THE ASH COLE TRADING SERIES

BOOK 5 — COMPANION 1

Source Code Blueprint

The complete build blueprint for the scikit-learn AI trading bot, module by module

Ash Cole

This companion document is distributed as part of Book 5 of the Ash Cole Trading Series. The code you build from this blueprint is yours; the example snippets reproduced from the book are shared for personal and educational use. No warranty is expressed or implied. Past performance does not guarantee future results. Trade demo first, trade small second, trade live only after the evidence bar from Chapter 10 has been met.

How to Use This Package

This document is the build blueprint for the Book 5 bot. It lays out the full repository you will create as you work through the book — one folder per chapter, each holding the Python modules that chapter teaches you to write, plus a short README and a minimal test harness. Every file referenced in the book is specified here, in the same section order, with a one-paragraph note on what it does and how it connects to the modules around it. This is a specification to build from, not a download.

Read the blueprint alongside the book, not in isolation. As you write each module, give it a header that back-references the chapter and section it came from, and a short README that records the commands needed to reproduce the output shown in the book. The book is always the canonical source: if your build behaves differently from the book, trust the book and debug your code. Questions and errata: agenticedgelab.ai, the address on the last page of this document.

The bot you build runs on any machine with Python 3.11, scikit-learn 1.4 or newer, pandas, numpy, and the MetaTrader5 Python bindings. No GPU is required. No neural network is used. Every model fits in memory on a laptop with eight gigabytes of RAM, and the entire walk-forward pipeline from Chapter 9 completes in under twenty minutes on a modern CPU.

Repository Structure

The package is laid out so that each chapter's code can be inspected, tested, and run independently, while the assembled bot in Chapter 8 imports from every earlier chapter's folder in a single clean dependency graph. There are no circular imports. Every module has a single responsibility. Every folder has a README.

```
/book5_code/
  config.yaml              # master configuration — single source of truth
  requirements.txt         # pinned versions for reproducibility
  README.md                # repository-level orientation
  ch02_data_pipeline/      # collector, cleaner, aligner
  ch03_features/           # ICT feature functions (FVG, OB, sweeps, structure)
  ch04_model/              # labeler, training script, sample_model.pkl
  ch05_sessions/           # killzone scheduler, DST-safe clock, macro windows
  ch06_mtf/                # HTF bias, LTF filter, 3x3 conflict matrix
  ch07_risk/               # ATR sizer, drawdown scaler, correlation cap
  ch08_bot/                # ai_bot.py — the assembled system
  ch09_backtest/           # walk_forward.py, fold_report.py
  ch10_validation/         # monte_carlo.py, df_budget.py, stability.py, stopping_rule.py
  ch11_deployment/         # systemd unit, dashboard (Flask), alerts, kill switch
  ch12_continuous/         # decay detectors, retrain job, shadow A/B, journal
  tests/                   # pytest suite — one file per chapter folder
```

The dependency direction runs strictly forward: Chapter 3 imports from Chapter 2, Chapter 4 from Chapter 3, and so on. Chapter 8 imports from every chapter before it. Chapters 9 through 12 are validation and operational layers that import the bot from Chapter 8 but are not imported by it. This is the acyclic structure the book argues for and the package enforces.

File Header Convention

Every Python file in the package opens with the same header block. The block is there so that any single file can be opened in isolation and the reader can reconstruct its context without having to hunt through the rest of the repository. The header records four things: the chapter and section it comes from, the page number of its first appearance in the book, the module's single responsibility in one sentence, and the minimal set of imports it depends on.

```
"""
ch07_risk/atr_sizer.py
Chapter 7, section "Layer one — volatility scaling"
(page ~195).
Returns a position size in standard lots given
equity, risk %, ATR, and
stop distance. Single responsibility: volatility-
normalized sizing.
Dependencies: pandas, numpy, config.yaml.
"""
```

No file in the package breaks this convention. If you add your own modules, follow the same header — it is the simplest form of documentation that actually survives refactoring.

Chapter-by-Chapter Walkthrough

Chapter 2 — The Data Pipeline

Three modules. collector.py pulls OHLCV bars from MetaTrader 5 using the MetaTrader5 Python bindings, handles broker timestamp offsets, and writes a Parquet file per symbol-timeframe pair. cleaner.py applies the de-duplication, gap-detection, and forward-fill rules from the chapter. aligner.py joins multi-timeframe data on the M5 index using strictly-prior joins so the higher-timeframe value at any M5 bar is the value that was closed before that bar, never the value that was still forming. The strictly-prior join is the most common source of leakage in retail ML bots and the book's discussion of it is the reason this module has its own tests file.

Run order: collector.py → cleaner.py → aligner.py. Output: a single Parquet file per symbol containing aligned M5, M15, H1, H4, and D1 columns, ready for Chapter 3.

Chapter 3 — ICT Feature Engineering

Five feature modules, one per ICT concept. fvg_features.py computes fair value gap presence, size, location within the recent range, age in bars, and fill ratio. ob_features.py computes order block strength, recency, a tested-or-untouched flag, and the per-timeframe hierarchy. sweep_features.py computes

liquidity sweep detection, magnitude, session context, follow-through, and prior swing depth. structure_features.py computes trend direction, break-of-structure and change-of-character counts, and range position. session_features.py encodes the killzone and macro-window flags plus the cyclical time encodings. Each module takes the Chapter 2 aligned DataFrame and returns new columns; none of them drop rows; none of them look at future bars.

The combined feature matrix carries around seventy-six features before the stability filter and Chapter 3's simplicity principle reduce it to the fifteen to twenty that the final model actually uses. The feature-count ceiling and its rationale are in Chapter 10's degrees-of-freedom budget.

Chapter 4 — The Random Forest Classifier

Three modules. labeler.py turns raw bar data into binary labels using the forward-looking rule described in the chapter (did price reach the take-profit before the stop-loss within N bars, where N is a config parameter). train_rf.py runs the training pipeline: load features, load labels, split by time, fit a RandomForestClassifier with 300 trees, maximum depth 8, minimum samples per leaf 20, class_weight balanced, and save the model to disk. evaluate.py runs the held-out evaluation and produces the per-threshold precision and recall table from the chapter. You train this model once in Chapter 4 and save it as sample_model.pkl; Chapters 5 through

8 load that saved file, so you only pay for the training run once.

Suggested settings when you train it: EURUSD M5 over a recent two-year window, the fifteen features selected by stability score, and a confidence threshold around 0.62 to start. Treat the result as an illustrative model for learning the pipeline end to end — never as a production model to trade live.

Chapter 5 — Session-Aware Execution

Three modules. killzone_scheduler.py maps any UTC datetime to one of {Asian, London, NY_AM, NY_PM, NoTrade} according to the exact windows from the chapter. macro_windows.py encodes the full set of seven macro windows from the chapter — London open 2:33–3:00 AM, London close 4:03–4:30 AM, NY AM 9:50–10:10 AM and 10:50–11:10 AM, NY lunch 11:50 AM–12:10 PM, NY PM 1:10–1:40 PM and 3:15–3:45 PM, all New York time — plus the blackout periods around high-impact news events. dst_safe_clock.py is the short utility that converts broker local time to UTC in a way that survives daylight-saving transitions in both the US and Europe without the bot mis-firing on the Sunday of the switch. The tests file for this module is the longest in the package because DST is the quiet killer of session-based bots and every edge case in the chapter is covered.

Chapter 6 — Multi-Timeframe Analysis

Three modules. htf_bias.py computes the D1 and H4 bias from the last N closed bars. ltf_filter.py applies the M5 entry filter. conflict_matrix.py encodes the 3×3 HTF/LTF matrix from the chapter as a lookup table that returns one of {full_size, half_size, block}. The matrix is a table, not a function, because the book's chapter on it is explicit that the rules are discrete and the lookup is deterministic — there is nothing to tune.

Chapter 7 — Adaptive Risk Management

Four modules, one per layer. atr_sizer.py is the volatility scaler. confidence_scaler.py implements the formula risk_pct = base_pct × (1 + (p – threshold) / (1 – threshold)) from the chapter. drawdown_scaler.py implements the three-tier multiplier (1.0 up to 3% drawdown from peak, 0.7 between 3% and 6%, 0.4 between 6% and 10%, zero at 10% or beyond). correlation_check.py computes rolling correlation against open positions and applies the (1 – max_correlation) cap. The layers are composed in the order the chapter specifies: volatility first, confidence second, drawdown third, correlation last. The order matters because reversing it produces silently different sizes on most trades.

Chapter 8 — The Assembled Bot

One module: ai_bot.py. It is the file that imports every earlier module and wires them together into a single run_once() method. The file is just under four hundred

lines including imports, logging, and the defensive exception handling around every external call. The main loop is seventy lines. The arrangement itself is the deliverable, and the book's chapter on this module spends most of its space on the order of operations rather than the code, because the code is simple once the order is correct.

A matched boundary_tests.py file lives next to it and exercises the four boundary cases from the chapter: a model probability exactly at the threshold, a drawdown of exactly 10% from the trailing peak, a half_size flag combined with full model confidence, and a rolling correlation of exactly 1.0 to an open same-direction position.

Chapter 9 — Walk-Forward Backtesting

Two modules. walk_forward.py implements the 24-month train, 3-month test, 50-bar gap, non-overlapping step procedure from the chapter. fold_report.py produces the per-fold table that matches the format in the book, including the illustrative annotation that caps the reader's expectations about their own numbers.

Chapter 10 — Edge Validation

Four modules. monte_carlo.py implements trade-order permutation (not bootstrap) over ten thousand runs, compounding returns, producing the percentile distribution of final equity, maximum drawdown, and

longest losing streak. df_budget.py tracks the degrees-of-freedom count and enforces the budget table from the chapter. stability.py computes the feature stability score as 1 – (std of rank across folds / number of features), clipped to [0,1], with the 0.7 threshold from the book. stopping_rule.py evaluates the three-part criterion (combined Sharpe > 1.2, at least 75% of folds profitable, average per-fold Sharpe > 0.5) and returns a pass/fail plus a human-readable explanation. These three modules together implement the Edge Validation Checklist that Companion 2 reproduces in printable form.

Chapter 11 — Live Deployment

Six files. ai_bot.service is the systemd unit file from the chapter (Restart=always, RestartSec=10). dashboard.py is the Flask app that serves the traffic-light dashboard, the six numbers, the thirty-day equity chart, and the last ten log entries. alerts.py is the three-tier alert router with the thirty-minute per-condition rate limit. kill_switch.py watches for the STOP flag file and handles the clean shutdown. heartbeat.py writes the heartbeat timestamp once per bar. disconnection_recovery.py implements the four-step re-authenticate → reconcile → verify → resume sequence. Everything in this folder is supervised by systemd, and the README shows the exact sequence of systemctl commands to deploy it onto a fresh VPS.

Chapter 12 — Continuous Learning

Five modules. performance_decay.py, feature_drift.py, and regime_change.py are the three decay detectors, each firing independently on the rolling window and threshold described in the chapter. retrain.py runs the quarterly retraining job: pull two years of data, run the Chapter 9 walk-forward, evaluate with the Chapter 10 stopping rule, save the candidate to /candidates/. shadow_model.py runs a candidate alongside the incumbent for thirty days and reports the hypothetical Sharpe comparison. ab_promote.py performs the atomic config swap that promotes a shadow to incumbent (or rolls back). decay_journal.py is the append-only structured log of every retrain event, every detector firing, and every promotion or rollback. Every entry is a single JSON object on its own line (NDJSON), so the file is greppable, tail-friendly, and append-safe across restarts.

Example journal entry (three lines, one per event):

```
{"date":"2026-07-
01","event":"retrain","candidate_sha":"a1b2c3d","ver
dict":"pass","tests":"pass","outcome":"shadow_starte
d"}
{"date":"2026-07-
31","event":"shadow_eval","candidate_sha":"a1b2c3d",
"shadow_sharpe":1.48,"incumbent_sharpe":1.19,"outcom
e":"promoted"}
{"date":"2026-10-
01","event":"retrain","candidate_sha":"e4f5a6b","ver
dict":"fail","failing_criterion":"combined_sharpe","
reason":"combined Sharpe 0.9, below the 1.2
bar","outcome":"archived"}
```

Test Suite

Every chapter folder has a matching tests file under /tests/. The suite is not exhaustive — it covers the boundary cases the book discusses and the specific pitfalls the book warns about, which is a much better use of a retail trader's time than chasing 100% line coverage. Run the full suite with a single pytest command from the repository root. On a modern laptop it completes in under ninety seconds. A green run is a precondition for every deployment and every retrain promotion; the test-pass status is written into the decay journal entry as part of the metadata.

Version and License

Package version: 1.0.0 (April 2026), matching the first edition of Book 5. The code you build from this blueprint is yours; the example snippets reproduced from the book are shared for personal and educational use. Questions, errata, and any updates to this blueprint are at agenticedgelab.ai.

This blueprint is a starting point, not a finished product. Your build will be wrong somewhere. The point of the book is to give you the understanding to notice when it is wrong and the discipline to fix it before real money moves.