}; a number without it is not a latency claim.

Step 9.4 — Memorisation guard

Do. Build the library $\mathcal{W}$ once per checkpoint: every training window S2 used, standardised the same way, on the same target coordinates the sampler diffuses (n_{tgt}-dimensional, joint across all m instruments, so the guard catches a memorised *correlation* structure, not only a memorised single-instrument path). Freeze $\mathcal{W}$ to the checkpoint's version hash — a library built from a different training run measures nothing, see the trap below. For every generated path g in a scenario set, compute

$$d_{\min}(g) = \min_{w \in \mathcal{W}} \|g - w\|_2$$

on standardised returns [Course Book §23.5]; use a bounded-window DTW distance instead (window $r \leq 0.1H \approx 5$ bars, forgives small phase shifts without letting the warp collapse genuinely different paths) as a secondary check on any path the L2 rule flags, to separate a phase-shifted near-copy from a genuinely novel path. Compute the real-vs-real baseline once per library version: leave-one-out nearest-neighbour distances among the windows of $\mathcal{W}$ themselves, cached alongside it (an $O(|\mathcal{W}|^2)$ computation; an approximate index is acceptable if the reported percentile is validated exactly on a subsample [Course Book §23.5]). Alarm at the run level, and flag individual paths:

Pseudocode — Memorisation guard: per-run check

Require: generated paths $\{g_1, \ldots, g_N\}$ (target coordinates, standardised); library $\mathcal{W}$ tied to the current checkpoint hash; cached baseline first percentile p_1^{real}

for $i = 1, \ldots, N$ **do**
  $d_{\min}(g_i) \leftarrow \min_{w \in \mathcal{W}} \|g_i - w\|_2$ ▷ or bounded-window DTW
  `near_duplicate`$_i \leftarrow \left(d_{\min}(g_i) < p_1^{\text{real}}\right)$
end for
$p_1^{\text{gen}} \leftarrow$ first percentile of $\{d_{\min}(g_i)\}_{i=1}^{N}$
`run_alarm` $\leftarrow \left(p_1^{\text{gen}} < p_1^{\text{real}}\right)$ **or** (near-duplicate fraction $> 1\%$)
write report: histogram of $d_{\min}$, p_1^{gen}, p_1^{real}, `run_alarm`, per-path flags, distance metric, library and checkpoint version

Drop or down-weight every `near_duplicate` path before it reaches S5's product tables — do not silently fold suspected copies into a CVaR or drawdown distribution. Run the full check on every retrain (mandatory); between retrains, run it as a cheap canary on a random subsample of 100 generated paths per production run, and escalate to the full N if the subsample's flagged fraction looks anomalous.

The mathematics. $d_{\min}(g)$ above ([Course Book §23.5], eq. for the nearest- neighbour distance); alarm rule $p_1^{\text{gen}} < p_1^{\text{real}}$, calibrated against the platform's own historical windows rather than an independent-Gaussian scale reference, because real FX windows cluster far more tightly than independent draws do [Course Book §23.5].

Artifact. `stage4_memo_library_{version}.parquet` (frozen, hash-tied); `stage4_memo_baseline_{version}.json` (cached leave-one-out percentiles); `stage4_memo_report_{run_id}.json` (histogram, p_1^{gen} vs p_1^{real}, `run_alarm`, per-path flags, metric, versions).

Pass / fail. Pass: $p_1^{\text{gen}} \geq p_1^{\text{real}}$ and the near-duplicate fraction is at or below 1% (a judgement: the guard already targets the first percentile, so a well-calibrated generator should show roughly that base rate of borderline cases). Fail: blocks S5 and S6 outright — nothing computed from this batch may be cited anywhere, including in a slide [Course Book §23.1].

Trap — A library that drifts out from under its report

Testing a retrained sampler's paths against yesterday's library, or against yesterday's cached real-vs-real baseline, measures nothing: the guard's entire claim rests on comparing a checkpoint to exactly the data it was trained on. Version both $\mathcal{W}$ and the baseline to the sampler checkpoint's hash, refuse to run the guard on a mismatch, and treat that refusal as a hard stop, not a warning to route around.

Step 9.5 — Scenario products and the consistency check

Do. For each (instrument, TF) the sampler serves and each bar $k = 1, \ldots, H$, take the empirical quantiles $\hat{q}_\tau(k)$ for $\tau \in \{10, 20, 50, 80, 90\}$ (the live envelope's own published set, canon) from the N generated cumulative-return paths at bar k. Before computing anything else, run the consistency check: pull the live calibrated quantiles $q_\tau(k)$ for the *same* bar from `timesfm.sqlite`, aligned by the conditioning context's as-of timestamp s to the live row's timestamp exactly — comparing forecasts issued from different information sets is not a consistency check, it is noise, so treat a mismatched timestamp as a hard stop, the same discipline pb03 applies to external feeds. Turn canon's qualitative requirement (agreement "to within Monte Carlo error" [Course Book §23.5]) into an executable rule: for each (τ, k) cell compute the fraction of generated paths at or below the live quantile, $\hat{p}_\tau(k)$, its z-statistic against the binomial null, and alarm only past a Bonferroni-corrected threshold, since one (instrument, TF) run tests $5 \times 50 = 250$ cells at once and an uncorrected per-cell test would alarm on noise routinely. For max-drawdown, use the convention fixed in [Course Book §23.5] exactly: running peak includes the entry level. For stop-before-target, evaluate on bar closes and report the sample size N beside every probability — a probability without its N is not actionable [Course Book §23.5]. For CVaR, run every path through Stage 3's actual execution rule (entry, stop, target, time exit) against the built cost surface (pb02), producing realised trade P&Ls $L_1, \ldots, L_N$, then take the coherent Rockafellar–Uryasev estimator, never the naive plug-in, since the two differ by double digits and only the coherent value may enter an optimisation or a risk limit [Course Book §14.5]. Report at $a \in \{0.90, 0.95, 0.99\}$, the levels Stage 3's sizing caps are audited against.

The mathematics.

$$\hat{p}_\tau(k) = \frac{1}{N}\sum_{g=1}^{N} \mathbf{1}\{C_g(k) \le q_\tau(k)\}, \qquad \mathrm{SE} = \sqrt{\tfrac{\tau}{100}\left(1 - \tfrac{\tau}{100}\right)/N}, \qquad z_{\tau,k} = \frac{\hat{p}_\tau(k) - \tau/100}{\mathrm{SE}}.$$

The divergence check, [Course Book §23.5].

$$M_k = \max_{0 \le j \le k} C_j \ (C_0 = 0), \qquad D_k = M_k - C_k, \qquad \mathrm{MDD} = \max_k D_k.$$

The drawdown convention, [Course Book §23.5].

$$N_{\text{required}} = \frac{1 - p}{p \cdot \text{rel_err}^2}$$

for a target relative error on a tail probability p, [Course Book §23.5].

$$\widehat{\mathrm{CVaR}}_a = \min_{c \in \mathbb{R}} \left\{ c + \frac{1}{N(1-a)} \sum_{i=1}^{N} (L_i - c)_+ \right\}$$

the sample Rockafellar–Uryasev estimator, [Course Book §14.5]. Symbols: $C_g(k)$ the cumulative log return of generated path g at bar k; $\mathbf{1}\{\cdot\}$ the indicator; M_k, D_k the running peak and drawdown of one path under the book's convention (peak includes $C_0 = 0$); L_i the realised loss ($-$P&L) of path i under the actual execution rule.

Artifact. Five output tables:

Table	Grain	Key columns
`scenario_quantiles`	instrument × TF × run × bar k × τ	`q_hat, q_live, z_stat, alarm_flag`
`scenario_mdd`	instrument × TF × run	`mdd_p50/p75/p90/p95/p99, mean_mdd, n_paths`
`scenario_touch`	instrument × TF × run × (stop, target)	`p_stop_first, p_target_first, p_neither, se, n_paths`
`scenario_cvar`	instrument × TF × run × a	`cvar_ru, cvar_plugin, cost_surface_version, n_paths`
`path_cvar` (in `timesfm.sqlite`)	instrument × TF, latest	`cvar_95_ru, cvar_99_ru, sampler_version, generated_ts, staleness_bars`

Pass / fail. Pass: S4 cleared; no (τ, k) cell exceeds the Bonferroni-corrected threshold ($z_{\text{crit}} = 4.1075$ at family-wise $\alpha_{\text{FW}} = 0.01$ over 250 cells); all five tables written with `n_paths` recorded on every row. Fail: blocks S6 and the `path_cvar` write; Stage 3 keeps the last known-good row, staleness flag set (S7).

Trap — The two CVaR conventions disagree by double digits

The plug-in average of losses at or beyond $\widehat{\text{VaR}}_a$ and the Rockafellar–Uryasev value can differ by more than ten per cent on identical data, because the plug-in dilutes the tail with the VaR observation itself [Course Book §14.5]. Quote the plug-in only when the phrase "average loss beyond the VaR" is meant literally in a report; quote the coherent value, and only the coherent value, anywhere it feeds a sizing decision or a risk limit.

Step 9.6 — Stress augmentation for evaluation

Do. Condition the sampler on a stressed context — the regime router's high-volatility, risk-off state (pb07) and the RORO/VolRegime as-of gauges (pb03) fed through the pinned context channels — to draw a stress batch distinct from S5's current-conditions batches. Before use, clear two gates. First, re-run S4's memorisation guard on this specific stress-conditioned batch: a differently conditioned generation can memorise differently than an unconditional one, so the guard is re-run per batch, never assumed from the checkpoint's last pass. Second, run the stylised-fact battery against the *real historical stressed-regime windows*, not the whole unconditional history, and prefer the platform's own measured autocorrelation as the reference band over the course book's illustrative literature ranges, which are explicitly not fitted to this data [Course Book §1.4]:

Check	Compares	Band
excess kurtosis κ	generated regime-conditional κ vs. real regime-matched κ [Course Book §1.4]	±25% relative (recommendation: kurtosis is high-variance even in-sample [Course Book §1.4])
|returns| ACF, lags $\{1, 5, 20\}$	generated vs. platform-measured real ACF	positive, same order of magnitude as the platform's own measured values
return ACF, lag 1	generated linear autocorrelation	near zero, matching real — a generator introducing spurious linear structure fails here

Only a batch clearing both gates may be appended to a CPCV evaluation set (pb04 S2: $N = 8$ groups, $k = 2$ test groups by default, $\varphi = 7$ real paths [Course Book §16.4]), and only as a separate, clearly labelled evaluation-only axis — "stress-CPCV" results reported alongside, never merged into, the primary real-data CPCV numbers. Synthetic blocks never

enter a training group: they are not part of the N contiguous real-time groups at all. Flag every ledger row whose evaluation set touched a stress block with `synthetic_input` and `synthetic_stress_run_id` (S1's schema), and count that evaluation as an additional trial in the deflation accounting [Course Book §15.4].

The mathematics. Excess kurtosis $\kappa = \mathbb{E}[(r-\mu)^4]/\sigma^4 - 3$ [Course Book §1.4]; the ACF check compares $\text{corr}(|r_t|, |r_{t-k}|)$ generated vs. real at the stated lags, no new formula [Course Book §1.4].

Artifact. Stress-scenario library (separate from S5's current-conditions batches); fidelity report (κ and ACF comparisons, guard rerun result); CPCV harness config with the stress axis labelled and isolated; flagged ledger rows.

Pass / fail. Pass: both gates clear for the specific batch. Fail: the batch is discarded, the failure is logged with its reason for audit, and evaluation proceeds on real data only — a failed batch is never silently retried until it passes.

Trap — Synthetic material inside a training fold

A future refactor of the CPCV group builder is the likeliest way a stress block ends up inside $G_1, \ldots, G_N$ by accident, since nothing about the combinatorics forbids it once a synthetic block looks like any other contiguous group. Wire a CI assertion into the harness build (alongside S1's import-graph check) that scans every training-fold's row range for `synthetic_input = TRUE` and fails the build if it finds one.

Step 9.7 — Integration

Do. Deploy exactly as S1 requires: Stage 4 is its own sidecar service, never imported into the C# engine or the live inference sidecar. It triggers after bar close and after the live inference sidecar reports its cycle complete — polling a readiness signal (a last-inference-done timestamp), never assuming a fixed offset — so it never contends for the GPU at the moment live inference needs it. On the pinned 3060, with the 5080's role canon-unresolved, two GPU-sharing mechanisms are available: (a) strict yield-on-demand priority, Stage 4 holds an exclusive lock it releases the instant a live-inference request arrives, checked *between diffusion steps*, not only between batches, since one batch at $T_d = 50$, $B = 256$ can otherwise hold the GPU for up to 0.8 s uninterrupted (S3); (b) NVIDIA MPS time-slicing, which cuts context-switch overhead but does not guarantee latency isolation — a busy Stage-4 kernel can still queue behind or in front of a live-inference kernel, so MPS alone does not satisfy "leaves the live path untouched" [Course Book §23.4]. Recommend (a) as the default and reserve MPS for the day Stage 4 moves to a dedicated card, where contention is moot. Record the 5080's role as an explicit open decision, not a design choice this chapter makes; when it resolves, Stage 4 is the natural first workload to move, since it is already a fully separate async process with no live-path coupling. Staleness: if Stage 4 is killed mid-run (a contention timeout) or its `path_cvar` row is older than $2\times$ the timeframe's bar length (one full missed cycle of grace), Stage 3's sizing layer falls back to its own bar-level CVaR from the live quantile head, the same fallback-chain pattern pb08 S7 already establishes — it never blocks and never uses a stale value silently.

Artifact. Sidecar service definition; GPU-lock/priority mechanism (lock table or CUDA-stream-priority spec); staleness policy constants; `path_cvar` with its `staleness_bars` column; an open-decision log entry for the 5080.

Pass / fail. Pass: a live-inference latency benchmark taken with Stage 4 running concurrently shows no more than a 5% p95 regression against the S3 baseline (recommendation: keeps

Stage 4 provably non-load-bearing on the path that is load-bearing), and the staleness fallback is exercised at least once in rehearsal — kill Stage 4 mid-run, confirm Stage 3 falls back cleanly. Fail: Stage 4 is disabled until contention is fixed (Kill criterion).

Stage gate — Stage 4

Two conjunctive conditions, both gated on a precondition that is checked, not scored. **Precondition.** S4's memorisation guard clears and S5's divergence alarm is silent and S6's fidelity bands clear for every batch cited in the P&L claim below — this is a precondition, not part of the scorecard [Course Book §23.7]: a lower CRPS, a better-looking scenario fan, or agreement with the quantile head is not evidence for the second gate, only a licence to compute it. **Gate 1 — latency.** S3's measured wall(N, T_d, B)/bar length $\leq f$ on every timeframe Stage 4 serves, and S7's contention test shows no material live-inference regression [Course Book §23.4, §23.7]. **Gate 2 — incremental value.** Sizing that consumes S5's `path_cvar` beats Stage 3's existing sizing (pb08) on out-of-sample net-of-cost P&L, measured under CPCV [Course Book §16.4], deflated for the trial count including S6's stress-augmented evaluations [Course Book §15.4]. Both gates must pass; neither substitutes for the other.

Kill criterion — Stage 4

On failing Gate 1 (latency or contention): Stage 4 drops its async sidecar trigger and moves to a nightly offline batch job, once per maintenance window, producing the same S5 tables for S6's evaluation harness and for manual sizing-calibration review only — never polled intra-day by Stage 3. On failing Gate 2 while the precondition and Gate 1 both pass: Stage 4 still remains, unremoved, as offline/nightly-consulted tooling, because that outcome is a success, not a consolation [Course Book §23.7] — it is the only source in the programme for drawdown distributions, first-passage probabilities and correlated multi-asset stress. What is actually killed in either case: the `path_cvar` column stops being polled by Stage 3's live sizing calculation, which reverts fully to its own bar-level CVaR (the established fallback chain, pb08 S7, unchanged). No other stage is affected, since Stage 4 was never on the direction path. Pre-committed wording to sign before Stage 4's first live run: *"Stage 4 is disabled or downgraded to nightly-offline; sizing continues on Stage 3's own* CVaR*; nothing already shipped depended on Stage 4 output for direction, so no other stage is affected."*

Artifacts — Stage 4

- `stage4_scope_guard` policy file, CI import-graph check, ledger schema migration (`synthetic_input`, `synthetic_source_run_id`) — S1.
- Per-timeframe sampler checkpoints, configs and training logs — S2.
- `stage4_latency_report_{date}.csv` and the async budget table — S3.
- Memorisation-guard library, cached real-vs-real baseline, per-run reports, both version-tied to the sampler checkpoint hash — S4.
- `scenario_quantiles`, `scenario_mdd`, `scenario_touch`, `scenario_cvar`, and the `path_cvar` feed into `timesfm.sqlite` (table-drop contract, canon) — S5.
- Stress-scenario library, fidelity reports, isolated CPCV stress axis, flagged ledger rows — S6.
- Sidecar service spec, GPU-priority mechanism, staleness policy constants, 5080 open-decision log entry — S7.

PART IV

The Market Versions



LUNA LABS

10

The Market Versions: v2 Options, v3 Pairs, v4 Commodities

This chapter assumes v1 is built: the Stage −1 preconditions of pb02–pb04 hold, and at least Stage 1 of pb05–pb09 has run to a recorded verdict. What follows adds three market heads to the existing FX spine — options, equity pairs, commodities — and does not repeat the shared-trunk mechanics of Chapter 19 (multi-task loss weighting, transfer protocols, instrument embeddings) beyond citing them; it covers what each market head needs *in addition*: its own data, its own signal, its own cost model, its own evaluation, its own gate.

Key idea — Pool representations, silo calibration, unify the ledger

One rule governs everything below, restated by every version in its own vocabulary rather than re-derived. The shared trunk trains on every market's data jointly and is the correct default. Each market's conformal band, cost surface and walk-forward confirmation is built, calibrated and confirmed on that market's own exchangeable history alone, never another market's. Every configuration any version's search has tried, kept or discarded, reports to the one trial ledger that prices Chapter 15's Deflated Sharpe correction for the whole programme, not per version. [Course Book §19.5-19.6]

Steps below are grouped by version; each version ends with its own stage gate, kill criterion and artifact register, because a market head that fails its own gate does not ship regardless of how the others perform. Shared operations that belong to no single version — calibration silos, cold-start onboarding, rollout order — close the chapter.

Trap — Verify current specifications — every time, every venue

No contract multiplier, tick size, settlement method, expiry calendar, exercise style, margin rate, borrow fee, short-locate availability, roll calendar or session time in this chapter is a live number. Every one of them is set by an exchange or a broker and changes without notice; what follows states the accounting structure and the decision rule, never a number to hardcode. Before a single order routes under v2, v3 or v4, pull the current specification from the venue or broker API and store it with a capture timestamp, exactly as [Course Book §20.6] insists for options. An out-of-date multiplier turns a correct model into a wrong position size; an out-of-date short-locate assumption turns a correct pairs signal into an unfillable order.

10.1 v2 — Options

v2 is gated on v1's Stage 1 (pb06) passing: the whole thesis is that the Stage-1 engine's calibrated $\hat{\sigma}_r$ can be sold or bought against the market's own $\hat{\sigma}_{\mathrm{IV}}$, direction-free [Course Book §20.5]. A v2 built

on an unvalidated width model wagers real capital, at wider vol-point costs than FX, on a sigma engine of unknown quality.

Step 10.1 — Options chain and IV-surface capture

Do. Select a data vendor against three criteria, in this order: (i) delivers snapshots timestamped at the moment of capture, not end-of-day-only summaries — a point-in-time requirement, not a convenience; (ii) covers both legs of the committed v2 universe — ASX index and single-name options and the US index/ETF/single-name complex; (iii) preserves every surface it has ever served, so a later vendor restatement never silently overwrites what the platform saw at capture — the options analogue of the FRED/ALFRED vintage discipline pb03 already enforces. Store one row per (underlying, expiry, strike, option type) per snapshot: bid, ask, implied volatility as quoted, spot at capture, and *two* timestamps — `capture_ts` (when the snapshot was pulled) and `ref_price_ts` together with `ref_price_convention` (official settlement value for index options; last trade or NBBO mid for single names). Record the convention explicitly per row; never infer it later, and never pair a settlement price against an underlying close struck at a different instant.

Artifact. `options_chains.sqlite`, table `chain_snapshot(underlying, expiry, strike, opt_type, bid, ask, iv_quoted, spot_at_capture, capture_ts, ref_price_ts, ref_price_convention, vendor)`.

Pass / fail. Every row's `capture_ts` and `ref_price_ts` are populated and distinct wherever the settlement convention makes them so; re-querying the vendor for a historical snapshot returns bit-identical quotes to what is stored, never a restated surface.

Step 10.2 — The volatility edge

Do. For each (underlying, expiry), pull the Stage-1 engine's forecast of realised volatility over the contract's remaining life, $\hat{\sigma}_r$ — the machinery pb06 already validated, not a fresh model — and compare it to the chain's own at-the-money implied volatility $\hat{\sigma}_{\mathrm{IV}}$ at the matching `capture_ts`. The edge is $e = \hat{\sigma}_{\mathrm{IV}} - \hat{\sigma}_r$ in volatility points: positive means the market prices more variance than the engine expects (sell), negative means less (buy).

The mathematics. Delta-hedged P&L is dollar-gamma-weighted realised-minus-implied variance, exactly, under no assumption that either volatility is constant:

$$\Pi_T = \int_0^T e^{r(T-t)} \tfrac{1}{2} \Gamma_t S_t^2 \left(\sigma_{r,t}^2 - \sigma_I^2\right) \mathrm{d}t.$$

$\Gamma_t S_t^2$ is the position's dollar gamma (P&L per unit of squared proportional move), $\sigma_{r,t}$ the instantaneous realised-volatility path, σ_I the implied volatility the position was struck at, r the financing rate. [Course Book §20.5]

Artifact. `vol_signal(underlying, expiry, capture_ts, sigma_r_hat, sigma_iv_atm, edge_vol_pts, sign)`.

Pass / fail. $|e|$ exceeds a pre-registered minimum, stated before a single trade is looked at (a judgement call, not a computation), before the pair proceeds to Step 3; below it, log the row and skip the trade.

Step 10.3 — Delta-hedged straddle protocol

Do. Express the edge as an at-the-money straddle — one call, one put, same strike nearest the forward, same expiry. Sell when $e > 0$, buy when $e < 0$. Hedge delta at a fixed cadence

(recommend: once per session, plus whenever $|\Delta|$ drifts past a rebalance band of 10–15% of the position's initial delta-neutral notional — a judgement tuned to the venue's hedging cost, not a computed value) using the underlying or, where available, the front-month future. For an account that cannot carry naked short gamma, substitute the defined-risk alternate: an iron condor (short strangle, long a wider strangle against it), which caps the loss at the wing distance minus premium received, at the cost of surrendering part of the premium and most of the gamma to the long wings.

Artifact. `straddle_positions(trade_id, underlying, expiry, strike, structure, entry_ts, entry_edge_vol_pts, exit_ts, exit_reason);` `hedge_log(trade_id, ts, underlying_px, delta_before, delta_after, qty_traded)`, one row per rebalance.

Pass / fail. Every rebalance in `hedge_log` fires within the stated band; no position sits past expiry without an explicit assignment/exercise decision logged.

Step 10.4 — Options cost accounting, in vol points

Do. A straddle round trip crosses the market four times (two legs, entered and exited); each crossing costs half the quoted bid–ask width at that moment, in the same volatility-point units the edge is quoted in. Pull the venue's current quoted width at the actual trading time — never an average, never a historical assumption — and build the cost surface the way pb02 built the FX one: from recorded quotes, not mid-price candles. Widen the assumed cost in the wings, at the open, and in stress, where quoted width is structurally worse.

The mathematics. Round-trip cost in volatility points, $c_{\text{rt}} = n_{\text{cross}} \times (w/2)$, $n_{\text{cross}} = 4$ for a straddle, w the quoted width at trade time. A trade clears cost only if $|e| > c_{\text{rt}}$, before a single forecast error is counted. Table 10.1 sizes this generically against a sensitivity range of w; it is a structure to compute against your own venue's quotes, not a number to trade on — verify current specifications [Course Book §20.6].

Artifact. `options_cost_surface(underlying, session, quoted_width_vol_pts, captured_at)`.

Pass / fail. Cost-doubling control (pb04's leak-injection suite): OOS Sharpe of the overlay falls when every cost term is doubled; if it does not, the cost surface is under-measured, not the edge over-stated.

Quoted width w (vol pts)	Round-trip cost c_{rt} (vol pts)
0.25	0.50
0.50	1.00
1.00	2.00
2.00	4.00

Table 10.1. Round-trip options cost as a function of quoted market width, $c_{\text{rt}} = 4 \times (w/2)$. Delta-hedging costs and financing are additional. Verified: `verify_pb10.py` §2.

Step 10.5 — Score the forecast, the distribution, and the book

Do. Three separate scoring passes, never conflated. (1) Score $\hat{\sigma}_r$ as a forecast against realised variance with pb06's own QLIKE loss (already stated there), and run a Diebold–Mariano test of the Stage-1 engine against the naive benchmark that uses $\hat{\sigma}_{\text{IV}}$ itself as the forecast — if the engine cannot beat the market's own quote, there is no edge and no trade, regardless

of what the P&L ledger later shows. (2) Score the conditional distribution with pb06's PIT and Christoffersen diagnostics ([Course Book §17.2]) on realised outcomes, checking coverage specifically in the tail the book is short — the 99th percentile during a volatility spike, not the unconditional 80% band. (3) Score net-of-cost P&L on CPCV paths (pb04), purged by *option life*: two contracts whose lives overlap share the same realised path and are not independent, so an embargo shorter than the longest contract's life leaks. The trial count for the Deflated Sharpe correction includes every strike- and expiry-selection rule tried, not only the model variants.

The mathematics. Diebold–Mariano on the loss differential $d_t = \mathcal{L}(e_{1t}) - \mathcal{L}(e_{2t})$: $\mathrm{DM} = \bar{d}/\sqrt{\hat{V}(\bar{d})}$, $\hat{V}(\bar{d})$ a Newey–West HAC variance, with the Harvey–Leybourne–Newbold small-sample correction $\mathrm{DM}^* = \mathrm{DM}\sqrt{(T+1-2h+h(h-1)/T)/T} \sim t_{T-1}$, h the forecast horizon, T the sample size. [Course Book §17.1]

Artifact. `trial_ledger` rows tagged `universe='v2-options'` (one per model/strike/expiry-rule configuration, run or abandoned); QLIKE/DM report; coverage report.

Pass / fail. DM rejects in the engine's favour at a pre-registered level surviving the trial count; the PIT/coverage battery passes in-regime, not only on the unconditional aggregate.

Stage gate — v2 — the options overlay

Deploy only if every condition holds, reconciling Chapter 20's four-condition detail with Course Book Appendix E's one-line version:

(1) v1 has already cleared Stage 1 (pb06) — the sigma engine is the whole premise; no amount of options-side evidence substitutes for it.

(2) The Stage-1 engine beats the $\hat{\sigma}_{\mathrm{IV}}$-as-forecast benchmark on QLIKE, with a DM p-value surviving the number of model variants tried.

(3) The overlay's cost-net P&L has a Deflated Sharpe Ratio above zero on CPCV purged by option life.

(4) The combined v1+v2 book's Deflated Sharpe exceeds v1's alone by a margin that survives the added trial count — an overlay that only re-expresses v1's own volatility exposure is correlation, not diversification.

(5) The combined book's CVaR under the platform's fat-tailed conditional distribution stays inside budget in the worst historical regime in the sample, not on average.

Kill criterion — v2 fails

Do not deploy the overlay. No renegotiation of the threshold, no re-test on a narrower strike/expiry universe chosen after seeing which subset would have passed. The Stage-1 volatility engine keeps earning its keep inside v1 exactly as before; v2 is shelved, not the engine.

Artifacts — v2 output register

`options_chains.sqlite` (chain snapshots); `vol_signal` table; `options_cost_surface`; straddle/iron-condor position and hedge logs; QLIKE/DM report; PIT/coverage report; v2 rows in the global trial ledger; the signed gate memo.

10.2 v3 — Pairs

v3 does not depend on the volatility engine — its edge thesis is cointegration, not calibrated width — so it can be researched alongside v1 [Course Book §22.5]. Its own kill gate is severe on its own terms: pair mining is one of the largest multiple-testing factories in quantitative finance, and the arithmetic must be confronted before a line of screening code runs [Course Book §21.6].

Step 10.6 — Universe construction: ASX 200 and US large caps

Do. Build two point-in-time universe tables, one per market, each a list of (ticker, member_from, member_to) rows reconstructed from historical index-membership disclosures — never today's constituent list applied retroactively, which silently deletes every name later delisted for a takeover, fraud or bankruptcy: precisely the event that breaks a cointegrating relation. Attach a corporate-actions table sourced independently of the price series, and adjust every price series by applying factors as of their *announcement* timestamp, never retroactively from a currently adjusted series (a currently adjusted series encodes future corporate actions into past prices). Mark a delisting bar as a forced liquidation at a realistic price, never a silent deletion.

Artifact. `universe_asof(market, ticker, member_from, member_to)` for ASX 200 and US large cap; `corp_actions(ticker, action_type, factor, announce_ts, effective_ts)`; adjusted price series with a provenance column recording which factors were applied and when.

Pass / fail. A universe query as-of any historical date reproduces that date's actual index membership, including names since delisted; re-adjusting a price series from stored factors reproduces the stored series bit-for-bit.

Step 10.7 — Economic pre-filter and the candidate count

Do. Before any price data is touched, restrict candidate pairs to those with a stated economic reason to cointegrate: same GICS sub-industry, dual-class share pairs, ADR/ordinary pairs, index/constituent-basket pairs. Write the rule down before looking at a single return series — this is the only stage that shrinks the candidate count without spending statistical power. Log the raw pool every candidate universe offers, before filtering, as the search's starting denominator.

The mathematics. A universe of n names offers $\binom{n}{2} = n(n-1)/2$ unordered candidate pairs. [Course Book §21.6]

Artifact. `candidate_pairs(pair_id, ticker_a, ticker_b, market, filter_reason)` — one row per pair surviving the economic filter, plus a logged count of the raw pool it was drawn from.

Pass / fail. The raw-pool count and the survivor count are both written to the ledger before the next step runs; Table 10.2 reproduced with the platform's own n.

Universe	n	$\binom{n}{2}$
ASX 200	200	19,900
US large cap	500	124,750
Combined (siloed, summed)	700	144,650

Table 10.2. Raw candidate-pair counts by universe, before any filter. ASX and US legs are screened as separate silos, never cross-joined — a cross-market pair shares no trading session, currency or settlement, so nothing here pairs an ASX name against a US one. Verified: `verify_pb10.py` §1a.

Step 10.8 — Engle–Granger cointegration batch

Do. For every surviving pair, and a pre-registered rule for which leg is left-hand (the more liquid one, applied uniformly), run the Engle–Granger two-step procedure in batch. Fix the ADF augmentation-order rule before the search starts (an information criterion over $p = 0, \ldots, p_{\max}$, $p_{\max} \approx \lfloor 12(T/100)^{1/4} \rfloor$) and apply it identically to every pair — a per-pair-tuned p is itself a multiple-testing device. Require both orderings to reject, or run both regressions and keep the stricter result.

The mathematics. Step 1, OLS $y_t = \mu + \beta x_t + z_t$, keep $\hat{\beta}, \hat{z}_t$. Step 2, ADF on $\hat{z}_t$ with no intercept, statistic $= \hat{\gamma}/\mathrm{se}(\hat{\gamma})$ from $\Delta\hat{z}_t = \gamma\hat{z}_{t-1} + \sum_i \delta_i \Delta\hat{z}_{t-i} + \varepsilon_t$, compared to *Engle–Granger* critical values (Table 10.3), never Dickey–Fuller ones — using the wrong table manufactures cointegration out of pure noise, at a rate the arithmetic below makes concrete. [Course Book §21.2]

Artifact. `cointegration_results(pair_id, beta_hat, mu_hat, augmentation_p, adf_stat, eg_crit_5pct, reject_flag, ordering)`.

Pass / fail. `reject_flag` set only against the Engle–Granger table at the pre-registered α; every tested pair has a row, rejected or not — an untested or silently dropped pair is a leak in the count, not an absence of one.

	1%	5%	10%
Engle–Granger, $n = 2$ variables, constant in Step 1	−3.90	−3.34	−3.04

Table 10.3. Engle–Granger critical values for the Step-2 residual test, two variables, constant in the Step-1 regression. [Course Book §21.2]

Step 10.9 — OU half-life filter

Do. Fit the AR(1) form of the spread on every cointegration survivor, recover the half-life, and keep only pairs whose half-life sits inside the platform's screening band — roughly three to thirty days. Both bounds come from the round-trip cost, not folklore: the hard floor is pair-specific — recompute it from the pair's own OU fit and cost rather than reusing another pair's number.

The mathematics. Regress $\hat{z}_t = \hat{c} + \hat{\phi}\hat{z}_{t-1} + \hat{\eta}_t$; $\hat{\theta} = -\ln\hat{\phi}/\Delta$, $\hat{t}_{1/2} = \ln 2/\hat{\theta} = -\Delta \ln 2/\ln\hat{\phi}$, Δ the bar spacing [Course Book §21.3]. The net-optimal half-life and the hard floor below which no entry/exit gap clears cost: $t^*_{1/2} = (2c/g)^2$, floor $= t^*_{1/2}/4 = (c/g)^2$, $g = \Delta z\,\hat{\sigma}/\sqrt{2\ln 2}$, $\Delta z = z_{\text{in}} - z_{\text{out}}$, $\hat{\sigma}$ the spread's instantaneous volatility, $c = 4c_{\text{leg}}$ the round-trip toll. [Course Book §21.4]

Artifact. `ou_fits(pair_id, phi_hat, theta_hat, halflife_hat, halflife_floor, in_band_flag)`.

Pass / fail. `in_band_flag` true only inside $[3, 30]$ days *and* above the pair's own computed floor; a pair failing on speed is logged, not silently dropped from the count.

Step 10.10 — The honestly-counted N and its deflation bar

Do. Before a single pair enters backtesting, total N_{eff}: every candidate that survived the economic pre-filter, multiplied by every lookback-window and entry/exit threshold combination that will actually be tried against it (state the grid before running it). This is the number that *could have been selected*, not the number of pairs eventually traded, and it

is logged, timestamped, alongside the trial ledger before results exist. Feed N_{eff} and the ledger's own cross-trial Sharpe variance V (read off the ledger, not assumed) into pb04's Deflated Sharpe pipeline; no champion is believed unless its Deflated Sharpe clears the resulting bar.

The mathematics. N_{eff} is the count that could have been selected: every pair that failed the cointegration or half-life screen, every threshold grid point, every lookback — not merely the pairs eventually traded. pb04's expected-max $\text{SR}_0(N, V)$ (Euler–Mascheroni formula, already stated there) is evaluated at this N_{eff}, never at a smaller post-hoc count. [Course Book §21.6]

Artifact. A signed pre-registration record, filed alongside the trial ledger, recording N_{eff}, the grid that produced it, V, and SR_0 — written before the first pair-level backtest runs.

Pass / fail. N_{eff} and SR_0 are on record with a timestamp earlier than the first CPCV result; a champion reported after the fact without this row is not evaluated, full stop.

Instantiated at the platform's own combined universe: $\binom{200}{2} + \binom{500}{2} = 19{,}900 + 124{,}750 = 144{,}650$ raw pairs (Table 10.2). Budgeting a 3% economic-filter keep rate (a planning recommendation, not a measured figure — [Course Book §21.6] characterises this filter as keeping a low single-digit percentage) gives $N_{\text{prefilter}} = 4{,}340$; a grid of three lookback windows by four entry/exit threshold pairs (a recommendation: wide enough to bracket the 3–30 day half-life band, narrow enough to keep the count honest) gives $N_{\text{eff}} = 4{,}340 \times 12 = 52{,}080$. At $V = 0.04$ (the Course Book's own illustrative cross-trial dispersion [Course Book §21.6] — replace with the platform's measured V the moment real trials exist), pb04's $\text{SR}_0(52{,}080, 0.04) = 0.8494$ annualised: fifty-two thousand worthless pairs, at this dispersion, are expected to throw up a champion this good by chance alone, before a single leg cost is paid. Skipping the economic pre-filter and deflating against the full 144,650-pair pool instead raises the bar only to 0.8941 — confirmation of [Course Book §21.6]'s own finding that SR_0 grows like the square root of the log of N, so pre-filtering buys real but bounded protection. Verified: `verify_pb10.py` §1b–1d.

Step 10.11 — Trading rule: rolling z-score and the double-leg toll

Do. Standardise the spread with a rolling window $L \geq 6\hat{t}_{1/2}$, computed strictly on bars before the decision bar — never the full-sample mean and standard deviation, and never a $\hat{\beta}$ fit once on the whole sample and applied from bar one. Enter against the dislocation at $|z_t| > z_{\text{in}}$ (recommend 2), exit at $|z_t| \leq z_{\text{out}}$ (recommend 0.5, not exactly zero — the last half-sigma takes as long as the first one and a half), stop and flatten at $|z_t| > z_{\text{stop}}$ (recommend 3–4). Before enabling a pair, check its own break-even gap against its own measured leg cost.

The mathematics. $z_t = (\hat{z}_t - \hat{\mu}_t)/\hat{\sigma}_t$, $\hat{\mu}_t, \hat{\sigma}_t$ rolling over L bars strictly before t. Round-trip cost $c = 4c_{\text{leg}}$ (two legs, entered and exited); trade only if $(z_{\text{in}} - z_{\text{out}})\hat{\sigma}_z > 4c_{\text{leg}}$, $\hat{\sigma}_z$ the spread's stationary standard deviation. [Course Book §21.4]

Artifact. `pair_rules(pair_id, L, z_in, z_out, z_stop, c_leg, breakeven_gap, cleared_flag)`.

Pass / fail. `cleared_flag` false halts the pair before it trades; a leak-injection rerun (pb04) pointed at the z-score pipeline specifically returns to baseline once any full-sample computation is removed.

Trap — The full-sample z-score is a time machine

Standardising with the whole sample's mean and standard deviation tells the strategy, at every bar, where today sits relative to a distribution not known for years — it enters short at

every high and long at every low by construction. The same leak has two disguises: fitting $\hat{\beta}$ once on the full sample and applying it from bar one, and choosing z_{in} after inspecting the full-sample distribution of z. All three must run on a window ending strictly before the decision bar.

Step 10.12 — Breakdown monitoring and structural stops

Do. Every live pair carries three monitors, evaluated on the same bar clock as the signal and logged to the pair-level ledger: (i) rolling Engle–Granger, re-running the Step-2 regression on a trailing window of at least $20\hat{t}_{1/2}$ bars at the same augmentation order, every bar; (ii) CUSUM on the standardised residual $\hat{z}_t/\hat{\sigma}_t$, drift and threshold calibrated to a target average run length under the null of a stable relation — CUSUM catches a persistent mean shift far faster than a rolling ADF, whose power against a small permanent shift is poor; (iii) hedge-ratio stability, logging the standardised innovations of whatever hedge-ratio estimator is in use, flagging a run outside ± 3. Retirement is automatic and pre-registered, not a discretionary call once the numbers exist.

Artifact. `pair_monitor(pair_id, ts, rolling_eg_stat, cusum_stat, innovation_z, alarm_flag)`.

Pass / fail. The pair is closed and quarantined the instant the rolling statistic fails to reject at 10% for twenty consecutive bars, or CUSUM fires, whichever comes first; re-entry requires the full candidate-pipeline re-run from a fresh sample, never a resumption.

Step 10.13 — Pair-level evaluation harness

Do. Run every surviving pair through the full pb04 harness: net-of-cost P&L with four leg tolls and financing on both legs; CPCV with purge and embargo, grouped by constituent name so two pairs sharing a leg never straddle a fold boundary; Deflated Sharpe against N_{eff}; the Probability of Backtest Overfitting from CSCV over the pair-selection procedure itself ([Course Book §15.5]); the leak-injection suite pointed first at the z-score and hedge-ratio pipelines.

Artifact. `trial_ledger` rows tagged `universe='v3-pairs'`, one per pair per configuration, run or abandoned, carrying CPCV path statistics, DSR, PBO.

Pass / fail. Every tested configuration has a row before its result is read; PBO and DSR are both computed against the N_{eff} logged at candidate-pipeline time, never a smaller post-hoc count.

Step 10.14 — Portfolio assembly and shared-leg risk cap

Do. Pairs sharing a leg share risk. Size the surviving portfolio on the pair-return covariance matrix, not pair by pair, and cap aggregate exposure to any single constituent name across every pair that holds it. Report the deployed roster's leg count against the raw screening pool as a standing sanity check on how much the pipeline actually filtered.

Artifact. `portfolio_weights(pair_id, weight, contributing_names)`; per-name aggregate exposure report.

Pass / fail. No single name's aggregate exposure exceeds the pre-registered cap; the deployed roster is a small fraction of the raw pool — at the platform's committed scale, on the order of 40 legs against the 144,650-pair raw pool, a reduction of roughly 3,616× ([Course Book §19.4]; verified: `verify_pb10.py` §1e) — and that ratio is itself evidence the multiplicity discipline,

not luck, produced the survivors.

Stage gate — v3 — the pair set

v3 proceeds to paper trading only if the surviving portfolio's net-of-cost out-of-sample Sharpe ratio, measured on the most recent and most crowded years of the sample, exceeds SR_0 evaluated at the N_{eff} counted before backtesting began, with DSR > 0.95. [Course Book §21.6] This is deliberately stricter than v1's Stage-0 bar of DSR $> 1/2$: pair mining is one of the largest multiple-testing factories in the programme, and the bar reflects it.

Kill criterion — v3 fails

Discard the pair set. Do not re-test it at a shrunk N chosen after seeing which subset would have passed — that is the exact manoeuvre the deflation bar exists to catch. Failing this gate is the expected outcome, not a research failure: it costs a few weeks of compute and avoids funding a decay curve documented for decades in the pairs literature.

Artifacts — v3 output register

`universe_asof` and `corp_actions` tables; `candidate_pairs`; `cointegration_results`; `ou_fits`; the signed N_{eff}/SR_0 ledger snapshot; `pair_rules`; `pair_monitor` log; `trial_ledger` rows tagged `v3-pairs`; `portfolio_weights`; the signed gate memo.

10.3 v4 — Commodities

v4 extends the accounting the platform already needs for the three commodity CFDs inside v1's own 21-instrument roster (XAU, XAG, WTI) and widens the covariate set for a curve-tracked book planned to grow beyond them; it rides the same pb04 harness and reports to the same global ledger, with one extra gate the others do not need. [Course Book §22.5]

Step 10.15 — Roll and financing accounting

Do. Split every commodity CFD's ledger entry into two components, never conflated: the *forecast* component (bar-by-bar mark-to-market on the tracked price — the only component the Stage 0–3 engine has an opinion about) and the *carry* component (the sum of realised roll drag at every roll boundary — a property of the futures curve the engine has no opinion about and never will). Pull both quoted prices (F_1 the expiring contract, F_2 the next) at the moment of each roll; do not assume the curve's current shape persists to the next one.

The mathematics. Per-roll drag $\delta = (F_2 - F_1)/F_1$ for a long position (short is $-\delta$), positive in contango, negative in backwardation. Annualised at 12 rolls/year: simple 12δ, compounded $(1+\delta)^{12} - 1$. [Course Book §22.1]

Artifact. `roll_ledger(instrument, roll_ts, f1, f2, delta, direction)`; P&L attribution split into `forecast_pnl` and `carry_pnl` columns per position.

Pass / fail. Every held position's total P&L reconciles to `forecast_pnl + carry_pnl` exactly; a backtest that cannot produce this split is rejected before it is read — an unaudited roll term compounds to double digits annually and is large enough to *be* the entire result.

Step 10.16 — Cost-surface extension to the commodities book

Do. Extend pb02's cost surface to the futures-linked sessions of WTI, XAU, XAG and any instrument later added to the wider curve-tracked book: widen the assumed spread and margin in the roll window specifically (providers routinely widen both around expiry, and in some cases force an early roll), and flag the days immediately around a physically-settled contract's expiry as a distinct session bucket — an ordinary week's spread data will not see a delivery-mechanics event coming. Verify current specifications for every multiplier, tick size and roll-timing rule against the venue before sizing a position.

Artifact. `cost_surface` rows extended with a `roll_window_flag` session bucket per commodity instrument.

Pass / fail. Cost-doubling control (pb04) applied to the extended surface specifically inside the roll window, not only on an ordinary-week average.

Step 10.17 — Covariate widening: real yields and the dollar

Do. Add real-yield and broad-dollar-index series to the Stage −1b feed registry — `feeds(series_id, reference_period, release_ts, value, vintage_ts)` — with their own release calendar and staleness monitor, exactly as every other feed already in that registry: no covariate reaches the regime router without a stored release timestamp distinct from its value timestamp. Extend the router's observable set to consume them; gold and silver are the natural first test case, since both already carry a documented real-yield relation and a shared USD leg.

Artifact. New rows in the pb03 feed registry; router observable-set configuration updated to include real yields and a broad-dollar index.

Pass / fail. The pb03 as-of join audit returns only values with `release_ts` $\leq t$ for the new feeds, identical to every existing one; a covariate earns its place in the router only if it survives CPCV and the Deflated Sharpe correction on out-of-sample, net-of-cost P&L — the same Stage-2 gate unchanged, not a relaxed one for commodities.

Step 10.18 — Seasonality claims, deflated before they are believed

Do. Test any seasonal claim as a single joint question first — do twelve monthly means differ at all — before asking which month; the moment the question narrows to "which cell", price every cell in the scan as a trial. Scanning the committed v4 roster across the full calendar is not a small search.

The mathematics. $r_t = \sum_{m=1}^{12} \beta_m D_{m,t} + \varepsilon_t$, $D_{m,t} = \mathbf{1}\{\text{month}(t) = m\}$, joint null $H_0 : \beta_1 = \cdots = \beta_{12}$, tested with an HAC-robust F-statistic. Once the question narrows to a specific cell, Bonferroni at m cells: $z_{1-\alpha/2m} = \Phi^{-1}(1 - \alpha/(2m))$. [Course Book §22.2]

Artifact. `seasonality_scan(instrument, month, mean_return, se, t_stat, bonferroni_survivor_flag)`.

Pass / fail. At the committed six-commodity roster, the scan is $6 \times 12 = 72$ cells; nominal 5% expects 3.6 false positives from noise alone, and the two-sided Bonferroni bar is $z = 3.392$ (against an unadjusted 1.960) — no seasonal input reaches the router or the sizing layer without clearing this bar, or without an out-of-sample replication and a mechanism argued *before* the scan. [Course Book §22.2] Verified: `verify_pb10.py` §3.

Stage gate — v4 — commodities

Same CPCV + Deflated Sharpe + PBO ≤ 0.5 net-of-cost bar v1's surviving champion cleared, with two extra conditions specific to this version: (i) roll/financing accounting verified by a cost-doubling-style control applied inside the roll window specifically; (ii) any seasonality input surviving the Bonferroni correction of the seasonality step before it reaches the router or the sizing layer. [Course Book §22.5]

Kill criterion — v4 fails

Do not deploy. If only the seasonality leg fails, drop that specific feature alone and re-gate the remainder — the carry/forecast accounting split is not optional and its failure kills the whole version, but a seasonality input is severable.

Artifacts — v4 output register

`roll_ledger` with forecast/carry P&L split; extended `cost_surface` with roll-window buckets; feed-registry additions for real yields and the dollar index; `seasonality_scan` table; v4 rows in the global trial ledger; the signed gate memo.

10.4 The Shared Spine: Calibration, Cold Start, Rollout

Three operations belong to no single version, because every version depends on them identically.

Step 10.19 — Calibration silos: the non-negotiable per-market rule

Do. For every market version, independently maintain: (i) its own calibration set, drawn from its own exchangeable history; (ii) its own conformal width correction, refit on its own nonconformity scores; (iii) its own cost-model ledger entries, built from its own measured spread, slippage, rollover or roll-yield behaviour. Never let one market's calibration scores enter another's conformal fit, and never let one market's cost surface stand in for another's, even temporarily — an options bid–ask scaling with moneyness and time to expiry has no FX analogue to borrow, and a pairs leg's execution cost is a joint property of two names' liquidity, not one.

The mathematics. No new formula; a validity statement. Conformal calibration and test scores must be exchangeable; scores drawn from two markets with distinct cost structures, tick conventions or volatility regimes are not exchangeable with each other, so a band calibrated on one and applied to another is not a weaker guarantee, it is a claim the coverage theorem simply does not cover. [Course Book §19.5]

Artifact. Per-market `calibration_registry(market, calib_set_id, refit_ts, coverage_check)`; a per-market cost-vintage stamp on every ledger row (pb02's vintage rule, restated here as cross-market law).

Pass / fail. A registry audit finds zero calibration or cost rows whose source market differs from the market they are applied to; this check runs inside every version's own leak-injection rerun, not as a separate one.

Step 10.20 — Cold-start onboarding for a new instrument

Do. When a new instrument joins an already-shipped version (a new options underlying, a new pair leg, a widened commodities book), zero-initialise its instrument-level embedding

row rather than drawing it at random. It inherits its asset-class row exactly, carrying every class-level regularity the trunk has already learned, with zero training steps on the new instrument's own history. This is a statement about the point forecast's representation only — it says nothing about calibration. Run the new instrument through a shadow period, sized and confirmed on its own accruing history, before it trades capital: the cold-start trick buys a good starting point forecast, never a calibrated band on day one.

The mathematics. $z_j = e_j + e^{\text{class}}_{c(j)} (+e^{\text{global}})$, e_j instrument j's embedding row, $c(j)$ its asset class, $e^{\text{class}}_{c(j)}$ the class row, e^{global} an optional single row shared by every instrument. Cold start sets $e_{j^*} \leftarrow \mathbf{0}$, so $z^{(0)}_{j^*} = e^{\text{class}}_{c(j^*)} (+e^{\text{global}})$ exactly, and ordinary gradient descent specialises it from there as data accrues. [Course Book §19.4]

Artifact. New embedding-table row, initialised to zero and logged with an onboarding timestamp; shadow-period record (start date, duration, calibration-set accrual status).

Pass / fail. The new instrument does not trade capital until its own calibration set independently passes the calibration-silo registry audit above; a zero-initialised row that has already drifted away from zero at the shadow period's start is a defect — it means training data leaked in before onboarding was logged.

Step 10.21 — Rollout order and the reconciled gate table

Do. Sequence capital, not research. v1 gates on Stage -1 through Stage 3 in full and has no upstream dependency; every other version waits on some part of it. v2 gates specifically on v1's Stage 1 (pb06) — the calibrated $\hat{\sigma}$ is the whole thesis. v3 does not depend on the volatility engine and may be researched alongside v1, though it still reports to the one global ledger and rides the same harness. v4 rides the harness too, with its own extra roll/seasonality gate. No version's capital allocation precedes its own signed gate memo, and no gate memo is read until pb04's leak-injection suite has passed on the exact code that produced it. [Course Book §22.5]

Artifact. Table 10.4, reconciling this chapter's four gateboxes with Course Book Appendix E's tear-out version rows; Figure 10.1, the dependency graph.

Pass / fail. Every row of Table 10.4 traces to a signed gate memo in this chapter's artifact registers before the corresponding version trades capital.

Version	Depends on	Gate (one line)	Kill action
v1 — FX	none (Stage -1 onward)	Stage 0–3 chain: envelope skill, then conditional width, then regime router, then bespoke model, each on deflated cost-net P&L	Per-stage kill wording, pb05–pb08
v2 — options	v1 Stage 1	Incremental deflated cost-net P&L of the overlay over the v1 book alone	Do not deploy the overlay; v1 keeps the engine
v3 — pairs	none (rides the harness only)	Deflated Sharpe of the mined pair set at the honestly counted $N = \binom{n}{2} \times$ grid, run pair-level, DSR > 0.95	Discard the pair set; never re-test at a shrunk N
v4 — commodities	v1 harness (cost/CPCV/DSR/PBO machinery)	Same net-of-cost bar as v1, roll accounting verified, seasonality Bonferroni-cleared	Do not deploy; drop the seasonality leg alone if that is the only failure

Table 10.4. Rollout order and the reconciled one-line gate table, matching Course Book Appendix E's tear-out sheet. [Course Book Appendix E]

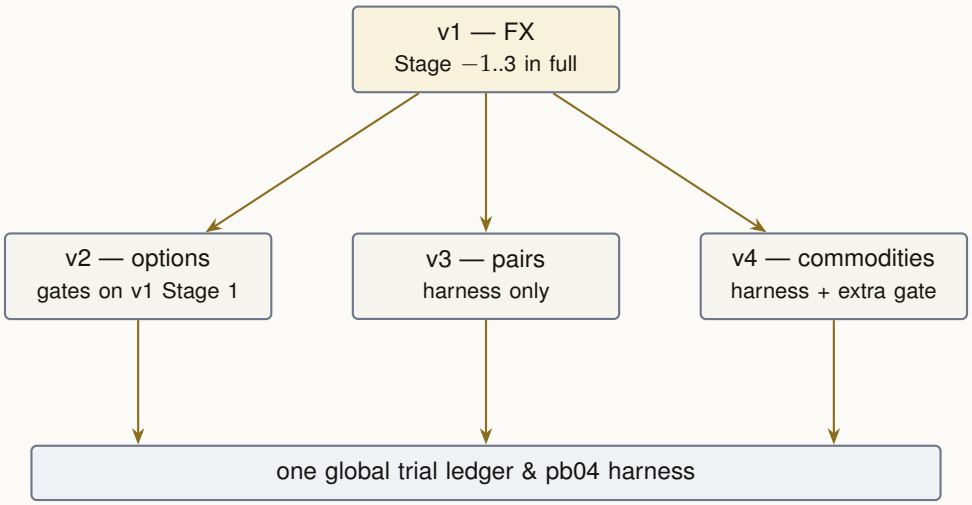


Figure 10.1. Rollout dependency graph: v2 gates specifically on v1's Stage 1; v3 depends only on the harness and may run alongside v1's research; v4 rides the harness with its own added gate. Every version reports to the one ledger. [Course Book §19.6]

PART V

The Overall Steps



LUNA LABS

11

The Overall Steps

Chapters 2–10 are the territory: 97 executable steps with schemas, thresholds and sign-off wordings. This chapter is the map — the entire programme as one numbered sequence you can hold in your head, with the governing quantity of each move stated compactly and the gate that ends each phase. Use it to know where you are; use the stage chapters to take the next step. Formula details and all derivations remain where they live: the step cited in brackets, and the Course Book behind it.

11.1 The three standing rules

Key idea — Rule 1 — Nothing is believed until deflated

Every evaluated configuration — run, abandoned, or superseded — writes a ledger row before its results are read, and every performance claim is judged by DSR $= \Phi\big((\widehat{\mathrm{SR}} - \mathrm{SR}_0)\sqrt{T-1}\,/\,\sigma_{\mathrm{SR}}\big)$, where SR_0 is the expected maximum Sharpe that luck alone hands the best of N ledger trials [Course Book §15.4]. Marginal results are kills.

Key idea — Rule 2 — Kill wordings are signed before results exist

Each stage's kill criterion is written and signed before that stage's first run (Steps 5.8, 6.10, 7.8, 8.5 carry the templates). After results are visible, the wording may not be renegotiated — the Course Book's Chapter 15 is the proof that post-hoc judgement cannot be trusted, including by its author.

Key idea — Rule 3 — A red CI suite freezes everything

The four leak-injection controls (oracle in, Sharpe explodes; labels shuffled, Sharpe dies; timestamps scrambled, Sharpe dies; costs doubled, Sharpe falls reconcilably) must be green for any result to be read [Course Book §18.4]. Red suite, closed book.

11.2 The master sequence

Phase A — Preconditions (Stage -1; Chapters 2–4)

A1. Pull truth prices. Re-pull three years of bid/ask M1 candles for all 21 instruments; the mid-only candles on hand cannot price a fill. Spread $=$ ask $-$ bid, in pips. [Step 2.2]

A2. Build the cost surface. Cell = (instrument, session bucket, news flag, rollover flag); per cell the empirical spread quantiles and a slippage term; round trip $c = \text{spread} + 2\,\text{slip} + \text{commission}$, stressed at $1.5c$ and $2c$. Freeze with a vintage stamp. [Steps 2.1–2.10] *Gate: coverage $\geq 99\%$ of bars in cells with $n \geq 100$; reconciles with realised fills.*

A3. Make every feed point-in-time. One truth schema (series, reference period, release timestamp, value, vintage); the as-of join admits a value at t only if release_ts $\leq t -$ lag. COT gets its Friday-release calendar with holiday shifts; FRED becomes ALFRED vintages; RORO and VolRegime are recomputed backward-only and persisted. [Steps 3.1–3.7] *Gate: the standing audit query finds no future value reachable.* [Step 3.8]

A4. Build the harness. Bar-loop engine ($\mathcal{F}_t$-measurable decisions, $t{+}1$-open fills, cost lookup by vintage); CPCV splitter (N groups, $k = 2$, purge one horizon each side, embargo per timeframe, time slices spanning all instruments; paths $= k\binom{N}{k}/N$); metrics module (per-path net P&L $\to$ Sharpe $\to$ DSR; coverage battery). [Steps 4.1–4.3]

A5. Open the ledger. Schema with config hash, universe, cost vintage, split config, status including *abandoned*; a named owner; write-before-read discipline. [Step 4.4]

A6. Turn the CI suite green. The four controls of Rule 3, wired to run on every harness change. [Steps 4.5–4.9] *Gate: all green. Nothing downstream starts before this.*

Phase B — Stage 0: the honesty test (Chapter 5)

B1. Regenerate the forecast history at live settings. The stored history is partial, stale and at the wrong context; a verdict on it is a verdict about a different model. All 21 instruments, M30/H1/H4, live context, generation config stamped. [Step 5.1]

B2. Declare the policy class. Envelope features (width $q_{90}-q_{10}$, its change, band position, width percentile); a finite grid of linear, bucket and threshold policies — declared in full before any run, because the grid size times the searched timeframes is the ledger N. [Steps 5.2–5.3]

B3. Fix the benchmarks. Constant fractional risk, and volatility targeting $w_t = \sigma^*/\hat{\sigma}_t$ at matched average exposure — the null model any envelope policy must beat [Course Book §14.4]. [Step 5.4]

B4. Run, stress, deflate. CPCV per timeframe at cost vintage; rerun at $1.5c$, $2c$; DSR on the paired difference against each benchmark over the full ledger count. [Steps 5.5–5.6]

B5. Read by the pre-committed table. Pass, kill, or marginal — and marginal *is* kill. [Step 5.7; memo template Step 5.8]

> **Stage gate — Stage 0**
>
> DSR > 0.5 against *both* benchmarks, the worst-decile CPCV path positive, and the result standing at $1.5\times$ costs. Fail $\Rightarrow$ no forecasting model is built on the envelope; the effort redirects to execution and portfolio construction. Pass $\Rightarrow$ the winning policy ships as a size multiplier behind the existing engine, and it becomes the thing Stage 1 must beat. [Step 5.9]

Phase C — Stage 1: the adaptive width engine (Chapter 6)

C1. Choose the vol proxy per timeframe (finer-bar realised variance where it exists; QLIKE as the fit loss). [Step 6.1]

C2. Fit the conditional engines. Log-HAR per (instrument, timeframe) with back-transform correction; GARCH(1,1) alternative with a persistence sanity bound; select per timeframe by out-of-sample QLIKE. [Steps 6.2–6.3]

C3. Manufacture the band. Horizon variance = sum of k-step forecasts; quantiles by scaling ($\sigma\,\Phi^{-1}(\tau)$, or the unit-variance t_ν variant chosen by validation pinball). [Step 6.4]

C4. Wrap it. CQR scores in a rolling buffer; ACI level adaptation $\alpha_{t+1} = \alpha_t + \gamma(\alpha - \mathrm{err}_t)$ with the clip-the-level-not-the-state convention. [Steps 6.5–6.6]

C5. Monitor and shadow. Rolling coverage, PIT buffer, two-tier CUSUM alarms; six-plus

weeks of shadow running beside the frozen `width_mult`, sidecar-owned adaptive table joined at read time. [Steps 6.7–6.9]

C6. Accept per regime. Christoffersen battery inside stress buckets; the Stage-0 policy rerun on the new band, deflated. [Step 6.10]

Stage gate — Stage 1

Stress-bucket conditional coverage recovers (the independence half of the battery no longer rejects where n supports a read), *and* the sizing policy on the new band adds deflated cost-net P&L over the Stage-0 baseline. Calibration-only improvement ships as risk information but does not open Stage 2's P&L thesis.

Phase D — Stage 2: regimes and covariates (Chapter 7)

D1. Assemble the observable vector as-of (VolRegime, RORO, lagged COT z, realised-vol ratio). [Step 7.1]

D2. Fit the router on train only; deploy filtered forever. Baum–Welch with seeds and BIC; live use is the causal forward filter — the smoothed-label ban has its own audit query and CI control. [Steps 7.2–7.4]

D3. Wire the three actions: width-model selection, size throttle, model gate. [Step 7.5]

D4. Run the covariate probe (licence check first; zero-shot in-context inference over the regenerated history; no training). [Step 7.6]

D5. Test increments only. Baseline is Stage 1; every router and probe configuration is a ledger row; deflate. [Steps 7.7–7.8]

Stage gate — Stage 2

Incremental deflated cost-net P&L over the Stage-1 baseline with per-regime coverage not degraded. Fail $\Rightarrow$ the covariates carry no tradeable conditional signal at these horizons, and *Stage 3 is cancelled* — its premise is this gate.

Phase E — Stage 3: the bespoke model (Chapter 8)

E1. Build the dataset (RevIN, patches of 32 over context 1536, typed covariates from the as-of feeds, uniqueness weights). [Step 8.1]

E2. Fix the config (d=256, four blocks, one variate-attention block, monotone five-quantile head; ∼3.8M parameters, budget reconciled to the Course Book's counting). [Step 8.2]

E3. Train on the blended objective — pinball warmup, then pinball + decision-focused net-P&L term with the cost surface inside + CVaR penalty; environment locked; the search sized by the maintenance-window budget, and that size *is* the ledger count. [Steps 8.3–8.4]

E4. Run the six-run gauntlet in order: baseline gauntlet, CPCV vs the Stage-2 ensemble, DSR at the search count, PBO over the search population, per-regime coverage, leak-injection rerun. [Step 8.5]

E5. Freeze, integrate, protect. ONNX with parity tolerance; table-drop integration with a three-level fallback chain; retrain-is-a-new-strategy; rollback protocol. [Steps 8.6–8.9]

Stage gate — Stage 3

Beats the Stage-2 ensemble deflated; PBO ≤ 0.5; the strictly-sequential walk-forward lands inside the CPCV path band. Fail $\Rightarrow$ ship the Stage-2 ensemble.

Phase F — Stage 4 and the market versions (Chapters 9–10)

F1. Stand up the generative tooling off the live path — conditional diffusion sampler, DDIM steps, batch latency measured against the async budget; memorisation guard calibrated real-vs-real; scenario products (path quantiles, drawdown distributions, stop-touch probabilities, path CVaR) feeding sizing and CPCV stress only, ledger-flagged. [Steps 9.1–9.7] *Gate: latency inside budget and incremental tail-adjusted P&L; otherwise it remains offline scenario tooling.*

F2. v2 options, gated on Stage 1: timestamped chains and surfaces; forecast-RV versus quoted IV monetised through the dollar-gamma identity; costs in vol points; CPCV purged by option life. [Steps 10.1–10.5]

F3. v3 pairs: survivorship-safe universe, economic pre-filter *before* testing, Engle–Granger with its own critical values, OU half-life filter, double-leg costs, rolling moments only, breakdown monitors — and deflation at the honestly-counted candidate N. [Steps 10.6–10.14]

F4. v4 commodities: roll and financing inside the P&L engine; cost surface and covariates extended; seasonality claims through the same deflation as everything else. [Steps 10.15–10.18]

F5. Operate the spine: calibration and cost silos per market, one global ledger, class-embedding cold starts with shadow periods, and the rollout order v1 → (v2 after Stage 1) → v3/v4 as capacity allows. [Steps 10.19–10.21]

11.3 The programme at a glance

Phase	Question it answers	Deciding metric	On fail
A (Stage −1)	Can any result be trusted at all?	CI suite green; audits pass	fix; nothing runs
B (Stage 0)	Is the envelope tradeable as sized?	DSR vs both benchmarks	no model is built
C (Stage 1)	Does adaptive width beat frozen width?	regime coverage + deflated P&L	ship calibration only
D (Stage 2)	Do the macro feeds carry signal?	incremental deflated P&L	Stage 3 cancelled
E (Stage 3)	Does a learned model beat the ensemble?	deflated P&L, PBO, walk-forward	ship Stage 2
F (Stage 4)	Do scenarios improve tail sizing?	latency + tail-adjusted P&L	offline tooling
F (v2–v4)	Does each market add to the book?	per-version deflated increment	version parked

Kill criterion — The programme's own kill

If Phase B fails and Phase C yields calibration only, the honest terminal state is: a better-calibrated risk display, a world-class evaluation harness, and *no new forecasting model* — with the effort redirected to execution quality and portfolio construction, where the unexploited variance demonstrably lives. That outcome is a success of the method, recorded as such in the ledger. The harness, not any model, is the asset this programme cannot lose.

Artifacts — What exists when everything has run

Truth prices and a vintaged cost surface; point-in-time feed store with calendars and audits; the harness, CI suite and global trial ledger; the Stage-0 verdict memo; the adaptive band engine with its monitors; filtered regime series and router config; either a frozen bespoke model with its gauntlet reports or a recorded decision not to build one; scenario tooling with memorisation guards; and per-market silos for every version that earned deployment. Every artifact versioned, every decision signed, every number deflated.