COMILLAS
UNIVERSIDAD PONTIFICIA

ICAI

GRADO EN INGENIERÍA MATEMÁTICA E INTELIGENCIA ARTIFICIAL

TRABAJO FIN DE GRADO

Retornos fraccionados, memoria y predicción en series financieras

Author: Francisco López-Alvarado Ortega

Director: Álvaro Guinea Juliá

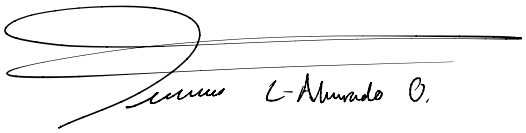
Madrid, June 2026

I declare, under my responsibility, that the Project submitted under the title

Fractional differencing, memory and forecasting in financial time series

at the ICAI School of Engineering of Comillas Pontifical University in the academic year 2025/2026 is my own work, original and unpublished, and has not previously been submitted for any other purpose.

The Project is not plagiarized, either wholly or partially, and the information taken from other documents is duly referenced.



Signed: Francisco López-Alvarado Ortega

Date: 15 / 06 / 2026

Submission of the project is authorized

THE PROJECT DIRECTOR

Signed: Álvaro Guinea Juliá

Date: 15 / 06 / 2026

THE PROJECT CO-DIRECTOR (if applicable)

Signed:

Date: ..... / ..... / 2026

Acknowledgments

I would like to express my sincere gratitude to my thesis director, Álvaro Guinea Juliá, for all the time and dedication he has devoted to this project. His guidance, patience and continual feedback have been invaluable at every stage of the work, and this thesis would not have been possible without his support.

RETORNOS FRACCIONADOS, MEMORIA Y PREDICCIÓN EN SERIES FINANCIERAS

Autor: Francisco López-Alvarado Ortega
Director: Álvaro Guinea Juliá

Resumen

Las series de precios financieros son no estacionarias, por lo que deben transformarse antes de modelarlas. La práctica habitual, tomar retornos, logra estacionariedad pero destruye la memoria de largo plazo. La diferenciación fraccionaria ofrece un punto intermedio: el mínimo orden que estaciona la serie conservando memoria. Este trabajo evalúa, con una metodología honesta (rival de paseo aleatorio, métrica MASE y test de Diebold-Mariano (DM)), si la diferenciación fraccionaria mejora la predicción de series financieras con redes neuronales (LSTM/RNN) y si el orden que la red aprende coincide con el que maximiza la memoria. Un control sintético valida que el método detecta memoria cuando existe.

Palabras clave: diferenciación fraccionaria; memoria larga; predicción financiera; redes LSTM; paseo aleatorio; eficiencia de mercado.

Resumen ejecutivo

1 Introducción

Contexto. El trabajo se sitúa en la predicción de series temporales financieras y, en particular, en la tensión entre *estacionariedad* y *memoria*: los precios no son estacionarios, pero al diferenciarlos para estacionarlos (retornos) se destruye su memoria de largo plazo. La diferenciación fraccionaria permite diferenciar "lo justo", conservando memoria.

Motivación. Si esa memoria conservada fuese útil, alimentar con ella a una red neuronal debería mejorar la predicción frente a los retornos; y si la red puede aprender el orden de diferenciación, revelaría el orden óptimo para predecir. Comprobar ambas cosas con rigor, y descartar que una aparente mejora sea un artefacto de medición, es lo que motiva este trabajo.

2 Objetivos

El objetivo principal es determinar, con una metodología honesta, si la diferenciación fraccionaria mejora la predicción de series financieras con redes neuronales. Como objetivos secundarios: (i) construir un estimador del orden de memoria d^* anclado al estadístico ADF, sin

pesos arbitrarios; (ii) hacer que la red aprenda el orden de diferenciación por *backpropagation* y comparar ese óptimo predictivo con d^*; y (iii) validar todo el procedimiento mediante un control sintético con memoria conocida.

3 Modelo

La solución combina tres piezas. **(1)** Como entrada a la red se usa la *diferenciación fraccionaria* del log-precio, $(1-L)^d$ truncada a K pesos: al variar d entre 0 y 1 se recorre un continuo entre el precio (toda la memoria, no estacionario) y el retorno (estacionario, sin memoria), como muestra la Figura 1. **(2)** Un estimador *diferenciable* del orden de memoria d^*, anclado al test ADF en lugar de a pesos arbitrarios (objetivo O1; sigue la receta de mínimo orden de López de Prado [1], raíz de Granger–Joyeux [2] y Hosking [3]). **(3)** Redes recurrentes (LSTM/RNN) en las que d es un parámetro *entrenable*, aprendido por *backpropagation* (objetivo O2); el orden resultante d_{opt}, el "mejor para predecir", se compara con d^*. Todo se mide con honestidad: contra un *paseo aleatorio*, con la métrica MASE y el test de Diebold-Mariano (DM), y validando antes el pipeline en un *control sintético* con memoria conocida que el método debe detectar (objetivo O3).

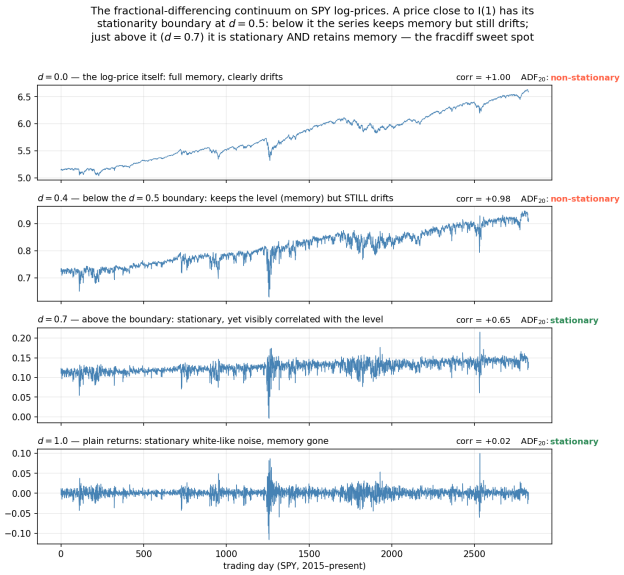


Figura 1: El continuo de la diferenciación fraccionaria sobre un log-precio real: al subir d de 0 (precio) hacia 1 (retorno), la serie pierde memoria y gana estacionariedad. El estimador d^* elige el punto justo.

4 Resultados

Control sintético 6/6: el método detecta la memoria inyectada y no inventa ninguna cuando no la hay, lo que valida el pipeline. **Precios 0/8:** sobre precios diarios reales ningún modelo fraccionario bate al paseo aleatorio, y el resultado resiste a más memoria, más capacidad y modelos clásicos (ARIMA/ARFIMA). **Los dos óptimos solo se comparan donde hay memoria:** en precios la curva de error es plana (no hay "mejor d"; correlación con d^* casi nula, +0,18), pero en sintético con memoria real d_{opt} y d^* *se mueven juntos* (Spearman +1,00). **Volatilidad 5/6:** el mismo sistema, apuntado a la volatilidad, gana en 5 de 6 activos ($p < 0{,}001$) con un 24–28 % menos de error (Figura 2). La memoria explotable está en *cuánto* se mueven los precios, no en *hacia dónde* [4].

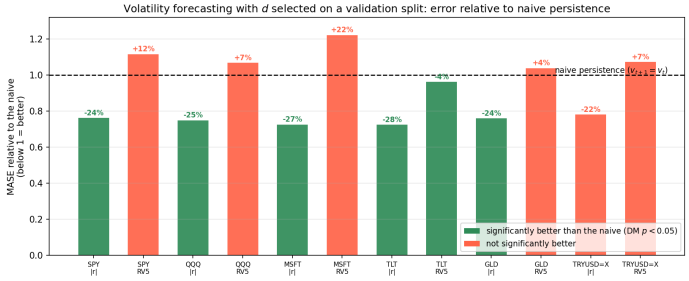


Figura 2: Volatilidad bajo el protocolo honesto: MASE relativo al método ingenuo (por debajo de la línea = mejor; verde = significativo al 5 %). Los dos proxies son el log-retorno absoluto $|r_t| = |\log p_t - \log p_{t-1}|$ y la volatilidad realizada a 5 días $\mathrm{RV5}_t = \sqrt{\frac{1}{5}\sum_{i=0}^{4} r_{t-i}^2}$. El mismo sistema que no mejoraba los precios (0/8) sí mejora la volatilidad (5/6).

5 Conclusiones

La diferenciación fraccionaria es *válida* y d^* es *significativo*, pero solo donde hay memoria sobre la que actuar: los precios diarios se comportan como un paseo aleatorio (sin memoria explotable), mientras que la volatilidad sí la tiene. El negativo es *informativo*: convierte la eficiencia de mercado en un *mapa operativo* de cuándo merece la pena la herramienta. Aportaciones: (i) un estimador diferenciable de d^* anclado al ADF; (ii) una red que aprende d de extremo a extremo; (iii) un protocolo honesto con control sintético, que por sí solo desmonta varias mejoras "aparentes" de la literatura; y (iv) evidencia de cuándo ayuda la diferenciación fraccionaria y cómo se relacionan los dos óptimos.

6 Referencias

[1] M. López de Prado, *Advances in Financial Machine Learning*. Wiley, 2018.

[2] C. W. J. Granger y R. Joyeux, "An Introduction to Long-Memory Time Series Models and Fractional Differencing", *Journal of Time Series Analysis*, vol. 1, n.º 1, págs. 15-29, 1980.

[3] J. R. M. Hosking, "Fractional Differencing", *Biometrika*, vol. 68, n.º 1, págs. 165-176, 1981.

[4] Z. Ding, C. W. J. Granger y R. F. Engle, "A Long Memory Property of Stock Market Returns and a New Model", *Journal of Empirical Finance*, vol. 1, n.º 1, págs. 83-106, 1993.

FRACTIONAL DIFFERENCING, MEMORY AND FORECASTING IN FINANCIAL TIME SERIES

Author: Francisco López-Alvarado Ortega
Director: Álvaro Guinea Juliá

Abstract

Financial price series are non-stationary, so they must be transformed before modelling. The standard practice, taking returns, achieves stationarity but destroys the long-range memory of the series. Fractional differencing offers a middle ground: the minimum differencing order that renders the series stationary while preserving memory. This work rigorously evaluates, under an honest methodology (a random-walk benchmark, the MASE metric and the Diebold-Mariano test (DM)), whether fractional differencing improves neural-network forecasting (LSTM/RNN) of financial series, and whether the differencing order learned by the network matches the one that maximises memory. A synthetic control validates that the procedure detects memory when it is present.

Keywords: fractional differencing; long memory; financial forecasting; LSTM networks; random walk; market efficiency.

Executive Summary

1 Introduction

Context. This work lies in the forecasting of financial time series and, specifically, in the tension between *stationarity* and *memory*: prices are non-stationary, but differencing them to achieve stationarity (returns) destroys their long-range memory. Fractional differencing makes it possible to difference "just enough", retaining memory.

Motivation. If that retained memory were useful, feeding it to a neural network should improve forecasts over plain returns; and if the network can learn the differencing order, it would reveal the order that is optimal for prediction. Verifying both claims rigorously, and ruling out that any apparent gain is a measurement artefact, is the motivation of this work.

2 Objectives

The main objective is to determine, under an honest methodology, whether fractional differencing improves the forecasting of financial series with neural networks. The secondary

objectives are: (i) to build an estimator of the memory order d^* anchored to the ADF statistic, free of arbitrary trade-off weights; (ii) to let the network learn the differencing order by *backpropagation* and compare this predictive optimum with d^*; and (iii) to validate the whole procedure with a synthetic control of known memory.

3 Model

The solution combines three pieces. **(1)** The network is fed the *fractional differencing* of the log-price, $(1-L)^d$ truncated to K weights: sweeping d between 0 and 1 traces a continuum from the price (full memory, non-stationary) to the return (stationary, no memory), as Figure 1 shows. **(2)** A *differentiable* estimator of the memory order d^*, anchored to the ADF test rather than to arbitrary weights (objective O1; it follows the minimum-order recipe of López de Prado [1], rooted in Granger–Joyeux [2] and Hosking [3]). **(3)** Recurrent networks (LSTM/RNN) in which d is a *trainable* parameter, learned by *backpropagation* (objective O2); the resulting order d_{opt}, the "best for prediction", is compared with d^*. Everything is measured honestly: against a *random-walk* benchmark, with the MASE metric and the Diebold-Mariano test (DM), and validating the pipeline first on a *synthetic control* of known memory that the method must detect (objective O3).

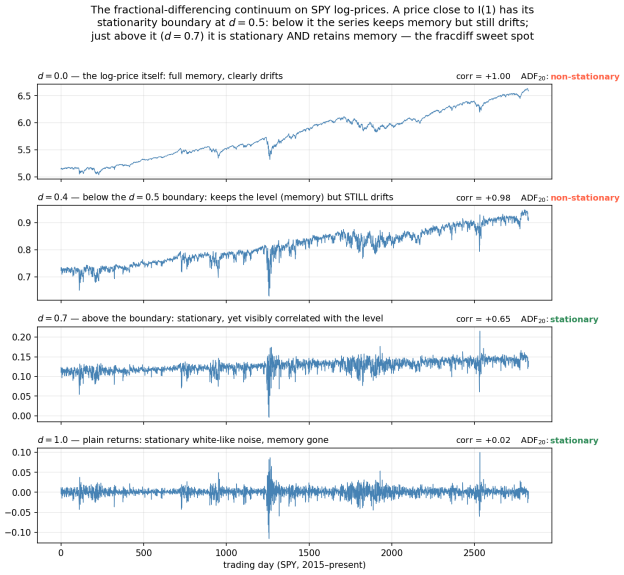


Figure 1: The fractional-differencing continuum on a real log-price: as d rises from 0 (price) towards 1 (return), the series loses memory and gains stationarity. The estimator d^* picks the right point.

4 Results

Synthetic control 6/6: the method detects the injected memory and invents none when there is none, which validates the pipeline. **Prices 0/8:** on real daily prices no fractional model beats the random walk, and the result survives more memory, more capacity and classical models (ARIMA/ARFIMA). **The two optima can only be compared where memory exists:** on prices the error curve is flat (no "best d"; correlation with d^* essentially zero, +0.18), but on synthetic series with genuine memory d_{opt} and d^* *move in lockstep* (Spearman +1.00). **Volatility 5/6:** the same system, pointed at volatility, wins in 5 of 6 assets ($p < 0.001$) with 24–28% less error (Figure 2). The exploitable memory is in *how much* prices move, not in *which way* [4].

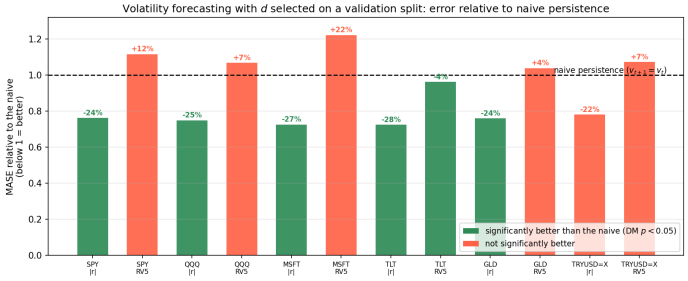


Figure 2: Volatility under the honest protocol: MASE relative to the naive benchmark (below the line = better; green = significant at 5%). The two proxies are the absolute daily log-return $|r_t| = |\log p_t - \log p_{t-1}|$ and the 5-day realised volatility $\mathrm{RV5}_t = \sqrt{\frac{1}{5}\sum_{i=0}^{4} r_{t-i}^2}$. The same system that could not improve prices (0/8) does improve volatility (5/6).

5 Conclusions

Fractional differencing is *sound* and d^* is *meaningful*, but only where there is memory to act on: daily prices behave like a random walk (no exploitable memory), whereas volatility carries it. The negative result is *informative*: it turns the efficient-market hypothesis into an *operational map* of when the tool is worth using. Contributions: (i) a differentiable estimator of d^* anchored to the ADF test; (ii) a network that learns d end-to-end; (iii) an honest evaluation protocol with a synthetic control, which by itself overturns several "apparent" gains in the literature; and (iv) evidence of when fractional differencing helps and how the two optima relate.

6 References

[1] M. López de Prado, *Advances in Financial Machine Learning*. Wiley, 2018.

[2] C. W. J. Granger and R. Joyeux, "An introduction to long-memory time series models and fractional differencing", *Journal of Time Series Analysis*, vol. 1, no. 1, pp. 15-29, 1980.

[3] J. R. M. Hosking, "Fractional differencing", *Biometrika*, vol. 68, no. 1, pp. 165-176, 1981.

[4] Z. Ding, C. W. J. Granger, and R. F. Engle, "A long memory property of stock market returns and a new model", *Journal of Empirical Finance*, vol. 1, no. 1, pp. 83-106, 1993.

Contents

List of Figures

List of Tables

Chapter 1 Introduction

Forecasting financial time series is one of the hardest problems in quantitative finance, and part of the difficulty is structural. Price series are *non-stationary*: they trend and wander instead of returning to any fixed level, so their statistical behaviour keeps changing over time. Most models, statistical or machine-learning alike, work best on *stationary* inputs, where that behaviour stays stable. The usual fix is to *difference* the series, that is, to model returns ($p_t - p_{t-1}$, or log-returns) rather than prices. This does make the series stationary, but at a high cost: it erases almost all of its *long memory*, the slowly fading link between distant observations that might, in principle, carry predictive information.

That trade-off, *stationarity versus memory*, is where this thesis begins. Ordinary (integer) differencing forces an all-or-nothing choice: either the raw price (all memory, but non-stationary) or first differences (stationary, but no memory). *Fractional differencing*, introduced by Granger and Joyeux [1] and Hosking [2], removes that dilemma by letting the differencing order d be any real number, not just 0 or 1. The operator $(1 - L)^d$ (where L is the lag operator) then becomes an infinite weighted sum of past values, and for $0 < d < 1$ one can difference *just enough* to reach stationarity while keeping a tunable amount of memory. López de Prado [3] brought this idea into machine learning as a feature-engineering step: pick the *smallest* order d that makes a price stationary, so the input is admissible for a model yet keeps as much memory, and hopefully as much signal, as possible.

A natural hypothesis follows. If we feed a neural network this stationary-but-memory-preserving version of the price, instead of plain returns, it should forecast better. This thesis puts that hypothesis to a strict test. It also asks a deeper question. Suppose we make the differencing order d a *trainable parameter* and let the network tune it by backpropagation to minimise forecast error; the network then reveals the d that is best *for prediction*. Is that predictive-best order the same as the *memory-best* order d^*, the one that preserves the most memory while staying stationary? In plain words: is the representation that forecasts best also the one that best preserves memory?

These two questions are stated precisely as:

- **RQ1 (forecasting).** Does fractional differencing improve the forecasting of financial series with recurrent neural networks, relative both to plain returns and to a *random-walk* benchmark?

- **RQ2 (representation).** Does the differencing order learned by the network (the predictive optimum) coincide with the order d^* that maximises memory subject to stationarity (the memory optimum)?

Answering these questions honestly turns out to be as important as the questions them-

selves. A recurring trap in this literature is methodological: many reported gains are artefacts of an unfair comparison rather than real predictive skill. Avoiding that trap is a core part of this work. Every model is judged against an honest *random-walk* benchmark, with a scale-free error measure (the Mean Absolute Scaled Error, MASE) and a significance test (the Diebold-Mariano test). And to tell apart "the method cannot find memory" from "the data have no memory to find", the whole pipeline is first run on *synthetic* series built to contain a *known* amount of long memory (fractionally integrated processes). This is a positive control, in the laboratory sense: if the method cannot recover memory we injected on purpose, no negative result on real data can be trusted.

The main contributions of this work are: (i) a differentiable estimator of the memory-optimal order d^*, anchored to the Augmented Dickey-Fuller statistic rather than to arbitrary trade-off weights; (ii) a network architecture in which the differencing order is learned end-to-end together with the forecasting weights; (iii) an honest evaluation protocol equipped with a synthetic positive control; and (iv) empirical evidence, on both real markets and synthetic data, that characterises *when* fractional differencing helps and how the predictive and memory optima relate.

The remainder of this document is organised as follows. Section 2 reviews the literature on long memory in financial series and on fractional differencing, and identifies the gap addressed here. Section 4 describes the models, the estimator of d^*, the learned-d network and the evaluation protocol. Section 5 reports the experiments on real and synthetic data, and Section 6 draws conclusions and outlines future work.

Chapter 2 State of the Art

This chapter presents the ideas and the previous work that this thesis builds on. It starts with the concept of *long memory* and where it comes from. Then it introduces *fractional differencing* and the *ARFIMA* model, the standard tools to describe long-range dependence. After that, it reviews the evidence on long memory in financial data, separating returns from volatility, and the mixed forecasting record of these models. It also presents the *recurrent neural networks* used in this work and the metrics used to evaluate forecasts. The chapter ends with the most recent research line, where fractional differencing meets machine learning, and with the gap that this thesis tries to fill.

2.1 Long memory and the Hurst exponent: a story that begins on the Nile

The idea of long memory did not start in finance. It started with the river Nile. In the first half of the twentieth century, the British engineer Harold Edwin Hurst worked in Egypt for decades on the design of the water reservoirs for the Aswan dam. His question sounds simple: how large must a reservoir be so that it never runs dry and never overflows? The answer depends on how the yearly inflows of the river behave over very long periods. The Nile has records that go back centuries, so Hurst could study this question in depth. He measured the *rescaled range*: for a window of n consecutive years, he computed the range R of the accumulated deviations of the inflow from its mean, and divided it by the standard deviation S of that window.

If the years were independent from each other, as in a random walk, this rescaled range should grow like $\sqrt{n}$. Hurst found something different. For the Nile, and for dozens of other natural series, he obtained

$$\mathbb{E}\left[\frac{R(n)}{S(n)}\right] \sim c\, n^{H}, \qquad \text{with } H \approx 0.73, \tag{2.1}$$

which is clearly above the value $H = 0.5$ that independence implies [4]. Wet years tend to follow wet years, and dry years tend to follow dry years, over very long periods. This behaviour became known as the *Hurst phenomenon*, and the exponent H as the *Hurst exponent*. Its interpretation is standard:

- $H = 0.5$: no memory. The increments behave like white noise (a random walk).
- $H > 0.5$: *persistent* behaviour, that is, long memory. A movement tends to be followed by movements of the same sign, and the autocorrelations decay very slowly.

- $H < 0.5$: *anti-persistent* behaviour. The series tends to revert to its mean.

The reference process with this property is *fractional Brownian motion* (fBm), introduced by Mandelbrot and Van Ness [5]. It is the simplest random path whose memory is set by a single number, the Hurst exponent H. Ordinary Brownian motion is the special case $H = 0.5$, where each step ignores the past; with $H > 0.5$ steps tend to continue in the same direction (persistence), and with $H < 0.5$ they tend to reverse (anti-persistence). Its steps, called *fractional Gaussian noise*, keep that correlation at *every* time scale, and the path is *self-similar*: zooming into a piece gives the same statistical shape as the whole, $B_H(at) \overset{d}{=} a^H B_H(t)$. All of this is captured by its covariance, $\mathbb{E}[B_H(t)B_H(s)] = \frac{1}{2}\big(|t|^{2H} + |s|^{2H} - |t-s|^{2H}\big)$.

Figure 1 shows the phenomenon and how we measure it. We build the three paths ourselves as fractional Brownian motions with $H < 0.5$, $H = 0.5$ and $H > 0.5$. The recipe has two steps: first we fractionally integrate white noise, $x_t = (1-L)^{-(H-0.5)}\varepsilon_t$, to obtain fractional Gaussian noise (using the same $(1-L)^d$ machinery as the rest of this work); then we add it up, and that cumulative sum is the fBm path. The three are visibly different: rough and mean-reverting for $H < 0.5$, an ordinary Brownian path for $H = 0.5$, and long, smooth swings for $H > 0.5$. Fitting $\log(R/S)$ against $\log n$, with the Anis–Lloyd–Peters small-sample correction, recovers the H that generated each one.

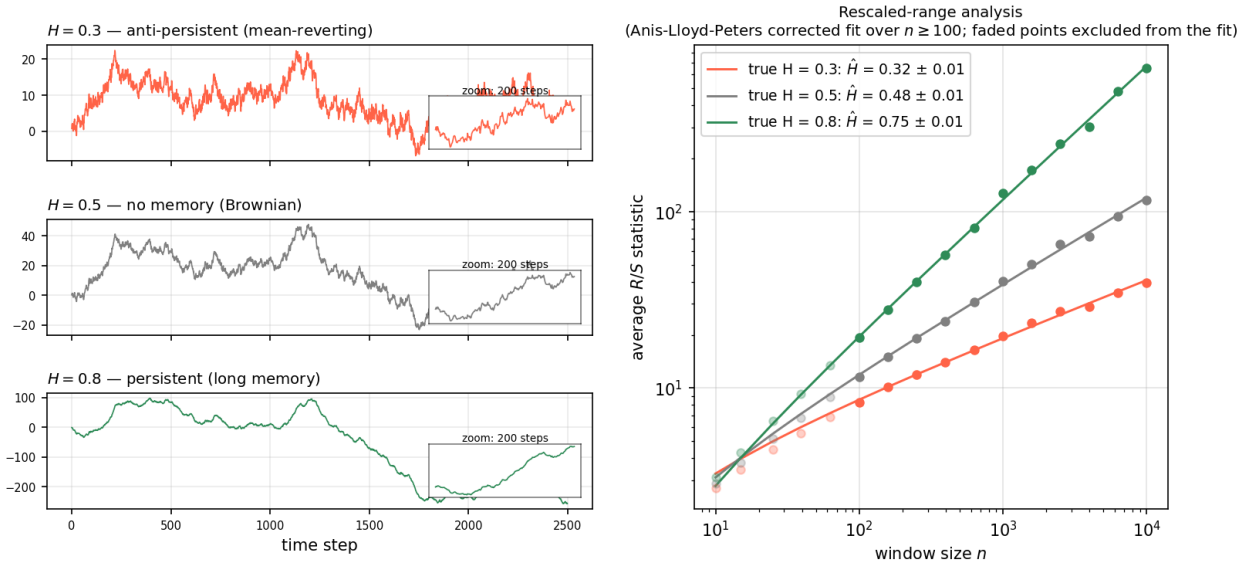


Figure 1: Left: 2,500-step excerpts of accumulated fractional noises with Hurst exponents $H = 0.3$ (anti-persistent), $H = 0.5$ (no memory) and $H = 0.8$ (persistent); the insets zoom into 200 steps, where the rough texture of $H = 0.3$ and the smooth swings of $H = 0.8$ become visible. Right: rescaled-range analysis computed on the full series of length $N = 40{,}000$, with the Anis-Lloyd-Peters small-sample correction and the fit restricted to windows $n \geq 100$ (faded points are excluded from the fit). The estimates recover the true exponents: $\hat{H} = 0.32 \pm 0.01$, 0.48 ± 0.01 and 0.75 ± 0.01.

Hurst did not stop at the Nile. For more than fifty years he repeated the same analysis on hundreds of natural records: river and lake levels, rainfall, tree rings, sediments, even sunspot numbers. The result was almost always the same, a value of H clearly above 0.5. So the effect could not be something special about one river: it looked like a general property of natural series. His method, plotting $\log(R/S)$ against $\log n$ and reading H from the slope, is still the standard textbook way to estimate H; later techniques, such as detrended fluctuation analysis or wavelet methods, are refinements of the same idea. Why was this result surprising? Because classical statistics predicted the opposite. When many independent random shocks accumulate, what happened long ago should stop mattering. Hurst's series did not behave like that: they seemed to remember their distant past.

Benoit Mandelbrot understood what these results meant: the classical models, in which memory dies out quickly, could not explain them, so a new mathematical object was needed. Together with Van Ness [5] he introduced *fractional Brownian motion* (fBm), a generalisation of ordinary Brownian motion controlled by the same parameter H. Its key property is that, when $H > 0.5$, its increments stay correlated even when they are very far apart in time, which is exactly the behaviour that Hurst had measured. Mandelbrot gave this slow persistence a biblical name, the *Joseph effect*, after the seven years of plenty followed by seven years of famine: long stretches where the series stays on the same side of its mean. He contrasted it with the *Noah effect*: sudden, extreme jumps, like the flood. With fBm, long memory stopped being a curiosity of hydrology and became a precise mathematical model that could be applied to many natural and economic series. In finance, the Hurst exponent is used today as a simple thermometer of market efficiency: a value near 0.5 says that prices move like a random walk, in agreement with the efficient-market hypothesis, while a stable value far from 0.5 would point to structure that, in principle, could be exploited.

2.2 From integer to fractional differencing

Long memory creates a practical problem. Time-series models usually need *stationarity*: a mean and an autocovariance structure that do not change over time. A price series p_t is clearly not stationary. The textbook solution is the *difference operator*. Writing L for the lag operator ($L\,x_t = x_{t-1}$), the first difference gives the returns, $(1 - L)\,p_t = p_t - p_{t-1}$. In general, a series is integrated of order d, written $I(d)$, if it must be differenced d times to become stationary. Financial prices are usually treated as $I(1)$ and modelled in returns.

Seen this way, the choice is binary: either $d = 0$ (the raw price, not stationary, with all its memory) or $d = 1$ (stationary returns, with the long-run memory removed). The key insight of Granger and Joyeux [1] and, independently, Hosking [2] is that the order does not need to

be an integer. Using the binomial series, the operator $(1-L)^d$ is well defined for any real d:

$$(1-L)^d = \sum_{k=0}^{\infty} \binom{d}{k} (-L)^k = \sum_{k=0}^{\infty} \omega_k L^k, \qquad \omega_0 = 1, \quad \omega_k = \omega_{k-1} \frac{k-1-d}{k}. \tag{2.2}$$

Fractional differencing is therefore a filter with infinitely many weights ω_k that fade out slowly as we look further into the past. Each transformed value is a weighted sum of the *whole* history of the series, with recent observations counting most and older ones counting a little less. By tuning d between 0 and 1 we slide smoothly between the raw price and its returns, and we obtain a whole family of series with these key properties:

- It is well behaved (*stationary* and *invertible*) as long as $|d| < 0.5$.
- For $0 < d < 0.5$ it has *long memory*: the link between two points k steps apart, $\rho(k)$, fades only slowly, like a power law $\rho(k) \sim c\,k^{2d-1}$, instead of the much faster (geometric) decay of ordinary short-memory ARMA models. In plain words, the past keeps influencing the present for a very long time.
- This is the same long memory seen from another angle. If we split the series into fast movements (quick day-to-day jumps) and slow movements (long, gentle swings), almost all of its weight sits on the slowest ones. A series with no memory would share its weight evenly across all speeds; long memory instead piles it up on the long, slow cycles.
- It speaks the same language as the Hurst exponent: the two are tied by $d = H - 0.5$. So no memory ($H = 0.5$) means $d = 0$, and the persistence Hurst found on the Nile ($H > 0.5$) is simply a positive d.

What about a real price, which is not stationary ($d \geq 0.5$)? The operator can still be applied. The idea is then to look for the *smallest* d that already makes the series stationary. That series, the *fractional returns*, gives up only as much memory as it must in order to be stationary, and keeps the rest. This balance between staying stationary and keeping memory is the central idea of this thesis.

The easiest way to get a feel for the operator is to look at two pictures: its weights, and what it actually does to a real price. Figure 2 shows the weights ω_k for several orders. At $d = 1$ only the first two weights are non-zero: this is just plain returns, $p_t - p_{t-1}$, which use today and yesterday and nothing else. For a fractional d the weights shrink gradually instead of stopping abruptly, so each transformed value still gathers information from a long stretch of the past.

Figure 3 applies the operator to the log-price of the S&P 500 ETF (SPY). A price behaves almost like an $I(1)$ series, so applying $(1-L)^d$ leaves a series of order close to $1-d$; the border between non-stationary and stationary therefore sits at $d = 0.5$. The figure shows

what happens on each side of that border. Just below it (the panel with $d = 0.4$) the series still tracks the price level almost perfectly (correlation +0.98): it has kept nearly all of its memory, but it still drifts, and a strict ADF test with 20 lags refuses to call it stationary. Just above it ($d = 0.7$) the series is now stationary and still keeps a good part of the memory (correlation +0.65). Full differencing ($d = 1$) also makes it stationary, but it destroys the memory almost completely (correlation +0.02). That middle band, stationary but still carrying memory, is exactly what fractional differencing offers and what integer differencing ($d = 0$ or $d = 1$) can never give. The figure also hints at a detail that will matter later: the exact point where the stationary region begins depends on how strict the stationarity test is. Section 2.4 explains why, and Chapter 4 measures it.

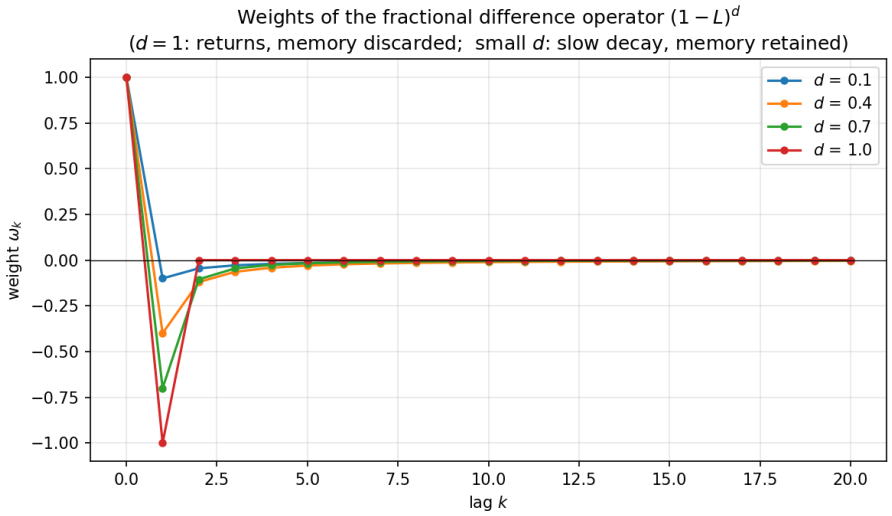


Figure 2: Weights ω_k of the fractional difference operator $(1-L)^d$ for several orders d. Integer differencing ($d = 1$) truncates the past abruptly; fractional orders decay hyperbolically and thus preserve long memory.

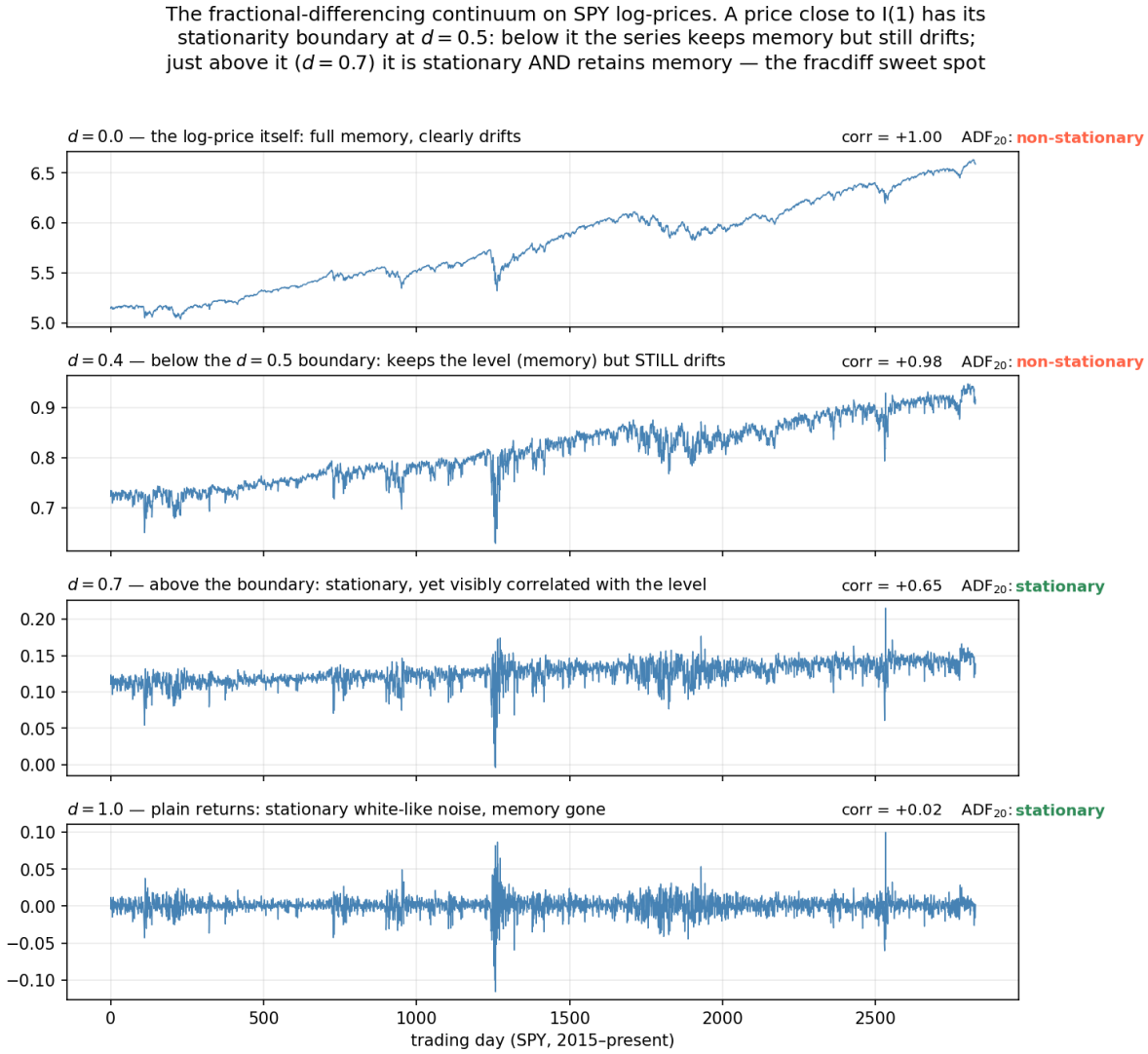


Figure 3: The fractional-differencing continuum on SPY log-prices (since 2015). Each panel reports the correlation of the transformed series with the original log-price (memory kept) and the verdict of an ADF test with 20 lags. For a price close to $I(1)$ the stationarity boundary is at $d = 0.5$: with $d = 0.4$ the series keeps the memory but still drifts; with $d = 0.7$ it is stationary and still keeps memory (the band that fractional differencing makes available); with $d = 1$ it is stationary but the memory is destroyed.

2.3 The ARFIMA model

Putting fractional differencing inside the classical Box-Jenkins framework gives the *Autoregressive Fractionally Integrated Moving Average* model, ARFIMA(p, d, q):

$$\phi(L)\,(1-L)^d\,(x_t-\mu)=\theta(L)\,\varepsilon_t, \tag{2.3}$$

where $\phi(L)$ and $\theta(L)$ are the autoregressive and moving-average polynomials of orders p and q, and ε_t is white noise. The fractional parameter d captures the long-run dependence, and the ARMA part captures the short-run dynamics. ARFIMA generalises ARIMA by letting d take non-integer values. Granger also gave an economic reason for this model: the sum of many independent short-memory components can itself produce long memory, which explains why it appears in aggregate economic series.

Estimating d is a research field in itself. Semiparametric estimators work in the frequency domain. The best known is the log-periodogram regression of Geweke and Porter-Hudak [6], which uses the λ^{-2d} behaviour of the spectrum near zero. Parametric approaches use maximum likelihood or the Whittle approximation. Baillie [7] surveys these techniques and the role of fractional integration in econometrics. One point matters especially for this thesis: in all these methods, d is chosen with a *statistical* criterion (how well it fits the autocovariance or the spectrum), never with a *predictive* one. This difference is what motivates research question RQ2.

2.4 Stationarity and unit-root testing

Everything in fractional differencing depends on a clear notion of stationarity and on a way to test it. A series is (weakly) stationary if its mean and autocovariances do not change over time. In financial prices, non-stationarity usually appears as a *unit root*: an autoregressive root equal to one, as in the random walk $p_t = p_{t-1} + \varepsilon_t$. The standard test is the *Augmented Dickey-Fuller* (ADF) test [8]. It estimates the regression

$$\Delta y_t = \alpha + \gamma\, y_{t-1} + \sum_{i=1}^{p} \delta_i\, \Delta y_{t-i} + \varepsilon_t, \tag{2.4}$$

and tests the null hypothesis $\gamma = 0$ (unit root, so not stationary) against $\gamma < 0$ (stationary). The lagged differences Δy_{t-i} "augment" the regression: their job is to absorb the short-run autocorrelation. The test statistic is the t-ratio of $\hat{\gamma}$. Under the null it does not follow a normal distribution but the Dickey-Fuller distribution. With a constant and no trend, its 5% critical value is approximately -2.86, although the exact value depends on the sample size and the deterministic terms included. If the statistic is below this value, the test rejects the unit-root null, and the series is treated as stationary for the purposes of this procedure.

The KPSS test [9], which takes stationarity as the null instead, is a common complement. The ADF statistic plays a central role in this thesis: following López de Prado, the memory-optimal order d^* is defined as the smallest d for which the fractionally differenced series first passes the ADF test. The critical value -2.86 is therefore the natural anchor of the optimisation.

Two practical properties of the ADF test will matter later. First, its verdict depends on the number of lags p. The lagged differences absorb *short-run* autocorrelation. But when the alternative is a *fractionally* integrated process, whose autocorrelations decay very slowly, a regression with very few lags cannot absorb that dependence. The remaining persistence loads onto $\hat{\gamma}$ and pushes the statistic down, so an ADF with few lags tends to declare stationarity *too early*. Second, the boundary itself is fractional: applying $(1-L)^d$ to an FI(δ) series gives an FI($\delta - d$) series, which is stationary exactly when $\delta - d < 0.5$. The smallest order that makes it stationary is therefore $\delta - 0.5$, not δ. Chapter 4 measures both properties on synthetic series whose true order is known.

2.5 Long memory in financial data: returns versus volatility

Do financial series really have long memory that can be exploited? This has been one of the most debated empirical questions in the field, and the answer depends a lot on *what* is measured.

Returns: weak and disputed. Under the efficient-market hypothesis [10], asset returns should be essentially unpredictable, and in fact they show very little linear autocorrelation. Early claims of long memory in returns were challenged by Lo [11]. He built a *modified* rescaled-range statistic that is robust to short-range dependence and to changes in variance. With this correction, he found little significant evidence of long memory in U.S. stock returns. Lobato and Savin [12] reached a similar conclusion: they carefully separated *real* from *spurious* long memory and found that returns show little genuine long-range dependence. Granger and Hyung [13] added another warning. Occasional *structural breaks* in a short-memory series can look exactly like long memory, so an estimated d may reflect breaks instead of true persistence. More recent studies that track the Hurst exponent over time [14], [15] find that any apparent memory in returns is unstable: it appears around crises, or in some emerging or less liquid markets, and then disappears. It is not a stable property that one could trade on.

Volatility: a solid regularity. The picture changes completely for the *second moment*. Ding, Granger and Engle [16] showed that, although returns are almost uncorrelated, their absolute values (and powers of them) show strong autocorrelation that decays very slowly

and survives over hundreds of lags. This is genuine long memory in volatility. The finding produced an influential family of long-memory volatility models: the Fractionally Integrated GARCH (FIGARCH) of Baillie, Bollerslev and Mikkelsen [17], the fractionally integrated EGARCH of Bollerslev and Mikkelsen [18], and later generalisations such as the HYGARCH model of Davidson [19], which contains FIGARCH as a special case. In the modern realized-volatility literature, long-memory and heterogeneous-autoregressive models are still the standard tools for volatility forecasting [20], [21], and the phenomenon keeps being found in newer asset classes such as carbon and energy exchange-traded funds [22]. The message is consistent: in financial data, the persistent memory lives mainly in *risk* (volatility), not in the average direction of returns. This is a crucial warning for any attempt to use memory to forecast returns. Figure 4 reproduces this fact on the data used later in this thesis. The autocorrelations of daily SPY returns are statistically zero after the first lag. The autocorrelations of the absolute returns start near 0.4 and are still clearly positive after forty lags.

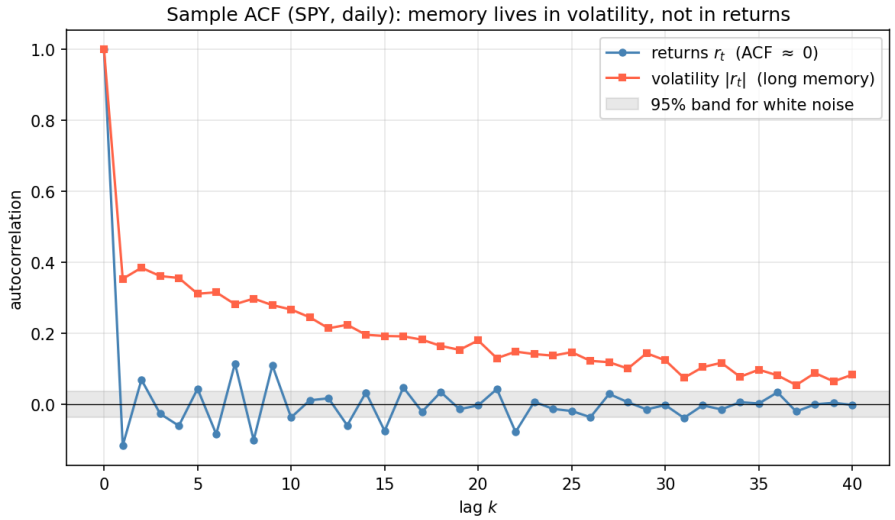


Figure 4: Sample autocorrelation functions of daily SPY returns and absolute returns. The long memory documented by Ding, Granger and Engle [16] is clearly visible in volatility and absent in returns.

Beyond equities. Fractional integration also describes well several macroeconomic and fixed-income series, where mean reversion exists but is slow. Tsay [23] and Gil-Alana [24] model U.S. real and long-term interest rates as fractionally integrated processes, outside the rigid choice between $I(0)$ and $I(1)$. Baillie, Chung and Tieslau [25] find strong long-memory, mean-reverting behaviour in the inflation of several countries with an ARFIMA-GARCH model. All this confirms fractional integration as a useful *descriptive* tool. But describing a series well is not the same as forecasting it better, as the next subsection shows.

2.6 Forecasting with long-memory models

It is not clear at all that long-memory models give better *point forecasts*. Bhardwaj and Swanson [26] report the most positive evidence: ARFIMA models can beat short-memory benchmarks for some macroeconomic and financial series, especially at long horizons. On the other side, Ellis and Wilson [27] compare ARFIMA$(0, d, 0)$ forecasts (with d known and with d estimated) against simple models and a *random walk*, for horizons of up to one hundred steps. They find that the simple models, and the random walk in particular, tend to *beat* the ARFIMA forecasts, and they openly question its value as a forecasting tool. This mixed-to-negative record reflects a general fact of financial econometrics: at short horizons, the naive random walk is very hard to beat. That is exactly the benchmark this thesis adopts for research question RQ1.

2.7 Evaluating forecasts: metrics and the random-walk benchmark

To know if a model really forecasts well, and does not simply fit the data, the evaluation must be out-of-sample, and the metric must be chosen with care. Scale-dependent errors like the root mean squared error (RMSE) and the mean absolute error (MAE) are intuitive, but they cannot be compared across series with different price levels. Computed on prices, they also reward a model just for staying close to the current level. Hyndman and Koehler [28] proposed a scale-free alternative, the *Mean Absolute Scaled Error* (MASE). It divides the mean absolute error of the model by the in-sample mean absolute error of the naive (random-walk) forecast:

$$\mathrm{MASE} = \frac{\frac{1}{n}\sum_t |y_t - \hat{y}_t|}{\frac{1}{m-1}\sum_{t=2}^{m} |y_t - y_{t-1}|}. \tag{2.5}$$

A MASE below one means the model beats the naive forecast; a value of one means it only matches it. The *random walk*, predicting that tomorrow will be equal to today, is the natural benchmark in finance, and a very demanding one: in an efficient market it is close to optimal by construction. Finally, a difference in error between two models can be real or just sampling noise. The *Diebold-Mariano* test [29] answers this question. It builds a statistic from the per-observation loss differences; under the null hypothesis of equal accuracy it is asymptotically standard normal, and the correction of Harvey, Leybourne and Newbold [30] adjusts it for small samples. Together, MASE, directional accuracy and the Diebold-Mariano test against a random walk form the honest evaluation protocol that, as this chapter argues, is too often missing from claims that long-memory preprocessing improves forecasting.

2.8 Recurrent neural networks and Long Short-Term Memory

The models used in this thesis to learn from the (fractionally differenced) series are *recurrent neural networks* (RNNs), the natural deep-learning architecture for sequences. A simple (Elman) RNN [31] processes a sequence $\{x_t\}$ by keeping a hidden state h_t that summarises the past and is updated at every step:

$$h_t = \tanh(W_x\, x_t + W_h\, h_{t-1} + b)\,, \qquad \hat{y}_t = W_o\, h_t + b_o. \tag{2.6}$$

The same weights are reused at every step, so the network can in principle handle sequences of any length. It is trained by *backpropagation through time*: the recurrence is unrolled and the chain rule is applied across steps.

In practice, however, simple RNNs have problems with *long-range* dependencies. Bengio, Simard and Frasconi [32] showed that, when the distance between relevant events grows, the gradients that travel back through time either *vanish* (they shrink towards zero exponentially) or *explode*. With vanishing gradients, the network cannot learn to connect distant causes and effects. For a problem about long memory, this is an ironic limitation. The solution was the *Long Short-Term Memory* (LSTM) network of Hochreiter and Schmidhuber [33]. The LSTM adds an explicit *cell state* c_t, a protected memory channel, controlled by three multiplicative *gates*: a forget gate f_t, an input gate i_t and an output gate o_t:

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)\,, \qquad i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)\,, \tag{2.7}$$

$$\tilde{c}_t = \tanh(W_c[h_{t-1}, x_t] + b_c)\,, \qquad o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)\,, \tag{2.8}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \qquad h_t = o_t \odot \tanh(c_t), \tag{2.9}$$

where σ is the logistic sigmoid and $\odot$ is the element-wise product. Figure 5 shows the resulting information flow. The additive update of the cell state creates a path through which gradients can travel many steps without vanishing. This lets the network keep information for long periods and learn much longer dependencies than a plain RNN.

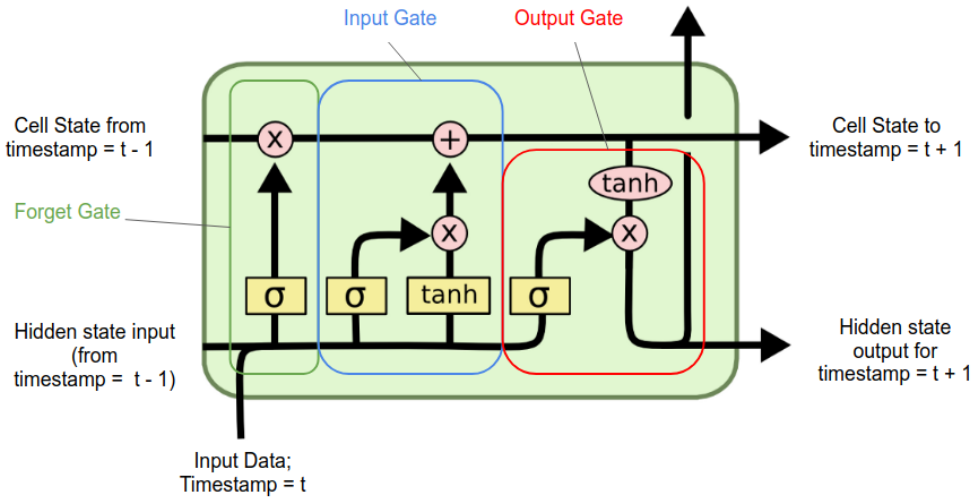


Figure 5: Schematic of an LSTM cell. The cell state c_t runs along the top as a protected memory channel. The forget, input and output gates, all functions of $[h_{t-1}, x_t]$, control what is erased, what is written and what is exposed at each step. *Source:* [34].

The Gated Recurrent Unit (GRU) of Cho et al. [35] appeared later as a lighter alternative with only two gates and similar performance. RNNs, and LSTMs in particular, are standard tools for financial time-series forecasting today. Even so, the same sobering pattern appears here too: out of sample, deep sequence models often fail to beat the random walk on returns, which agrees with the efficiency of liquid markets. This work deliberately uses a single-layer LSTM and a plain RNN of the same size as its two architectures, so that the conclusions do not depend on one particular network design.

2.9 Fractional differencing meets machine learning: the frontier and the gap

The most recent research line, and the closest one to this thesis, joins fractional differencing with machine learning. The main reference is López de Prado [3]. In *Advances in Financial Machine Learning* (Chapter 5, "Fractionally Differentiated Features") he argues that feeding returns to ML models wastes information: by over-differencing to guarantee stationarity, practitioners destroy the memory that predictive models could use. His proposal is the *fixed-width window* fractional difference: apply $(1 - L)^d$ with the weight sequence cut when the weights become very small, and choose the *minimum* order d for which the series first passes a stationarity test, usually the ADF test. He shows that, for many financial series, a small order (often $d < 0.5$) already gives a stationary series that still has a correlation above 0.9

with the original price level, while full differencing ($d = 1$) brings that correlation close to zero. The example makes very visible how much memory integer differencing throws away. This recipe of "maximum memory subject to stationarity" has become a popular feature-engineering step in quantitative finance. Figure 6 reproduces the recipe on SPY log-prices exactly as it appears in the original, including its 1-lag ADF test: the correlation with the level is still 0.98 at the first order that passes the test, $d^* \approx 0.4$. The figure should be read together with the warning of Section 2.4: with a stricter ADF specification the crossing point moves clearly to the right. Chapter 4 measures this sensitivity, which the recipe itself does not examine.

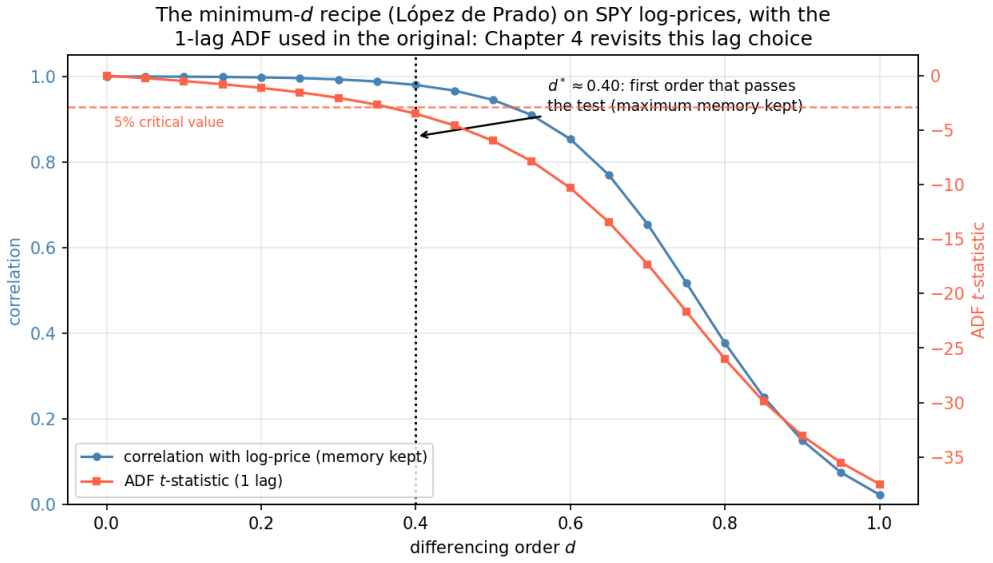


Figure 6: The minimum-d recipe of López de Prado [3] on SPY log-prices: memory retained (correlation with the log-price, left axis) and the 1-lag ADF statistic (right axis) as functions of the differencing order. The recipe selects the first d whose statistic crosses the 5% critical value.

The recipe caught on quickly because it became very easy to use. Free, open-source libraries compute fractional differencing fast and plug straight into `scikit-learn`; one of them is even written in a *differentiable* way in PyTorch, which means the operator can sit *inside* a neural network and be trained together with it by gradient descent [36], and there are GPU versions for very large datasets. In practice, these fractionally differenced features are usually combined with the rest of López de Prado's toolkit (triple-barrier labels, meta-labelling and purged cross-validation) and then fed to gradient-boosted trees, random forests and, more and more often, deep sequence models.

A growing number of applied papers report that this step *improves* predictions: Walasek and Gajda [37] on four equity indices, Tang [38] on Chinese futures, Bieganowski and

Ślepaczuk [39] with supervised autoencoders, and Stempień and Gajda [40] with an LSTM all find lower errors or better trading performance than with integer differencing. Several of them honestly add that the gains are fragile and can vanish under noise or a change of hyper-parameters.

Taken together, however, this evidence is methodologically weak. The comparison is almost always the same, *fractional versus integer* differencing judged by error or trading metrics, while two basic checks are missing: a *random-walk* benchmark (the forecast that tomorrow equals today) and a significance test such as Diebold-Mariano. Most positive results are also unreviewed preprints. This matters because a fractionally differenced series sits *between* the price level (almost a random walk) and the returns (almost pure noise): the more of the level it keeps, the more predictable it *looks*, simply because prices barely move from one day to the next. An apparent edge can therefore be an artefact of the comparison rather than genuine forecasting skill.

Two questions therefore remain open, and they define this thesis. *First* (RQ2): in nearly all the work above, d is fixed beforehand by a statistical rule and frozen before training; nobody has let a network *learn* d by backpropagation and checked whether the prediction-best order matches the memory-best order d^*. *Second*: the claim is almost never tested *honestly*, against a random walk, with a scale-free metric and a significance test, and validated on data with *known* memory. This thesis closes both gaps: it makes d learnable, compares the memory-best and prediction-best orders across many assets and two architectures, and evaluates everything against a random-walk benchmark together with a *synthetic ARFIMA control*: artificial price series generated by an ARFIMA process with a *known* amount of long memory, which the whole pipeline must detect before any result on real data is trusted (the positive control of Section 4.6).

Chapter 3 Motivation and Objectives

3.1 Motivation

The state of the art reviewed in Section 2 leaves the field in an awkward position. On one hand, fractional differencing has become a standard first step in financial machine learning: it comes built into mature open-source libraries, it is taught as part of a widely used methodology [3], and a growing line of applied papers claims that it improves forecasts [37], [38], [39], [40]. On the other hand, the evidence is shaky: the studies almost always compare fractional against integer differencing on the price itself, rarely include a random-walk benchmark or a significance test, and never validate the method on data whose memory is known in advance. As Section 2.9 explained, this gap is not harmless: a fractionally differenced series lies between the price level and noise-like returns, so an apparent improvement may only reflect how much of the price level it keeps, rather than real predictive skill. The recipe's main promise, that the memory it preserves actually helps prediction, has never been tested honestly, and only a rigorous design can show whether it holds.

There is a second, more conceptual gap. In every study reviewed, the order d is fixed in advance by a statistical rule (keep as much memory as possible while staying stationary) and then frozen while the model is trained. But the order that best preserves memory need not be the order that best helps prediction: the *memory optimum* d^* depends only on the series, whereas the *predictive optimum* d_{opt} also depends on the model and on the error it minimises. Differentiable versions of the operator now let us ask the question directly: place $(1-L)^d$ inside the network, let backpropagation tune d, and compare it with d^*. Whether the two optima coincide, move together, or are unrelated has never been studied, and, as this thesis shows, under flexible models d_{opt} may not even exist.

Finally, there is a reason of method. In financial forecasting, negative results usually tell us little, because the reader cannot separate "there is no signal" from "the method could not find it". This work is built so that its conclusions, positive or negative, can be interpreted: every part of the pipeline is first tested on synthetic series with known long memory before being trusted on real data, and every claim of improvement must beat the random-walk benchmark and pass a significance test. Building and validating such a protocol is itself a contribution that outlasts the specific results reported here.

3.2 Research Questions and Objectives

The two research questions raised in the Introduction can now be stated in a precise, testable form.

- **RQ1 (forecasting).** Does fractional differencing make a recurrent network's one-day-ahead price forecast more accurate than plain returns and, above all, than the naive random walk?

- **RQ2 (representation).** Is the differencing order that forecasts best the same as the one that best preserves memory? This breaks into two parts:

  - **RQ2a.** On real assets, does the predictive order d_{opt} coincide with the memory order d^*?

  - **RQ2b.** Does a single best order d_{opt} even exist, and if so, how does it relate to d^*?

RQ1 is deliberately strict about what counts as "more accurate". The forecasts come from two recurrent networks, an LSTM and a simple RNN, and are compared against two references: ordinary returns (full differencing, $d = 1$) and the naive random walk, which predicts that tomorrow will equal today and is the honest baseline that any genuine skill must beat. Accuracy is measured with MASE, an error metric that does not depend on the price scale of each asset and can therefore be compared across markets, and every difference is confirmed with the Diebold-Mariano significance test, so that an apparent gain cannot simply be due to chance.

RQ2 contrasts two orders that answer different questions. The *memory optimum* d^* depends only on the series: it keeps as much long memory as possible while the series stays stationary. The *predictive optimum* d_{opt} also depends on the model and on the error it minimises: it is the order that gives the best forecasts in practice. RQ2a asks, as the literature would, whether the two move together across assets; to keep the answer from depending on the search method, d_{opt} is found in two independent ways, learned by backpropagation or located by scanning every candidate value. RQ2b is more basic, and the split is not cosmetic: comparing two quantities only makes sense if both are real. If the forecasting error barely changes with d (a flat surface), the order that happens to minimise it is essentially random, and a near-zero correlation with d^* would mean nothing. RQ2b therefore checks whether a genuine best order exists at all (the question of *identifiability*) and answers it on synthetic series whose true memory is known in advance, so that there is a correct answer to compare against. Chapter 4 develops this analysis and the tools used to measure d; Chapter 5 reports the answers.

These questions lead to one main objective and three more specific ones, which follow the original project definition:

- **Main objective.** To find out, using a fair and honest evaluation protocol, whether fractional differencing really improves the forecasting of financial time series with recurrent neural networks (RQ1), and how the order that forecasts best relates to the order that preserves the most memory (RQ2).

- **O1.** To build a *differentiable* estimator of the memory-optimal order d^*, that is, one that can be plugged directly into gradient-based training. Rather than balancing memory and stationarity with arbitrary weights, it is anchored to the Augmented Dickey-Fuller (ADF) test, a standard statistical test of stationarity, which gives the estimator a clear meaning. This objective also includes calibrating it, in particular measuring how sensitive it is to the number of lags chosen for the ADF test (Section 4.3).

- **O2.** To build a forecasting model in which the differencing order d is not fixed in advance but is a *trainable parameter*, learned at the same time as the network weights. The order the model settles on, d_{opt}, is then compared with d^*, both on real assets and on synthetic series whose memory is known (Sections 4.4 and 4.7; results in Section 5.3).

- **O3.** To add a *synthetic positive control* to the evaluation: a set of artificial series built to contain a known amount of memory. Before any conclusion drawn from real data is accepted, the whole pipeline must pass a test on these series, detecting and using the memory when it is present and finding nothing when it is absent. The idea is the same as a positive control in a laboratory: a method that cannot find memory we have deliberately put in cannot be trusted to find it in real markets (Section 4.6; results in Section 5.1).

Each objective can be checked. O1 is achieved if the estimator recovers the known stationarity boundary on synthetic series whose order we set ourselves. O2 is achieved by describing the order the model learns and the shape of the forecasting error as a function of d. O3 is achieved if the synthetic control passes in every one of its cases. The main objective is achieved by answering RQ1 and RQ2 with statistically solid evidence, regardless of whether the answers turn out positive or negative.

Chapter 4 Methodology

This chapter builds the machinery for the two questions of this work, and it helps to keep them apart from the outset. **RQ1** asks whether fractional differencing improves forecasting at all; **RQ2** asks how the order that *forecasts* best, d_{opt}, relates to the order that preserves the most *memory*, d^*. Most of the chapter builds and validates the forecasting experiment behind RQ1: the problem and its notation (4.1), fractional differencing inside the network (4.2), the forecasting architectures (4.4), the evaluation protocol (4.5), and the synthetic control that gates every conclusion (4.6). Two subsections serve RQ2 instead, and together form the conceptual spine of the chapter: the estimator of d^* (4.3) and the identifiability analysis that explains why d_{opt} is so hard to measure in the first place (4.7). Section 4.8 then collects the entire design in a single table. Throughout, design decisions are justified where they are introduced; the theoretical background (long memory, ARFIMA, ADF, MASE, Diebold-Mariano, LSTM) was established in Section 2 and is not repeated.

4.1 Problem Formulation and Notation

Let p_t be the daily closing price of an asset, and let $x_t = \log p_t$ be its *log-price*. We always work with the log-price, not the raw price, for three simple reasons. First, a full difference of the log-price gives exactly the *log-return*: $(1 - L)\, x_t = x_t - x_{t-1} = \log(p_t/p_{t-1})$. So as the differencing order d moves from 0 to 1, the series moves smoothly from the log-price ($d = 0$) to the log-return ($d = 1$), the form finance normally works with. Second, prices move in percentages, not in fixed amounts: a 1% change means the same whether the price is 10 or 1000. The logarithm turns those percentage changes into plain additions, which keeps the size of the moves roughly constant over time and makes the series behave like the $I(1)$ case that the ADF test and the operator $(1-L)^d$ assume. Third, log-returns have no units, so the same model and the same error metric (MASE) can be compared across assets whose prices are on very different scales.

The goal is *one-step-ahead forecasting*: on each day t, using only what is known up to t, predict tomorrow's price $\hat{p}_{t+1}$. The model is fed a sliding window of the last W log-prices, $\mathbf{x}_t = (x_{t-W+1}, \ldots, x_t)$, and is asked to predict the *next log-return*,

$$r_{t+1} = x_{t+1} - x_t, \tag{4.1}$$

which is standardised (centred and scaled) with the mean and standard deviation of the returns in the training set. From the model's predicted return $\hat{r}_{t+1}$, the price forecast is rebuilt in one step, starting from the *real* price of the previous day,

$$\hat{p}_{t+1} = p_t\, e^{\hat{r}_{t+1}}, \tag{4.2}$$

Because every forecast starts from the real previous-day price, and not from a previously predicted one, errors do not pile up day after day.

Predicting the return instead of the price level is a deliberate choice. The return (Equation (4.1)) is *stationary*: its typical size stays roughly the same over time, so the model always sees numbers on a stable scale. Predicting the price level would be very different. The level would have to be rescaled using values measured during training, and as soon as the test prices rise above anything seen in training, the model would be forced to *extrapolate* into unfamiliar territory. Since financial prices trend upward for long stretches, this happens constantly, and the large errors it produces would be a side effect of the setup rather than a real failure of the model (an early version of this pipeline broke in exactly this way; see Section 5.1). The baseline the model must beat is the naive *random walk* (RW), which simply predicts that tomorrow's price equals today's, $\hat{p}_{t+1}^{RW} = p_t$. This is just the special case $\hat{r}_{t+1} = 0$ of Equation (4.2): predicting a zero return *is* the random walk. So the model can only beat it by finding genuine predictable structure and placing it into $\hat{r}_{t+1}$. Figure 7 summarises the whole pipeline.

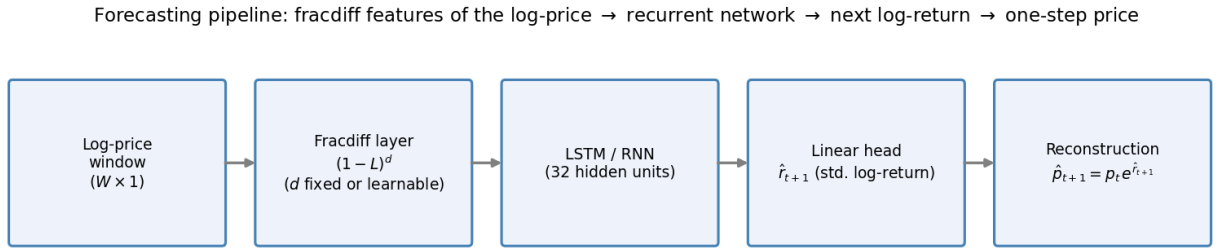


Figure 7: The forecasting pipeline. A window of log-prices is fractionally differenced inside the network (with d fixed or learnable), processed by a recurrent network, and mapped to the next standardised log-return (an *ordinary* return, not a fractional one); the price forecast is reconstructed in one step from the true previous price.

Two quantities are compared throughout this work, and fixing their names now avoids ambiguity later:

- The **memory optimum** d^*: the smallest differencing order that makes the fractionally differenced log-price stationary. This is exactly López de Prado's "maximum memory subject to stationarity" order, and it depends on the *series* alone; it is estimated as described in Section 4.3.

- The **predictive optimum** d_{opt}: the order that gives the lowest forecasting error on unseen (out-of-sample) data. Unlike d^*, it depends not only on the series but also on

the model and the loss used to measure it, and that dependence turns out to be the crux of RQ2 (Section 4.7).

4.2 Fractional Differencing in Practice

The operator $(1-L)^d$ of Section 2.2 has infinitely many terms, so inside a network it has to be *truncated*. Keeping its weights ω_k (from the recursion of Equation (2.2)) only up to a maximum lag K, the fractionally differenced feature at time t becomes a finite weighted sum of the most recent values,

$$\tilde{x}_t^{(d)} = \sum_{k=0}^{K} \omega_k \, x_{t-k}, \tag{4.3}$$

This is a one-dimensional convolution, and its kernel, the weights ω_k, can be differentiated with respect to d. That single fact is what later lets d be *learned* like any other network parameter. Three practical details matter.

Truncation. Two lengths decide how far back each feature looks: the input window W and the filter length K. The main experiments use $W = 30$ and $K = 5$, kept deliberately identical to the original project design so that any change in the results comes from the corrected evaluation and not from re-tuning these numbers. The robustness analysis of Section 5.2 pushes them up to $W = 250$, $K = 100$, to confirm that no conclusion is merely an effect of using a short memory. The synthetic experiments, where long memory is known to be present, use $W = 120$, $K = 50$ throughout.

Does $K = 5$ keep enough of the filter? It is a fair worry. The whole idea of long memory is that the weights ω_k fade away *slowly* (like k^{-d-1}), so a long tail reaching far into the past is exactly what we want to keep, and cutting the filter at $K = 5$ throws part of that tail away. How much? We can measure it directly, as the fraction of the filter's total weight that survives the cut, $\sum_{k=0}^{K} |\omega_k| \big/ \sum_{k=0}^{\infty} |\omega_k|$. For the orders that matter on real assets (d^* between 0.06 and 0.45), $K = 5$ keeps about 73% of the weight at $d = 0.1$ and 86% at $d = 0.45$; in other words it discards a 14–27% tail, the slow part that carries the memory. Using $K = 100$ instead recovers 85–96%. So why keep $K = 5$ at all? Only to copy the original project design exactly, so that any change in the results comes from the corrected evaluation and not from re-tuning. It is best seen as a deliberately *short-memory* setting, not a faithful long-memory filter. And nothing important hangs on it: Section 5.2 re-runs everything at $W = 250$, $K = 100$, which puts the tail back, and every verdict stays the same.

Padding. At the very start of a window there are not enough past values to compute Equation (4.3), so the missing positions have to be filled. The library default, zero-padding, is wrong for log-prices: a value of 0 means a price of $e^0 = 1$, so it glues a fake price of 1 in front of a series whose real level is arbitrary, creating a sudden jump that the filter then

spreads through the output. Instead, the pipeline fills those positions by *repeating the first value of the window*, which introduces no artificial jump.

Scaling. The size of $\tilde{x}^{(d)}$ changes enormously with d: near $d \approx 0$ it is on the scale of the price level, while at $d = 1$ it is on the much smaller scale of returns. To stop these differences in magnitude from driving the comparison, each feature is divided by the standard deviation of the filtered *training* series for that same d. Every model then receives inputs of similar size, so a difference in performance reflects the information in the feature, not its scale.

One point needs to be spelled out, because the learned-d mode makes it tricky. When d is fixed (the baseline and fixed-d modes), the divisor is just the training standard deviation at that d, and there is no issue. But when d is learned it changes at every step, so which d do we use for the divisor? We compute it *once*, at the starting d, and then leave it fixed: it is not updated as d moves, and no gradient passes through it. This keeps training stable, but it has a price, the learned-d gradient is not perfectly corrected for the change in size that comes with d. One could worry that *this* is what makes the loss surface flat. It is not. In the fixed-d grid of Section 5.3, where each d *is* scaled with its own correct standard deviation, the curve comes out just as flat. So the flatness is real, not a side effect of the frozen scaling, and its true cause is the identifiability problem of Section 4.7.

4.3 A Differentiable, ADF-Anchored Estimator of d^*

As anticipated in the chapter map, d^* is one of the two orders that RQ2 will compare (Section 4.7); this subsection builds the estimator that measures it, before the forecasting machinery of the following subsections turns to RQ1.

The memory optimum d^* is the answer to a constrained problem: keep as much memory as possible, but only as far as the series stays stationary,

$$d^* \;=\; \arg \max_{d \in [d_{\min}, d_{\max}]} \; m(d) \quad \text{subject to} \quad t_{ADF}\big(\tilde{x}^{(d)}\big) \le c, \tag{4.4}$$

Here $m(d)$ measures how much memory survives the filtering; following López de Prado, it is the Pearson correlation between the filtered series and the original log-price. The estimator also supports a second, fully differentiable memory measure: the Hurst exponent, estimated by the rescaled-range (R/S) analysis of Section 2.1 and re-implemented in PyTorch so that, exactly like the correlation, its gradient flows back to d and can drive the optimisation. The two are interchangeable in the objective below; all d^* values reported in this work use the correlation, with the Hurst variant available as a cross-check. The constraint uses the ADF statistic t_{ADF}, and $c = -2.86$ is its approximate 5% critical value (with a constant term): the series counts as stationary once its statistic drops below c. Since $m(d)$ always decreases as d grows, keeping the most memory means differencing as little as possible, so the best d

sits exactly on the line where the series just turns stationary. The estimator's only job is to find that line *smoothly*, so that the search can be carried out by gradient descent. Two ingredients make this possible.

A differentiable ADF statistic. The ADF test of Section 2.4 is, underneath, an ordinary least-squares regression, and everything it needs, the regression coefficients, the residual variance and the standard error of the key coefficient $\hat{\gamma}$, can be written with explicit formulas in terms of the data matrix. Re-implementing those formulas with differentiable tensor operations turns the test into a quantity $t_{ADF}(\tilde{x}^{(d)})$ that can itself be differentiated: gradients flow from the statistic back to d through the filter weights. The number of extra (augmentation) lags p in the regression is left as an explicit setting, because, as shown next, it matters a great deal.

A penalty objective anchored to the test. The constraint is turned into a cost to be minimised,

$$\mathcal{L}(d) \;=\; -\,m(d) \;+\; \lambda \left[\,\mathrm{ReLU}\big(t_{ADF}(\tilde{x}^{(d)}) - c\big)\right]^2, \tag{4.5}$$

minimised with the Adam optimiser, after rewriting $d = d_{\min} + (d_{\max} - d_{\min})\,\sigma(\theta)$ so that the order can never leave its allowed range. The key is the penalty term: it is *exactly zero* while the filtered series already passes the ADF test, and grows with the square of the violation once it does not. So λ is not an arbitrary weight trading memory against stationarity; it is simply how firmly a real constraint, fixed at the published critical value, is enforced, and the result barely changes once λ is above a moderate level. The left panel of Figure 8 shows this on a synthetic series of known order: the penalty acts like a wall that pushes d out of the non-stationary region, leaving the minimum right at the boundary, as intended.

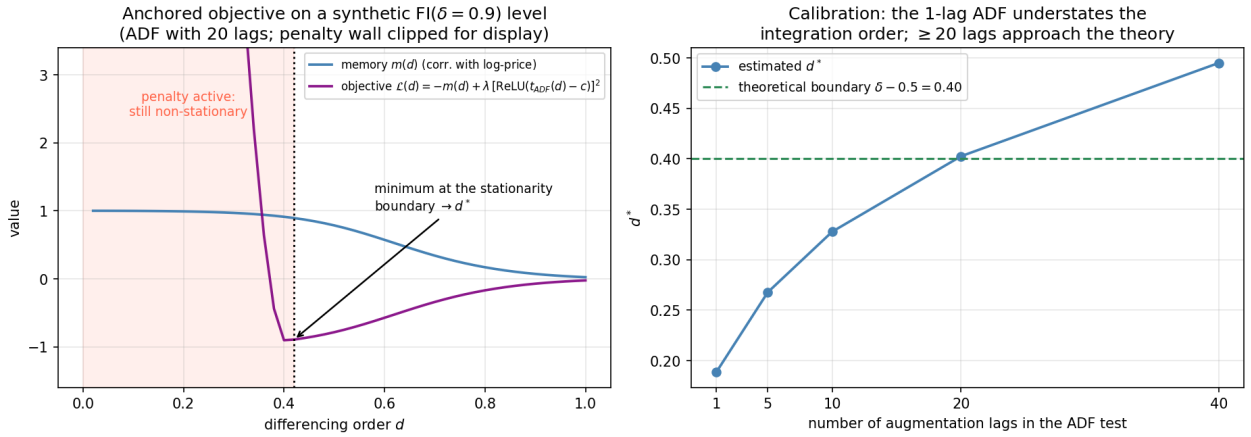


Figure 8: The d^* estimator. Left: anchored objective of Equation (4.5) on a synthetic FI($\delta = 0.9$) log-price; the minimum lies at the stationarity boundary. Right: calibration of the ADF lag choice on the same series, whose theoretical boundary is $\delta - 0.5 = 0.40$; the 1-lag test understates the order ($d^* = 0.19$), while 20 augmentation lags recover the theoretical value ($d^* = 0.40$).

Calibration: how many ADF lags. The ADF test has one setting that turns out to matter a lot: how many *lags* it includes. Section 2.4 warned that, with too few lags, the test wrongly declares a long-memory series stationary too early. We can measure exactly how big this error is, because we can build a synthetic series whose answer we already know: a fractionally integrated series of order δ becomes stationary precisely at $d = \delta - 0.5$. So we run the estimator on it with different numbers of lags and compare each result against that known value. For $\delta = 0.9$, where the right answer is 0.40, the estimate grows steadily as we add lags: 0.19, 0.27, 0.33, 0.40 and 0.49 for 1, 5, 10, 20 and 40 lags (Figure 8, right). Too few lags fall well short; around 20 lags it lands on the correct 0.40; beyond that it overshoots, because piling on even more lags makes the test lose power. Two things follow. First, the popular 1-lag choice, the one used in the original minimum-d recipe, makes long-memory series look *less* integrated than they really are, which squashes the genuine differences in d^* from one asset to another. Second, for this reason, whenever d^* is measured in this work to be compared with d_{opt} (the mechanistic study of Section 4.7 and the cross-sectional analysis of Section 5.3), it is always computed with 20 lags, a value fixed in advance by this calibration; the 1-lag number is shown beside it only so that the results stay comparable with the literature.

4.4 Forecasting Architectures: FracReturnNet and the Three d-Modes

Every price-forecasting experiment uses the same network, *FracReturnNet*, and the data flows through it in three simple steps. First, the window of log-prices is fractionally differenced with Equation (4.3) (using replicate padding and train-based scaling). Second, the filtered window is read by a recurrent network with 32 hidden units. Third, a linear layer turns the last hidden state into the prediction: the standardised next log-return. We run this with two kinds of recurrent cell, an LSTM and a plain Elman RNN of the same size, so that no conclusion depends on the choice of cell. The network is trained to minimise the mean-squared error of the standardised return with Adam (learning rate 10^{-3}, batch size 256).

A word on the size of the network. We do not use a single recurrent layer because it is the only option we tried; deeper stacks are perfectly possible, and we did test them. We keep one layer as the standard on purpose. The aim of this work is not to build the strongest possible forecaster, but to isolate what *fractional differencing* contributes. A small model has little capacity of its own, so most of its accuracy has to come from how well the input is represented, that is, from the differencing order. With a large model the network could compensate for a poor choice of d and hide its effect. Keeping the model small is therefore the cleaner test: it lets the benefit of fractional differencing show through as clearly as possible.

The architecture is instantiated in three modes, which differ only in how the differencing

order is set (Table 1):

Table 1: The three modes of FracReturnNet. The only difference between models is the treatment of the differencing order d; everything else (architecture, training, evaluation) is identical.

Mode	d	Trained?	Learning rate of d	Role
Baseline	1 (returns)	no	n/a	integer-differencing reference
Fixed-d	d^* (Sec. 4.3)	no	n/a	the memory optimum (maximises memory)
Learned-d	free parameter	yes	0.05	the predictive optimum, end-to-end

The learned mode needs some care, and it is worth seeing exactly why. The differencing order is stored as a free parameter, kept inside the range $(0.01, 1.0]$ and *initialised at* 0.1. We start at this low end on purpose: a small d means very little differencing, and therefore the *most memory retained*, which is precisely the regime fractional differencing exists to exploit. Starting from maximum memory (minimum differencing) and letting the optimiser add differencing only if prediction genuinely demands it is the natural way to search, since it keeps as much of the original signal as possible for as long as possible. The upper limit of 1.0 is set because going past plain returns ($d = 1$) only hurts performance, as the loss landscape in Section 5.3 confirms.

The delicate part is how fast d is allowed to learn. It is given a *dedicated learning rate* of 0.05, fifty times larger than the one used for the network weights, and this is not a cosmetic choice. We audited the optimisation and found something subtle: the gradient of the loss with respect to d is computed correctly (it agrees with a finite-difference check to five decimal places), but it is extremely small, and the loss surface along d is almost flat. With the ordinary learning rate, d barely moves from where it started over an entire training run, so any "learned order" it reports would simply echo its initial value rather than reflect anything in the data. The larger, dedicated learning rate lets d actually traverse its range instead of staying frozen at the start.

We have to be honest about what this buys us, and what it does not. When the loss surface is this flat, letting d move freely does *not* make it settle on a clean, data-driven value; it simply drifts under the random noise of each minibatch and stops wherever the geometry happens to push it, usually at a clamp edge. So the bigger learning rate fixes *one* problem, a d that just echoes its starting value, but it cannot create a real optimum where the loss has none. The final d must therefore be read *alongside* the identifiability analysis of Section 4.7, not trusted on its own as a clean estimate of d_{opt}. In fact, this near-flat surface is exactly the symptom that section sets out to explain.

One last choice keeps the experiment honest, and we arrived at it the hard way. An earlier version of the pipeline did attach a stationarity penalty to the learned order. We removed it

once we realised it made the central comparison in RQ2, between the learned order and d^*, unfair: penalising the learned d for being non-stationary pulls it towards d^* by construction, so the two ending up close would have been a property of the loss, not a finding about the data. The learned mode is therefore trained with a *purely predictive* loss, with no stationarity penalty attached, which is what makes that comparison meaningful.

4.5 Evaluation Protocol

Data and split. Every series is a set of daily closing prices (auto-adjusted) downloaded from Yahoo Finance, from 2015-01-01 to 2026-06-10. We cut each series in time order: the first 80% is the training part and the last 20% is the test part (for a series with the full history, this places the test period roughly from early 2024 onwards). Nothing that happens after the cut point is ever used to fit anything. The network predicts the next *ordinary* log-return, not a fractional one.

We use a *single* time split, not a rolling (walk-forward) one, and we say so openly: it means each verdict is read off just one market period, the most recent 20% of the series. We accept this for two reasons. First, our main conclusions do not rest on any single series; they are patterns that show up again and again across many assets, both network cells, the synthetic gate and the 24-asset cross-section, which a one-off quirk of the test period could not produce. Second, every setup refits d^*, the feature scales and the full network, so a proper walk-forward would multiply an already heavy set of experiments many times over. A rolling evaluation is the obvious next step, and we list it among the limitations of Section 5.5.

Metrics and decision rule. We measure accuracy with the MASE of Section 2.7. Its denominator (the average size of the naive one-step error) is computed *using the training part only*, so the yardstick never sees the test data. To check whether a difference is real and not luck, we use the Diebold-Mariano test on the squared errors at horizon one, with the Harvey-Leybourne-Newbold correction for small samples [30]. The rule is strict: a model "beats the random walk" only if its DM statistic is negative *and* $p < 0.05$. Any MASE difference without that significance is treated as noise, whichever way it points.

One mismatch is worth flagging on purpose: MASE is built on *absolute* errors, while the DM significance test uses *squared* errors. We keep both deliberately. The squared-error version is the standard, original form of the Diebold-Mariano test; MASE is reported because it is scale-free and easy to read across assets. With the heavy tails of financial data the two could in principle point different ways, so we checked, and in every case here they agree: whenever MASE says a model is better or worse than the random walk, the squared-error DM test says the same. No "beats / does not beat" decision depends on which error we use.

Leakage control. A handful of quantities are dangerous because, if computed on the

wrong data, they leak the future into the model. We list them explicitly and force all of them to be computed on training data alone: the statistics used to standardise the target, the feature scale of Section 4.2, the estimate of d^*, and the MASE denominator. On top of that, no input window is allowed to cross the train/test cut. This list is not red tape: these are exactly the points where subtle leakage slipped into earlier versions of this project and into comparable published work.

Honest model selection. Sometimes a hyperparameter has to be *chosen* from the data instead of fixed in advance (here, only the differencing order of the volatility study, Section 5.4). When that happens we never look at the test set to choose: we set aside an internal validation split (the last 20% of the training windows), pick the value there, retrain the model on the whole training set with that value, and then evaluate the test set *exactly once*. Choosing on the test set and then reporting significance on that same test set, which is easy to do by accident when scanning many d values, inflates the results, so we avoid it deliberately.

Determinism. Every experiment fixes the random seed (42; the multi-seed probe of Section 4.7 uses seeds 42 to 51), so anyone re-running the public code reproduces the numbers in this document. The only caveat is that the data provider keeps serving prices up to the day you run it, so running later shifts the last decimals; every verdict reported here is stable to that.

4.6 The Synthetic Control as a Gate

Sections 4.4 and 4.5 fixed the models and how they are scored; this subsection adds the one safeguard that makes the whole RQ1 experiment trustworthy.

If the pipeline says "no predictability" on real data, that answer only means something if we are sure the pipeline *could* have found predictability when it was really there. So we feed it a test case where we already know the answer. We build artificial log-returns that contain long memory on purpose, using an ARFIMA$(0, d_{true}, 0)$ process,

$$u_t = (1 - L)^{-d_{true}} \varepsilon_t, \qquad \varepsilon_t \sim \mathcal{N}(0, 1), \tag{4.6}$$

which we generate with the binomial weights of the *inverse* operator cut off at 4,000 lags, discarding the first 3,000 observations as burn-in and rescaling the returns to a realistic 1% daily volatility; the synthetic price is then $p_t = p_0 \exp(\sum_{s \le t} u_s)$ with $N = 4{,}000$ observations. The key is to know *where* the memory lives: we put it in the *returns* u_t, not in the price. The number d_{true} controls how much. We use three values:

- $d_{true} = 0$: the returns are pure noise ($u_t = \varepsilon_t$), so the next return cannot be guessed from the past. This is the *negative* control. (The price is still a random walk, which

is highly persistent; what is unpredictable is the next *return*, and that is what matters here.)

- $d_{true} = 0.30$ and $d_{true} = 0.45$: the returns now have real long memory, meaning past returns genuinely help predict the next one, while the series stays stationary.

Each of these is forecast with both architectures, using exactly the same protocol we will later use on real data.

Now the test is simple. If $d_{true} = 0$, there is nothing to predict, so *no* model should beat the random walk; if one does, the pipeline is inventing a signal that does not exist. If $d_{true} > 0$, the memory is really there, so the pipeline *must* beat the random walk; if it fails even here, where we built the memory in ourselves, then a "no predictability" result on real data would mean nothing. These six runs (three values of $d_{true} \times$ two architectures) are the *gate*: a pass/fail rule, decided before we look at any real data, that says the real-asset results are only worth reading if all six pass. Section 5.1 reports the outcome, including the time the gate did its job and caught a broken pipeline.

4.7 The Identifiability Problem and the Probe Design

With the RQ1 experiment now fully specified and gated, we turn to RQ2. We have in fact already glimpsed the difficulty: in Section 4.4 the learned order barely moved because its loss surface was nearly flat, and we treated that as a mere optimisation nuisance. It is not. That near-flatness is the visible symptom of a deeper problem, and this subsection confronts it head-on.

RQ2 compares the predictive optimum d_{opt} (the order that forecasts best) with the memory optimum d^* (the order that keeps most memory while staying stationary). But before we can compare them, we have to ask a more basic question: does d_{opt} even *exist* as a well-defined number? The answer turns out to shape the whole empirical strategy.

Why d_{opt} may not exist. Fractional differencing (Equation (4.3)) is just a *linear* filter of the input window, and over the orders we care about it is almost *invertible*: changing d only re-writes the *same* information in a slightly different linear form. Now feed that to a very flexible model. An LSTM is a universal approximator, so it can effectively *undo* the filter and reach the same best loss no matter which d we picked, because any structure it can extract from one version of the window it can extract from all of them. The result is a forecasting loss that is essentially *flat* in d, up to optimisation noise. Three things follow. (i) A learnable d gets no useful gradient, so it just drifts to wherever the loss geometry happens to push it, in practice its clamp boundary or its starting value. (ii) The lowest point of a flat curve is pure sampling noise, so correlating each asset's best d with its d^* is correlating noise with a real

number: the expected correlation is zero *even if a true relationship exists*. (iii) To measure d_{opt} at all, we need a model that is *constrained enough that the choice of d actually matters*. This is argued here in advance and then checked empirically in Section 5.3, first through the behaviour of the learned order and then by retraining exhaustively over a grid of fixed d.

The fix: an anchored one-tap probe. The previous argument said that a flexible model can undo the filter and hide d. So we go the opposite way: we use the *simplest* possible model, one so constrained that it can no longer compensate for the choice of d on its own, but that is still *fair* to the random walk we are trying to beat. Concretely, an ordinary-least-squares regression with just three terms,

$$\hat{y}_t \;=\; \beta_0 + \beta_1\, x_t + \beta_2\, \tilde{x}_t^{(d)}, \tag{4.7}$$

where y_t is the target, the (per-window standardised) log-price one step ahead, and $\hat{y}_t$ is its forecast from a constant β_0, the current standardised log-price x_t (this is the *random-walk anchor*), and a *single* fractionally differenced term $\tilde{x}_t^{(d)}$ (the memory feature, built with order d).

Why is x_t called the anchor? First, recall that the coefficients $\beta_0, \beta_1, \beta_2$ are not chosen by hand: least squares fits them to forecast as well as possible, so the level x_t and the memory term are used *together*, each as much as it helps. Now, the random-walk forecast is just "tomorrow equals today", i.e. $\hat{y}_t = x_t$. Because x_t is in the regression, the random walk is always *available* as a special case (set $\beta_1 = 1$, $\beta_2 = 0$), for *any* d; it acts as a safety net. The point is not that the model is forced to use it, but that it can always fall back on it, so the probe can never lose to the random walk for a silly reason. The consequence is what we are after: since the level is guaranteed by x_t, the memory term only has to supply *memory*, and any change in performance as we vary d reflects *only* how much real memory that order captures. Without the anchor there would be a trap. For some orders the truncated fracdiff weights add up to almost zero, so the filtered series throws away the price level; with no x_t to restore it, the model would be left blind to where the price is and would forecast badly, for a reason that has *nothing* to do with memory. The anchor removes that artefact, keeping the measurement clean.

And why a *single* memory term? Because with only one coefficient β_2 the model cannot hide a poor representation behind extra parameters: all the predictive power must come from the representation itself, that is, from choosing a good d. This is exactly what we want, because now the loss curve over d shows a genuine minimum precisely where real memory exists. We summarise performance with relMASE(d), the probe's test MAE divided by the random walk's, averaged over ten independent runs of each process (so relMASE < 1 means it beats the random walk). Figure 9 contrasts the two regimes.

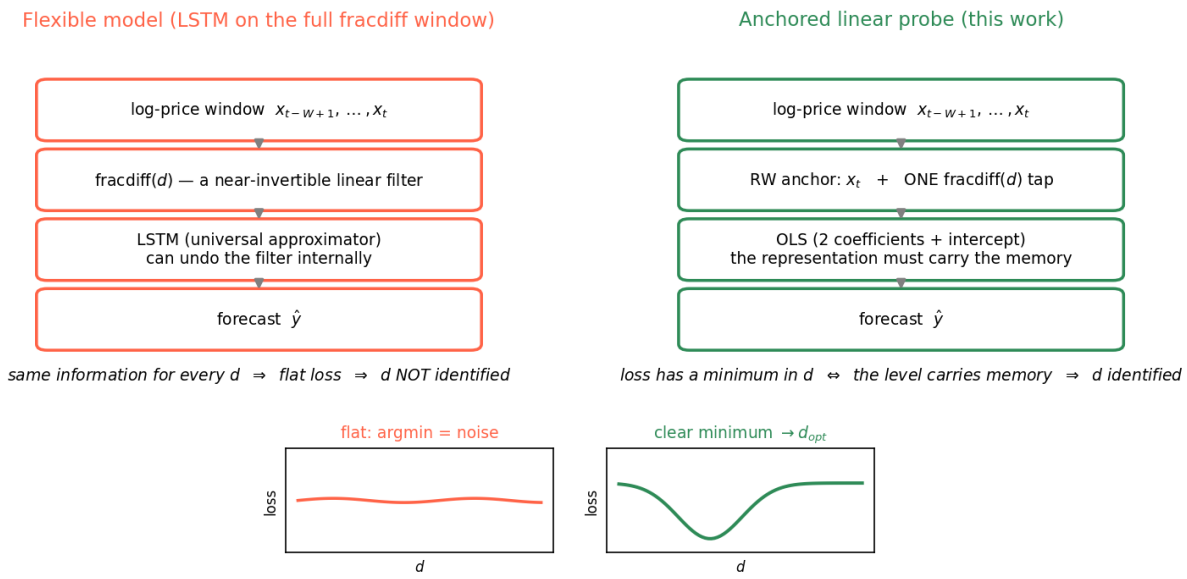


Figure 9: Why a constrained probe is needed to measure d_{opt}. Left: a flexible model can undo the (near-invertible) fracdiff filter, so its loss is flat in d and its lowest point is just noise. Right: the anchored one-tap probe contains the random walk for every d and forces the single memory term to do the work, so the loss in d shows a real minimum exactly when the level carries predictive memory. Bottom: a case is only kept if the depth of its loss curve clears the noise floor measured on the $\delta = 1$ (pure random walk) null.

Deciding when a minimum is real: a filter calibrated on noise. Even with the probe, the loss curve could show a tiny dip just by chance, and we would not want to mistake luck for a real optimum. So we set a simple rule: a minimum only counts as *real* ("identified") if the curve is deep enough. We measure that depth as the relative gap between its highest and lowest points,

$$\text{depth} = \frac{\max_d \text{relMASE}(d) - \min_d \text{relMASE}(d)}{\min_d \text{relMASE}(d)},$$

and we require it to be larger than 0.5%. Where does that 0.5% come from? We measured it on a case that has *no* optimum at all by construction: a pure random walk. Run through the exact same procedure, its curve still wobbles with a depth of 0.07% purely from noise. Our cut-off of 0.5% therefore sits about seven times above that noise floor, so anything we accept is clearly more than random wobble. If a case does not clear the threshold, we declare its best d *undefined* and it contributes no d_{opt} to any later comparison.

Synthetic families where we know the true answer. To test the probe we build

series whose answer we already know. The starting point is a simple idea: in any price series, memory can live in *two different places*. It can sit in the *level* (the price itself), or it can sit in the *returns* (the day-to-day changes). To describe both situations with a single number we use δ, the integration order of the *level*, which we can read as a dial for how much memory the price carries:

- $\delta = 1$ is the reference: a *pure random walk*. The price has full memory (every shock is permanent), but the returns are pure noise, i.e. unpredictable. This is the "normal" financial case.
- $\delta < 1$ means the price has memory, but *less* than a random walk; the returns are then *anti-persistent* (a rise tends to be followed by a pull-back). The memory lives in the *level*.
- $\delta > 1$ means there is so much memory that it spills into the *returns* themselves, which become *persistent* (a rise tends to be followed by another rise).

Our two families are simply two stretches of this dial, each with a known truth:

- **Family B, memory in the *level*.** The price is $x_t = (1 - L)^{-\delta}\varepsilon_t$ with $\delta \in \{0.6, 0.7, 0.8, 0.9, 1.0\}$. The returns are anti-persistent, and theory tells us exactly how much we must difference to reach stationarity: $d^*_{th} = \delta - 0.5$. The extreme $\delta = 1.0$ is the pure random walk, the degenerate "no-optimum" case we used right above to calibrate the 0.5% filter.
- **Family A, memory in the *returns*.** Now the returns themselves follow an ARFIMA$(0, d_{true}, 0)$ process with $d_{true} \in \{0.25, 0.45\}$. This makes the level FI(δ) with $\delta = 1 + d_{true} > 1$, so the returns are persistent. It is the same regime used by the gate of Section 4.6.

So where do *real* prices fall on this dial? Right in the middle, between the two families. Real daily returns are almost uncorrelated, which places real assets at $\delta \approx 1$, exactly the degenerate point: the random-walk case where, as we saw, no predictive optimum d_{opt} really exists and the loss curve is flat. In other words, real data sit at the single hardest spot on the whole map. To chart that map, for every synthetic case we also compute d^* per seed with the 20-lag setting of Section 4.3, so we can put d_{opt} and d^* side by side wherever both are defined, and finally we run the very same probe on the real cross-section. This design answers RQ2b directly, and reduces RQ2a to one simple question: where on this map do the real assets actually land?

4.8 Experimental Design

Table 2 lists every experiment in one place: what data it uses, what instrument it uses, and where its result appears. The main benchmark runs on four headline assets, picked to cover different asset classes and trend behaviours: CADUSD=X (a liquid exchange rate), TLT (a bond ETF), MSFT (a trending large-cap stock) and TRYUSD=X (an emerging-market currency that depreciates persistently). The cross-sectional analyses widen this to 24 assets spanning currencies, bonds, equity indices, sectors, commodities and individual stocks, and the volatility study uses six. The headline setup ($W = 30$, $K = 5$, 50 epochs, 32 hidden units) deliberately copies the original project's design so the comparison is fair; the synthetic experiments use longer windows and more taps ($W = 120$, $K = 50$, 40 epochs). Every hyperparameter, random seed and library version, plus a table linking each figure and table in this document to the exact script that regenerates it, is collected in Appendix B. The whole battery is reproduced by a single orchestrator that runs the gate *first* and aborts everything if the gate fails.

Table 2: Experimental design. DM = Diebold-Mariano against the random walk; probe = anchored linear probe of Section 4.7.

Experiment	Data	Instrument	Question (section)
Synthetic gate	ARFIMA, $d_{true} \in \{0, .3, .45\}$	LSTM+RNN, MASE+DM	validity (5.1)
Price benchmark	4 assets, since 2015	LSTM+RNN, 3 modes	RQ1 (5.2)
Memory/capacity sweeps	MSFT, TRYUSD=X	LSTM+RNN	RQ1 robustness (5.2)
Classical models	SI=F futures	ARIMA vs ARFIMA	RQ1 robustness (5.2)
Learned-d audit	4 assets	Learned-d, both cells	RQ2a (5.3)
Loss landscape	4 assets	Fixed-d grid retrain	RQ2a (5.3)
Cross-sectional scan	24 assets	Fixed-d grid, d_1^*, d_{20}^*	RQ2a (5.3)
Mechanistic study	FI(δ) families, 10 seeds	probe + LSTM check	RQ2b (5.3)
Volatility extension	6 assets, $\lvert r_t \rvert$ and RV5	LSTM, validation-selected d	extension (5.4)

Chapter 5 Results and Discussion

This chapter is built on one simple idea: each step must earn the reader's trust before the next one is presented. Section 5.1 begins by checking the tool on synthetic data whose memory we already know, so that every later verdict can be believed. Section 5.2 then takes the main question, RQ1, and tests it honestly on real prices, with its robustness checks. Section 5.3 works through RQ2 in three narrowing steps, the order the network learns on its own, the loss landscape when we scan every order by hand, and a controlled study that explains what both show, and then reconciles them. Section 5.4 points the same machinery at volatility, and Section 5.5 gathers everything into a single map, together with the study's limitations. Every number in this chapter comes from the public pipeline of Chapter 4 with fixed seeds; the complete tables are collected in Appendix A.

5.1 Validation on Synthetic Data

Before any verdict on real data can be trusted, the pipeline must pass a test whose answer is known in advance. Table 3 reports that test, the gate of Section 4.6: three levels of injected memory, two architectures, the full protocol, each run on a synthetic series of $N = 4{,}000$ daily observations (a further 3,000 are generated and discarded as burn-in). Two columns carry the verdict. *MASE* is the test error, so a lower value is better, and the number to beat is the random walk (RW), the naive "tomorrow equals today" forecast. The *p-value*, from the Diebold-Mariano test, says whether a gap to the RW is real or just luck: below 0.05 it is real. A model therefore *beats* the random walk only when both hold at once, a lower MASE *and* $p < 0.05$.

Read that way, the pattern is exactly the required one, **six cases out of six**. When $d_{true} = 0$ the series is a pure random walk, and no model significantly beats the benchmark in either architecture; the pipeline does not manufacture signal where none exists. When $d_{true} \in \{0.30, 0.45\}$ the pipeline beats the random walk with $p < 0.001$ in all four cases, cutting MASE by up to 28% at $d_{true} = 0.45$; when memory is present, the pipeline finds it and turns it into significant forecasting gains. The estimated memory order behaves correctly too, rising steadily with the injected memory ($d^* = 0.143, 0.487, 0.706$ for $d_{true} = 0, 0.30, 0.45$; Figure 10).

It is worth pausing on why d^* comes out *larger* than d_{true}, because the two numbers describe different series. The long memory is injected into the *returns*, so d_{true} is a property of the returns. But d^* is read off the *price*, and it measures something else: how much fractional differencing the price needs before it becomes stationary. The price is the running sum of the returns, so it behaves like a random walk with the returns' memory riding on top of it. To reach stationarity d^* has to undo both the random-walk part *and* that extra

memory, which is why it always sits above d_{true}. And that gap is not a problem, because all the gate has to check is that the instrument *reacts*: when we inject more memory, d^* should rise. That is exactly what happens ($0.143 \to 0.487 \to 0.706$ as d_{true} goes $0 \to 0.30 \to 0.45$). We do not need d^* to equal d_{true}; we only need the two to move together, in the same order. That is what shows the tool genuinely notices the memory we put in.

Table 3: The synthetic gate (test MASE; p = Diebold-Mariano p-value against the random walk). "Beats RW" requires a negative DM statistic with $p < 0.05$ for at least one fractional model. Expected: NO for $d_{true} = 0$, YES otherwise. Outcome: 6/6.

d_{true}	Cell	d^*	RW	Baseline	Fixed-d^* (p)	Learned-d (p)	Beats RW?
0.00	LSTM	0.143	1.451	1.456	1.450 (.881)	1.455 (.196)	NO (expected NO)
0.00	RNN	0.143	1.451	1.461	1.448 (.666)	1.461 (.002)†	NO (expected NO)
0.30	LSTM	0.487	4.451	4.009	4.818 (.006)†	4.012 (.000)	**YES** (expected YES)
0.30	RNN	0.487	4.451	4.033	4.443 (.247)	4.032 (.000)	**YES** (expected YES)
0.45	LSTM	0.706	3.302	2.373	2.943 (.012)	2.377 (.000)	**YES** (expected YES)
0.45	RNN	0.706	3.302	2.404	2.898 (.000)	2.403 (.000)	**YES** (expected YES)

† significant in the *wrong* direction: the model is significantly worse than the RW.

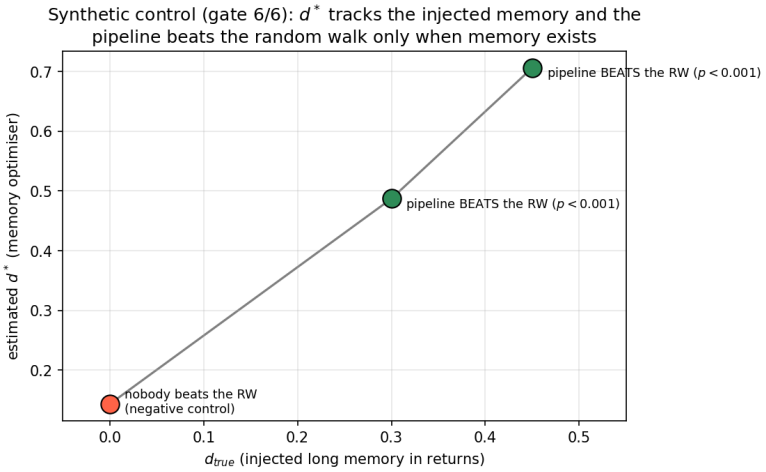


Figure 10: Summary of the gate: the estimated d^* tracks the injected memory, and the pipeline beats the random walk only when memory exists.

One detail of Table 3 deserves attention, because it foreshadows RQ2. The two fractional models treat the differencing order differently. *Fixed-d^** keeps it frozen at d^*, the order that retains the most memory. *Learned-d* is free to move it during training, and it always drifts up to the cap $d = 1$, which is simply the ordinary return; that is why its error matches the returns baseline almost exactly (for instance 4.012 against 4.009 at $d_{true} = 0.30$). And it is these returns-based models, the baseline and the learned-d one, that beat the random walk,

while Fixed-d^* does not: at $d_{true} = 0.30$ it is in fact significantly *worse* (the † in the table marks exactly this, a gap that is real but in the wrong direction). The reason is the very question RQ2 will pursue: even on series built to contain long memory, the order that keeps the most memory (d^*) is not the order that forecasts best (here, the plain return, $d = 1$). So the gate validates the *pipeline*, not the recipe.

For completeness, an earlier design of this project *failed* this same control. That version forecast the normalised price level directly instead of the return, and its scaler, fitted on training data only, forced the network to extrapolate beyond its training range on trending series, inflating MASE by two orders of magnitude. The redesign of Chapter 4 (return target, replicate padding, one-step reconstruction) followed from that diagnosis, and the corrected pipeline passes 6/6. The episode is worth mentioning because it is the practical argument for gates: without the synthetic control, that defective pipeline would have produced a confident, and wrong, negative answer to RQ1.

5.2 RQ1: Forecasting Daily Prices

With the instrument validated, we can now turn it on the real question. Table 4 lists the results for the four headline assets, and Figure 11 draws the same numbers as MASE relative to the random walk, alongside the robustness sweeps. The reading rule is the same as before: to beat the random walk a model needs a lower MASE *and* a p-value below 0.05. The verdict is blunt. **In 0 of 8 cases** (four assets × two architectures) does any model beat it, neither the fractional models nor the returns baseline. Every MASE ratio sits around 1.00 (from 0.97 to 1.18), so the models all forecast essentially like the random walk. And the few gaps that are statistically real point the *wrong* way: the model loses, not wins. On MSFT the learned-d model is significantly *worse* ($p = 0.033$ for the LSTM, $p = 0.001$ for the RNN), and Fixed-d^* is significantly worse on TRYUSD=X with the RNN ($p = 0.022$) and never better anywhere.

Table 4: RQ1 benchmark: test MASE per asset, architecture and model ($W = 30$, $K = 5$, 50 epochs). Bold marks the best fractional model per row; no model attains DM < 0 with $p < 0.05$ against the RW in any row. Full DM statistics in Appendix A.

Asset	Cell	d^*	RW	Baseline	Fixed-d^*	Learned-d
CADUSD=X	LSTM	0.063	0.654	0.653	0.677	**0.653**
CADUSD=X	RNN	0.063	0.654	0.653	0.661	**0.653**
TLT	LSTM	0.169	0.693	0.695	**0.693**	**0.693**
TLT	RNN	0.169	0.693	0.700	**0.692**	0.699
MSFT	LSTM	0.393	2.341	2.396	**2.339**	2.378
MSFT	RNN	0.393	2.341	2.412	**2.339**	2.411
TRYUSD=X	LSTM	0.446	0.033	**0.032**	0.039	0.033
TRYUSD=X	RNN	0.446	0.033	**0.034**	0.042	0.036
Verdict: 0/8 cases beat the random walk (DM < 0 and $p < 0.05$)						

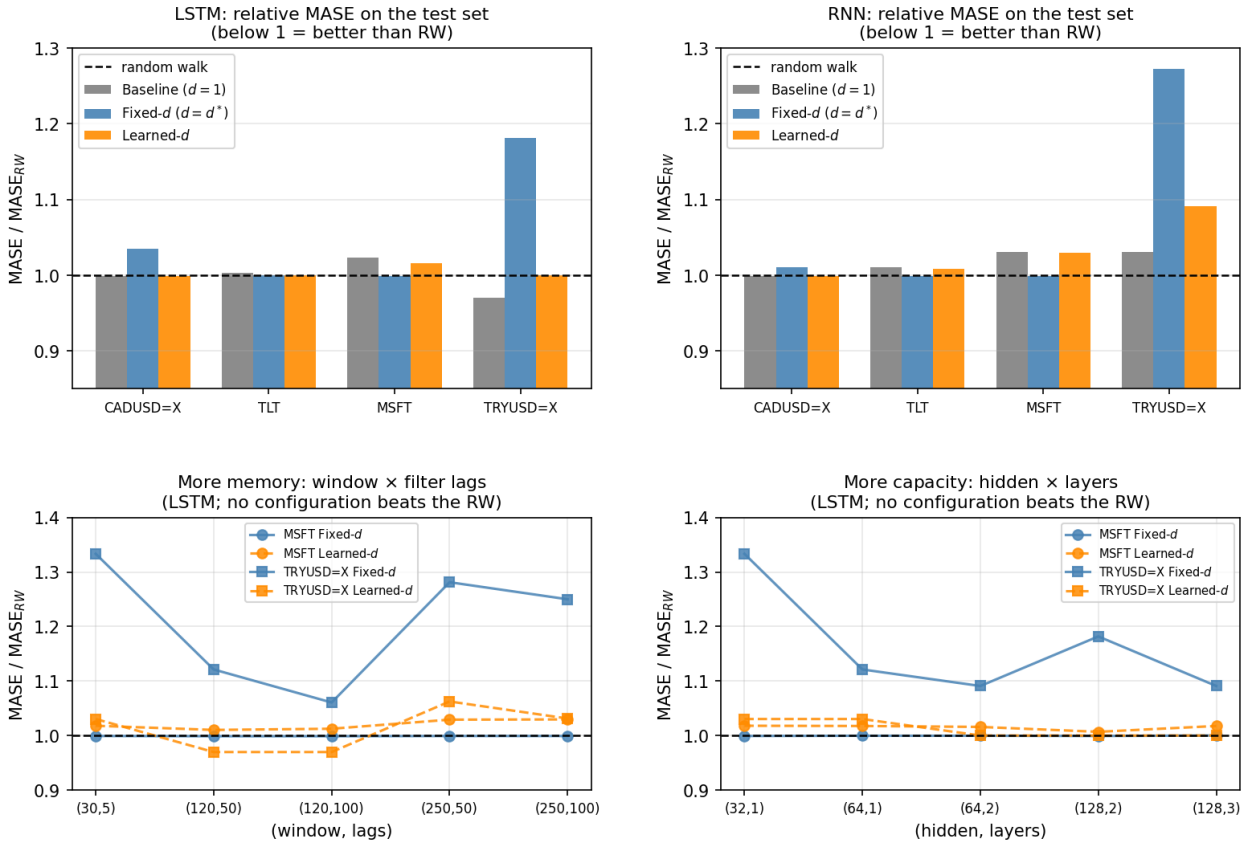


Figure 11: RQ1 in one view. Top: MASE relative to the random walk for both architectures; all bars cluster at 1.0. Bottom: robustness sweeps (LSTM): enlarging the memory available to the model ($W \times K$ up to 250×100) or its capacity (up to 128 units $\times$ 3 layers) leaves the picture unchanged.

Robustness. A fair worry is that this negative comes from a badly chosen setup. So three obvious objections are checked one by one, and each gives the same answer.

- *Did the model simply have too little memory to work with?* No. Raising the window and filter length from $(30, 5)$ to $(120, 50)$, $(120, 100)$, $(250, 50)$ and $(250, 100)$ changes nothing: on MSFT the best fractional model stays within 1.000 ± 0.002 of the random walk, the learned order still saturates at or near 1, and nothing turns significant in the right direction.

- *Was the network too small?* No. Growing it from 32×1 to 128×3 again leaves every configuration at or above the benchmark.

- *Is a neural model the wrong tool altogether?* No. The classical counterparts on the SI=F futures series give the same answer: ARIMA$(1, 1, 0)$ reaches MASE 4.353 ($p = 0.332$, directional accuracy 48.6%) and ARFIMA$(1, 0, 1)$ reaches 4.361 ($p = 0.754$, 52.0%), against 4.334 for the random walk. Neither beats it, and the directional accuracy is no better than a coin flip.

Answer to RQ1. At a daily horizon, under an honest benchmark, fractional differencing does not improve one-step price forecasts, not with more memory, not with more capacity, not with a different cell, and not with a classical estimator. And this negative is *informative*, not a dead end: since the very same pipeline clearly exploits memory when it is there (Section 5.1), what is missing must be in the data, not in the method. This matches the literature of Section 2.5, where daily returns carry essentially no exploitable linear memory [11], [12] and the random walk stays the benchmark to beat [27], and it fits the fragility of the positive fracdiff-plus-ML results in Section 2.9, none of which ever cleared a random-walk test.

5.3 RQ2: The Predictive Optimum versus the Memory Optimum

RQ1 says fractional differencing does not help on prices; RQ2 asks the deeper question of whether the predictive optimum even exists there, and how it relates to the memory optimum d^*. The comparison proceeds as a funnel that tightens at each step. First the *literal* instrument: a single d learned by the network itself. Then the *robust* instrument: an exhaustive scan over d that removes the optimiser from the picture. Then a mechanistic study on data with known memory, which explains what the first two observe. A final subsection reconciles everything into the answers to RQ2a and RQ2b.

5.3.1 The end-to-end learned order is degenerate

The idea here is simple. If the loss really knew where the best differencing order was, then the d that the network learns on its own (Section 4.4) should drift towards it. It does not; it does instead exactly what the invariance argument of Section 4.7 warned it would do, and goes nowhere useful. Look at where it lands: across the eight benchmark runs the learned order ends pinned at its ceiling, $d = 1.000$, in seven of them, and at 0.595 in the eighth (TLT, LSTM). That is a tiny band, $[0.595, 1.000]$, and it has nothing to do with the assets' true memory optima, which run from 0.063 to 0.446 (Figure 12). In other words, the network parks d at the top no matter what the data look like.

This is not because the maths is broken: the optimisation audit confirms the gradient is computed correctly. The gradient is simply too weak and too shapeless to pull d towards any data-driven value, so the final d is not chosen by the data at all. It is chosen by the shape of the loss surface, which lets d slide up into the ceiling, and, when the learning rate is low, by wherever d happened to start; an apparent "learned $d \approx 0.9$" from an early version of this project turned out to be exactly that, an artefact of the starting value rather than a real estimate. So the literal version of RQ2a has a sharp answer: **the order learned by backpropagation does not estimate anything**. It cannot match d^*, because it is not measuring any property of the data in the first place. What is left open is *why*, whether it is the optimiser that is too weak or there is simply no information in the loss to begin with, and that is the question the next two instruments settle.

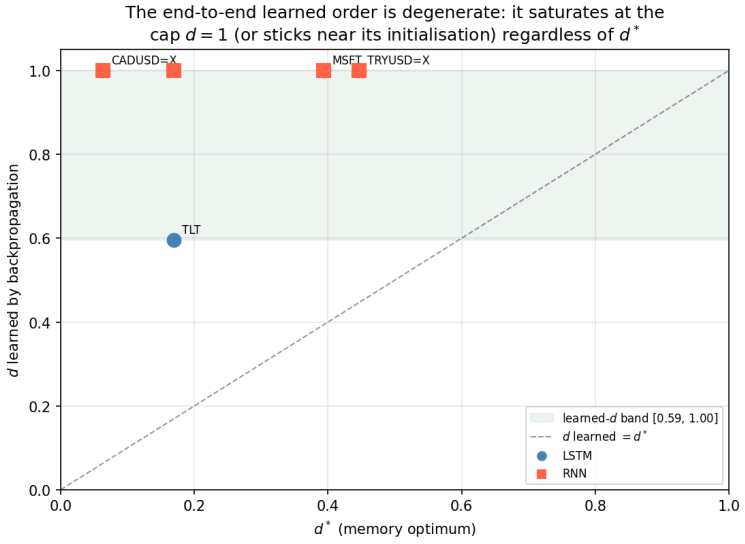


Figure 12: The learned differencing order versus the memory optimum on the four benchmark assets and both cells. The learned order saturates at the cap independently of d^*.

5.3.2 Flat loss landscapes on real prices; no d_{opt}–d^* relation

The previous test left one doubt open: maybe the optimiser was just too weak, and a good d is out there waiting to be found. The robust instrument settles it by taking the optimiser out of the picture entirely. Instead of letting the network choose d, we fix d to each value on a grid, retrain the model from scratch at every one of them, and plot the resulting test error against d. If a predictive optimum existed, this curve would dip at it. Figure 13 shows what actually happens on the four headline assets: the curves are *flat, and flat at the level of the random walk*. Changing d barely changes the error at all. For the three liquid assets the entire curve moves by only 0.7% to 3.8% in relative MASE, no point drops meaningfully below 1.0, and the lowest point lands at $d = 1.0$, 0.9 and 0.1 respectively, never at the memory optimum d^* (which spans 0.06 to 0.45) and never at any consistent place. (TRYUSD=X looks more jagged only because its tiny price scale magnifies rounding; it is an artefact, not a signal.)

A second version of the scan confirms the diagnosis. When we run it on synthetic series that genuinely have memory injected *in the returns* (the same regime as the gate), the best predictive d pins at $d \approx 1.0$ at every memory level, while d^* climbs steadily from 0.14 to 0.71. So even where memory is real, the two move independently: the best-forecasting d and the maximum-memory d^* are simply answering different questions.

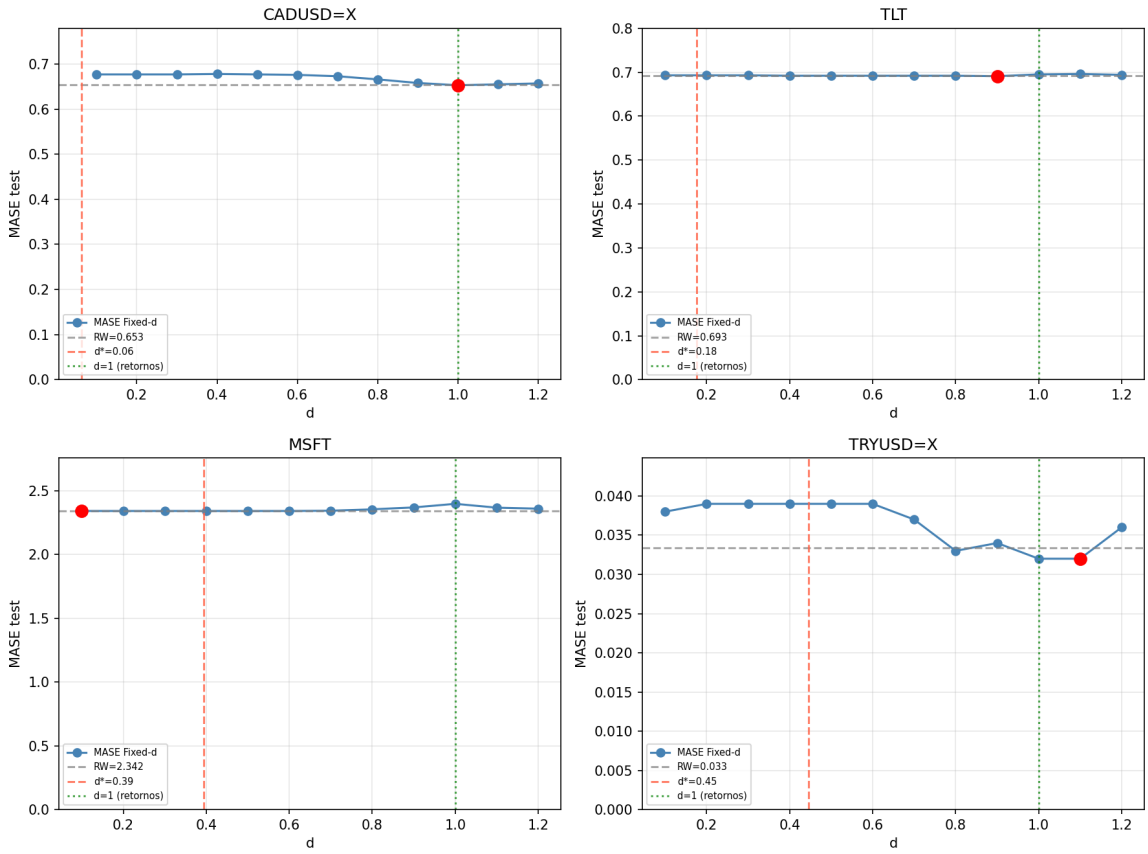


Figure 13: Loss landscape over d on real assets (Fixed-d retrained at each grid point; MASE relative to the RW). The curves are flat: every representation forecasts like the random walk, and the argmin (red) is unrelated to d^* (dashed).

If the curve is flat for each asset on its own, the natural next question is cross-sectional: *across* assets, does d_{opt} correlate with d^*? The flatness already implies the answer. Over 24 assets spanning four classes, the correlation between the per-asset predictive argmin and the memory optimum is statistically indistinguishable from zero: Pearson $+0.16$ ($p = 0.45$) with the 1-lag d^*, and $+0.11$ ($p = 0.62$) with the calibrated 20-lag d^* (Spearman $+0.15$ and $+0.10$; Figure 14). A correlation links two numbers, d_{opt} and d^*, so it can fail for two different reasons: a bad d^*, or a bad d_{opt}. One might blame d^*, since the estimator of Section 4.3 is known to read a bit low. But measuring d^* better cannot fix anything, because the broken number is d_{opt}. Recall how d_{opt} is obtained: we sweep d, draw the error curve, and pick the d at its lowest point. That only means something if the curve actually has a bottom. Here it does not, it is flat, so every d gives almost the same error and the "winner" is decided by a tiny random wiggle, different from one run to the next. The lowest point is therefore not telling us the best d, it is telling us where the noise dipped. A d_{opt} chosen that way is essentially a random number, and a random number cannot line up with d^* no matter how

precisely we measure d^*.

This is exactly why we cannot stop at RQ2a and conclude that the two optima are "decoupled". A near-zero correlation here proves nothing, because one of the two quantities does not really exist on real prices. To test whether d_{opt} and d^* are genuinely unrelated, we first need a setting where the predictive optimum *does* exist, with a curve that has a real, non-accidental minimum. Building that setting is the job of RQ2b.

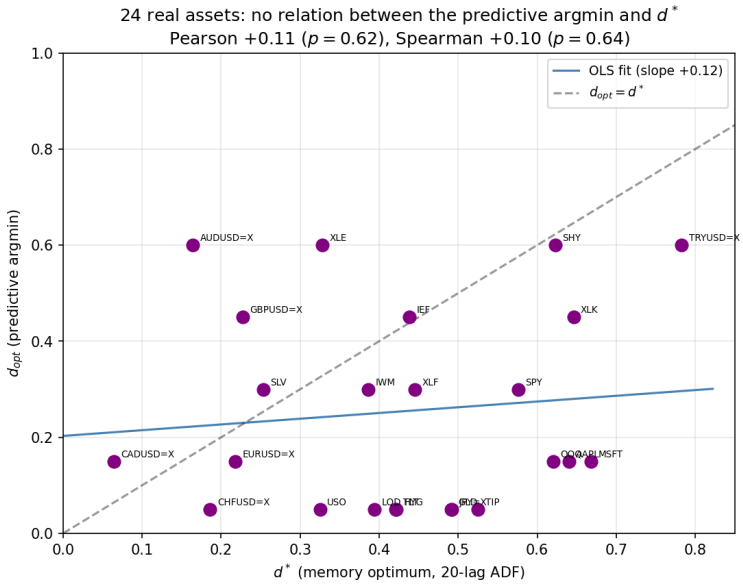


Figure 14: Cross-section of 24 real assets: predictive argmin versus the calibrated (20-lag) memory optimum. No significant relation, as expected when the underlying loss curves are flat.

5.3.3 The mechanistic study: co-movement where memory exists

So here we build that setting on purpose, with two ingredients. The first is the model: the anchored one-tap probe of Section 4.7, a deliberately simple linear model with a single memory term. Because it is so rigid, it cannot quietly undo the choice of d the way a flexible network can, so the error curve now has a real bottom and d_{opt} genuinely exists. The second is the data: instead of real prices, we use synthetic FI(δ) series ($N = 4{,}000$ daily observations, ten seeds per case) whose true order we set ourselves, which gives us a known answer to check d_{opt} and d^* against. With a real optimum and a ground truth in hand, we can finally ask whether the two move together. Table 5 and Figure 15 report the answer, the central result of this thesis.

Table 5: Mechanistic study (anchored linear probe, 10 seeds per case; d^* estimated per seed with the 20-lag ADF; mean ± s.d.). A case is *identified* if its curve depth exceeds the 0.5% threshold calibrated on the $\delta = 1$ null (measured noise floor 0.07%). $d^*_{th} = \delta - 0.5$ is the theoretical stationarity boundary.

Case (δ)	d^*_{th}	d^* (20 lags)	d_{opt}	depth (%)	min relMASE	identified
B $\delta = 0.6$	0.10	0.040 ± 0.003	0.35 ± 0.17	4.1	0.920	yes
B $\delta = 0.7$	0.20	0.074 ± 0.037	0.40 ± 0.18	3.3	0.950	yes
B $\delta = 0.8$	0.30	0.175 ± 0.083	0.45 ± 0.19	2.0	0.976	yes
B $\delta = 0.9$	0.40	0.287 ± 0.109	0.60 ± 0.25	0.6	0.995	yes
B $\delta = 1.0$ (RW)	0.50	0.354 ± 0.127	undefined (0.00 ± 0.38)	0.1	1.003	**no**
A $\delta = 1.25$	0.75	0.629 ± 0.140	0.70 ± 0.11	5.3	0.931	yes
A $\delta = 1.45$	0.95	0.845 ± 0.102	0.80 ± 0.05	28.4	0.667	yes

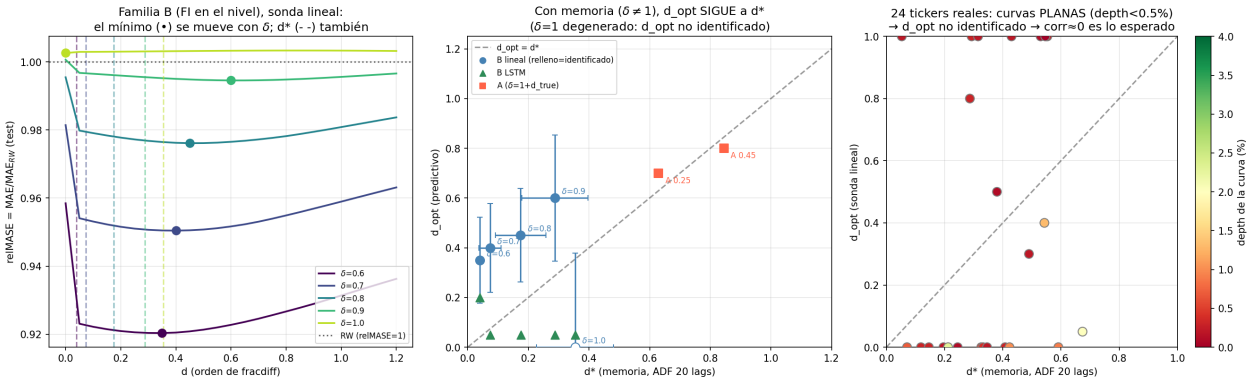


Figure 15: The mechanistic study. Left: probe loss curves on family B; the minimum moves right as δ grows, and so does d^*; at $\delta = 1$ the curve is flat at the noise floor. Centre: d_{opt} versus d^* across all identified synthetic cases (blue: level memory; red: return memory); the two optima co-move tightly. Green triangles: the flexible LSTM on the same data does *not* localise a tracking optimum, as the invariance argument predicts. Right: the same probe on 24 real assets; flat curves, the degenerate $\delta \approx 1$ regime.

Before the three facts, a one-line reminder of the two quantities being compared: d_{opt} is the differencing order that forecasts *best* (the bottom of the error curve), and d^* is the order that just makes the price *stationary* (the memory recipe). The question of RQ2 is whether these two are related.

Three facts stand out. **First, whenever the price really has memory ($\delta \neq 1$), a best d truly exists.** The error curves now have a clear bottom, not a flat line: their depth runs from 0.6% to 28.4%, all above the threshold, and at that bottom the probe actually beats the random walk (lowest relMASE 0.92, 0.95, 0.98 at $\delta = 0.6, 0.7, 0.8$, and 0.93, 0.67 in family A). **And this best d rises and falls together with the memory order d^*.**

Put the six cases in order: the more memory we inject, the higher *both* d_{opt} and d^* become. The correlation between the average d_{opt} and the average d^* is Pearson $+0.974$ ($p = 0.001$), and Spearman is $+1.000$, which means they rank the six cases in exactly the same order. Even comparing all sixty individual runs the link stays strong, $+0.68$ and $+0.73$ ($p < 10^{-4}$), and the fitted line $d_{opt} = 0.37 + 0.54\, d^*$ describes it. They *move together, but they are not equal*: the best d for forecasting lands a little above the order that just reaches stationarity. That gap is expected, because the two numbers are just two different ways of reading the *same* hidden amount of memory, one by minimising forecasting error, the other by passing a stationarity test. **Second, when there is no memory, the method correctly finds nothing.** At $\delta = 1$, a pure random walk, the error curve is flat at the noise floor (0.1%), the probe cannot beat the benchmark (relMASE 1.003), and its lowest point jumps around at random from run to run (0.00 ± 0.38). There is no best d to find when there is nothing to predict. **Third, the flexible model fails exactly as the theory said it would.** Repeat the whole battery with the LSTM in place of the simple probe and the tracking vanishes: the LSTM's best d sticks to the smallest grid value for almost every δ, and its correlation with d^* is -0.61 ($p = 0.28$). For $\delta \geq 0.9$ it cannot even beat the random walk. The lesson is that pinning down d is a property of the *model* you use, not of the data: a rigid model reveals the optimum, a flexible one hides it. That is precisely why the fully flexible learned-d network of Section 5.3.1 never had a chance of measuring it.

Finally, we run the very same probe on the 24 real assets, and this closes the loop (Figure 15, right): their curves are flat, just like the no-memory case above. Six assets do pass the depth threshold on paper, but each of them still has a lowest relMASE between 0.98 and 1.01, meaning no real forecasting gain, and several of their best points sit right at the edges of the grid, the tell-tale sign of a noisy argmin rather than a true optimum. The correlation with d^* stays non-significant ($+0.18$, $p = 0.40$). The pattern is therefore clean and one-directional: *no real asset shows both a genuine best d and real predictability, while every synthetic case that has memory ($\delta \neq 1$) shows both.* In the language of this map, real daily prices simply sit at its dead point, $\delta \approx 1$, the same spot as a pure random walk, so there is no optimum there to line up with d^* in the first place.

5.3.4 Reconciliation: answers to RQ2

All the pieces of the funnel now fit into a single story, which we can state as the two answers RQ2 was asking for.

RQ2a: do the two optima match on real prices? The honest answer is that, on real prices, the question *cannot be answered*, because one of the two quantities is missing. The forecasting error barely changes with d (the curve is flat), exactly as the invariance argument predicted and as the learned order, the landscapes and the probe all confirm. With a flat curve there is no real "best d" to speak of, so the near-zero correlation with d^* tells us nothing:

it does *not* mean the two optima are unrelated, it only means there is no optimum to compare in the first place. So when other studies report a precise "optimal d" for forecasting daily prices, what they are really reporting, on this evidence, is noise.

RQ2b: when the optimum *does* exist, are the two related? Yes, and strongly. Whenever the price genuinely has fractional memory ($\delta \neq 1$), the best forecasting d exists and rises in lockstep with the memory order d^* (Spearman +1.00), and this holds across very different worlds, from anti-persistent to persistent, because both numbers are reading the same underlying memory. The one caveat, repeated from above: they track each other but are not identical: there is an offset and a gentler slope, so the data do *not* support the literal equality $d_{opt} = d^*$.

Read together, this gives a balanced verdict on the López de Prado recipe, neither a rescue nor a dismissal. The idea behind the recipe, that the stationarity boundary is a predictively useful quantity, is *right* when the price level actually has memory. It fails on daily price forecasting not because the idea is wrong, but because daily prices are the worst possible case for it ($\delta \approx 1$, returns almost pure noise), where the very thing the recipe tunes leaves no trace in the forecasting error. The same now explains the end-to-end learnable d (this work's objective O2): it cannot work on real prices, and the reason is identification (there is nothing to detect), not a faulty optimiser. The conditions under which it *would* work, real memory in the level plus a constrained model, are exactly the ones the probe has just mapped out.

5.4 Extension: Forecasting Volatility

So far the verdict on prices has been negative, and we argued the reason is *where* the memory lives, not the method. Section 2.5 made that point: the robust long memory of financial series is in their volatility (how big the moves are), not in their direction (whether they go up or down). This gives a clean test. If RQ1 came out negative only because prices have no usable memory, then aiming the *exact same* machinery at a volatility target, where memory does live, should now succeed. We try it on six assets, each with two ways of measuring volatility: the size of the daily move, $|r_t|$, and a smoother 5-day realised-volatility measure (RV5). Each is forecast one step ahead with the same architecture, compared against the naive "tomorrow equals today" benchmark $\hat{v}_{t+1} = v_t$, under the identical MASE-plus-DM protocol. The differencing order d is picked on the internal validation split (Section 4.5), and the test set is used only once.

The contrast with RQ1 could hardly be sharper (Table 6, Figure 16). Using the size-of-move proxy $|r_t|$, the model now beats the naive benchmark in **five of six assets, all with** $p < 0.001$, cutting the forecasting error by 24% to 28% (this includes a 24% cut on GLD; there both MASE values happen to be above 1 only because the volatility level shifted between the training and test periods, but the model is still clearly better than the naive). The single

Table 6: Volatility extension: d selected on validation, single test evaluation. ACF_{20} is the lag-20 autocorrelation of the volatility proxy in the training segment. Beats requires DM < 0, $p < 0.05$.

Asset	Proxy	ACF_{20}	d_{sel}	Model MASE	Naive MASE	DM	p	Beats?
SPY	$\|r\|$	0.205	0.7	0.734	0.962	−4.12	.000	**yes**
SPY	RV5	0.355	0.0	1.062	0.951	−0.71	.476	no
QQQ	$\|r\|$	0.168	0.5	0.754	1.006	−5.31	.000	**yes**
QQQ	RV5	0.366	0.1	1.057	0.990	−1.04	.297	no
MSFT	$\|r\|$	0.101	0.7	0.726	1.000	−5.34	.000	**yes**
MSFT	RV5	0.242	0.7	1.180	0.965	−0.74	.462	no
TLT	$\|r\|$	0.132	0.5	0.629	0.868	−8.39	.000	**yes**
TLT	RV5	0.269	0.3	0.854	0.887	−4.67	.000	**yes**
GLD	$\|r\|$	0.049	0.3	1.136	1.496	−4.76	.000	**yes**
GLD	RV5	0.194	0.1	1.459	1.404	−1.64	.101	no
TRYUSD=X	$\|r\|$	0.149	0.5	0.199	0.255	−1.72	.086	no
TRYUSD=X	RV5	0.305	0.3	0.250	0.233	−0.88	.380	no

exception, TRYUSD=X, just misses significance ($p = 0.086$). On the smoother RV5 proxy only TLT wins (1/6), and that is expected rather than a failure. The 5-day averaging makes the series so smooth, with consecutive days correlating above 0.9, that the naive benchmark itself becomes near-perfect: when "tomorrow equals today" is already almost exact, there is hardly any error left for *any* model to remove. The weak score on RV5 therefore reflects a benchmark that is very hard to beat, not a failure of fractional differencing. The headline is the jump from RQ1: the same pipeline, the same metric and the same test that scored 0/8 on prices now score 5/6 on volatility. *The memory you can actually exploit in financial series is in how big the moves are, not in their direction.* This is the classic regularity of Ding, Granger and Engle [16], here reproduced out-of-sample with a genuine forecasting gain. Two honest caveats keep the claim in proportion. First, the benchmark is only naive persistence, not a tuned volatility model, so a fair comparison against HAR or GARCH is left for future work. Second, the chosen orders $d_{sel} \in [0.3, 0.7]$ should be read as "some differencing helps", not as exact values, because the error-versus-d curve, although no longer flat as it was on prices, is still fairly shallow.

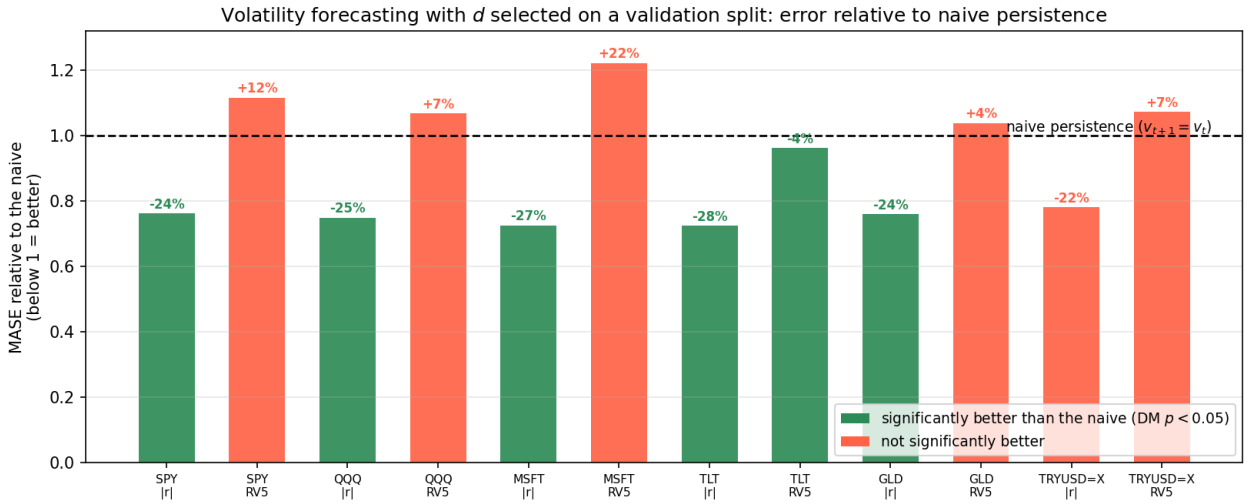


Figure 16: Volatility forecasting under the honest protocol: test MASE of the model *relative* to the naive persistence benchmark, per asset and proxy (below the dashed line = lower error than the naive; green = significant at 5%). The two proxies are the absolute daily log-return $|r_t| = |\log p_t - \log p_{t-1}|$ and the 5-day realised volatility $\mathrm{RV5}_t = \sqrt{\frac{1}{5}\sum_{i=0}^{4} r_{t-i}^2}$. The relative scale removes train/test volatility-regime shifts that make absolute MASEs incomparable across assets.

5.5 Synthesis and Limitations

The Introduction promised to characterise *when fractional differencing helps and how the predictive and memory optima relate*. The results compose that characterisation, summarised in Table 7.

Table 7: Synthesis: when does fractional differencing help, and when is its order even measurable?

Regime	Signal?	d_{opt} identified?	Beats benchmark?	Relation d_{opt} vs d^*
Real daily prices ($\delta \approx 1$)	no	no (flat loss)	no (0/8)	not identified; corr ≈ 0 is the predicted outcome
Synthetic FI level, $\delta \neq 1$	yes	yes (probe)	yes	tight co-movement (Spearman +1.00), with offset
Volatility ($\lvert r_t \rvert$)	yes	shallow but usable	yes (5/6, $p < .001$)	$d_{sel} \in [0.3, 0.7]$: "some memory helps"

Read top to bottom, the three rows tell one simple story. Fractional differencing is a

good way to represent long memory, and the memory-optimal order d^* is a genuinely useful quantity, *but only where there is memory to represent in the first place*. Daily prices of liquid assets are not such a place: in practice they look just like the degenerate $\delta = 1$ random walk, and there the differencing order simply cannot help, because changing it leaves no trace in the forecasting error. Where memory really is present, the synthetic FI worlds and, above all, volatility, the very same machinery delivers significant gains. So none of this contradicts the efficient-market view of Section 2.5; it just turns it into a practical map of *where* the tool works and where it cannot.

Limitations. The conclusions hold within the following bounds.

- **One split per asset.** Each asset is evaluated on a single chronological 80/20 split. A rolling-origin design would test the verdicts over more periods, but was too costly to run; the breadth of the panel (24 assets, four classes) partly makes up for it.
- **One-day, point forecast.** The horizon is a single day and the target is a point forecast. Nothing here speaks to longer horizons, where the slow decay of fractional memory might matter more, nor to distributional or trading-based objectives.
- **Simple volatility benchmark.** The volatility comparison is against naive persistence under a deliberately symmetric protocol; testing against HAR- or GARCH-type models is left for future work.
- **ADF lag choice.** The number of ADF lags is a free choice of the researcher. We fixed it in advance by calibrating on known ground truth (Section 4.3) and report both variants, but other stationarity tests (KPSS-anchored, spectral) were not explored.
- **Live data.** Prices are downloaded live, so re-running shifts the last decimals. All the verdicts survive a refresh, and every random step is seeded.
- **Pure synthetic processes.** The synthetic families are clean FI processes. Adding short-memory ARMA structure or structural breaks [13] would stress-test the probe further.
- **Short filter in the headline runs.** The headline runs cut the fractional filter at $K = 5$, keeping only 73–86% of the filter weight for the orders that matter on real assets (Section 4.2). They are best read as a short-memory setting; true long memory is covered by the separate $K = 100$ robustness runs, not by the headline itself.
- **Imperfect scaling in learned-d.** In learned-d mode the input scaling is frozen at the starting order, so that gradient is not perfectly size-corrected. The fixed-d grid (scaled correctly at every order) shows this is *not* what flattens the landscape, but a version that updates the scaling as d moves was not implemented.

Within these bounds, the claims of this chapter are exactly the ones the evidence supports: no more, and no less.

Chapter 6 Conclusions and Future Work

This thesis asked a deliberately simple pair of questions about a popular idea. Fractional differencing promises the best of two worlds: it removes the trend from a price series, like an ordinary difference, while keeping more of its past, its *memory*, than an ordinary difference would. The hope is that feeding a neural network this "memory-preserving" version of a price should help it forecast. **RQ1** asked whether that hope holds up under an honest test. **RQ2** asked a deeper question: if there is a differencing order that forecasts best, is it the same order that preserves the most memory? The work answered both, and the answers only make sense together.

6.1 Answers to the research questions

RQ1: fractional differencing does not improve daily price forecasting. Across four headline assets and two architectures, no fractional model, and not even the plain-returns baseline, beat the random walk: **0 of 8** cases (Section 5.2). The result is robust to more memory, more network capacity and a switch to classical ARIMA/ARFIMA estimators, and the few statistically real differences point the *wrong* way. This is not a bug in the pipeline, because the very same pipeline passes a synthetic positive control in all 6/6 cases (Section 5.1): it detects memory we deliberately inject and finds nothing when there is none. The honest reading is that the missing ingredient is in the *data*, not in the method: daily returns of liquid assets carry essentially no linear memory to exploit.

RQ2a: on real prices, the two optima cannot even be compared. The forecasting error barely changes with the differencing order, so the curve is flat and has no real bottom (Section 5.3). When there is no genuine "best order" to measure, the near-zero cross-sectional correlation with d^* ($+0.18$, $p = 0.40$) is not evidence that the two are unrelated; it is exactly what a non-identified quantity must produce. A reported "optimal d" for daily price forecasting is, on this evidence, noise.

RQ2b: where the optimum exists, the two move together tightly. On synthetic series that genuinely carry memory ($\delta \neq 1$), the predictive optimum does exist, and it rises in lockstep with the memory optimum (Spearman $+1.00$; fitted relation $d_{opt} = 0.37 + 0.54\, d^*$). The two are *not equal*, they track each other with an offset, because they are two readings of the same underlying integration order: one through forecasting error, the other through a stationarity test. Crucially, this tracking only appears with a constrained model; the fully flexible learned-d network cannot recover it, which shows that identifying d is a property of the *instrument*, not just of the data.

The positive case: memory lives in volatility, not in direction. Pointed at a

volatility target, the identical machinery beats the naive benchmark in **5 of 6** assets with $p < 0.001$, cutting error by 24–28% (Section 5.4). The same pipeline that scored 0/8 on prices scores 5/6 on volatility. This is the decisive confirmation that the negative of RQ1 is about *where* the memory is, and reproduces the classical Ding–Granger–Engle regularity out-of-sample with a real forecasting gain.

6.2 Achievement of the objectives

- **O1 (a differentiable d^* estimator) – achieved.** The estimator is anchored to the ADF test rather than to arbitrary weights, and on synthetic series of known order it recovers the theoretical stationarity boundary once the ADF lag count is calibrated (Section 4.3).
- **O2 (a trainable differencing order) – achieved, with a sharp finding.** The order can indeed be learned end-to-end, but the analysis shows *why* it is uninformative on real prices: the loss is flat in d, so the learned order is decided by initialisation and the loss geometry, not by the data (Section 5.3). This is a characterisation, not a failure.
- **O3 (a synthetic positive control) – achieved.** The control passes 6/6 and, in doing so, caught a real defect in an earlier version of the pipeline. It is what lets us trust the negative answer to RQ1 as a fact about markets rather than an artefact of code.
- **Main objective – achieved.** RQ1 and RQ2 are both answered with statistically solid, honestly benchmarked evidence, independently of whether the answer was the hoped-for "yes".

6.3 Main contributions

The work contributes, beyond the individual answers: (i) an *honest evaluation protocol* for this problem, built around a random-walk benchmark, the Diebold–Mariano test and a synthetic gate, which by itself overturns the optimistic picture painted by parts of the literature; (ii) an *identifiability lens* that reframes a confusing negative ("the optima do not match") into a precise statement ("the optimum is not identified on a near-random-walk"); and (iii) an *operational map* (Table 7) of when fractional differencing helps and when its order is even measurable. Together they sharpen, rather than contradict, the efficient-market baseline of Section 2.

6.4 Future work

The limitations of Section 5.5 point directly to the natural next steps. The most promising is to follow the memory: since volatility is where the exploitable structure lives, the same fractional machinery should be benchmarked there against tuned HAR- and GARCH-class models, not only against naive persistence. Beyond that, four directions stand out: evaluating with a rolling-origin scheme instead of a single split; extending to longer horizons and to distributional or trading-based objectives, where slowly decaying memory may matter more; stress-testing the d^* estimator and the probe on contaminated processes (short-memory ARMA structure, structural breaks) and with alternative stationarity instruments (KPSS-anchored, spectral); and implementing a learned-d variant whose input scaling updates as d moves, to close the last methodological gap. More broadly, the identifiability framework, "a preprocessing choice can only be learned where it leaves a trace in the loss", is not specific to finance and could guide the use of fractional differencing in any long-memory domain, such as climate, network traffic or physiological signals, where the same flat-loss trap can appear.

6.5 Closing remark

The headline result is negative, but it is an *informative* negative. Fractional differencing is a sound tool, and the order that maximises memory is a meaningful quantity, but only where memory exists for it to act on. Daily liquid prices are simply not such a place; volatility is. The contribution of this thesis is to have drawn that line carefully, with a protocol strict enough that the line can be trusted.

Chapter 7 References

[1] C. W. J. Granger and R. Joyeux, "An introduction to long-memory time series models and fractional differencing", *Journal of Time Series Analysis*, vol. 1, no. 1, pp. 15-29, 1980. DOI: 10.1111/j.1467-9892.1980.tb00297.x

[2] J. R. M. Hosking, "Fractional differencing", *Biometrika*, vol. 68, no. 1, pp. 165-176, 1981. DOI: 10.1093/biomet/68.1.165

[3] M. López de Prado, *Advances in Financial Machine Learning*. Hoboken, NJ: John Wiley & Sons, 2018, ISBN: 978-1-119-48208-6.

[4] H. E. Hurst, "Long-term storage capacity of reservoirs", *Transactions of the American Society of Civil Engineers*, vol. 116, pp. 770-808, 1951.

[5] B. B. Mandelbrot and J. W. Van Ness, "Fractional brownian motions, fractional noises and applications", *SIAM Review*, vol. 10, no. 4, pp. 422-437, 1968. DOI: 10.1137/1010093

[6] J. Geweke and S. Porter-Hudak, "The estimation and application of long memory time series models", *Journal of Time Series Analysis*, vol. 4, no. 4, pp. 221-238, 1983. DOI: 10.1111/j.1467-9892.1983.tb00371.x

[7] R. T. Baillie, "Long memory processes and fractional integration in econometrics", *Journal of Econometrics*, vol. 73, no. 1, pp. 5-59, 1996. DOI: 10.1016/0304-4076(95)01732-1

[8] D. A. Dickey and W. A. Fuller, "Distribution of the estimators for autoregressive time series with a unit root", *Journal of the American Statistical Association*, vol. 74, no. 366, pp. 427-431, 1979. DOI: 10.2307/2286348

[9] D. Kwiatkowski, P. C. B. Phillips, P. Schmidt, and Y. Shin, "Testing the null hypothesis of stationarity against the alternative of a unit root", *Journal of Econometrics*, vol. 54, no. 1-3, pp. 159-178, 1992. DOI: 10.1016/0304-4076(92)90104-Y

[10] E. F. Fama, "Efficient capital markets: A review of theory and empirical work", *The Journal of Finance*, vol. 25, no. 2, pp. 383-417, 1970. DOI: 10.2307/2325486

[11] A. W. Lo, "Long-term memory in stock market prices", *Econometrica*, vol. 59, no. 5, pp. 1279-1313, 1991. DOI: 10.2307/2938368

[12] I. N. Lobato and N. E. Savin, "Real and spurious long-memory properties of stock-market data", *Journal of Business & Economic Statistics*, vol. 16, no. 3, pp. 261-268, 1998. DOI: 10.1080/07350015.1998.10524760

[13] C. W. J. Granger and N. Hyung, "Occasional structural breaks and long memory with an application to the S&P 500 absolute stock returns", *Journal of Empirical Finance*, vol. 11, no. 3, pp. 399-421, 2004. DOI: 10.1016/j.jempfin.2003.03.001

[14] D. O. Cajueiro and B. M. Tabak, "The hurst exponent over time: Testing the assertion that emerging markets are becoming more efficient", *Physica A: Statistical Mechanics and its Applications*, vol. 336, no. 3-4, pp. 521-537, 2004. DOI: 10.1016/j.physa.2003.12.031

[15] M. Vogl, "Hurst exponent dynamics of S&P 500 returns: Implications for market efficiency, long memory, multifractality and financial crises predictability by application of a nonlinear dynamics analysis framework", *Chaos, Solitons & Fractals*, vol. 166, p. 112 884, 2023. DOI: 10.1016/j.chaos.2022.112884

[16] Z. Ding, C. W. J. Granger, and R. F. Engle, "A long memory property of stock market returns and a new model", *Journal of Empirical Finance*, vol. 1, no. 1, pp. 83-106, 1993. DOI: 10.1016/0927-5398(93)90006-D

[17] R. T. Baillie, T. Bollerslev, and H. O. Mikkelsen, "Fractionally integrated generalized autoregressive conditional heteroskedasticity", *Journal of Econometrics*, vol. 74, no. 1, pp. 3-30, 1996. DOI: 10.1016/S0304-4076(95)01749-6

[18] T. Bollerslev and H. O. Mikkelsen, "Modeling and pricing long memory in stock market volatility", *Journal of Econometrics*, vol. 73, no. 1, pp. 151-184, 1996. DOI: 10.1016/0304-4076(95)01736-4

[19] J. Davidson, "Moment and memory properties of linear conditional heteroscedasticity models, and a new model", *Journal of Business & Economic Statistics*, vol. 22, no. 1, pp. 16-29, 2004. DOI: 10.1198/073500103288619359

[20] M. Martens, D. van Dijk, and M. de Pooter, "Forecasting S&P 500 volatility: Long memory, level shifts, leverage effects, day-of-the-week seasonality, and macroeconomic announcements", *International Journal of Forecasting*, vol. 25, no. 2, pp. 282-303, 2009. DOI: 10.1016/j.ijforecast.2009.01.006

[21] D. P. Louzis, S. Xanthopoulos-Sisinis, and A. N. Refenes, "Forecasting stock index realized volatility with an asymmetric HAR-FIGARCH model: The case of S&P 500 and DJIA stock indices", 2010, SSRN Working Paper. DOI: 10.2139/ssrn.1524861

[22] J.-H. Chen et al., "The comparison of the long memory in volatility for carbon and energy exchange-traded funds", *Spanish Journal of Finance and Accounting*, 2025. DOI: 10.1080/02102412.2025.2563941

[23] W.-J. Tsay, "Long memory story of the real interest rate", *Economics Letters*, vol. 67, no. 3, pp. 325-330, 2000. DOI: 10.1016/S0165-1765(99)00272-4

[24] L. A. Gil-Alana, "Long memory in the U.S. interest rate", *International Review of Financial Analysis*, vol. 13, no. 3, pp. 265-276, 2004. DOI: 10.1016/j.irfa.2004.02.009

[25] R. T. Baillie, C.-F. Chung, and M. A. Tieslau, "Analysing inflation by the fractionally integrated ARFIMA-GARCH model", *Journal of Applied Econometrics*, vol. 11, no. 1, pp. 23-40, 1996.

[26] G. Bhardwaj and N. R. Swanson, "An empirical investigation of the usefulness of ARFIMA models for predicting macroeconomic and financial time series", *Journal of Econometrics*, vol. 131, no. 1-2, pp. 539-578, 2006. DOI: 10.1016/j.jeconom.2005.01.016

[27] C. Ellis and P. Wilson, "Another look at the forecast performance of ARFIMA models", *International Review of Financial Analysis*, vol. 13, no. 1, pp. 63-81, 2004. DOI: 10.1016/j.irfa.2004.01.005

[28] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast accuracy", *International Journal of Forecasting*, vol. 22, no. 4, pp. 679-688, 2006. DOI: 10.1016/j.ijforecast.2006.03.001

[29] F. X. Diebold and R. S. Mariano, "Comparing predictive accuracy", *Journal of Business & Economic Statistics*, vol. 13, no. 3, pp. 253-263, 1995. DOI: 10.1080/07350015.1995.10524599

[30] D. Harvey, S. Leybourne, and P. Newbold, "Testing the equality of prediction mean squared errors", *International Journal of Forecasting*, vol. 13, no. 2, pp. 281-291, 1997. DOI: 10.1016/S0169-2070(96)00719-4

[31] J. L. Elman, "Finding structure in time", *Cognitive Science*, vol. 14, no. 2, pp. 179-211, 1990. DOI: 10.1207/s15516709cog1402_1

[32] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult", *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157-166, 1994. DOI: 10.1109/72.279181

[33] S. Hochreiter and J. Schmidhuber, "Long short-term memory", *Neural Computation*, vol. 9, no. 8, pp. 1735-1780, 1997. DOI: 10.1162/neco.1997.9.8.1735

[34] "LSTMs Explained: A Complete, Technically Accurate, Conceptual Guide with Keras", Analytics Vidhya (Medium), Accessed: Jun. 15, 2026. [Online]. Available: https://medium.com/analytics-vidhya/lstms-explained-a-complete-technically-accurate-conceptual-guide-with-keras-2a650327e8f2

[35] K. Cho et al., "Learning phrase representations using RNN encoder-decoder for statistical machine translation", in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1724-1734. DOI: 10.3115/v1/D14-1179

[36] Fracdiff contributors, *Fracdiff: Fractional differentiation of time series (python package with a differentiable pytorch implementation)*, https://github.com/fracdiff/fracdiff, 2023.

[37] R. Walasek and J. Gajda, "Fractional differentiation and its use in machine learning", *International Journal of Advances in Engineering Sciences and Applied Mathematics*, vol. 13, no. 2-3, pp. 270-277, 2021. DOI: 10.1007/s12572-021-00299-5

[38] F. Tang, *Application of supervised learning models in the Chinese futures market*, arXiv:2303.04581 [q-fin.TR], 2023.

[39] B. Bieganowski and R. Ślepaczuk, *Supervised autoencoders with fractionally differentiated features and triple barrier labelling enhance predictions on noisy data*, arXiv:2411.12753 [q-fin.CP], 2024.

[40] D. Stempień and J. Gajda, *Comparative analysis of financial data differentiation techniques using an LSTM neural network*, arXiv:2505.19243 [q-fin.ST], 2025.

Chapter A Complete Result Tables

This appendix collects the full numerical results summarised in Chapter 5. All values are test-set quantities produced by the public pipeline with fixed seeds.

A.1 RQ1 benchmark with full Diebold-Mariano statistics

Table 8 expands Table 4 with the DM statistic and p-value of every model against the random walk. Negative DM means the model has lower squared error; significance requires $p < 0.05$.

Table 8: Full RQ1 benchmark ($W = 30$, $K = 5$, 50 epochs). Each cell: MASE / DM / p.

		Baseline ($d = 1$)			Fixed-d^*			Learned-d		
Asset	Cell	MASE	DM	p	MASE	DM	p	MASE	DM	p
CADUSD=X	LSTM	0.653	−0.06	.954	0.677	+1.46	.144	0.653	−0.40	.691
CADUSD=X	RNN	0.653	+0.33	.740	0.661	+0.17	.867	0.653	+0.04	.965
TLT	LSTM	0.695	+0.46	.648	0.693	−0.86	.393	0.693	−0.01	.991
TLT	RNN	0.700	+2.25	.025	0.692	−0.17	.868	0.699	+2.14	.032
MSFT	LSTM	2.396	+2.20	.029	2.339	+0.36	.721	2.378	+2.14	.033
MSFT	RNN	2.412	+3.09	.002	2.339	+0.82	.415	2.411	+3.26	.001
TRYUSD=X	LSTM	0.032	−1.59	.113	0.039	+1.22	.224	0.033	−0.85	.394
TRYUSD=X	RNN	0.034	−0.36	.719	0.042	+2.29	.022	0.036	+0.50	.618

A.2 Robustness sweeps

Tables 9 and 10 report the full memory and capacity sweeps on the two most informative assets (the trending equity MSFT and the persistently depreciating TRYUSD=X). In no configuration does any model attain DM < 0 with $p < 0.05$ against the random walk.

A.3 Cross-sectional scan: 24 assets

Table 11 reports the per-asset quantities behind Figure 14: the memory optimum with both ADF specifications, the predictive argmin of the Fixed-d scan, and the corresponding MASEs. Correlations: d_{opt} vs d_1^*: Pearson +0.163 ($p = .446$), Spearman +0.154 ($p = .471$); d_{opt} vs d_{20}^*: Pearson +0.106 ($p = .622$), Spearman +0.099 ($p = .644$).

Table 9: Memory sweep: window W × filter lags K (40 epochs). d_{lrn} = final learned order.

Asset	Cell	(W, K)	d^*	d_{lrn}	RW	Fixed-d^*	Learned-d	p (Lrn)
MSFT	LSTM	(30,5)	0.346	1.000	2.341	2.339	2.383	.026
MSFT	LSTM	(120,50)	0.347	0.826	2.330	2.327	2.354	.037
MSFT	LSTM	(120,100)	0.347	0.911	2.330	2.327	2.359	.030
MSFT	LSTM	(250,50)	0.346	1.000	2.299	2.297	2.366	.003
MSFT	LSTM	(250,100)	0.346	1.000	2.299	2.297	2.367	.003
MSFT	RNN	(30,5)	0.346	1.000	2.341	2.339	2.401	.004
MSFT	RNN	(120,50)	0.347	1.000	2.330	2.329	2.394	.006
MSFT	RNN	(120,100)	0.347	1.000	2.330	2.331	2.394	.006
MSFT	RNN	(250,50)	0.346	1.000	2.299	2.295	2.377	.003
MSFT	RNN	(250,100)	0.346	1.000	2.299	2.295	2.377	.003
TRYUSD=X	LSTM	(30,5)	0.391	1.000	0.033	0.044	0.034	.394
TRYUSD=X	LSTM	(120,50)	0.393	1.000	0.033	0.037	0.032	.577
TRYUSD=X	LSTM	(120,100)	0.393	1.000	0.033	0.035	0.032	.570
TRYUSD=X	LSTM	(250,50)	0.395	1.000	0.032	0.041	0.034	.690
TRYUSD=X	LSTM	(250,100)	0.395	1.000	0.032	0.040	0.033	.716
TRYUSD=X	RNN	(30,5)	0.391	1.000	0.033	0.040	0.034	.764
TRYUSD=X	RNN	(120,50)	0.393	1.000	0.033	0.038	0.032	.478
TRYUSD=X	RNN	(120,100)	0.393	1.000	0.033	0.038	0.032	.474
TRYUSD=X	RNN	(250,50)	0.395	1.000	0.032	0.037	0.034	.732
TRYUSD=X	RNN	(250,100)	0.395	1.000	0.032	0.037	0.034	.750

A.4 Anchored probe on real assets

Table 12 reports the probe of Section 4.7 applied to the 24 real assets ($W = 100$, $K = 50$, one tap; d^* with 20 lags). Six assets nominally exceed the 0.5% depth threshold, but all minimum relMASE values lie in $[0.981, 1.013]$: no asset combines an identified optimum with material predictability. Correlation over all 24: Pearson +0.178 ($p = .405$); over the six nominally identified: +0.333 ($p = .519$).

Table 10: Capacity sweep: hidden units × layers ($W = 30$, $K = 5$, 40 epochs).

Asset	Cell	(H, L)	d_{lrn}	RW	Fixed-d^*	Learned-d	p (Lrn)
MSFT	LSTM	(32,1)	1.000	2.341	2.339	2.383	.026
MSFT	LSTM	(64,1)	1.000	2.341	2.340	2.382	.022
MSFT	LSTM	(64,2)	1.000	2.341	2.340	2.378	.005
MSFT	LSTM	(128,2)	1.000	2.341	2.339	2.357	.351
MSFT	LSTM	(128,3)	1.000	2.341	2.341	2.382	.067
MSFT	RNN	(32,1)	1.000	2.341	2.339	2.402	.004
MSFT	RNN	(64,1)	1.000	2.341	2.340	2.397	.006
MSFT	RNN	(64,2)	1.000	2.341	2.362	2.359	.060
MSFT	RNN	(128,2)	1.000	2.341	2.341	2.406	.001
MSFT	RNN	(128,3)	1.000	2.341	2.344	2.405	.001
TRYUSD=X	LSTM	(32,1)	1.000	0.033	0.044	0.034	.394
TRYUSD=X	LSTM	(64,1)	1.000	0.033	0.037	0.034	.559
TRYUSD=X	LSTM	(64,2)	1.000	0.033	0.036	0.033	.078
TRYUSD=X	LSTM	(128,2)	1.000	0.033	0.039	0.033	.276
TRYUSD=X	LSTM	(128,3)	1.000	0.033	0.036	0.033	.262
TRYUSD=X	RNN	(32,1)	1.000	0.033	0.040	0.034	.764
TRYUSD=X	RNN	(64,1)	1.000	0.033	0.047	0.035	.873
TRYUSD=X	RNN	(64,2)	1.000	0.033	0.047	0.034	.907
TRYUSD=X	RNN	(128,2)	1.000	0.033	0.047	0.040	.000
TRYUSD=X	RNN	(128,3)	1.000	0.033	0.033	0.061	.000

Table 11: Cross-sectional scan over 24 assets (grid $d \in \{.05, .15, .30, .45, .60, .75, .90\}$, $W = 30$, $K = 5$).

Asset	d_1^*	d_{20}^*	d_{opt}	best	RW	Asset	d_1^*	d_{20}^*	d_{opt}	best	RW
EURUSD=X	.077	.218	.15	0.868	0.866	SPY	.347	.576	.30	1.823	1.825
GBPUSD=X	.085	.227	.45	0.796	0.789	QQQ	.356	.620	.15	2.216	2.227
CADUSD=X	.064	.064	.15	0.661	0.653	IWM	.208	.386	.30	1.526	1.504
AUDUSD=X	.075	.164	.60	0.841	0.831	XLF	.250	.445	.30	1.552	1.533
CHFUSD=X	.074	.186	.05	1.149	1.144	XLE	.190	.328	.60	1.499	1.477
TRYUSD=X	.449	.783	.60	0.032	0.033	XLK	.385	.646	.45	2.892	2.919
JPY=X	.250	.492	.05	1.460	1.450	GLD	.249	.491	.05	3.542	3.522
TLT	.174	.421	.05	0.709	0.693	SLV	.088	.253	.30	4.461	4.407
IEF	.199	.438	.45	0.962	0.937	USO	.152	.325	.05	1.047	1.034
SHY	.343	.623	.60	1.298	1.326	MSFT	.391	.668	.15	2.396	2.341
LQD	.188	.394	.05	1.024	1.002	AAPL	.382	.640	.15	2.498	2.449
HYG	.211	.422	.05	0.809	0.802	TIP	.240	.525	.05	0.876	0.880

Table 12: Anchored linear probe on real assets. "Id." = depth $\geq 0.5\%$.

Asset	d^*_{20}	d_{opt}	depth%	min rel.	Asset	d^*_{20}	d_{opt}	depth%	min rel.
EURUSD=X	.196	0.00	0.2	0.997	SPY	.490	0.30	0.2	0.992
GBPUSD=X	.071	0.00	0.7^{Id}	1.002	QQQ	.530	1.20	0.1	0.989
CADUSD=X	.053	1.20	0.2	0.997	IWM	.315	1.20	0.2	1.000
AUDUSD=X	.119	0.00	0.3	1.001	XLF	.380	0.50	0.1	0.998
CHFUSD=X	.146	0.00	0.3	1.000	XLE	.287	0.80	0.3	0.998
TRYUSD=X	.674	0.05	2.0^{Id}	0.993	XLK	.553	1.20	0.3	0.985
JPY=X	.408	0.00	0.3	0.997	GLD	.423	0.00	1.0^{Id}	0.993
TLT	.326	0.00	0.5	1.002	SLV	.212	0.00	2.3^{Id}	1.000
IEF	.332	0.00	0.4	1.003	USO	.245	0.00	0.1	1.001
SHY	.543	0.40	1.3^{Id}	0.981	MSFT	.591	0.00	0.9^{Id}	1.013
LQD	.293	1.20	0.3	1.003	AAPL	.546	1.20	0.0	0.995
HYG	.346	0.00	0.2	0.999	TIP	.430	1.20	0.2	0.996

Chapter B Hyperparameters and Reproducibility

B.1 Hyperparameters by experiment

B.2 Environment and reproduction

All experiments run on CPU with Python 3.11, PyTorch 2.5.1, statsmodels, scikit-learn, NumPy/SciPy and Matplotlib; daily prices are downloaded with `yfinance` (auto-adjusted closes from 2015-01-01). The complete code, including every diagnostic referenced in this document, is available at:

`https://github.com/flopezalvaradoo/TFG-fracdiff-forecasting`

The repository is organised around a single orchestrator, `run_all.py`, which regenerates every table and figure of this thesis in one batch: it runs the synthetic gate first and *aborts* if the gate does not pass 6/6, after which each experiment writes its canonical output to `resultados/`; the figures of this document are produced by `Documentacion/make_memoria_figs.py` from those outputs. All random seeds are fixed (42; the probe uses 42 to 51), so results are deterministic up to one caveat: the data provider serves prices up to the day of execution, so decimals drift when the battery is re-run at a later date. Every verdict reported in this thesis (gate 6/6; RQ1 0/8; volatility 5/6 on $|r_t|$; the tracking statistics of the mechanistic study) is stable under refresh.

Table 13: Hyperparameters of every experiment. All networks use Adam (lr 10^{-3}, batch 256, MSE on the standardised target); the learned order uses a dedicated learning rate of 0.05, clamp $(0.01, 1.0]$ and initialisation 0.9. Train/test split: chronological 80/20. Seed 42 unless stated.

Experiment (script)	W	K	Epochs	Model	Grids / notes
Gate (`validate_synthetic.py`)	120	50	40	LSTM, RNN 32×1	$d_{true} \in \{0, .30, .45\}$, N=4000
Benchmark (`benchmark_real.py`)	30	5	50	LSTM, RNN 32×1	4 assets, 3 d-modes
Memory sweep (`sweep_fraclags_corr.py`)	30 to 250	5 to 100	40	LSTM, RNN	5 (W, K) configurations
Capacity sweep (`sweep_capacity_corr.py`)	30	5	40	up to 128×3	5 (H, L) configurations
Landscape (`landscape_real.py`)	30	5	50	LSTM Fixed-d	grid $d \in [0.1, 1.2]$, step .1
Return-target synthetic (`sweep_dtrue.py`)	120	50	40	LSTM Fixed-d	$d_{true} \in [0, .45]$
Cross-section (`real_many_dstar20.py`)	30	5	40	LSTM (level)	24 assets; d_1^* and d_{20}^*
Mechanistic probe (`mechanism_when_correlation.py`)	100	50	n/a	OLS probe (1 tap)	grid $d \in [0, 1.2]$ step .05; seeds 42 to 51; LSTM check 40 epochs, seeds 42 to 44
Volatility (`vol_test.py`)	30	5	40	LSTM (level)	grid $d \in \{0, \ldots, 1\}$; d selected on last 20% of train
Classical (`compare_arima_arfima_corr.py`)	n/a	n/a	n/a	ARIMA/ ARFIMA	`statsmodels`, SI=F
d^* estimator (`hurst/adf_memory.py`)	200 Adam steps, lr .05; filter 50 to 100 lags; $c = -2.86$; $\lambda = 10$; ADF lags 1 or 20				