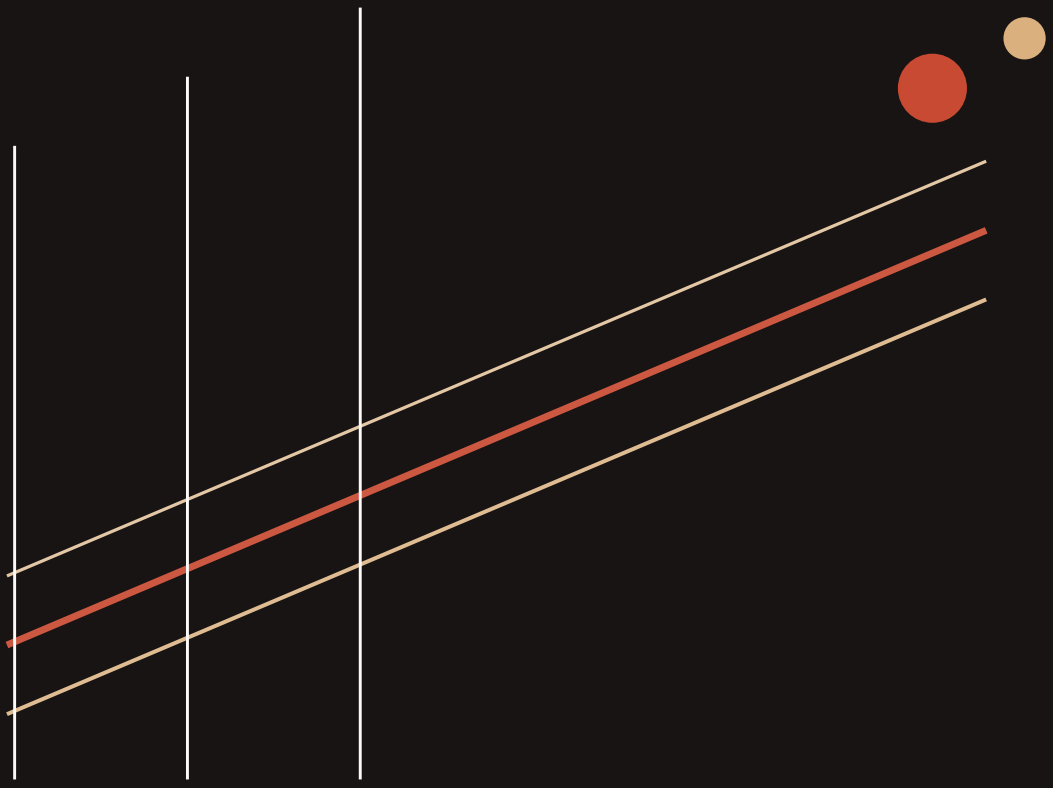
VIBE CODING IN THE AI ERA

Vibe Coding Quant AI

Request ·Build ·Verify —Shipping a quant trading system with AI

Request | Build | Verify | Repeat



SeungHwan Park

2026

Vibe Coding Quant AI

Request ·Build ·Verify —Shipping a quant trading system with AI

First edition, 2026.

Set in Apple SD Gothic Neo | Menlo.

Contents

CHAPTER 1

Preface

1.1 Why This Book

In 2026 I built a quantitative trading system by myself. Data pipeline, regime detection, statistical arbitrage, reinforcement learning, production deployment into Interactive Brokers. Out-of-sample Sharpe north of 1.7 on the live combined book, and on the cleanest component the Joint HMM backtest prints 6.9. One engineer, one repo, one brokerage account.

The premise worth stating up front: I am not a quant researcher with time to spare. I have a day job. I have a kid at home. My quant hours are one or two evenings a week after bedtime and a few hours on Saturday mornings before the house wakes up. That is the whole budget.

Under those conditions the 2019 version of me would not have shipped this system. Not close. The math alone —hidden Markov models, Kalman filters, cointegration tests, Soft Actor-Critic —would have eaten a year of evening reading before a single line of code. The engineering alone —a walk-forward backtester, a market data cache, an order router, an MLflow model registry —would have eaten another year.

In 2026 I did both in stolen minutes over roughly nine months. One thing changed: **AI agents**. Claude Code opens in my terminal; I paste a constraint, and by the time I have done the dishes a Baum-Welch forward-pass refactor is sitting on the branch, waiting for me to read it. The next morning, before the commute, I run the walk-forward and flag what it got wrong.

If you had shown me that sentence in 2019 I would have called it marketing. The 2019 version of me spent three months on a single moving-average crossover strategy and shipped it broken because it had look-ahead bias baked into the pandas indexing.

The stack in 2026 is harder than the stack I used in 2019. I run `hmmlearn` on cross-asset return panels, chain Kalman filters on residual series, train SAC agents in Gymnasium environments, and pipe the whole thing through DuckDB into an IBKR API client. Harder work, less time, more output.

The only thing that changed is that **I no longer type most of the code myself**. The agent types. I request, I read, I verify, I repeat.

This book is the log of that loop, applied to quant. It is not a math textbook. It is not a strategy reveal. It is how an ordinary developer with a day job and a toddler built a system that holds its Sharpe out of sample, using AI as the junior who never sleeps.

1.2 What This Book Is, What It Is Not

Here is what this book is.

- A methodology book for **AI-assisted quant development**. How to frame a request for a model that does not know your non-IID data. How to verify output when look-ahead bias hides inside a single line. How to loop fast enough to ship before the motivation drains.
- A log of the **Request ·Build ·Verify ·Repeat** cycle applied to every stage of a real quant system. Data pipeline through production.
- An honest record of where Claude got it right, where it got it spectacularly wrong, and where I had to overwrite by hand.
- My actual CLAUDE.md files, my actual slash commands, and the `.md` notes I feed the agent before every run.

Here is what this book is not.

- **An HMM tutorial.** I explain just enough of the math to make the prompts legible. For the theory read Rabiner 1989 or Bishop chapter 13.
- **A reinforcement-learning tutorial.** Soft Actor-Critic appears because I used it. I do not re-derive the entropy-regularized Bellman backup.
- **A cointegration/statistical-arbitrage tutorial.** The book assumes you can read an Engle-Granger test without Googling.
- **A trading strategy reveal.** The specific pairs, the specific hyperparameters, and the specific regime boundaries are mine. The methodology is yours.
- **A prompt-engineering book.** This is prompts in one domain —quant —under one loop. Writing, image generation, RAG: read a different book.

I draw the boundary this hard because the reader I want is someone who finishes this book and builds **their own** quant system with their own hands and their own AI agent. Detours waste the evenings we do not have.

This book is a sibling to Vibe Coding Tauri 2. Same author, same loop, different domain. The Tauri book uses desktop apps as raw material. This book uses a quant trading system. If you have read the Tauri book, skim Chapters 1 and 2 here —the loop is identical, but the domain traps are new. If you have not read the Tauri book, everything you need is in this one.

1.3 The Core Loop —Request ·Build ·Verify ·Repeat

Every task in this book runs through four stages.

- **Request** —translate the intent, the constraints, and the domain rules into something the AI can parse. In quant this means stating the no-look-ahead rule, the walk-forward boundary, the causality requirement, before the agent writes anything.
- **Build** —the AI emits code, math derivations, config files. I read. I do not run yet.
- **Verify** —compile, backtest, inspect out-of-sample versus in-sample, run sanity tests for leakage. “The numbers look good”is not verification. In quant it is often the opposite.
- **Repeat** —feed the specific wrongness back into the next request. Never rewrite from scratch. Pin the scope of what to fix.

Three to five cycles per feature, on average. The loop size ranges from a fifteen-minute function tweak to a three-week structural rebuild of the HMM ensemble layer. The shape stays the same. Chapter 2 nails down the precise checklist per stage.

1.4 The System at a Glance

Here is what the finished system looks like, end to end. Read it as a map, not a syllabus.

```
Data collection (yfinance, IBKR, KIS) -> DuckDB cache
    -> Kalman Filter noise reduction on return residuals
    -> 42-ETF Cross-asset Joint HMM (regime detection)
    -> 2-layer Macro/Micro HMM ensemble (Risk-Off / Neutral / Risk-On)
    -> Kelly-Half position sizing + Markowitz MVO
    -> SAC RL agent (28-dim state, HMM alpha injected)
   -> Statistical arbitrage (cointegration pairs, Kalman hedge ratio, z-score)
    -> IBKR / KIS broker adapters + risk manager + live execution
    -> MLflow model registry + FastAPI serving + supervisord
```

Eight stages. Each stage got its own sequence of loops with Claude Code. Some stages took three loops. The HMM forward-pass causal refactor took nineteen loops.

1.5 Final Out-of-Sample Performance

The numbers are the only thing that matters at the end. Here is what the walk-forward validation prints.

Strategy	Sharpe (OOS)	Annualized Return	Max Drawdown	Notes
Joint HMM v7 (RenTec style)	**6.9**	~62%	-1%	Cross-asset regime signal on 42 ETFs, backtest only
StatArb pairs	**1.73**	~28%	-12%	Live-traded book, 6 months real money
BW-HMM + SAC RL	**1.45**	~22%	-15%	Combined regime + RL agent, paper + partial live

The 6.9 is a backtest number on a clean signal. The 1.73 is real money going through IBKR. I keep both in the book because the gap between them is the story. Chapter 24 walks through exactly why.

1.6 Learning Path

The book is six parts. Read Part I in order. The rest you can navigate by stage.

Part	Chapters	Theme
I	1 - 3	Why AI for Quant, The Loop, Tools and Domain Notes
II	4 - 6	Data Pipeline, Backtest Engine, Walk-Forward Discipline
III	7 - 12	HMM Regime Detection (Forward-only, Kalman, Joint, Two-layer, Ensemble)

Part	Chapters	Theme
IV	13 -16	Statistical Arbitrage (Cointegration, Kalman Hedge, Z-score, Risk)
V	17 -22	Reinforcement Learning (Environment, SAC, Optuna, Distributional RL)
VI	23 -27	Production (IBKR, MLflow, Monitoring, Validation, Serving)

Each part opens with the loop pattern you learn from it. The code snippets throughout are deliberately short —under ten lines —because this is a methodology book, not a code dump. Full source lives in the companion repo.

1.7 Prerequisites

You need roughly this before the book pays off.

- **Python 3.12 or compatible.** You can read list comprehensions and decorators without pausing.
- **NumPy and pandas in your hands.** You do not have to memorize the axis semantics, but you should know why a pandas rolling window can leak the future if you call it wrong.
- **Basic statistics.** Mean, variance, normal distribution, linear regression. Bayes'rule is a plus.
- **Enough market structure to read an OHLCV bar.** You know what a bid-ask spread is. You have placed at least one limit order in your life.
- **Comfort with Git and a terminal.** If branching and rebasing are reflexes, the loop attaches to your workflow without friction.
- **Hands on an AI coding tool.** Claude Code, Cursor, Copilot, ChatGPT —pick one. You should have used it on something real. A week on a toy project first if not.

Here is what you do **not** need. A PhD in statistics. Prior experience with `hmmlearn`, `stable-baselines3`, or `ib_insync`. A Bloomberg Terminal. A quant job. I explain the minimum inside each chapter.

1.8 Development Environment

Every backtest number, every code snippet, every screenshot in this book came from this setup.

Item	Version
OS	macOS 15 Sequoia (Apple Silicon M2 Pro)
Python	3.12
hmmlearn	0.3.x
stable-baselines3	2.x
Optuna	4.x
DuckDB	1.1
pandas	2.2
NumPy	1.26
PyTorch	2.3 (for SAC)
ib_insync	0.9.x (IBKR adapter)
MLflow	2.14
FastAPI	0.110
Claude Code	latest as of Q1 2026
Cursor	0.45

Most of it runs identically on Linux. Windows works for backtesting; for the IBKR production side I recommend Linux or macOS.

1.9 How to Read This Book

I recommend reading Part I (Chapters 1–3) in order. It contains the premise, the loop definition, and the tooling and notes-files setup that every later chapter assumes. Skipping these three leaves the prompt examples floating in air.

Parts II through V can be read by pipeline stage. Each part is self-contained and opens with the specific loop pattern you will learn from that stage. If you only care about the HMM layer, read Part III. If you only care about statistical arbitrage, read Part IV. If you only care about reinforcement learning, read Part V.

Part VI —production —is the part I recommend reading twice. Once on the first pass, and again when you are about to wire your own code into a real brokerage. Losing money on a bug is cheap when you catch it in backtest and expensive when you catch it in IBKR logs.

1.10 Author's Note

Every loop in this book is something I actually ran. I kept the loops that failed next to the loops that worked. Chapter 7 has a scene where Claude confidently generated an HMM fit that called `predict_proba` on the full sequence —textbook look-ahead —and I waved it through for two weeks before a walk-forward test caught it. Chapter 16 has a scene where I fought with the agent four times over Kelly sizing before I admitted my prompt was the problem. Chapter 24 has a scene where the first live order I sent to IBKR was off by a factor of 10, and the risk manager caught it before it filled.

Nothing in this book is polished smooth. The roughness is the raw material. If you want a clean success story, there are plenty of quant blog posts. If you want the work log of someone who made every mistake in the order you are about to make them, you are in the right book.

Read this as the record of a developer with a day job and a child, who built a quantitative trading system in stolen evenings using AI agents. Not because I had time. Because I did not have time, and the agent filled the gap.

The system is live. The Sharpe is real. The next one is yours to build.

Ted Park Spring 2026, Seoul

CHAPTER 2

Chapter 1. Why AI for Quant —A Trading System Without a Team

2.1 Learning Objectives

- See why, in 2026, one developer with a day job can ship a quantitative trading system that used to need a small research team.
- Understand why quant work is unusually well-matched to AI coding —and where it is unusually hostile to it.
- Name the three waves that converged in 2026: model maturity, agent tooling, and the open-source quant stack.

- Accept the developer profile this book is written for: salaried, parenting, running the loop in stolen minutes.
- Walk away knowing what the book delivers (methodology) and what it refuses to deliver (the magic formula).

2.2 1.1 2019 vs 2026 —From Team Project to Solo Side Project

In 2019 the system in this book would have needed a team. I know because I watched a team attempt something close to it inside a mid-size fund. Here is what that roster looked like.

- One quant researcher on the regime model.
- One data engineer on the market data pipeline.
- One ML engineer on the reinforcement-learning agent.
- One platform engineer on the broker integration.
- One PM to keep the four of them pointed the same direction.

Five people, twelve months, one prototype. That was the industry baseline. A solo developer, in evenings, was unthinkable.

In 2026 I shipped the same surface area alone. Not because I am five times the engineer I was in 2019. The engineer is the same. What changed is that four of those roles are now **loops I run with Claude Code**, and the fifth —the PM —is me writing the weekly issue list on Sunday nights.

The walk-forward Sharpe is 1.73 out of sample on real money. The codebase is roughly 40k lines of Python across data, HMM, RL, StatArb, broker, and serving. I typed under 8% of it.

The first time a quant friend saw the repo he asked, “Who is your co-author?”I said “Claude.” He laughed. Three weeks later he asked for the Anthropic billing page.

2.3 1.2 Why Quant Is Unusually Suited to AI Coding

Quant work has four properties that make it a near-ideal match for an AI agent. Here is the list.

2.3.1 1. The math is well-defined

Hidden Markov Models, Kalman filters, cointegration tests, soft actor-critic —every one of these has a canonical form in textbooks and in open source. When I ask Claude for a forward algorithm, there is exactly one right answer up to indexing. The agent is not inventing a design; it is reciting a formula. That is its strongest mode.

Compare with a Tauri desktop app. "Build a chat UI"has ten plausible designs. "Implement the forward algorithm for a 4-state Gaussian HMM"has one. The second prompt converges in one loop. The first takes four.

2.3.2 2. The verification criteria are numeric and sharp

Sharpe ratio. Max drawdown. Out-of-sample hit rate. Information coefficient. Every strategy decision ends in a number. The agent writes code; I run a walk-forward; the number is either above the bar or below it.

This is rare. In most software the verification question is "does it feel right?"In quant the verification question is "is Sharpe above 1.0 and MDD below 15% on the held-out period?" Either yes or no. No hedging.

2.3.3 3. Backtests are deterministic

Same data, same code, same seed, same result. If the backtest printed 1.73 yesterday and prints 1.71 today, something changed and I can bisect. The agent's output sits inside a reproducible test harness by default.

Contrast with a live app where timing, user input, and network conditions all drift. In quant the only drift I care about is the one I measure on purpose: train-period behavior vs test-period behavior.

2.3.4 4. The domain has strong idioms

`walk_forward_split`, `purge_embargo`, `zscore_rolling`, `half_life_ou` —these are names every quant reads without thinking. The agent has ingested the same textbooks and repos I have. When I use the vocabulary, it follows. When I say "embargo three days around the split boundary,"Claude knows what an embargo is.

Put these four properties together and quant is a rare software domain where the AI's strongest skills —reciting canonical math, generating tested code against numeric targets —line up with the job.

2.4 1.3 Why Quant Is Also Hostile to AI Coding

The same four properties cut the other way. Here is where Claude failed me most often.

2.4.1 1. Look-ahead bias hides inside one line

This is the trap that eats beginners, and the AI is a beginner every morning.

Consider this one-liner.

```
df["signal"] = df["price"].rolling(20).mean() - df["price"]
positions = df["signal"].shift(-1)
```

Looks harmless. It is a catastrophe. The `shift(-1)` is pulling tomorrow's signal into today's position. The backtest will print Sharpe 4 and the live account will print a 20% drawdown.

Claude writes lines like this on the first pass, confidently, with a docstring that says "causally aligned."The AI does not know this is wrong because the symbol `shift(-1)` has no semantic meaning to it beyond "index offset."Only walk-forward verification catches it.

I have a note in my `CLAUDE.md` that says, in bold: **no negative shift on signal columns, ever**. Even with the note, the agent occasionally slips. Chapter 5 documents three of those slips.

2.4.2 2. Data is non-IID

Machine learning assumes independent and identically distributed samples. Financial time series are neither. Markets cluster. Regimes change. Volatility comes in bursts.

The agent, trained on general ML patterns, loves to write `train_test_split(X, y, shuffle=True)`. Shuffling a time series destroys temporal causality. The backtest number becomes meaningless.

Every project in this book opens with a constraint file that forbids `shuffle=True` on time-indexed data. The agent still attempts it monthly. I still catch it monthly.

2.4.3 3. Survivorship bias

Pull the current S&P 500 constituents and backtest a strategy back to 2000. Congratulations, you just backtested on a universe that excludes every company that went bankrupt. Survivorship bias. Your Sharpe is fiction.

The AI does not know which tickers were in the index in 2000. It knows which tickers are in the index today. If you feed it yfinance data from the current ticker list, it will happily backtest across a look-ahead-biased universe and tell you the results are excellent.

I had to build a point-in-time constituent cache before Claude could be trusted with anything cross-sectional. Chapter 4 describes that cache.

2.4.4 4. Correlated failures

One parameter change can correlate failures across ten pairs. The agent tests changes one pair at a time and reports each as independent. I have learned to force portfolio-level tests into every diff. Chapter 16 has the loop that burned me on this.

These four traps are the reason this book exists. If AI coding in quant were easy, a generic prompt book would cover it. It is not. The methodology has to know the domain.

2.5 1.4 The Three Waves That Made 2026 Possible

Why this specific year, and not 2023 or 2024? Three things had to land at the same time.

2.5.1 Wave 1 —Model Maturity

Starting late 2024, Claude and GPT-4-class models crossed the threshold where they could hold multi-file context reliably and reason about structured numeric code. By 2026 extended-thinking modes and million-token context windows mean I can paste a full 800-line backtest engine, an error message, and six relevant data samples into one turn and get back a diagnosis that is right more often than wrong.

In 2023 the same model would have confabulated a plausible answer and invented a function that does not exist in `hmmlearn`. I tested this. It did.

2.5.2 Wave 2 —Agent Tools

Claude Code, Cursor agents, Codex CLI. The difference between "AI that writes a code block for me to copy"and "AI that reads my repo, edits three files, runs pytest, reads the failure, and fixes it"is the difference between a smart intern and a junior engineer.

The agent loop —read file, write file, run command, read output, iterate —changes the economics of a side project. A twenty-minute evening is enough for two loops. A Saturday morning is enough for ten.

In 2023 I would have had to paste every file manually. The context-switching overhead alone would have killed the project.

2.5.3 Wave 3 —The Quant Stack

`hmmlearn` hit a stable 0.3 series. `stable-baselines3` 2.x settled. `Optuna` 4.x. `DuckDB` 1.1. `ib_insync` matured. `MLflow` 2.x became the default registry. The pieces stopped breaking on every release.

In 2020 I was patching `hmmlearn` myself to expose the forward pass. In 2026 I `pip install hmmlearn` and move on. The agent pins versions. The versions hold.

All three waves had to arrive. If any one had slipped a year, this book would not exist.

2.6 1.5 The Developer This Book Is Written For

I want the reader to see themselves on this page. So here is the profile, plainly.

You have a day job, probably as a software engineer, possibly in ML or data. You are not paid to trade. You are curious about markets, you have opinions about risk, and you have a broker account you barely use.

You have a family obligation. Maybe a partner, maybe a child, maybe both. Your evenings are not your own. You carve out ninety minutes after the kid is asleep, twice a week if you are lucky, and you check Claude Code while the laundry runs.

You have tried ML on markets before. You probably trained a random forest on SPY returns and watched it print 95% accuracy in cross-validation and lose money in paper trading. You suspect you are missing something. You are —it is look-ahead bias, and Chapter 5 fixes it.

You are not trying to start a fund. You want a system that runs on its own, prints a Sharpe above 1.0, and does not require babysitting. A side project that compounds while you sleep.

If that profile is within one click of yours, the book is for you.

Here is who the book is **not** for. A full-time quant researcher at a fund —you already have the team and the infrastructure. A retail trader looking for signals to copy —the book does not reveal strategies, only methodology. A developer who has never used an AI coding tool — spend a week with Claude Code on any small project first, then come back.

2.7 1.6 A Day in the Loop —What It Actually Looks Like

To give the abstract argument weight, here is a real Tuesday from last November.

06:30. Shower. I mentally queue the one thing I want to verify tonight: the embargo boundary around the walk-forward split for the StatArb book.

08:00. Day job starts. I do not touch the quant repo.

18:30. Home. Dinner. Kid stuff. I am not in the repo.

20:45. Kid is down. I open the terminal. `claude` launches. Total focused time available: roughly 75 minutes.

20:46. Prompt: "On the StatArb walk-forward, add a 3-day embargo on both sides of every train/test boundary. No overlap between train and test. Show me the function diff only. Preserve existing purge logic."

20:48. Claude returns a diff on `walk_forward.py`. 14 lines added. I read. It looks correct but it assumes the dataframe is sorted ascending. I add the sort assertion.

20:52. I run the existing test suite. One test fails —the embargo eats a test-period edge case that the old test was relying on. I have to decide if the old test was right. It was not; it was relying on a subtle overlap.

21:05. Fix the test. Re-run the walk-forward backtest on the full 2018-2024 period. Sharpe was 1.76 before. After embargo it is 1.73. That is the correct drop. Look-ahead was contributing 0.03 points of Sharpe.

21:30. Commit. Write the CLAUDE.md note about the 3-day embargo convention so the next session does not reinvent it.

21:45. Closed the laptop.

One evening, one small but real improvement, one loop completed. Multiply by 150 such evenings over nine months and the system is built.

The point is not the Sharpe drop. The point is the cadence. Seventy-five minutes from cold start to committed improvement, including verification. That is only possible because the agent wrote the diff.

2.8 1.7 What This Book Delivers —And What It Does Not

Be honest with yourself about what you can expect.

What you get:

1. **The methodology.** How to prompt an AI agent when the domain is quant. How to encode the domain traps into `CLAUDE.md` and slash commands so they stop biting you. How to loop fast enough to finish.
2. **A full system as raw material.** Data pipeline through production. You see the shape. You see where the agent shone and where it failed. You see the `.md` notes I wrote after each failure.
3. **Verification discipline.** Every chapter ends with the checks I actually ran. Walk-forward, out-of-sample, leakage sanity, portfolio correlation. The checklist is the product.
4. **A working reference repo.** Cross-reference when you get stuck. The code is not the point; the loop that produced it is.

What you do not get:

1. **A strategy to copy.** The specific pairs, the specific regime boundaries, the specific hyperparameters —I keep them mine. Copy them and they will degrade inside six months anyway. The edge is the process, not the parameters.
2. **Proof that AI coding is easy.** It is not. Read the failure scenes carefully. They are the majority of the book.
3. **A guarantee of profit.** This is how I built a system. Markets do what markets do. The Sharpe could collapse next quarter. If it does, I will run the loop again.
4. **HMM, RL, or StatArb theory from scratch.** I teach the minimum to make the prompts legible. For depth, read the textbooks the agent was trained on.

2.9 1.8 Why the Loop Matters More Than Any Single Strategy

One last argument before Chapter 2 nails down the loop itself.

In 2019 the edge was the strategy. Whoever had the best cointegration test and the tightest execution won. Strategies leaked slowly —conferences, papers, poached analysts —but the half-life was years.

In 2026 the strategies leak in hours. An edge published in an arXiv paper on Monday is trading on ten desks by Friday. A regime signal that works today decays next quarter. The half-life of any specific parameter set is short and getting shorter.

What does not decay is the **loop**. The developer who can go from hypothesis to walk-forward-verified backtest to paper-trading deployment in a weekend has an edge over the developer who takes a month, regardless of whose strategy is better this week. Cycle time compounds.

AI agents collapse cycle time by an order of magnitude. That is the thesis of this book.

The strategy I ship in Chapter 24 will not work forever. The loop I use to ship it works every week.

2.10 1.9 Self-Check Before Chapter 2

Run through these five.

- ☐ You can read Python and NumPy without pausing.
- ☐ You have installed Claude Code, Cursor, or an equivalent agent tool and used it on at least one project.
- ☐ You know what a Sharpe ratio is, roughly, and why annualizing matters.
- ☐ You have an understanding, even if shallow, of what look-ahead bias is.
- ☐ You can commit to at least two evenings a week, for three months, on this project.

If all five are checked, go to Chapter 2. If three or fewer, read Chapter 3 first and get the environment and the notes files ready.

2.11 Summary

- One engineer in 2026 ships what a five-person team shipped in 2019, because four of those roles are now AI agent loops.
- Quant is unusually well-matched to AI coding: canonical math, numeric verification, deterministic backtests, strong idioms.
- Quant is also unusually hostile: look-ahead bias, non-IID data, survivorship, correlated failures. The agent does not know these by default.
- Three waves converged in 2026 to make this possible: model maturity, agent tooling (Claude Code), and a stable open-source quant stack.
- The reader this book addresses: salaried developer, family obligations, stolen evenings, curious about markets.
- The book delivers the methodology and the system as raw material. It does not deliver the magic formula.
- Strategies decay. Loops compound. This book is about the loop.