# Balancing ML Models in Quantitative Trading

Rafan Ahmed
UNC Charlotte
College of Computing and Informatics

05.01.2025

# When we think of “machine learning” we think of this:

Claude

Gemini

But these are complex LLMs. Let’s break this down simpler:

# We have all interacted with with machine learning in our everyday lives...

Google Maps

NETFLIX

TikTok

Phenomenal demonstrations of computing! However, it is not always perfect...

Let's take a look at Google Maps for example

Show of hands—who here has blindly followed Google Maps and ended up more lost and further away from a destination?

Or your Google Maps made you take a completely wrong turn

Some stock market traders do the same exact thing

—except the ‘wrong turn’ costs millions of dollars.

Why? And how can we balance and prevent this as quantitative traders?

# What is Quantitative Trading?

## Stock Market Trading

Buy low, sell high, profit the difference.

Ex. You buy Amazon stock when it was at $90 and sold it when it went up to $110


## What makes it *quant*?

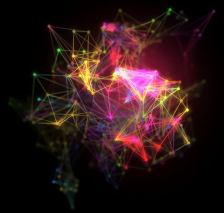
Instead of gut feelings, we let math decide — like Netflix's algorithm picks media for you to watch.

Ex.

*Data → Math*
*Rules → Buy/Sell decision*

## Machine Learning?

ML is pattern-spotting on steroids: give a computer thousands of past price movements of a stock; it can mathematically guess tomorrow's price.



# Machine Learning Tools can be Incredibly Powerful for Stock Market Trading

But treating these tools as magic or as a dogma for successful trading often backfires.

# Why Newer Quants Fall for Flashy ML Models

## Big accuracy on “homework” data

These complex ML models perform extremely well on training datasets (historical prices, stock highs/lows etc.), giving the illusion that they’re superior.
Flashy ML models often do great on historical data—like homework they’ve already seen. But that doesn’t mean they’ll work in the real market.

## The myth of Fancy = Profitable

Flashy, complicated ML models look impressive but don’t necessarily perform better in real-world finance and stock market trading.
Just because a model is complicated doesn’t mean it makes more money. In fact, fancy models often fail faster.

## Career kudos for flashy models

There's social/professional reward for using cutting-edge tools—even if they’re overkill.
There’s social pressure to use trendy tools—even when simpler models would work better.

# So why can these overly flashy ML models be bad?

Because being impressive on GitHub isn’t the same as surviving Wall Street.

(Yes, this is from personal experience.)

# Why Fancy ML Models Fail in the Real Market

**Markets change constantly**

The market you trained your model on **is not the market you'll trade in.** News, volatility, interest rates, and trader behavior all evolve—like Trump's tariff announcement or after Fed announcements—so yesterday's signal might be today's noise.

**Even great predictions can fail**

A model can "guess right" and still lose money if it trades too often, enters late, or ignores costs like slippage and commission.
**Prediction ≠ Profit.**

**Simpler models last longer**

Basic models (moving averages of stock price or logistic regression) are easier to understand, easier to fix when something goes wrong. They usually don't rely on hundreds of tiny assumptions about the market. Because of that, they're more stable over time



This is the model's backtest—when it's evaluated on the same data it was trained on. Everything looks bullish! High performance, smooth equity curve... surely this success will carry over to live trading, right?

But in reality, the model falls apart. It was too focused on memorizing patterns in the past—cluttered with noise and unnecessary parameters. When market conditions changed, it couldn't adapt. The strategy underperforms and turns bearish.

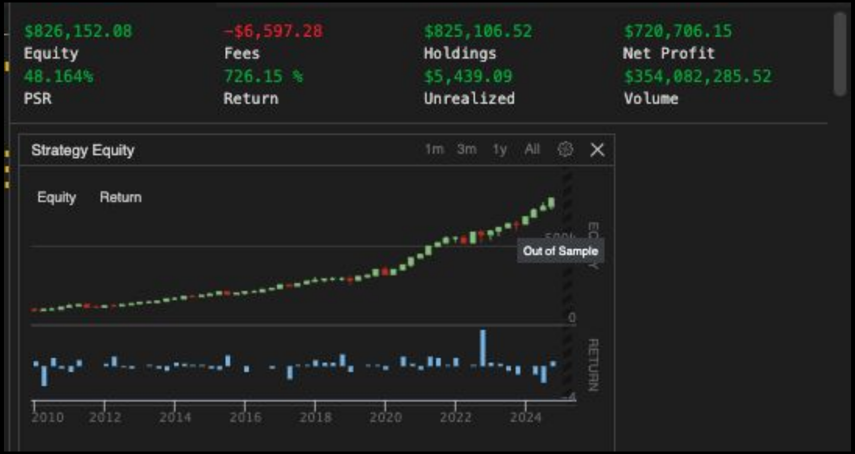
**When your model fits everything, it learns nothing. This is whats called overfitting.**

# I have a confession to make:

I Was (and still am) That Newer Quant (And I Overfit Badly)

For my first Quantitative Trading Strategy, I built a Recurrent Neural Network Strategy model that looked incredible in backtests.

Tracking the ETF $SPY, I simulated a portfolio that started at $100k start equity with the training simulation running from 2010-2025

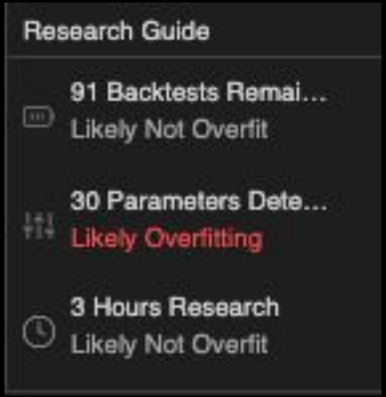



My backtests results for my Recurrent Neural Network Strategy.
726% return from $100k start equity!!

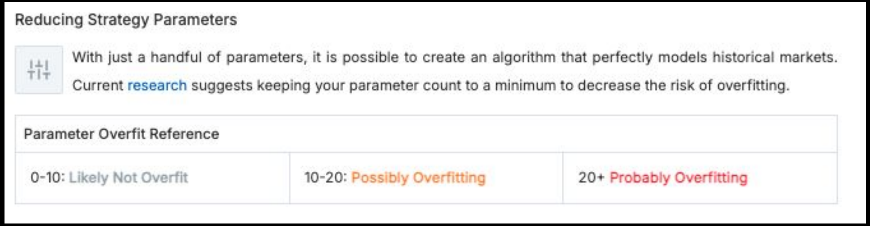
It made money on paper. I felt like a **quant genius.**

But there was a huge catch that the QuantConnect cloud backtesting platform notified me about:

My model was **overfitting...**




For context, 20+ strategy parameters is what is considered overfit according to QuantConnect's documentation. I had 30.


Reducing Strategy Parameters

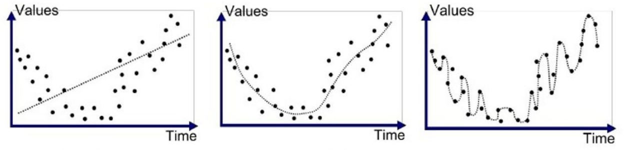
With just a handful of parameters, it is possible to create an algorithm that perfectly models historical markets. Current research suggests keeping your parameter count to a minimum to decrease the risk of overfitting.

| Parameter Overfit Reference | | |
|---|---|---|
| 0-10: Likely Not Overfit | 10-20: Possibly Overfitting | 20+ Probably Overfitting |

# What Is Overfitting? (And Why My Model Fell for It)

## What is Overfitting?

- Overfitting happens when a model learns the training data too well—including random noise, quirks, and exceptions that won't repeat.
- It doesn't just learn patterns. It **memorizes mistakes.**
- The model gets an "A+" on past data, but **flunks in reality and current market data/trends.**
- It's like studying only last year's exam answers and expecting to ace a completely new test.




Underfitted | Good Fit/Robust | Overfitted

## How My Strategy Overfit (based on internal code review)?

- I trained my RNN on data from **2000–2025**, but then tested it on **2010–2025**—this means it had already *seen* the test data.
- The model learned **patterns it would never have access to in real life**—that's called **lookahead bias**.
- The backtest looked great: **15% annual return**, **68% win rate**, and a **Sharpe ratio of 0.87** (returns looked strong relative to the amount of risk taken).
- But the Probabilistic Sharpe Ratio (or PSR) was just 48%—meaning there's only a 48% chance the model's performance would actually hold up in the real world. Basically, it's a coin flip and a major indicator of overfitting
- It was **too tuned to past market conditions**, not robust to new ones.

# So is ML in Quant Trading really THAT bad?

**NO!** It is all about achieving a balanced use of ML models while avoiding overfitting

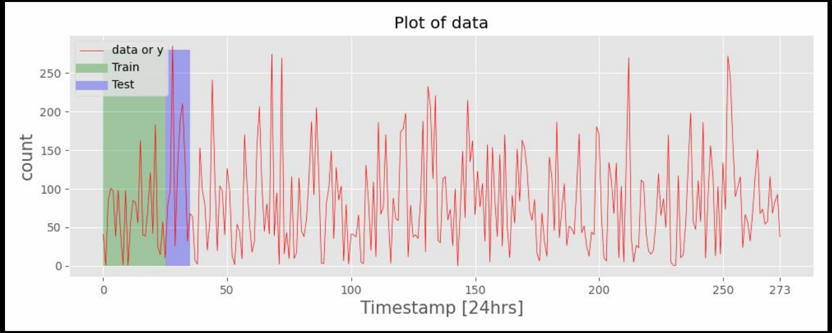
Here are methods pro quant traders do to **avoid overfitting** without giving up the powers of ML models:

# Rolling-Window Testing - *Test Like Time Moves*

- Break your data into moving time chunks — train on past data, test on the next few days, then slide forward.

- This is how models work in real life: they learn from yesterday to predict today, not the far future.

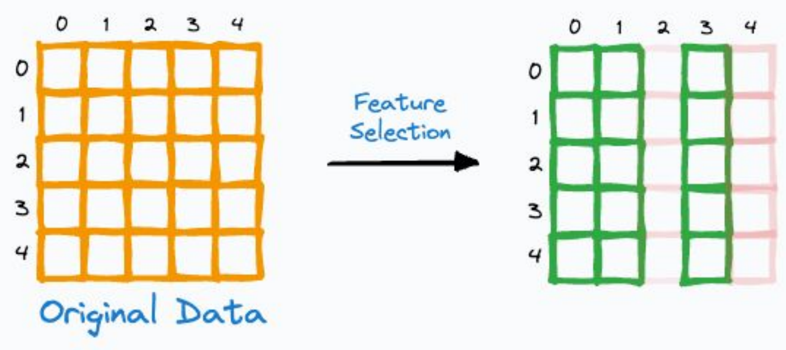
- This helps you catch weak strategies before they lose real money.




Instead of training once and testing at the end, we train on a chunk of data, test on the next day, then slide forward. This is how real strategies stay updated as the market moves.

# Fewer, Cleaner Features

- More inputs ≠ more insight. More data doesn’t always mean better results.
- Extra indicators can drown real patterns. Too many signals can confuse the model.
- Stick to high-quality, well-understood variables; drop the rest. Use only the most important info—leave out the stuff that doesn’t help.
- Cleaner data → simpler model → lower risk of overfitting.



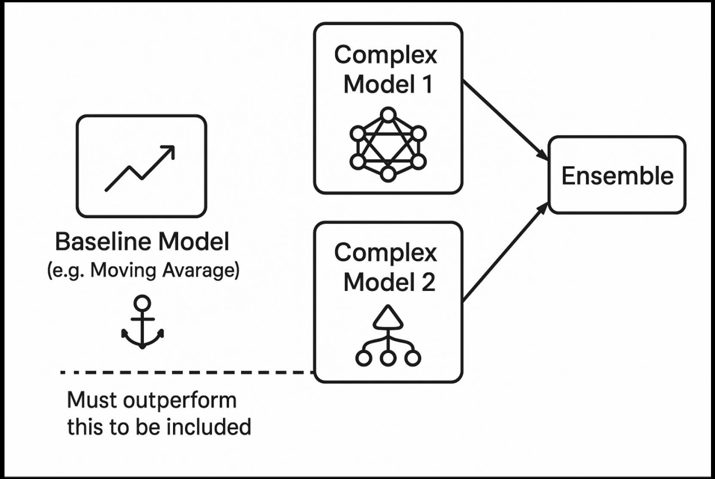

Start with all your data... but only keep the columns that actually help.
Feature selection is like decluttering your model—fewer, more useful inputs = a smarter, cleaner strategy.

# Anchor the Ensemble - *Simple First, Fancy Later*

- Begin with a simple model that's easy to understand.

- **Only** add complex stuff if it actually performs better on new, unseen data.

- The simple model acts as a **reality check**—so you know your fancy model isn't just showing off.



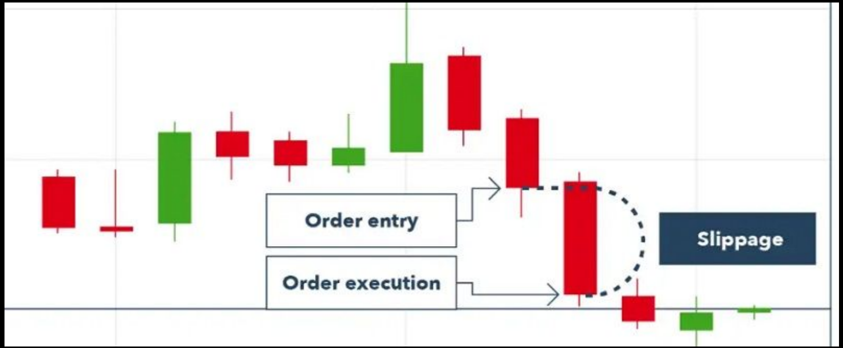

Start simple. Only keep the complex models if they actually perform better than the basic one.

# Cap Turnover & Test Slippage - *Trade Less, Keep More*

- Every time you make a trade, you quietly lose a bit of money through things like the gap between buy and sell prices (spreads), broker fees, and slippage—when your order fills at a worse price than expected.

- Set a limit on how often it can buy and sell (called a "turnover ceiling") so it doesn't rack up hidden costs like fees and bad prices.

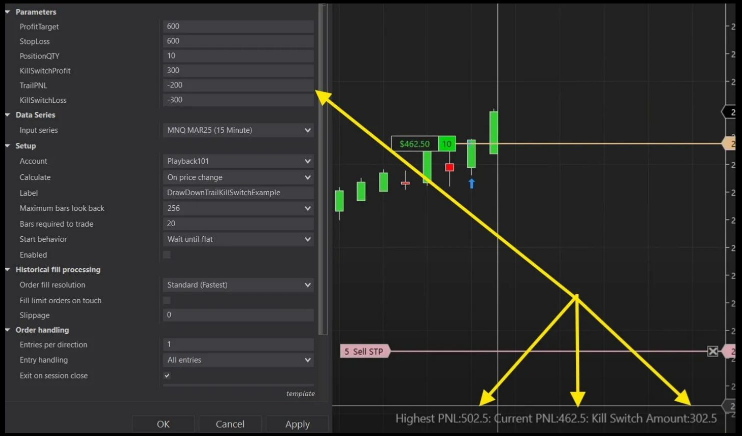
- If your strategy still performs well with those real-world frictions, it's more likely to hold up when real money's on the line.




Slippage = the price you expected vs. the price you actually got. In fast-moving markets, trades don't always fill at your ideal price — and those tiny gaps add up over time.

# Live Drift Alerts / Kill-Switches

- Markets are always changing, so your model shouldn't run on autopilot forever.
- Set up alerts (called **drift monitors**) that warn you when your model's predictions start to go off track.
- If the market shifts and your model starts making bad calls, a **kill-switch** can automatically pause trading.
- This gives you time to check what went wrong and re-train the model—**before small mistakes turn into big losses.**




If profits drop too far, the system cuts off trading to prevent bigger losses. That's your kill-switch in action.

# Takeaways for Building Balanced ML Quant Strategies

- **Don't fall for flash:** Complex ≠ better. Simpler models are often more stable and easier to debug.
- **Avoid overfitting:** Use rolling-window testing and watch for lookahead bias.
- **Clean your data:** Fewer, high-quality features beat a bloated dataset.
- **Anchor complex models:** Only keep them if they outperform simple baselines on new data.
- **Account for frictions:** Slippage, spreads, and fees add up—simulate real-world costs.
- **Balance is key:** Combine machine learning's power with human judgment and practical safeguards.

# THANK YOU

LinkedIn

GitHub

Instagram