 DOWNLOAD PDF

What algo trading actually takes

six months · 700+ hours · six futures markets traded by machine, 24/5

> None of this is the strategy. The quant research is what everyone pictures when they think about algo trading, and it was maybe a quarter of the hours. This is the other three quarters: the infrastructure, the failures, what each one cost, If you are building your own automated setup, these are the potholes we already hit so you can drive around them.

Somewhere between 700 and 1,000 hours over six months, most days ending past midnight, and the list below is still mostly things we found out by losing money or sleep. That is worth saying plainly because of how much advice is out there from people who have never run a system live. Backtesting is not running it. A strategy that works on paper and a strategy that survives a 5am network flap, a rollover prompt, a half-written file and a platform that says "Connected" while the feed is dead are different projects, and the second one is most of the work.

The test we apply to anyone offering guidance, ourselves included: ask what broke last month and what it cost. Anyone who has actually run this has a long, specific, slightly embarrassing answer. Anyone who does not has been backtesting.

THE ORDER WE DID IT IN

If we started over, this is the sequence. Each stage exists because skipping it is how people lose money on a system that never had an edge in the first place, or had one and could not run it.

1. Get data you trust before you test anything

Before a single strategy idea, get years of minute bars and prove they are correct. Check the timezone. Check what happens at contract rollovers. Check the session boundary, which for futures is not midnight. We found bugs in our own data months in, and every result computed before those fixes was garbage. Data first, always, and verify it against a second source if you can.

2. Test the boring question first: is there anything here at all?

Not "how much would this make." The first question is whether the thing you noticed happens more than chance. Build the control before the strategy. If a pattern cannot beat a shuffle test, nothing downstream matters and you have saved yourself months.

3. Simulate the path, with costs, across every year

Once something survives a control, make the simulation honest: walk the price bar by bar, charge commissions, charge slippage, and split the results by year. Most ideas die here, and they should. An edge that only worked in one year is a regime you already missed.

4. Build the plumbing before you risk anything

This is the stage everyone skips. Data feed, order routing, position tracking, logging, monitoring, alerts. Get all of it running with zero risk attached and let it run for weeks. You are not testing the strategy here, you are finding out how your platform behaves at 3am and what happens when your internet blips.

5. Paper trade the actual system, not the idea

Run the full stack on simulated accounts, in real time, for at least a month. This is where you find that your backtest assumed fills you do not get, that your exits fire late, that your platform disables strategies after a reconnect. Compare the paper results to what the backtest predicted for the same days. If they disagree, find out why before you continue.

6. Go live at a size that cannot hurt you

One micro contract. Not because it is meaningful money, but because live money finds problems paper never will: rejected orders, margin edge cases, and the psychological reality of watching it work. Stay there until the live results match the paper results.

7. Scale only on evidence, and only one variable at a time

More size, more markets, more strategies: one change at a time, with enough time between changes to attribute the results. If you change three things and the equity curve turns, you have learned nothing about which one did it.

THE DISCIPLINE UNDERNEATH ALL OF IT

Do not trust profit as proof. A strategy can make money for weeks for reasons that have nothing to do with why you think it works, and then give it all back when that reason disappears. We test whether the thing that supposedly generates the edge is actually

generating it. Several of our best-looking results died to that question, and the ones that survived it are the ones still running.

THE MACHINE

Give it a dedicated box, wired

Trading software gets its own machine and nothing else. No gaming, no browsing, no updates you did not schedule. Wired ethernet, never wifi. A wifi hiccup you would not notice while browsing is a dropped connection that disables strategies and leaves positions unmanaged.

Windows will put your trading platform to sleep

Default power settings are built for laptops that should nap. Set the power plan to High Performance, disable USB selective suspend, disable "turn off hard disks," and turn off network adapter power management. That last one has its own checkbox buried in Device Manager and it is the one that bites: Windows powers down the NIC to save energy, the platform silently loses its feed, and everything looks fine on screen.

Cloud sync folders will corrupt your platform

WHAT IT COST US

Enabling OneDrive on the folder holding our platform's data destroyed a working day. Corrupted the historical data archive, crashed the platform on every compile because sync grabbed a DLL mid-write, and silently blocked our data exporter from writing files.

Trading platforms write to their database, logs and DLLs constantly. Any actively syncing folder is a race condition against your own software. Uninstall the sync client or move the platform outside the synced tree. Do not settle for pausing it.

THE FAILURE NOBODY WARNS YOU ABOUT

Silent stalls, not disconnects

Everyone builds for the disconnect. The platform says "Connection Lost," you reconnect, you move on. The failure that actually costs money is the **silent stall**: connection status stays green, order routing still works, and the market data feed is frozen solid. Your charts stop updating. Your automation keeps running on a dead tape.

> **WHAT IT COST US**
>
> Nine hours once, two hours forty-three minutes another time. Both overnight, both with the platform reporting "Connected" the entire time. We only found the first one when a human looked at a chart in the morning.

Two things to know. First, a reconnect command on a connection that already claims to be connected is a no-op. Only a full **disconnect, then reconnect** revives the feed. Second, and this is the part that took us longest: status monitoring can never catch this, because status is fine. You have to watch the **data**.

Watch the bars, not the file timestamps

We wasted a lot of time trusting file modification times. Dashboards, exporters and loggers all rewrite their files on schedule, so a file that was updated ten seconds ago can be full of data from nine hours ago. The only honest staleness check is reading the **content**: what is the timestamp on the newest bar, and how far is it from the clock right now?

BUILD A LAYERED WATCHDOG STACK

One watchdog is not enough because the failure modes are different shapes. We run three layers and each one catches things the others structurally cannot.

- **Connection watchdog.** Watches for hard disconnects and reconnects. Runs as a background timer, not a UI timer. Our first version used a UI dispatcher timer and it starved on a loaded machine, sat silent through two real drops, and logged nothing but "armed."
- **Data watchdog.** Timestamps every incoming bar. If the connection claims connected but no data has arrived in 120 seconds during market hours, it forces a disconnect and reconnect. This is the one that catches silent stalls.
- **Process supervisor.** Watches the other programs. If a component stops updating its own status file for three minutes, kill and relaunch it. Rate limit the healing, because a program that fails to start becomes a restart loop that hides the real problem.

THE GAP THAT GOT US ANYWAY

Our data watchdog was written, installed, and *not enabled* during a two-hour stall. It was built as a strategy, which has to be attached to a chart and turned on manually. It was not. Whatever you build, verify it is actually running by making it log a heartbeat, then check the heartbeat.

Everything writes a heartbeat

Every component we run writes a small file every cycle with a timestamp and its current state. A stale heartbeat file answers "is this thing alive?" without guessing from process lists. Process alive does not mean process working: we have had components sit at zero CPU, holding their port, frozen, looking perfectly healthy in Task Manager for forty minutes.

Raise your platform's auto-restart limits

Most platforms will restart your strategies after a connection loss, up to a limit. Ours defaulted to 10 restarts in 30 minutes. A network flap at 5am exceeded it, the platform auto-disabled every strategy including the data exporters, and the automation went blind for two hours with open positions. Find that setting and raise it to something like 50 in 60 minutes.

ALERTING THAT ACTUALLY WAKES YOU UP

We use three channels on purpose, because one channel means either alert fatigue or missed emergencies.

- **Push notification app** (we use Pushover) for genuine emergencies only. It supports a priority level that pierces silent mode and repeats every 60 seconds until you acknowledge it in the app. That is the setting that gets a human out of bed. Use it sparingly or you will start ignoring it.
- **Telegram** for detail. Fills, stops, session summaries. Things you want to read but that should not wake you.
- **Discord** for the permanent record. Trade cards, daily recaps, graded results. Searchable later, which matters more than you expect.

One rule that saved us from muting our own alerts: **deduplicate anything that can repeat**. A fault that recurs on a five-second loop will send 720 messages an hour and get muted, which costs you the one alert that mattered.

DATA INTEGRITY

Contract rollovers will silently desync two machines

WHAT IT COST US

Two machines running identical logic produced different trades for over a week. Root cause: one machine accepted the platform's rollover prompt and moved to the next contract month, the other dismissed it. Prices differed by about 1.2 points. Every calculation derived from price was therefore different, and nothing errored.

What we learned the hard way. Order routing instrument and chart data instrument are two different settings and they can silently disagree, so you signal on one month and fill on another. After a roll you must **rebuild your historical cache from scratch**, never append, because the price step at the join is too small to trip any sanity check but large enough to corrupt every calculation spanning it. And check whether your platform has remembered a dismissed rollover prompt permanently, because ours does, and a suppressed prompt never appears again on the machine that needs it most.

Guard against silent truncation

A data rebuild that runs before the platform flushes the session produces a cache that is quietly short. Nothing errors. We now refuse to overwrite a cache with one less than half the size of the existing file, keep the old, write the candidate to a rejected file, and shout. That guard caught a rebuild running on truncated data that had already produced bad results.

Files being written are not files ready to read

WHAT IT COST US

Our stop logic read the last line of a bar file to get the current price. It caught the file mid-write, parsed half a line into a nonsense number, and closed four positions that were nowhere near their stops. Same day, our monitoring read the same half-written files and reported one market as 820,603 minutes stale.

Two guards, both cheap. **Sanity bound** anything you parse: a price should be within a few average daily ranges of where the trade started, and a staleness reading measured in days is a parse error,

not a stall. **Require confirmation** before acting: see the condition on two consecutive checks, not one. Costs you a second or two on a real event and makes a single bad read harmless.

TESTING DISCIPLINE

Every number needs a control

This is the single highest-value habit we have. Before believing any statistic, ask what the number would be if the thing you are measuring carried no information at all.

We tested a widely shared claim that a market breaks both sides of its first fifteen-minute range 81% of the time. It reproduced on our data at 85%. Then we ran the same test on every other hour's fifteen-minute window and got up to 94%. The window was not special. A box that small gets engulfed by the day regardless of what time it formed.

The strongest control we use is a **shuffle test**: keep every trade timestamp, randomize the directions, run it twenty times. If your real results do not clearly beat that distribution, your signal is not carrying the money.

Simulate the path, not the endpoints

Any claim involving a stop requires walking the actual price path bar by bar. Entry price and exit price alone cannot tell you whether a stop was touched in between. We hold this as a rule because we have seen strategies look profitable on endpoints and unprofitable the moment the path was simulated honestly.

Look-ahead hides in reasonable-sounding rules

Our most common bug class, and it never looks like cheating. A rule that classifies a trade using a level that had not formed yet when the trade was entered. A filter using a session range that was not complete. A "same day" calculation using a calendar day when the futures session runs 18:00 to 17:00. That last one alone moved about 7% of our trades into the wrong bucket.

Overfitting is not a thing you avoid, it is a thing you measure

Everyone knows to worry about curve fitting. Almost nobody puts a number on it. These are the checks we actually run, and each one has killed something of ours that looked good.

Measure your noise band first. Before comparing any two versions, find out how much your headline metric moves from randomness alone. We ran our selection process on pure noise and

measured the spread. It was much wider than we expected. Now any improvement smaller than that band is treated as nothing, no matter how good the story is. If you have never done this, you do not know whether your last improvement was real.

Demand a plateau, not a spike. When you tune a parameter, look at its neighbors. A setting that scores well while the settings on either side also score well is a real region. A setting that scores brilliantly while its neighbors are mediocre is a coincidence you have found and named. We keep the middle of the plateau even when the peak scores higher.

Never take the extreme end of a curve that only goes one direction. If tighter is always better all the way to the edge of what you tested, you have not found an optimum, you have found the edge of your test. The true optimum is somewhere past it, or the effect is an artifact of your simulation being generous. Both are reasons not to ship the extreme.

One test result is not a result. A single backtest of a single configuration tells you almost nothing, because you had thousands of configurations to choose from and you are looking at the one that won. Run many draws. Compare the distribution of your approach against the distribution of random selection, not one number against one number.

Thin samples flatter themselves. Anything with few observations will produce extreme-looking rates, and a ranking process will therefore fill up with them. We apply a shrinkage adjustment that pulls low-sample results toward the average, and it removed a whole class of selections that looked excellent and were noise.

Test the configuration you will actually run. We spent real time optimizing simplified versions of our system, then found the conclusions did not transfer to the live setup. If your live system has ten markets, position limits, and a hedging rule, your test needs all three, or you are tuning a different machine.

Out-of-sample stops being out-of-sample the second time you look. Every time you go back to your holdout data to check another variation, you spend a little of it. We keep a period we have never touched, and we accept that once we look at it, it is used up.

Split by year before you believe anything

An improvement that shows up in one year and not the others is usually noise or a regime that has passed. We require an edge to be positive in every year we can test before it goes near live money, and we measure a single walk-forward's noise band so we know how big a difference has to be before it means anything. Ours is larger than most people would guess.

The delta is the claim

Absolute backtest numbers are always somewhat wrong: fills, slippage, and fees are estimates. So we test changes as A versus B in the same harness with the same assumptions. The difference between two arms survives assumptions that neither arm's absolute number does.

LIVE EXECUTION SAFETY

An exit must never be able to open a position

WHAT IT COST US

A manual flatten and an automated exit landed in the same cycle. The automation sent a covering order for a position that was already closed, which opened a new position in the wrong direction across four accounts.

Every exit now checks the actual account position first and clamps the quantity so it can only reduce, never cross zero. If the platform says flat, the exit becomes a no-op and logs why. This one gate has fired dozens of times since and each one was a position that would have been opened by accident.

Pin position size at entry

If you change a size setting while a position is open, your exit will compute the new size. If the new size is smaller, the exit closes part of the position and strands the rest, with nothing tracking it and no stop attached. Record the size a position was opened with, and have exits use that record rather than the current configuration.

Halt entries on stale data, but never halt exits

When your data feed goes stale, new entries should stop immediately. Exits should not. Closing toward flat is always the safe direction, and a blind window with open positions is exactly when you most need the ability to get out.

GOING LIVE IS HALFTIME, NOT THE FINISH

We thought the hard part was finding an edge. The hard part was everything between having an edge and actually collecting it. Roughly half our work happened after the first live order, and none of it was strategy.

Your exits are probably late and you do not know it

Our backtest booked exits at the moment a condition became true. Our live system checked for that condition on a timer, then acted on the next cycle, and the platform's own logic waited for the following bar. Nobody wrote a bug. The result was exits firing two to three minutes after the backtest said, which on fast moves is the difference between a good trade and a mediocre one.

Measure the gap yourself: for a sample of live trades, compare the time your system decided to exit against the time the order actually left. Then fix the biggest contributor. Ours was detection cadence, so we moved to emitting the exit at detection rather than at the next poll, and the lag dropped to seconds.

Fills are not backtest prices, and reopens are the worst offender

We measured about a point of slippage on a session-reopen entry, which was around a hundred dollars per contract on that trade. The backtest booked the reopen bar's open. The market had moved most of a point in the first minute. If your strategy trades reopens, gaps, or news reactions, your backtest is optimistic in a way that flat-market testing never reveals.

If you trade funded accounts, the rules are part of the strategy

Prop firm rules are not paperwork you sign and forget. They change which strategies are viable, and violating one can cost the account regardless of whether you were profitable.

- **Mandatory daily flatten times.** Ours requires flat by a set time each afternoon. That single rule ends about 40% of our trades by the clock rather than by their logic, and it costs measurable money. It has to be in your backtest, or your backtest is describing a system you are not allowed to run.
- **Trailing drawdown versus end-of-day drawdown.** These are different products and they suit different strategies. A strategy that goes deeply underwater before working is fine on an end-of-day account and lethal on an intraday trailing one. We route strategies to account types deliberately now.
- **Correlated-position rules across accounts.** This one cost us two accounts. Most firms prohibit hedging, and the definition is wider than people assume: being long one equity index in one account and short a different equity index in another account counts, because they group correlated products. Running the same system across multiple accounts is explicitly not an exemption.
- **Consistency and payout rules.** Some firms require no single day to be too large a share of profits. That can mean a great day actively hurts you, which is a strange thing to design around

but a real one.

Multiple accounts running one system is its own problem

The moment you run the same automation across several accounts at the same firm, you inherit questions you never had with one: what happens when one account gets a fill and another gets rejected, what happens when a position is opened on three and closed on two, and how you stop the combined book from taking positions that are individually fine but collectively a hedge.

We ended up enforcing one direction per correlated group across the entire book, which costs real money in skipped trades, and paying that cost is much cheaper than losing funded accounts. Whatever your rule is, it belongs in the code, not in your memory.

Reconciliation is a permanent job

What your system believes it holds and what the account actually holds will drift. Rejected orders, manual intervention, partial fills, platform restarts. We compare expected against actual continuously and alert on the difference, and we have a way to tell the system "I closed that by hand, forget it" without it fighting us. Before we had that, every manual trade started an argument with our own software.

SMALL THINGS THAT EACH COST US A DAY

None of these are conceptual. They are the specific, stupid, real details that broke something and took hours to find. Most of them appear in no documentation anywhere.

- **The platform's "flatten everything" button disables every strategy.** Including the ones exporting your data and tracking your state. You hit it in a hurry, positions close, and your automation is now blind and silent with no error anywhere. Know what that button actually does before you need it.
- **A bad character in a config file fails at the next restart, not now.** We pasted text containing a curly quote into a JSON config. Everything kept running fine because the file was already loaded. Hours later, on a routine restart, the program would not start. The failure and the cause were separated by half a day. Configs get written plain ASCII, always.
- **Emoji in console output silently kills background processes.** Our launcher printed a status line with an emoji in it. Run from a terminal it was fine. Run as a scheduled background task under a different encoding, the print threw, and the program died before it started. No window, no log, no error. It just was not running.

- **Scheduled tasks strip quotes from paths with spaces.** Create a task from the command line with a path like "Program Files" in it and the task will be created, look correct in the UI, and fail with a file-not-found error when it fires. Use the newer scheduling commands with an explicit working-directory parameter.
- **Changing directory in a script does not change it for programs you launch.** Our scripts changed directory then launched a helper with a relative path. The helper inherited the original directory, could not find its files, and exited instantly with no error. Always pass full paths and set the working directory explicitly.
- **The daily maintenance break will trip your alarms.** Futures have an hour each afternoon where nothing trades. Our "data is stale" monitor screamed through it every single day until we taught it the schedule. Same for holidays and half days. If your monitors do not know the calendar, you will train yourself to ignore them.
- **Your session day is not the calendar day.** The futures session runs from the evening open to the following afternoon. Group your data by calendar date and roughly 7% of your trades land in the wrong day, which quietly corrupts anything measured per-day.
- **Two clocks on the same machine can disagree.** We had one shell reporting a time an hour off from another, all day, and every estimate made from it was wrong. Our display timezone also flipped on remote-desktop reconnects. We stopped trusting wall clock in any investigation and use the timestamps on the bars themselves.
- **Launch things one way, every time.** We standardized on launcher scripts that set the right environment. Starting the same program directly, bypassing the launcher, produced a second instance that fought the first over the same files. If there is more than one way to start your system, someone will eventually use the wrong one at the worst time.
- **Killing the process does not stop the system.** Ours is supervised, so killing a component just means it comes back in three minutes. Knowing how to genuinely halt everything, and having practiced it, is its own piece of infrastructure.
- **Match accounts by exact identifier, never by prefix or nickname.** Firms hand out account numbers that share prefixes across products. We now derive the exact set and assert the count rather than pattern-matching, after nearly acting on the wrong account.

PROCESS RULES WE HAD TO LEARN TWICE

- **Verify it ran, not that it is configured.** We had a scheduled task that reported a valid next-run time and never fired, silently, for a week. Force-run any new automation once and confirm it

actually executed.

- **Boot it before you stage it.** A syntax check is not a run. Start the thing, watch it come up, then hand it to someone else.
- **Certify that a safety gate can both pass and fail.** A gate you have only seen allow things is not tested. Write the case that makes it fire.
- **Most of our bugs were in the fixes.** Changes to live paths get made in flat windows, with the position state checked first, never mid-trade because something looked urgent.
- **When you move a responsibility, verify the new owner exists.** We disabled a data refresh in one program intending another to take over. The other was never built. Nothing errored for days.

THE MINDSET THAT MATTERED MOST

Almost everything above came from a failure, and almost every failure was invisible while it was happening. Nothing crashed. No error appeared. A number was quietly wrong, or a program was quietly not running, or two machines quietly disagreed.

So the habit worth building is not writing more code. It is asking, for every part of your system: **if this were broken right now, how would I know?** If the answer is "I would notice eventually," that is the next thing to build.

Written from the running notes of a live automated futures operation. Nothing here is trading advice or a strategy. If it saves you one overnight incident it did its job.

leppyrdtracks.com