Zürcher Hochschule
für Angewandte Wissenschaften

A Comparison Between Traditional and Machine Learning Based Calibration of the One Factor Hull-White Model

Master's Thesis

Author: Benedikt Grimus

Study Program: Banking & Finance

Supervisor: Dr. Marc Weibel

ZHAW School of Management and Law

Submitted on: 18. November 2025

Abstract

Contents

1 Introduction

Interest rate models are a cornerstone of modern quantitative finance, providing the essential framework for pricing derivatives, managing risk, and formulating hedging strategies. Among the various term structure models, the one-factor Hull-White model has established itself as a benchmark due to its analytical tractability and its ability to perfectly fit the initial term structure of interest rates. Its application is widespread, from the valuation of simple options like caps and floors to more complex, path-dependent instruments such as Bermudan swaptions. However, the utility of any model is fundamentally dependent on its calibration—the process of aligning its parameters with observed market prices to ensure that it reflects current market conditions and expectations.

The calibration of the Hull-White model, particularly its mean-reversion and volatility parameters, presents a significant challenge. Traditional calibration techniques, predominantly based on numerical optimization algorithms like the Levenberg-Marquardt method, are well-established. While effective, these methods often suffer from notable drawbacks. They can be computationally intensive, requiring significant time to converge to a solution. Furthermore, they are susceptible to becoming trapped in local minima, potentially yielding suboptimal parameters that do not accurately represent the global market landscape, especially in volatile or rapidly changing market regimes. This computational latency and potential for instability create a critical bottleneck for financial institutions that require fast and robust model calibrations for real-time pricing and risk management.

The recent advancements in machine learning, and specifically in the field of neural networks, offer a promising alternative to overcome these limitations. Neural networks are exceptionally adept at learning complex, non-linear relationships directly from data. By training a network on historical market data and the corresponding "true" model parameters, it is possible to create a surrogate model that can approximate the calibration function almost instantaneously. This data-driven approach has the potential to transform the calibration process from a time-consuming optimization task into a rapid inference problem, thereby offering significant gains in speed and consistency. The motivation for this research stems from the pressing need for more efficient and robust calibration techniques in an increasingly fast-paced financial world.

The primary aim of this thesis is to provide a comprehensive comparison between the traditional, optimization-based calibration of the one-factor Hull-White model and a modern, machine learning-based approach. To achieve this, the research will be guided by the following key questions:

1. How does the accuracy of a neural network-based calibration, measured by the model's ability to replicate market swaption prices, compare to that of the traditional Levenberg-Marquardt algorithm?

2. What is the quantitative difference in computational speed between the two methods, considering both the initial setup (training) and the ongoing application (inference)?

3. How stable are the estimated model parameters over time?

To ensure a focused and rigorous analysis, this study is subject to several delimitations. The scope is confined to the one-factor Hull-White model, calibrated using a comprehensive dataset of European at-the-money (ATM) swaptions from the Euro (EUR) market. The analysis covers a continuous time period from June 1, 2025, to August 31, 2025, providing a view of the model's performance in a recent market environment. The study does not extend to multi-factor models or other classes of interest rate derivatives.

This thesis is structured as follows. Chapter 2 provides a detailed background on the theoretical foundations, including swaptions, yield curve bootstrapping, and a thorough review of the Hull-White model and its traditional calibration. Chapter 3 delineates the research methodology, covering the dataset, the implementation of the machine learning framework, the feature engineering process, and the metrics used for comparison. Chapter 4 presents the empirical results of the comparative analysis, focusing on accuracy, speed, and robustness. Finally, chapter 5 concludes the thesis by summarizing the key findings, discussing their implications for financial practitioners, and suggesting avenues for future research.

2 Background

Before delving into the comparative analysis that forms the core of this thesis, it is essential to establish a solid theoretical foundation. This chapter provides a comprehensive overview of the key financial instruments and mathematical models that underpin this research. The discussion begins with the fundamental characteristics of swaptions and their importance in the interest rate derivatives market. It then proceeds to detail the one-factor Hull-White model, explaining its mathematical specification, pricing solutions, and the conventional calibration process using numerical optimization methods. Finally, the chapter introduces the foundational concepts of neural networks, outlining the architectural and learning principles that enable their application as a modern calibration tool.

2.1 Swaptions: Definition, Characteristics, and Financial Relevance

A swaption, or swap option, is a derivative contract that grants the holder the right, but not the obligation, to enter into an interest rate swap at a predetermined date in the future and under predefined terms. Specifically, a payer swaption gives the holder the right to enter into a swap as the fixed-rate payer (and floating-rate receiver), while a receiver swaption gives the right to enter as the fixed-rate receiver (and floating-rate payer). Swaptions thus represent a combination of option and interest rate derivative characteristics and are a fundamental component of the interest rate derivatives market (Brigo & Mercurio, 2006, pp. 19–20).

Swaptions can be classified by the style of exercise they permit, which significantly influences their valuation complexity and hedging strategies.

European swaptions are the most common type and allow exercise only at a single specified future date (Brigo & Mercurio, 2006, p. 19), typically coinciding with the start of the underlying swap. Their valuation is relatively tractable and often performed using closed-form solutions such as Black's formula (Black, 1976).

American swaptions allow exercise at any time up to the expiry date (Karlsson et al., 2016). This added flexibility increases the option's value but also its computational complexity. Due to the path-dependency of optimal exercise, American swaptions are typically valued using numerical methods such as lattice models (Gurrieri et al., 2009) or finite-difference (Longstaff & Schwartz, 2001) schemes.

Bermudan swaptions occupy an intermediate position, allowing the holder to exercise the option on a set of predetermined dates, often corresponding to coupon dates of the underlying swap. Bermudan swaptions are widely traded (Karlsson et al., 2016) and are especially relevant for callable and cancellable structures in interest rate risk management (Rebonato, 2004, p. 22). Valuation requires techniques that account for multiple early exercise opportunities, such as backward induction (Karlsson et al., 2016) or least-squares Monte Carlo methods (Longstaff & Schwartz, 2001).

Asian swaptions are path-dependent options where the payoff depends on the average of the underlying swap rate over a certain period rather than a single rate observation at expiry. Because of this averaging feature, their valuation requires models that account for the entire path of interest rates rather than just their terminal value. Analytical or semi-analytical approximations—such as volatility expansion techniques can be used to price them (Baaquie, 2010).

Each of these exercise styles leads to distinct pricing and risk management considerations and determines the appropriate mathematical framework and numerical method for valuation. In addition to exercise style, a swaption is characterized by several structural and market parameters that determine its valuation and risk profile:

Option Maturity (Expiry): This denotes the time until the swaption can be exercised. It reflects the horizon over which uncertainty about future interest rate movements is resolved. Longer maturities generally imply greater sensitivity to the dynamics of the underlying yield curve and the volatility of interest rates.

Underlying Swap Tenor: The tenor is the length of the interest rate swap that begins upon swaption exercise (Brigo & Mercurio, 2006, p. 19). For example, a 5yX10y swaption refers to an option that expires in five years and, if exercised, initiates a ten-year interest rate swap. The swap tenor affects the duration and convexity of the position, and thus its sensitivity to changes in the yield curve.

Strike Rate: The strike rate is the fixed interest rate agreed upon in the swaption contract. At expiry, the decision to exercise the swaption depends on the relationship between this strike and the prevailing forward swap rate for the corresponding maturity and tenor (Brigo & Mercurio, 2006, p. 239).

Premium: The premium, or market price, of the swaption reflects the cost of acquiring the optionality embedded in the contract. It is influenced by all of the above parameters, as well as prevailing interest rates, the volatility environment, and the exercise style.

Implied Volatility: Swaptions are quoted in terms of implied volatility, which is extracted from market prices using inversion techniques based on models such as Black's formula (Hohmann et al., 2015, p. 3). Implied volatility encapsulates the market's expectation of future fluctuations in interest rates and plays a central role in option pricing and risk management. Market practitioners often use a two-dimensional volatility cube, where implied volatilities vary with both option expiry and swap tenor (Suo et al., 2009, p. 1).

Swaptions serve multiple purposes in financial markets, ranging from risk management to model calibration and investment strategies. They are fundamental tools used by financial institutions, borrowers, and specialized agencies to manage future interest rate risk. In particular, Bermudan swaptions are highly relevant for hedging callable and cancellable structures in the over-the-counter (OTC) market. For example, government-sponsored mortgage agencies in the USD market often purchase European and Bermudan swaptions to hedge exposure created by callable debt funding programs (Rebonato, 2004, p. 22). By granting the holder the right, but not the obligation, to enter into an interest rate swap at a predetermined future date, swaptions create optionality that protects against adverse movements in interest rates and makes sure that a company payments at each payment date have been capped to the fixed rate (Brigo & Mercurio, 2006, p. 17). The payoff of a European swaption can also be replicated by a continuously rebalanced portfolio of zero-coupon bonds, which provides valuable insight for managing the risk of positions in the derivative itself (Brigo & Mercurio, 2006, p. 241). Additionally, swaptions are used to manage volatility risk, which can be the largest component of risk in a derivative deal (Brigo & Mercurio, 2006, p. 242).

Beyond risk management, swaptions are critical for market pricing, valuation, and calibration of interest rate models (Brigo & Mercurio, 2006, pp. 132–136). Swaption prices are essential inputs for calibrating models such as the Hull-White model (Kladivko & Rusy, 2023) and the LIBOR Market Model (LMM) for instance, calibrating the LMM to the swaption market allows parameters like instantaneous correlations to be determined (Brigo & Mercurio, 2006, p. 290).

Swaptions are also used to adjust financing costs and optimize yields. Investors seeking yield enhancement and issuers in search of advantageous funding rates may sell swaption-type optionality, while swaptions or swaption-like features are frequently embedded in more complex financial products or callable structures offered to investors (Rebonato, 2004, p. 22).

Swaptions represent highly versatile derivatives within interest rate markets, serving several critical functions. Their primary application lies in risk management, where they provide market participants with effective tools to hedge against adverse interest rate movements and manage volatility exposure. Moreover, swaptions play a pivotal role in the calibration of interest rate models; market-implied volatilities from swaption prices are essential inputs for sophisticated frameworks like the Hull-White and LIBOR Market Models. Finally, beyond their utility in hedging and valuation, these instruments are strategically employed by investors to enhance portfolio yields and by issuers to optimize

the costs associated with debt financing.

2.2 Bootstrapping the Zero-Coupon Yield Curve from Swap Rates

The construction of a zero-coupon yield curve from market swap rates is an essential step in interest rate modeling. Since swap contracts are among the most liquid instruments across maturities, they serve as primary inputs for determining the risk-free term structure. The goal of the bootstrapping process is to derive discount factors $\{P(t_0, T_i)\}_{i=1}^{N}$ or equivalently continuously compounded zero rates $\{z(T_i)\}_{i=1}^{N}$ that are consistent with observed market swap rates $\{R_i\}_{i=1}^{N}$ (J. C. Hull, 2015, pp. 84–86).

2.2.1 Preliminaries and Definitions

Let t_0 denote the valuation date and $\{T_i\}_{i=1}^{N}$ the maturities of the quoted par swap rates. Each swap with maturity T_i exchanges a fixed rate R_i for a floating rate indexed to a short-term reference rate (e.g., EURIBOR 6M) at payment dates $\{t_{i,k}\}_{k=1}^{n_i}$ on the fixed leg.

The discount factor $P(t_0, t)$ represents the present value at time t_0 of one unit of currency to be received at time t. The continuously compounded zero rate $z(t)$ is defined by

$$P(t_0, t) = e^{-z(t)(t-t_0)}, \tag{1}$$

which implies

$$z(t) = -\frac{1}{t - t_0} \ln P(t_0, t). \tag{2}$$

2.2.2 Valuation of a Par Swap

A standard fixed-for-floating interest rate swap can be interpreted as the difference between a fixed-rate bond and a floating-rate note. The present value of the fixed leg at inception is given by

$$PV_{\text{fixed}}(t_0) = R_i \sum_{k=1}^{n_i} \alpha_{i,k} P(t_0, t_{i,k}), \tag{3}$$

where $\alpha_{i,k}$ denotes the accrual factor for the period $[t_{i,k-1}, t_{i,k}]$ under the fixed-leg day count convention.

The present value of the floating leg for a par swap (i.e., at inception) equals

$$PV_{\text{float}}(t_0) = 1 - P(t_0, T_i), \tag{4}$$

under the assumption that the first floating coupon is set at par.

At inception, the par swap has zero net present value, so that

$$PV_{\text{fixed}}(t_0) = PV_{\text{float}}(t_0), \tag{5}$$

which yields the par swap relation

$$R_i \sum_{k=1}^{n_i} \alpha_{i,k} P(t_0, t_{i,k}) = 1 - P(t_0, T_i). \tag{6}$$

2.2.3 Recursive Bootstrapping Procedure

Equation (6) establishes a nonlinear relationship between the par swap rate R_i and the discount factors $\{P(t_0, t_{i,k})\}$. If discount factors up to T_{i-1} are already known from previously bootstrapped maturities, one can rearrange (6) to isolate the unknown discount factor $P(t_0, T_i)$ as

$$P(t_0, T_i) = \frac{1 - R_i \sum_{k=1}^{n_i - 1} \alpha_{i,k} P(t_0, t_{i,k})}{1 + R_i \alpha_{i,n_i}}. \tag{7}$$

The procedure thus proceeds recursively:

1. Initialize the curve with known short-term instruments (e.g., deposits or short-dated FRAs) to obtain discount factors for the first maturities.
2. For each subsequent maturity T_i, solve (7) for $P(t_0, T_i)$ using previously determined discount factors.
3. Continue until the entire maturity spectrum $\{T_i\}_{i=1}^{N}$ is covered.

Once the discount factors are determined, the continuously compounded zero rates follow from (2).

2.2.4 Interpolation and Continuity of the Term Structure

The market provides only a discrete set of maturities $\{T_i\}$. To obtain a continuous term structure, the discount factors between quoted maturities are interpolated. A common and arbitrage-free choice is to interpolate linearly in log-discount space, i.e.,

$$\ln P(t_0, t) = (1 - \lambda) \ln P(t_0, T_j) + \lambda \ln P(t_0, T_{j+1}), \quad t \in [T_j, T_{j+1}], \tag{8}$$

where

$$\lambda = \frac{t - T_j}{T_{j+1} - T_j}. \tag{9}$$

This ensures that the discount function $P(t_0, t)$ is continuous, positive, and monotonically decreasing, thereby avoiding arbitrage opportunities.

The bootstrapping method yields a set of discount factors $\{P(t_0, T_i)\}$ that are internally consistent with observed swap rates $\{R_i\}$. The corresponding zero-coupon yield curve $\{z(T_i)\}$ satisfies:

$$z(T_i) = -\frac{1}{T_i - t_0} \ln P(t_0, T_i), \tag{10}$$

providing a continuous, arbitrage-free representation of the term structure of interest rates suitable for valuation, risk management, and model calibration.

2.3 Short Rate Models

Short-rate models are a class of term structure models that describe the evolution of interest rates by modeling the dynamics of the instantaneous short-term interest rate, denoted by r_t. These models serve as the foundational framework for pricing interest rate derivatives, particularly those with path-dependent features or early exercise options such as American-style options where traditional models like Black's formula fall short due to their static assumptions. The short rate r_t represents the risk-free rate applicable over an infinitesimally short time interval and evolves according to a stochastic differential equation (SDE), typically under the risk-neutral measure. Under this framework, the present value of future cash flows is derived by discounting at the short-rate path, allowing for a consistent valuation across a variety of financial instruments (J. C. Hull, 2015, pp. 706–735).

Short-rate models are flexible in that they can be constructed either to derive the term structure as a model outcome (as in equilibrium models) or to fit the observed term structure exactly (as in no-arbitrage models). Regardless of the approach, these models typically require numerical methods for implementation, including lattice-based techniques (e.g., trinomial trees) or Monte Carlo simulation, especially in the case of American or exotic derivatives. A key aspect of the usability of the model is its calibration to market data, which involves estimating parameters - such as mean reversion speed (a) and volatility (σ)—from prices of liquid instruments such as caps and swaps. In the post-2008 financial environment, the shift towards OIS discounting has led to the need for multiple-curve modeling frameworks, often requiring the simultaneous calibration of separate short-rate processes for both the discounting curve (e.g., OIS) and the forwarding curve (e.g., LIBOR) (J. C. Hull, 2015, pp. 706–735).

2.3.1 Equilibrium Models

Equilibrium short-rate models are constructed by specifying a stochastic process for the short-term interest rate based on underlying economic principles, such as investor preferences and macroeconomic fundamentals. Unlike arbitrage-free models, equilibrium models derive the entire term structure of interest rates as an endogenous output of the model, rather than fitting it directly to observed market data. A distinguishing feature is that the drift term in the short rate's SDE is generally independent of time, reflecting long-term economic forces rather than market-implied expectations.

One of the earliest examples, the Rendleman–Bartter model, assumes that the short rate follows a geometric Brownian motion without mean reversion, which makes it analogous to equity price modeling, but unsuitable for interest rates due to its unbounded and potentially negative outcomes (J. C. Hull, 2015, p. 708). In contrast, the Vasicek model introduces mean reversion, modeling the short rate as an Ornstein-Uhlenbeck process. Although analytically tractable and capable of capturing the tendency of interest rates to revert to a long-term mean, it allows for negative interest rates, a feature that may be undesirable in certain market environments (J. C. Hull, 2015, pp. 708–709). The Cox–Ingersoll–Ross (CIR) model addresses this limitation by specifying that the diffusion term is proportional to the square root of the short rate, thereby ensuring non-negativity

under suitable parameter conditions (J. C. Hull, 2015, p. 710).

Despite their elegance and analytical appeal, equilibrium models are often limited in practical applications because they do not guarantee consistency with the observed initial term structure. As such, they are more suitable for theoretical analysis and long-term economic forecasting than for precise pricing and hedging of interest rate derivatives in the current market environment (J. C. Hull, 2015, pp. 707–714).

2.3.2 No Arbitrage Models

No-arbitrage short-rate models are designed to exactly match the observed term structure of interest rates at inception by construction. These models ensure that there are no arbitrage opportunities in the pricing of fixed-income securities and derivatives by embedding the initial yield curve as an explicit input. The drift component of the stochastic differential equation of the short rate in these models is typically time dependent, calibrated such that the models' generated bond prices align with market prices at time zero. As a result, no-arbitrage models are preferred for pricing and risk management in practice, especially in environments where accuracy in capturing the term structure is essential (J. C. Hull, 2015, pp. 714–715).

The Ho–Lee model, introduced in 1986, was the first short -rate model to incorporate the no-arbitrage principle. It assumes constant volatility and introduces a time-dependent drift term to fit the initial yield curve, resulting in a normally distributed short rate. While simple and analytically tractable, the Ho–Lee model allows for negative interest rates and constant volatility across all maturities (J. C. Hull, 2015, pp. 715–716). To address the limitations of both the Vasicek and Ho–Lee models, the Hull–White one-factor model combines mean reversion with time-dependent drift, allowing for greater flexibility in fitting the term structure and interest rate volatility (J. C. Hull, 2015, pp. 716–718).

Further refinements include multi-factor extensions, such as the Hull–White two-factor model, which captures changes in both the level and slope of the yield curve through the interaction of two stochastic drivers. These enhancements are especially useful in capturing the empirical behavior of interest rates and in pricing complex derivatives such as Bermudan swaptions. The requirement to calibrate these models to market-observed instruments like caps, floors, and swaptions is central to their application, and they are widely used in practice for valuation, hedging, and risk management of interest rate products (J. C. Hull, 2015, p. 719).

2.3.3 Gaussian Short Rate Models

Gaussian short rate models form a specific subclass within the broader family of short rate models in which the evolution of the short-term interest rate—or a transformation thereof—is governed by a stochastic process with normally distributed increments. The defining feature of these models is the assumption that the short rate r_t (or its change) evolves under a Brownian motion dz_t with constant or time-dependent volatility, leading

to normally distributed outcomes (Brigo & Mercurio, 2006, p. 53). As such, these models offer analytical tractability and closed-form solutions for bond prices and some derivative instruments. However, they also imply the theoretical possibility of negative interest rates, which may or may not be consistent with the prevailing market environment or the model's intended application (J. C. Hull, 2015, pp. 719–720).

Formally, a Gaussian short rate model is characterized by a stochastic differential equation (SDE) of the general form:

$$dr_t = \mu(t, r_t)dt + \sigma(t)dz_t, \tag{11}$$

where $\mu(t, r_t)$ is the drift term, $\sigma(t)$ is the volatility function (possibly constant or time-dependent), and dz_t denotes the increment of a standard Wiener process under the risk-neutral measure (Brigo & Mercurio, 2006, p. 53). Because the increments of dz_t are normally distributed, so too are the increments of r_t, provided that the transformation applied to the short rate is linear.

Prominent examples of Gaussian short rate models include the Vasicek, Ho–Lee, and Hull–White (one-factor) models, each of which was discussed in prior sections. While differing in their specification of the drift and volatility terms, all three share the core Gaussian property of allowing for symmetric, unbounded movements in the short rate. This feature enhances analytical tractability—particularly in deriving closed-form solutions for zero-coupon bond prices—but also permits negative interest rates, a theoretical artifact that gained practical relevance in post-crisis monetary regimes where nominal yields fell below zero in several developed markets.

Importantly, Gaussian short rate models stand in contrast to lognormal models such as the Black–Derman–Toy and Black–Karasinski models, which restrict the short rate to remain strictly positive by modeling either the logarithm of the rate or its multiplicative structure . While these alternative models circumvent the issue of negative rates, they often sacrifice analytical tractability in favor of numerical implementation (J. C. Hull, 2015, p. 718).

This master thesis will focus in the following chapters on the one-factor Hull–White model, a prominent Gaussian short rate model that enhances the classical Vasicek formulation by introducing a time-dependent drift term to ensure an exact fit to the initial term structure of interest rates. Its balance between flexibility, analytical tractability, and market-consistency makes it a standard benchmark in both academic research and practical applications, particularly for the valuation and risk management of interest rate derivatives.

2.4 The One-Factor Hull-White Model

The one-factor Hull-White model represents a pivotal advancement in interest rate modeling, combining analytical tractability, empirical flexibility, and arbitrage-free consistency. As detailed in Hull and White's seminal 1990 paper (J. Hull & White, 1990), the

model extends the classical Vasicek framework by introducing time-dependent parameters, enabling exact calibration to the observed initial term structure of interest rates. As explained in Section 2.3.3, the Hull-White model falls within the class of Gaussian short rate models due to its assumption of normally distributed short rate dynamics. Furthermore, as discussed in Section 2.3.2, it enforces market consistency by incorporating a time-varying drift term.

The Hull-White model offers a favorable balance between model realism and computational tractability. Its ability to fit the observed term structure exactly, model time-varying volatilities, and provide analytic solutions for a wide class of derivatives positions it as a standard tool in fixed income modeling. It is particularly well-suited for applications such as: Pricing of swaptions, caps, and callable bonds, risk management and Value-at-Risk (VaR) calculations and simulation of future interest rate scenarios for balance sheet modeling. The model's allowance for negative interest rates, due to its Gaussian nature, has become a point of practical relevance in recent years, as monetary policy in several advanced economies has driven rates below zero.

2.4.1 Mathematical Specification

The Hull-White model is often referred to as the *extended Vasicek model*, formalized as follows under the risk-neutral measure $\mathbb{Q}$:

$$dr_t = [\theta(t) - ar_t]\, dt + \sigma(t) dz_t, \tag{12}$$

where:

Table 1: Notation for the Hull–White Short Rate Model

Symbol	**Meaning**
r_t	Instantaneous short rate at time t
$\theta(t)$	Deterministic function fitted to match the initial term structure
a	Constant mean reversion speed ($a > 0$)
$\sigma(t)$	Time-dependent volatility function
dz_t	Standard Brownian motion under the risk-neutral measure $\mathbb{Q}$

This formulation allows the short rate to mean-revert toward a time-varying level $\theta(t)/a$, ensuring flexibility in capturing the shape of the initial yield curve while retaining a parsimonious structure. The presence of $\sigma(t)$ permits the model to reflect changes in the term structure of interest rate volatilities, which is crucial for accurately pricing interest rate derivatives.

2.4.2 Bond Pricing and Functional Solutions

One of the key strengths of the Hull-White model is its closed-form solution for zero-coupon bond prices. The price $P(r_t, t, T)$ of a zero-coupon bond maturing at time T, given the short rate at time t, is given by:

$$P(r_t, t, T) = \exp\left[A(t, T) - B(t, T)r_t\right], \tag{13}$$

where the deterministic functions $A(t, T)$ and $B(t, T)$ are defined as:

$$B(t, T) = \frac{1}{a}\left(1 - e^{-a(T-t)}\right), \tag{14}$$

$$A(t, T) = \int_t^T \left[\theta(s)B(s, T) - \frac{1}{2}\sigma(s)^2 B(s, T)^2\right] ds. \tag{15}$$

These solutions emerge from solving the associated partial differential equation for contingent claim prices, under the assumption of a risk-neutral market and bounded market price of risk. The functional form of $A(t, T)$ links directly to the model's ability to replicate the observed yield curve.

2.4.3 Option Pricing

A notable advantage of the Hull-White model is its analytic tractability for European-style options on bonds, which makes it especially valuable for practical implementation. The lognormality of bond prices under the model's Gaussian short rate dynamics allows for closed-form expressions for European bond option prices. For instance, a European call option on a discount bond with maturity T and exercise at $t_1 < T$ has a pricing formula structurally analogous to the Black formula, with a volatility term given by:

$$\sigma_P = \sigma(t_1)B(t_1, T), \tag{16}$$

where the bond price volatility is independent of r_t, enabling straightforward evaluation of the option price. Options on coupon-bearing instruments can be priced via decomposition into portfolios of zero-coupon bond options.

For American-style options or more complex interest-rate-contingent claims, numerical techniques such as finite-difference methods or lattice-based approaches (e.g., trinomial trees) are required to solve the underlying PDE.

2.4.4 Swaption Pricing

In the one-factor Hull-White model, European swaptions - options that grant the right to enter into an interest rate swap at a predetermined fixed rate -can be priced analytically

using a technique known as *Jamshidian's decomposition* (Jamshidian, 1989). This method is particularly powerful for one-factor models, where all coupon bond prices at a given time are deterministic functions of the short rate.

The fundamental insight of Jamshidian's approach is that a payer (or receiver) swaption, which is an option on a portfolio of fixed payments, can be decomposed into a portfolio of options on individual zero-coupon bonds. Since the Hull-White model provides closed-form solutions for European options on zero-coupon bonds (as discussed in the previous subsection), this decomposition facilitates an analytical expression for the swaption price.

A European payer swaption gives the holder the right, but not the obligation, to enter at maturity T into a fixed-for-floating interest rate swap where fixed payments of K occur at payment dates $T_1, T_2, \ldots, T_n$. Let $P(t, T_i)$ denote the price at time t of a zero-coupon bond maturing at T_i, and let α_i denote the accrual factors (e.g., 0.5 for semiannual payments). The value at time t of the fixed leg of the swap is:

$$\text{FixedLeg}(t) = K \sum_{i=1}^{n} \alpha_i P(t, T_i). \tag{17}$$

The value of the floating leg at time t equals one minus the price of a discount bond maturing at the swap start date T_0, assuming the swap is par at initiation. The swaption payoff at expiry T is therefore:

$$\max\left(\sum_{i=1}^{n} \alpha_i P(T, T_i)(K - S(T)), 0 \right), \tag{18}$$

where $S(T)$ is the par swap rate at time T.

Jamshidian's decomposition allows this multi-cash-flow option to be re-expressed as a portfolio of individual bond options. Specifically, there exists a unique short rate r^* such that the present value of the fixed leg equals the strike value $K \sum_{i=1}^{n} \alpha_i P(T, T_i)$. Formally, we define r^* as the root of the equation:

$$\sum_{i=1}^{n} \alpha_i P(r^*, T, T_i) = \sum_{i=1}^{n} \alpha_i P(T, T_i), \tag{19}$$

where $P(r^*, T, T_i)$ is the model-implied bond price expressed as a function of the short rate. Once r^* is found, the swaption payoff can be decomposed into a sum of bond option payoffs with the same exercise date T:

$$\text{Swaption}(t) = \sum_{i=1}^{n} \alpha_i \cdot \text{Option}(P(t, T_i), K_i), \tag{20}$$

where each $K_i = P(r^*, T, T_i)$ acts as the strike price for the bond option maturing at T_i.

In the Hull-White model, the price of a European call option on a zero-coupon bond with maturity T_i, strike K_i, and exercise at T is given by the closed-form formula:

$$C(t) = P(t,T_i)N(d_1) - K_i P(t,T)N(d_2), \tag{21}$$

with

$$d_1 = \frac{\ln\left(\frac{P(t,T_i)}{K_i P(t,T)}\right)}{\sigma_P} + \frac{1}{2}\sigma_P, \quad d_2 = d_1 - \sigma_P, \tag{22}$$

and σ_P being the bond price volatility given by:

$$\sigma_P^2 = \int_t^T \left(\sigma(s)B(s,T_i) - \sigma(s)B(s,T)\right)^2 ds. \tag{23}$$

The full swaption price is obtained by summing the individual bond option prices across all cash flow dates.

$$\text{Swaption}(t) = \sum_{i=1}^{n} \alpha_i \left[P(t,T_i)N(d_1^{(i)}) - K_i P(t,T)N(d_2^{(i)})\right], \tag{24}$$

$$\text{with} \quad d_1^{(i)} = \frac{\ln\left(\frac{P(t,T_i)}{K_i P(t,T)}\right)}{\sigma_P^{(i)}} + \frac{1}{2}\sigma_P^{(i)}, \quad d_2^{(i)} = d_1^{(i)} - \sigma_P^{(i)}, \tag{25}$$

$$\text{and} \quad \left(\sigma_P^{(i)}\right)^2 = \int_t^T \left[\sigma(s)B(s,T_i) - \sigma(s)B(s,T)\right]^2 ds, \tag{26}$$

$$\text{where} \quad K_i = P(r^*,T,T_i), \tag{27}$$

Table 2: Notation for Swaption Pricing via Jamshidian's Decomposition

Symbol	**Meaning**
$P(t,T)$	Price at time t of a zero-coupon bond maturing at T
T	Expiry of the swaption
T_i	Payment date of the i-th swap cash flow ($i = 1,\ldots,n$)
α_i	Year fraction (accrual factor) between T_{i-1} and T_i
r^*	Unique short rate at expiry T solving the bond portfolio equation
K_i	Strike price of the bond option for maturity T_i, i.e., $P(r^*,T,T_i)$
$\sigma(s)$	Instantaneous volatility function of the Hull–White model
$B(s,T)$	Sensitivity function: $B(s,T) = \frac{1-e^{-a(T-s)}}{a}$
$\sigma_P^{(i)}$	Volatility of the bond price ratio $P(t,T_i)/P(t,T)$
$d_1^{(i)}, d_2^{(i)}$	Black-type variables for bond option i
$N(\cdot)$	Standard normal cumulative distribution function

Jamshidian's decomposition offers significant computational efficiency for one-factor models by reducing a complex multidimensional option pricing problem to a sequence of univariate bond option valuations. Moreover, this approach retains full analytical tractability under the Hull-White framework, allowing for fast and accurate pricing of European swaptions, a critical requirement in both risk management and market calibration contexts.

2.4.5 Calibration of the Hull-White Model

The calibration of the one-factor Hull-White model plays a central role in ensuring its effectiveness for interest rate derivative pricing and risk management. As established in previous sections, the Hull-White model provides analytical tractability and the ability to fit the initial yield curve exactly. However, to make the model reflect market-implied prices of interest rate derivatives, particularly caplets and swaptions, its dynamic parameters-mean reversion $a(t)$ and volatility $\sigma(t)$-must be appropriately calibrated.

In practical applications, $a(t)$ and $\sigma(t)$ are commonly assumed to be piecewise constant to simplify the numerical implementation (Gurrieri et al., 2009, p. 7). However, unconstrained piecewise constant forms can lead to overfitting and parameter instability (Gurrieri et al., 2009, p. 2). To overcome this, the model is typically calibrated using functional parameterizations, such as logistic forms for $a(t)$ and cubic splines for $\sigma(t)$, to ensure smoothness and stability (Gurrieri et al., 2009, p. 8).

The calibration process relies on three key inputs: the initial yield curve, which determines $\theta(t)$, market-observed prices or implied volatilities of caplets and swaptions, and a well-defined objective function, often the sum of squared differences between model and market prices or implied volatilities. In particular, European swaptions serve as the primary calibration instruments. These are priced analytically via Jamshidian's decomposition (see Section 2.4.4), allowing the decomposition of the swaption payoff into a sum of bond options. The analytical tractability of this decomposition facilitates efficient model evaluation during optimization.

Three main strategies are typically employed to calibrate the Hull-White model parameters:

1. Fixed Mean Reversion: In this approach, the mean reversion parameter $a(t)$ is fixed to a pre-determined constant value, often based on empirical considerations or desired product sensitivities (e.g., Bermudan swaptions). The volatility parameter $\sigma(t)$ is then optimized to fit a subset of swaption prices or implied volatilities. While this method simplifies the calibration and allows for fast implementation, it can be sensitive to market regime changes and may offer poor global fit if the chosen a does not reflect current market conditions (Gurrieri et al., 2009, p. 13).

2. Two-Step Estimation: This method separates the calibration of $a(t)$ and $\sigma(t)$. The mean reversion $a(t)$ is first estimated using an approximation method (e.g., the SMM approximation), which relates swaption implied volatilities in a manner independent of $\sigma(t)$. Once $a(t)$ is estimated and fixed, $\sigma(t)$ is calibrated to fit analytical swaption prices. This approach offers a balance between robustness and speed, making it suitable for

environments requiring frequent recalibration (Gurrieri et al., 2009, pp. 13–14).

3. Simultaneous Optimization: This strategy involves a joint, multi-dimensional optimization of both $a(t)$ and $\sigma(t)$ to minimize the misfit between model prices and market prices of swaptions. While computationally more intensive, this method typically yields the highest fit quality across a broader range of instruments. Constraints and functional parameterizations are critical to ensure numerical stability and prevent overfitting (Gurrieri et al., 2009, p. 14).

The choice of calibration instruments significantly impacts the stability and accuracy of the resulting parameters. Two principal approaches are distinguished (Gurrieri et al., 2009, p. 16):

1. Local (Subset) Calibration: Calibration is restricted to a limited set of co-terminal swaptions (e.g., those maturing at 10Y or 20Y). This reduces the dimension of the optimization problem and allows for good fits on selected instruments, but can result in instability and poor performance on non-calibrated instruments, especially if time-dependent mean reversion is used (Gurrieri et al., 2009, pp. 16–24).

2. Global (Full Matrix) Calibration: The entire swaption matrix is used in the calibration procedure. This approach provides superior fitting quality across the full market surface and stabilizes the estimation of time-dependent parameters. When functional forms for $a(t)$ and $\sigma(t)$ are used in combination with global calibration, the model can achieve robust and accurate fits without evidence of overfitting. This thesis employs this global strategy for empirical analyses (Gurrieri et al., 2009, pp. 24–29).

The actual calibration of the Hull-White model is performed by minimizing the discrepancy between observed market data and model-implied quantities (Alaya et al., 2021, p. 2). In practice, this is typically achieved by minimizing the squared differences between market-observed implied volatilities $\hat{\sigma}_{i,j}^{\text{market}}$ and the model-implied volatilities $\hat{\sigma}_{i,j}^{\text{model}}(\boldsymbol{\theta})$, where (i, j) index the swaption expiry and tenor combinations, and $\boldsymbol{\theta}$ denotes the vector of model parameters (e.g., the values of $a(t)$ and $\sigma(t)$). The optimization problem can be formulated as

$$\min_{\boldsymbol{\theta}} \sum_{(i,j)\in\mathcal{I}} w_{i,j} \left(\hat{\sigma}_{i,j}^{\text{model}}(\boldsymbol{\theta}) - \hat{\sigma}_{i,j}^{\text{market}}\right)^2, \tag{28}$$

where $\mathcal{I}$ denotes the set of index pairs corresponding to available market swaptions and $w_{i,j} \geq 0$ are weighting factors that may be used to emphasize certain instruments (e.g., highly liquid swaptions or those near the money) Alternatively, this objective can be defined in terms of option prices instead of implied volatilities, depending on the availability of reliable market data and numerical stability considerations. The choice of objective function should reflect both the intended use of the model and the data quality. This minimization problem can be solved by algorithms suitable for solving nonlinear least-squares problems such as the Levenberg-Marquardt algorithm (see Section 2.4.6).

Understanding how the parameters affect model output is essential. The volatility function $\sigma(t)$ predominantly controls the *level* of the implied volatility surface (Gurrieri et al., 2009, p. 9), while the mean reversion $a(t)$ influences its *shape* (Gurrieri et al., 2009, p. 9).

Specifically, increasing $a(t)$ generally leads to lower implied volatilities and a change in curvature. Accurately capturing market phenomena such as volatility humps necessitates time-dependent mean reversion.

2.4.6 Levenberg-Marquardt Algorithm

The Levenberg-Marquardt algorithm is a widely used method for solving nonlinear least-squares problems, particularly effective when estimating model parameters in contexts such as the calibration of interest rate models. It aims to minimize the sum of squared residuals between observed data points and the corresponding model predictions (Marquardt, 1963).

Given a nonlinear model $f(x;\beta)$ with parameters $\beta = (\beta_1, \ldots, \beta_k)$, and observations $\mathbf{y} = (y_1, \ldots, y_n)$, the objective is to minimize the cost function

$$\Phi(\beta) = \sum_{i=1}^{n} (y_i - f(x_i;\beta))^2 = \|y - \hat{y}(\beta)\|^2. \tag{29}$$

The algorithm interpolates between two classical optimization strategies:

1. Steepest descent method: Effective when far from the minimum but may converge slowly near it.

2. Gauss-Newton method: Efficient near the minimum but may diverge in highly nonlinear regions.

The Levenberg-Marquardt method combines these approaches by updating the parameter vector β iteratively. In each iteration, the nonlinear model is linearized via a first-order Taylor expansion:

$$\hat{y}(\beta + \delta) \approx \hat{y}(\beta) + J\delta, \tag{30}$$

where J is the Jacobian matrix of partial derivatives with elements $J_{ij} = \frac{\partial f(x_i;\beta)}{\partial \beta_j}$ evaluated at the current parameter vector. The correction vector δ is then determined by solving the following regularized normal equation:

$$(J^\top J + \lambda I)\delta = J^\top (y - \hat{y}(\beta)), \tag{31}$$

where λ is a damping parameter, and I is the identity matrix.

The damping parameter λ controls the balance between the Gauss-Newton and steepest descent directions. For large λ, the algorithm behaves like a steepest descent method: (λI) dominates the system matrix, ensuring stability. For small λ, the method approaches the Gauss-Newton direction: $J^\top J$ dominates, allowing for faster convergence.

The value of λ is adjusted dynamically during the iteration process: If the new parameter vector $\beta+\delta$ results in a lower cost function Φ, the step is accepted and λ is decreased. If the step does not yield improvement, λ is increased, making the algorithm more conservative.

To improve numerical stability and convergence behavior, especially when model parameters have different magnitudes, parameter scaling is often applied. This ensures that step sizes and gradients are appropriately balanced.

Due to its adaptive nature and robustness, the Levenberg-Marquardt algorithm is particularly well suited for the calibration of the Hull-White model, where the objective function is highly nonlinear and sensitive to changes in model parameters.

2.5 Neural Networks

Neural networks are computational models inspired by the structure and function of the human brain and its nervous system. Unlike traditional computing systems that operate through explicit programming and deterministic logic, neural networks are designed to learn from data by adjusting internal parameters in response to input stimuli. This adaptive capability enables them to approximate complex, non-linear functions and has led to their widespread use in fields such as image recognition, natural language processing, and financial forecasting.

2.5.1 Types of Problems Solved by Neural Networks

Neural networks have proven to be powerful and versatile tools for solving a wide variety of computational problems. While they are traditionally associated with regression and classification tasks, modern neural architectures are capable of addressing a broader spectrum of problems across supervised, unsupervised, and reinforcement learning paradigms. This section provides an overview of the principal problem types solvable by neural networks.

Regression Problems: In regression tasks, the goal is to predict continuous numerical values based on input features. Neural networks learn a mapping from input vectors to a real-valued output, typically by minimizing a loss function such as the mean squared error (Goodfellow et al., 2016, p. 99). Applications include time series forecasting, energy demand prediction, and asset price estimation.

Classification Problems: Classification involves assigning input data to one or more discrete categories (Goodfellow et al., 2016, p. 98). Neural networks trained for classification tasks typically use softmax activation in the output layer and categorical cross-entropy as a loss function. Common applications include image recognition, sentiment analysis, fraud detection, and medical diagnosis.

Clustering: Clustering is an unsupervised learning problem where the aim is to group similar data points based on inherent structure in the data (Aljalbout et al., 2018). While traditional algorithms like k-means are commonly used, neural network-based

methods—such as autoencoders, self-organizing maps, and deep clustering networks—can learn more flexible and data-adaptive representations for clustering tasks.

Dimensionality Reduction: Dimensionality reduction seeks to project high-dimensional data into a lower-dimensional space while preserving meaningful structure. Neural networks can perform this task using autoencoders, which compress the data into a latent representation and then reconstruct it (Hinton & Salakhutdinov, 2006). Such techniques are widely used in data visualization, noise reduction, and feature extraction.

Reinforcement Learning: In reinforcement learning (RL), agents learn to take actions in an environment to maximize a long-term reward signal (Sutton & Barto, 2015, pp. 2–3). Neural networks serve as function approximators for value functions or policies in deep reinforcement learning frameworks such as Deep Q-Networks (DQNs) and Actor-Critic methods. RL applications include robotics control, autonomous driving, game-playing agents, and resource optimization.

2.5.2 Architecture

At the fundamental level, the basic unit of a neural network is referred to as a *neuron* or *node*. Each neuron receives one or more input signals, which are combined using a set of learnable parameters called *weights* (Mienye & Swart, 2024, p. 4). These weighted inputs are summed and passed through a non-linear transformation known as an *activation function*, which determines the neuron's output (Mienye & Swart, 2024, pp. 4–5). The activation function introduces non-linearity into the model, enabling the network to learn complex patterns in the data. Commonly used activation functions in neural networks include the sigmoid function, the hyperbolic tangent (*tanh*), and the Rectified Linear Unit (ReLU). Each of these functions introduces non-linearity into the model, enabling the network to capture complex patterns in the input data. The sigmoid and *tanh* functions are smooth, bounded, and differentiable, making them suitable for problems requiring normalized outputs. In contrast, the ReLU function, defined as $f(x) = \max(0, x)$, is computationally efficient and mitigates the vanishing gradient problem, thereby facilitating the training of deep neural networks.

Neural networks are typically organized into a series of layers (Mienye & Swart, 2024, p. 4):

Input Layer: The input layer is the first layer of the neural network and serves solely as the interface for receiving external data. Each neuron in this layer corresponds to a single feature or variable in the input vector. Importantly, neurons in the input layer do not perform any computation; instead, they transmit the raw input signals to the subsequent layer in the network.

Hidden Layer(s): Hidden layers are situated between the input and output layers and consist of neurons that perform intermediate computations. These layers are termed "hidden" because they are not directly observable from the input or output. The primary function of hidden layers is to transform the input data into more abstract representations through successive linear and non-linear operations. Networks with more than one hidden layer are referred to as deep neural networks, which can model highly intricate data

relationships. Each neuron in a hidden layer is typically fully connected to all neurons in the preceding and succeeding layers, although sparse (partially connected) architectures are also used.

Output Layer: The output layer is the final layer in the network and produces the model's predictions. The structure and activation function of this layer are typically chosen based on the nature of the problem—for example, a softmax activation function is used for multi-class classification, while a linear activation may be used for regression tasks.

The design of a neural network—such as the number of hidden layers, the number of neurons per layer, and the choice of activation functions—determines its capacity to learn and generalize from data. Proper configuration and training of these networks are essential to achieving optimal performance on a given task (Sazli, 2006).

2.5.3 Learning in Neural Networks: The Back-Propagation Algorithm

A defining characteristic of artificial neural networks is their ability to *learn* and *adapt* based on interactions with their environment. In this context, learning is defined as the process through which the free parameters (i.e., synaptic weights) of the network are adjusted to minimize a discrepancy between the actual output and the desired target output. The nature of this adjustment process depends on the learning algorithm employed.

Among the various learning algorithms, the *back-propagation algorithm* is the most widely used method for training feed-forward neural networks. It operates under the framework of supervised learning, where the correct output (label) is known for each training example. The algorithm iteratively minimizes a loss function by updating weights in the direction of the negative gradient of the error.

Let $y_j(n)$ denote the actual output of neuron j at iteration n, and let d_j be the corresponding desired output. The difference between the desired and actual output is quantified using a measure known as the *error energy*. For each neuron j in the output layer, the error signal $e_j(n)$ is defined as:

$$e_j(n) = d_j - y_j(n) \tag{32}$$

The *instantaneous error energy* for neuron j at iteration n, denoted by $\epsilon_j(n)$, is calculated as half the square of the error signal:

$$\epsilon_j(n) = \frac{1}{2} e_j^2(n) \tag{33}$$

This squared formulation ensures that the error energy is always non-negative and penalizes larger deviations more strongly. To evaluate the total discrepancy for the entire output layer Q, the *total error energy* $\epsilon(n)$ is computed as the sum of the individual error energies:

$$\epsilon(n) = \sum_{j \in Q} \epsilon_j(n) \tag{34}$$

Minimizing this total error energy is the central objective of the back-propagation learning algorithm, which iteratively adjusts the network's synaptic weights to reduce the mismatch between predicted and target outputs. The error signal $e_j(n)$ for neuron j is defined as:

$$e_j(n) = d_j - y_j(n) \tag{35}$$

The instantaneous error energy $\epsilon_j(n)$ associated with neuron j is then given by:

$$\epsilon_j(n) = \frac{1}{2}e_j^2(n) \tag{36}$$

The total instantaneous error energy $\epsilon(n)$ over all neurons $j \in Q$ in the output layer Q is computed as:

$$\epsilon(n) = \sum_{j \in Q} \epsilon_j(n) \tag{37}$$

To reduce the total error energy $\epsilon(n)$, the synaptic weights $w_{ji}(n)$ are updated using the gradient descent method. The weight update rule is expressed as:

$$\Delta w_{ji}(n) = -\eta \frac{\partial \epsilon(n)}{\partial w_{ji}(n)} \tag{38}$$

where η is the learning rate, a small positive constant controlling the step size.

To compute the partial derivative $\frac{\partial \epsilon(n)}{\partial w_{ji}(n)}$, the chain rule is applied:

$$\frac{\partial \epsilon(n)}{\partial w_{ji}(n)} = \frac{\partial \epsilon(n)}{\partial e_j(n)} \cdot \frac{\partial e_j(n)}{\partial y_j(n)} \cdot \frac{\partial y_j(n)}{\partial w_{ji}(n)} \tag{39}$$

Each term in this product is derived as follows:

$$\frac{\partial \epsilon(n)}{\partial e_j(n)} = e_j(n) \tag{40}$$

$$\frac{\partial e_j(n)}{\partial y_j(n)} = -1 \tag{41}$$

$$\frac{\partial y_j(n)}{\partial w_{ji}(n)} = f'\left(\sum_{i=0}^{m} w_{ji}(n) y_i(n)\right) \cdot y_i(n) \tag{42}$$

where $f'(\cdot)$ is the derivative of the activation function of neuron j, and $y_i(n)$ is the output of neuron i in the preceding layer.

Substituting Equations 40, 41, and 42 into Equation 39 yields:

$$\frac{\partial \epsilon(n)}{\partial w_{ji}(n)} = -e_j(n) \cdot f'\left(\sum_{i=0}^{m} w_{ji}(n) y_i(n)\right) \cdot y_i(n) \tag{43}$$

Inserting this result into the weight update rule (Equation 38) gives the final expression for the synaptic weight adjustment:

$$\Delta w_{ji}(n) = \eta \cdot e_j(n) \cdot f' \left(\sum_{i=0}^{m} w_{ji}(n) y_i(n) \right) \cdot y_i(n) \tag{44}$$

This process of computing the output error, back-propagating it through the network, and updating the weights is performed iteratively for each training sample. Over time, the network's parameters converge such that the output increasingly approximates the desired targets, thereby enabling the network to generalize and make accurate predictions on unseen data. This iterative optimization process is typically continued until the total error $\epsilon(n)$ reaches an acceptably low threshold or a maximum number of training epochs is completed (Sazli, 2006).

2.5.4 The Hyperband Algorithm for Hyperparameter Optimization

The performance of neural networks depends critically on the choice of hyperparameters, such as learning rate, batch size, number of layers, or regularization strength. Since these hyperparameters are not learned from data, their optimal configuration must be determined externally through hyperparameter optimization. Traditional methods such as grid search or random search are computationally inefficient, particularly when training deep neural networks. The *Hyperband* algorithm, introduced by Li et al. (2017), provides an efficient, theoretically grounded framework for hyperparameter selection based on the principle of adaptive resource allocation and early stopping.

Let $\mathcal{H}$ denote the hyperparameter space, and let $f : \mathcal{H} \to \mathbb{R}$ be a performance function (e.g., validation loss or error) that maps a hyperparameter configuration $h \in \mathcal{H}$ to its corresponding validation loss after training with a finite computational budget. The goal is to find

$$h^* = \arg\min_{h \in \mathcal{H}} f(h), \tag{45}$$

subject to a total computational budget B_{total}.

Hyperband frames this optimization as a *multi-armed bandit* problem, in which each configuration corresponds to an arm, and computational resources correspond to the number of pulls allocated to each arm. The algorithm dynamically balances *exploration* (testing many configurations) and *exploitation* (allocating more resources to promising configurations).

The core component of Hyperband is the *Successive Halving* (SH) procedure. Suppose n configurations are sampled uniformly at random from $\mathcal{H}$ and each is trained for an initial budget r (e.g., number of epochs or training samples). Their performances $\{f_i\}_{i=1}^{n}$ are evaluated, and only the top fraction $\frac{1}{\eta}$ is retained, where $\eta > 1$ is the *reduction factor*. The surviving configurations are then trained for η times the previous budget, and the process repeats. Formally, the iterative process for SH can be summarized as:

$$n_j = \left\lfloor n \cdot \eta^{-j} \right\rfloor, \tag{46}$$

$$r_j = r \cdot \eta^j, \tag{47}$$

for stages $j = 0, 1, \ldots, s_{\max}$, where n_j is the number of configurations and r_j is the resource budget per configuration at stage j. At each stage, the best $\lfloor n_j/\eta \rfloor$ configurations are promoted.

While Successive Halving requires a predefined (n, r) pair, Hyperband systematically explores different trade-offs between the number of configurations n and the allocated resources r. Specifically, it defines multiple *brackets* indexed by $s \in \{0, 1, \ldots, s_{\max}\}$, where each bracket represents a different allocation strategy.

For each bracket s, Hyperband computes:

$$s_{\max} = \left\lfloor \log_\eta R \right\rfloor, \tag{48}$$

$$n_s = \left\lceil \frac{s_{\max} + 1}{s + 1} \eta^s \right\rceil, \tag{49}$$

$$r_s = \frac{R}{\eta^s}, \tag{50}$$

where R denotes the maximum allowable resource per configuration (e.g., maximum number of training epochs).

Each bracket executes a full Successive Halving run with its respective (n_s, r_s) values. This design enables Hyperband to balance between two extremes:

- Wide exploration (large n_s, small r_s): testing many configurations briefly.
- Deep exploitation (small n_s, large r_s): fully training fewer configurations.

The total computational cost per bracket is approximately constant, ensuring that the overall budget B_{total} is respected:

$$B_{\text{total}} \approx (s_{\max} + 1)R. \tag{51}$$

The Hyperband procedure can be summarized as follows:

1. For each bracket $s = s_{\max}, s_{\max} - 1, \ldots, 0$:
   (a) Sample n_s configurations $\{h_{s,i}\}_{i=1}^{n_s}$ uniformly from $\mathcal{H}$.
   (b) Run Successive Halving with initial resource r_s and reduction factor η.
   (c) Record the best-performing configuration h_s^* in each bracket.
2. Select the globally best configuration

$$h^* = \arg\min_s f(h_s^*). \tag{52}$$

The Hyperband algorithm enjoys strong theoretical guarantees under mild regularity assumptions. Li et al. (2018) show that it achieves an asymptotically optimal trade-off between the number of configurations and resources allocated, with expected simple regret bounded by

$$\mathbb{E}\left[f(h^*) - f^*\right] = \mathcal{O}\left(\sqrt{\frac{\log n}{B_{\text{total}}}}\right), \tag{53}$$

where $f^* = \min_{h \in \mathcal{H}} f(h)$ denotes the true optimal performance. This result demonstrates that Hyperband is provably more efficient than random search and grid search in expectation.

Hyperband provides a principled and computationally efficient framework for neural network hyperparameter optimization by combining random search with adaptive resource allocation. Its key strength lies in automatically balancing exploration and exploitation through multiple parallel Successive Halving procedures, resulting in near-optimal use of available computational resources without requiring manual tuning of training budgets.

Include descriptions of the following points: Layers Neurons Weights & Weights Updates Activation Functions Bias Kernel

2.6 Related Work

Alvarez et al. (2022) demonstrated that the prices generated by a one-factor Hull-White model which uses parameters calibrated by a NN which was trained to learn the inverse relationship between swaption prices and the model parameters α and σ were close to the observed market prices, suggesting that the NN is an effective method for accurately calibrating the model parameters.

Hernandez (2016) examines the use of artificial NNs to replace traditional global optimization methods in the calibration of the two-factor Hull-White model. While global optimization methods are better at avoiding local minima, they are slower than local optimizers. The proposed approach uses a NN to approximate the calibration function, shifting the computational effort to an offline training phase. This enables the actual calibration process to be performed in a fraction of the time compared to global optimizers. The results show that using NNs for calibration offers significant time savings, with the process being much faster and more consistent than traditional methods. However, the performance of the NN degrades over time, though this can be mitigated by periodically retraining the model.

Alaya et al. (2021) also investigate the use of NNs for the calibration of interest rate models, focusing on the G2++ model and the CIR intensity model. The paper proposes using Deep Calibration (DC) to improve this process, making calibration faster, more accurate, and easier to handle. Two main approaches for DC are presented: Indirect Deep Calibration (Indirect DC), which uses intermediate quantities like covariance matrices derived from market data, and Direct Deep Calibration (Direct DC), which uses market-observable zero-coupon (ZC) rate curves directly. Both methods demonstrate significant time savings, with Direct DC being particularly efficient due to its simplicity and lower computational costs. The paper also highlights the advantages of training on synthetic data, which allows for large datasets, stress-testing scenarios, and direct error measurement. The results show that DC significantly reduces calibration time compared to classical methods, with Direct DC being up to 7,600 times faster. Moreover, DC achieves good accuracy and is more resilient to noise in the data, outperforming traditional calibration methods in terms of global error across all parameters. The approaches also work well when applied to other models, such as the CIR intensity model. Overall,

the study demonstrates that DL-based DC methods offer a practical, efficient alternative for calibrating financial models, with potential applications in various financial contexts.

Moysiadis et al. (2019) focussed on the calibration of the mean reversion speed parameter in the one-factor Hull-White model and proposed a method in which a NN is utilized to learn the future movement of interest rates based on historical data. The primary aim was to extract the mean reversion parameter from the derivative of the function learned by the NN which approximates the future rate. The approach emphasizes the robustness of the model, particularly its ability to handle market turbulence. The results demonstrate that the NN-based method delivers mean reversion values comparable to those obtained using a rolling linear regression, a standard method.

3 Methodology

This chapter provides a detailed, step-by-step blueprint of the research design and analytical framework used to compare the traditional and machine learning-based calibration methods. It begins by describing the dataset, including the source and specifications of the swaption and interest rate data. Subsequently, it outlines the data preprocessing pipeline, covering the bootstrapping of zero-coupon curves and the feature engineering process designed to capture key market dynamics. The core of the chapter details the architecture of the neural network model, the hyperparameter optimization strategy, and the implementation of the traditional Levenberg-Marquardt algorithm. Finally, it defines the specific metrics for accuracy, speed, and robustness that will be used to conduct a rigorous and objective evaluation of both approaches.

3.1 Dataset

3.1.1 Time Period

The market data used for the calibration of the one-factor Hull–White model covers a continuous time span of 92 calendar days, from June 1, 2025, to August 31, 2025. Observations are available for each calendar day within this period; weekends and public holidays are not excluded. The dataset therefore represents a high-frequency sample of daily market conditions. All market snapshots were collected at a daily frequency, ensuring a consistent temporal resolution across the entire observation window.

3.1.2 Swaptions

The calibration is based on a comprehensive panel of European at-the-money (ATM) payer and receiver swaptions quoted in the Euro (EUR) market. Each daily market snapshot consists of a complete normal volatility surface containing 228 distinct volatility points, derived from combinations of option maturities and underlying swap tenors.

The set of option maturities includes: 1 month, 3 months, 9 months, 1 year, 2 years, 3 years, 4 years, 5 years, 6 years, 7 years, 8 years, 9 years, 10 years, 12 years, 15 years, 20 years, 25 years, and 30 years.

The corresponding underlying swap tenors are: 1 year, 2 years, 3 years, 4 years, 5 years, 7 years, 10 years, 12 years, 15 years, 20 years, 25 years, and 30 years.

The floating leg of each underlying swap resets on a semiannual (6-month) basis. Volatilities are quoted in normal (Bachelier) terms, expressed in basis points (bps) of the underlying swap rate, and represent the mid-market levels (average of bid and ask quotes). Only ATM swaptions are considered in the calibration.

All swaption data were obtained from Bloomberg using the `VCUB` function with `BVOL` as the source. Due to the university's Bloomberg license restrictions, direct data downloads were not permitted; therefore, the data were captured via screenshots and subsequently converted into Excel format for processing. Bloomberg employs the EUR OIS (ESTR) discounting curve, identified as `(514) EUR OIS (ESTR)`, and the EUR swap curve `(45) Euro` for the construction of the volatility cube. This ensures internal consistency between the volatility and yield curve data used in the Hull–White model calibration.

3.1.3 Swap Curves

The initial yield curves employed for the Hull–White model calibration are derived from the EUR (vs. 6M EURIBOR) Swap / Projection Curve, identified in Bloomberg as `45`. This curve represents a forward (projection) curve used to generate 6-month EURIBOR forward rates for pricing and valuation. The discounting is performed using the EUR OIS (ESTR) curve (`Bloomberg ID 514`), consistent with the multi-curve framework applied in current market practice.

The market instruments used in the bootstrapping of the projection curve include:

- Cash deposits: Overnight to 12-month maturities, anchoring the short end of the curve.
- Forward Rate Agreements (FRAs): Typically spanning 1×7 to 9×15 months, bridging the short and medium-term maturities.
- Interest-rate swaps: Par fixed-vs-6M EURIBOR swaps from 1 year up to 30 years, forming the long end of the term structure.

Day-count conventions follow market standards: ACT/360 for short and medium maturities and 30U/360 for the long end. All market quotes correspond to mid (bid/ask midpoint) levels. The market data source is the Bloomberg Generic (BGN) composite, which aggregates dealer contributions to provide representative market levels.

The curve is bootstrapped sequentially from cash deposits through FRAs to interest-rate swaps, producing a continuous zero-rate and forward-rate term structure. The interpolation is typically performed in a log-linear fashion on either discount factors or zero rates.

Within the Bloomberg volatility cube environment (`VCUB`), this curve appears as `Swap Curve (45) Euro` and provides consistent forward-rate inputs for the calibration of EUR 6M swaption volatilities.

3.1.4 MOVE Index

The MOVE index represents the Treasury market's implied volatility and serves as a primary external indicator of interest rate uncertainty. Daily open values of the MOVE index were collected from Yahoo Finance to capture the most up-to-date measure of market-implied interest rate volatility at the start of each trading day. By using the open rather than closing values, the neural network is able to predict Hull–White model parameters early in the day, leveraging the most recent available information without waiting for end-of-day data. The MOVE index is particularly relevant as it captures expected fluctuations in interest rates, which directly influence the calibration of the Hull–White model.

3.1.5 VIX Index

The VIX index provides a widely recognized measure of equity market implied volatility and serves as a general gauge of market fear and risk appetite. Daily open values of the VIX were downloaded from Yahoo Finance to reflect the initial market sentiment for each trading day. High VIX levels, indicative of a "risk-off" environment, can have spillover effects on bond and swap markets, thereby indirectly affecting European interest rates. Including VIX open values in the dataset allows the neural network to incorporate contemporaneous equity market volatility into the prediction of Hull–White parameters, potentially improving model responsiveness to market stress conditions.

3.1.6 EUR/USD Exchange Rate

The EUR/USD exchange rate reflects the relative economic and monetary conditions between the Eurozone and the United States. Daily open prices were obtained from Yahoo Finance to capture the most current cross-currency market information at the start of each trading day. Large movements in the EUR/USD rate may indicate capital flows or diverging monetary policy actions between the European Central Bank and the Federal Reserve, which have direct implications for European interest rates. By including the open EUR/USD rate in the feature set, the neural network can incorporate up-to-date FX market dynamics when predicting Hull–White model parameters.

3.2 Descriptive Analysis of Market and Model Inputs

This section provides a detailed examination of the key figures illustrating the underlying financial data and model inputs used in this study. Each figure is analyzed with respect to its economic interpretation and its implications for the modeling framework.

Figure 1: Time Series of Market-Wide Indicators (VIX, MOVE, and EUR/USD)

Figure 1 depicts the evolution of key market indicators between June and late August 2025. The overall market environment during this period is characterized by a notable decrease in volatility across major asset classes. The *VIX Index*, representing equity market volatility, begins near a relatively elevated level of 20 before declining sharply in late June and stabilizing within a range of 14–17. This pattern suggests a reduction in perceived equity market risk. Similarly, the *MOVE Index*, a measure of Treasury market volatility, exhibits an even more pronounced decline—from approximately 100 to below 80—indicating a substantial calming of interest rate expectations. Concurrently, the *EUR/USD* exchange rate shows a strengthening of the Euro against the U.S. Dollar, peaking near 1.18 in mid-July before a modest correction. These co-evolving patterns

reflect a broad reduction in cross-asset risk and uncertainty.

Table 3: Summary Statistics of External Market Factors

Factor	Mean	Median	Std. Dev.	Skewness	Kurtosis
VIX Index	17.05	16.71	1.89	0.89	0.34
MOVE Index	86.29	86.02	6.12	0.10	-1.10
EUR/USD Rate	1.162	1.164	0.011	-0.57	-0.54

A summary of the key external market factors over the analysis period. The positive skew in the VIX index and the non-zero kurtosis across all factors suggest deviations from a normal distribution.

Table 3 provides a quantitative summary of the external market factors depicted in Figure 1. The statistics confirm the visual analysis, detailing the central tendency and dispersion of each series. Notably, the VIX Index exhibits significant positive skewness (0.89), indicating that upward spikes in volatility are more pronounced than downward movements. Conversely, the EUR/USD rate shows a negative skew (-0.57). The kurtosis values, which deviate from the zero value of a mesokurtic distribution, provide further numerical evidence that these financial time series are not normally distributed, a crucial consideration for the modeling approach.

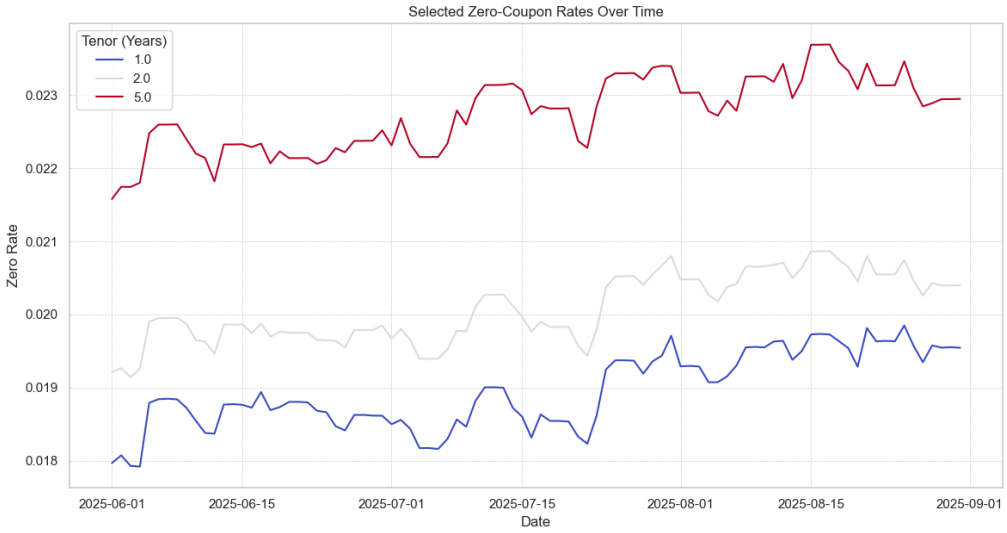

Figure 2: Yield Curve Dynamics for Selected Tenors (1Y, 2Y, 5Y)

Figure 2 illustrates the time series of yields across selected maturities. The yield curve retains its normal, upward-sloping shape throughout the observation window. A gradual upward shift is visible across all tenors, indicating a moderate tightening of monetary conditions. The largely parallel movement across maturities implies that changes are predominantly driven by a "level" factor. Minor deviations in spreads, however, suggest that "slope" and "curvature" components contribute modestly to the curve's dynamics.

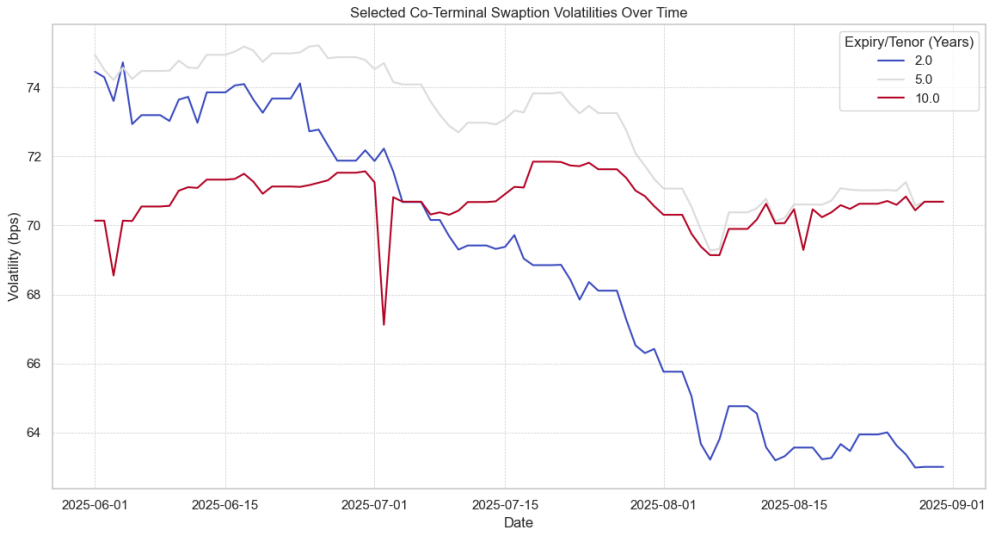

Figure 3: Time Series of Co-Terminal Swaption Volatilities

Figure 3 presents the evolution of the co-terminal swaption volatility term structure. The curve is generally downward sloping, with shorter maturities exhibiting higher volatility levels than longer maturities. The short-term (2-year) volatility declines sharply from over 74 bps to approximately 63 bps, whereas the long-term (10-year) volatility remains comparatively stable. This differential behavior leads to a pronounced flattening of the volatility term structure—a key empirical feature that any robust model should accurately capture.

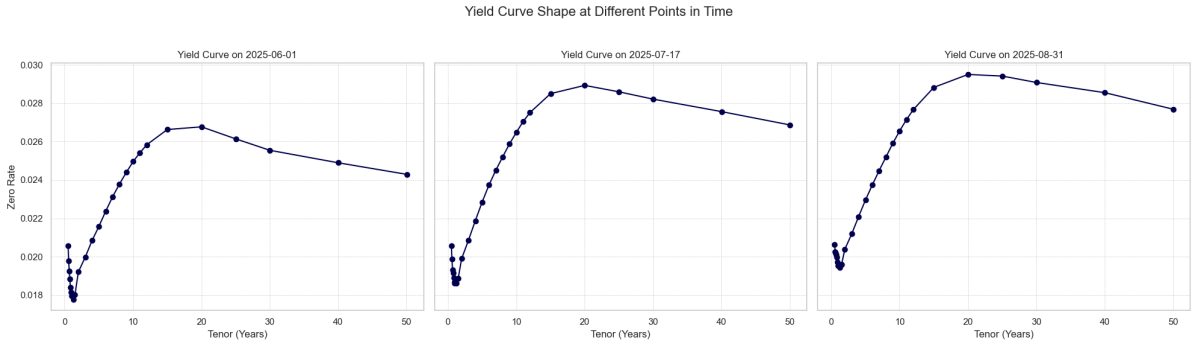

Figure 4: Yield Curve Snapshots over Time

Figure 4 visualizes static yield curve snapshots at different points in time. The yield curve consistently displays a concave, upward-sloping shape, typical for stable economic environments. The entire curve shifts upward from June to August, reinforcing the observed trend of rising interest rates. Moreover, the curvature appears to increase slightly over time, suggesting subtle changes in the underlying term structure dynamics.

Table 4: Descriptive Statistics for Key Yield Curve Tenors

Tenor	Mean Rate (%)	Std. Dev. (%)	Skewness	Kurtosis
1-Year	1.90	0.05	-0.02	-1.10
5-Year	2.27	0.05	-0.18	-0.88
10-Year	2.62	0.06	-0.25	-1.04
30-Year	2.76	0.12	-0.29	-1.22

Descriptive statistics for selected benchmark tenors of the zero-coupon yield curve. Mean Rate and Standard Deviation are expressed in percentage points.

Table 4 quantifies the characteristics of the yield curve by presenting descriptive statistics for key benchmark tenors. The increasing mean rate with tenor numerically confirms the upward-sloping nature of the term structure. A particularly important finding is the behavior of the standard deviation, which increases from 0.05% for the 1-year rate to 0.12% for the 30-year rate. This demonstrates that the long end of the yield curve is substantially more volatile in absolute terms, a stylized fact that is further explored in Figure 6.

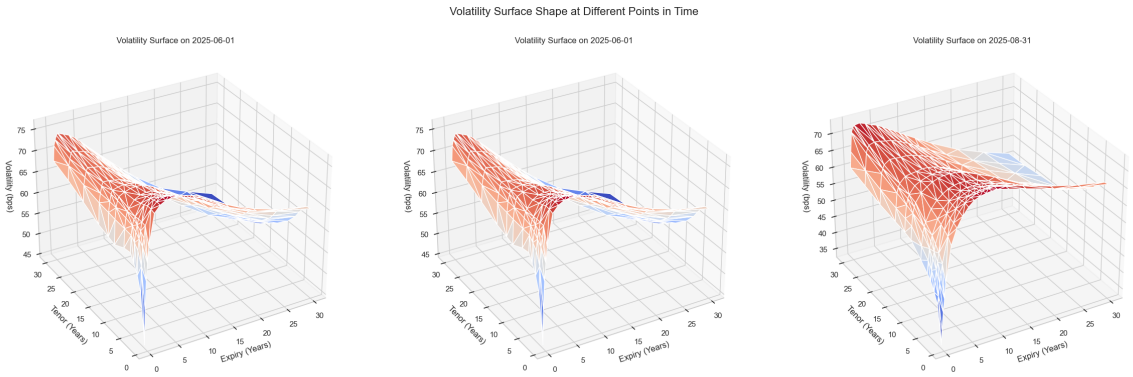

Figure 5: 3D Representation of the Swaption Volatility Surface

Figure 5 provides a three-dimensional representation of the swaption volatility surface. The surface exhibits pronounced structure, with the highest volatilities concentrated in short-expiry, short-tenor instruments. Volatility decreases along both the expiry and tenor dimensions, forming a characteristic "hump" at the short end. This structural complexity underscores the necessity of employing non-linear modeling approaches—such as neural networks—to accurately capture the surface's intricate shape and temporal dynamics.

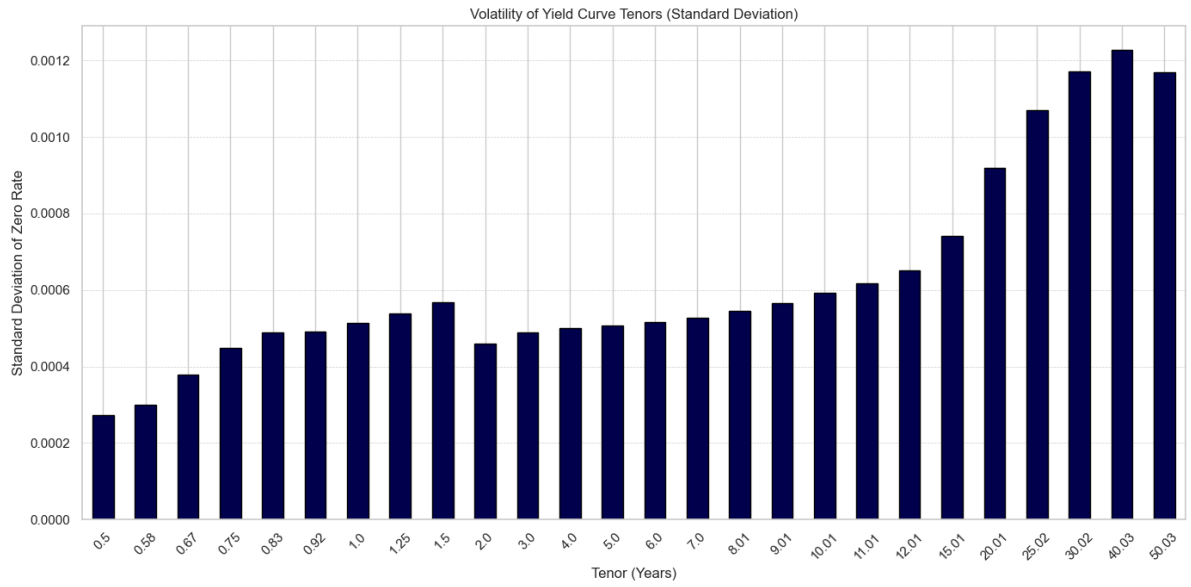

Figure 6: Standard Deviation of Daily Yield Changes by Tenor

Figure 6 shows the distribution of yield volatility across maturities. Volatility is lowest at the short end, increases moderately in the 1–2 year range, and rises significantly for longer maturities beyond 10 years. This pattern highlights that long-term yields exhibit the greatest absolute variability, a crucial consideration for risk management and model calibration.

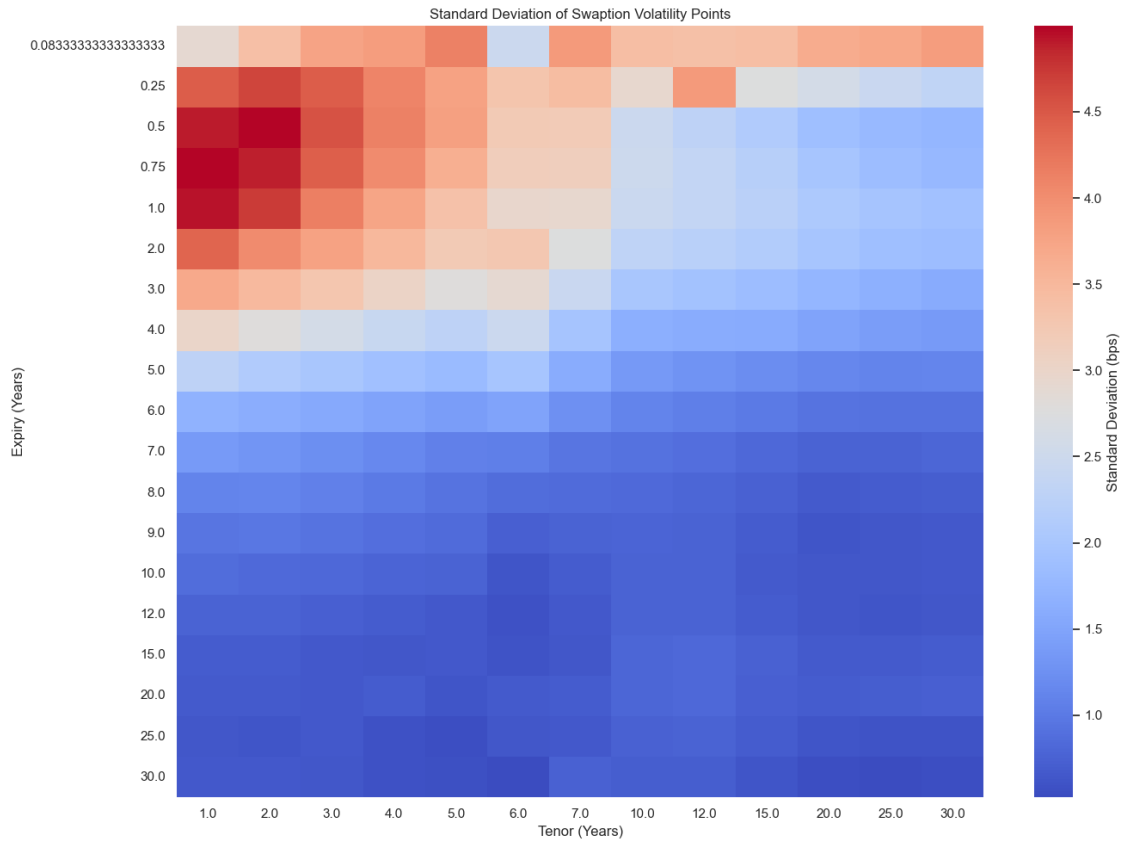

Figure 7: Heatmap of Swaption Volatility Surface Standard Deviations

Figure 7 displays the standard deviation of swaption volatilities across expiry and tenor dimensions. The highest volatility—represented by the most intense red shading—is concentrated in the short-expiry, short-tenor region. This indicates that the front end of the surface is particularly unstable, while the long end remains relatively static. A well-performing model must therefore excel at capturing the pronounced temporal variability in this region.

Table 5: Standard Deviation of Swaption Volatilities (in basis points)

Expiry	**1-Year Tenor**	**5-Year Tenor**	**10-Year Tenor**	**30-Year Tenor**
1-Year	4.93	3.37	2.45	1.92
5-Year	2.28	1.82	1.35	1.12
10-Year	0.86	0.75	0.76	0.65
30-Year	0.66	0.56	0.71	0.54

Standard deviation of daily swaption volatilities, measured in basis points (bps). The table shows the absolute volatility for key points on the expiry-tenor grid.

Table 5 provides a quantitative counterpart to the heatmap in Figure 7, detailing the magnitude of instability across the volatility surface. The data confirms that the highest volatility is located at the front end, with the 1-year expiry, 1-year tenor swaption

exhibiting a standard deviation of 4.93 bps. This value is nearly an order of magnitude larger than the 0.54 bps standard deviation observed for the 30-year expiry, 30-year tenor swaption. This sharp gradient in instability underscores the modeling challenge: the neural network must learn to produce highly dynamic outputs for the front end of the surface while generating stable outputs for the back end.

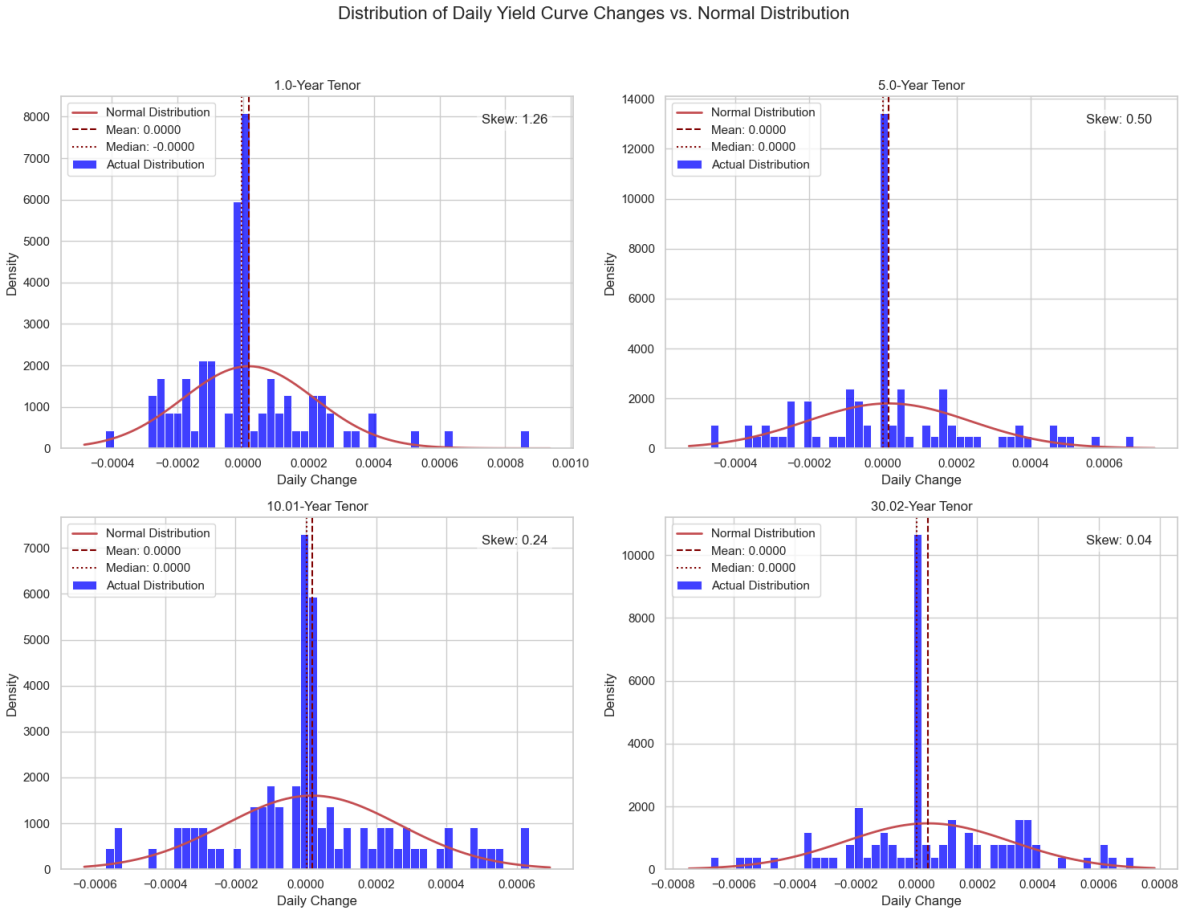

Figure 8: Histograms of Daily Yield Changes by Tenor

Figure 8 compares empirical histograms of daily interest rate changes with theoretical normal distributions. All tenors display leptokurtic distributions with fat tails, implying that extreme movements occur more frequently than predicted by Gaussian models. The 1-year tenor exhibits pronounced positive skewness (1.26), suggesting a tendency toward larger upward rate movements. Skewness gradually declines for longer maturities. These findings confirm that interest rate changes deviate substantially from normality.

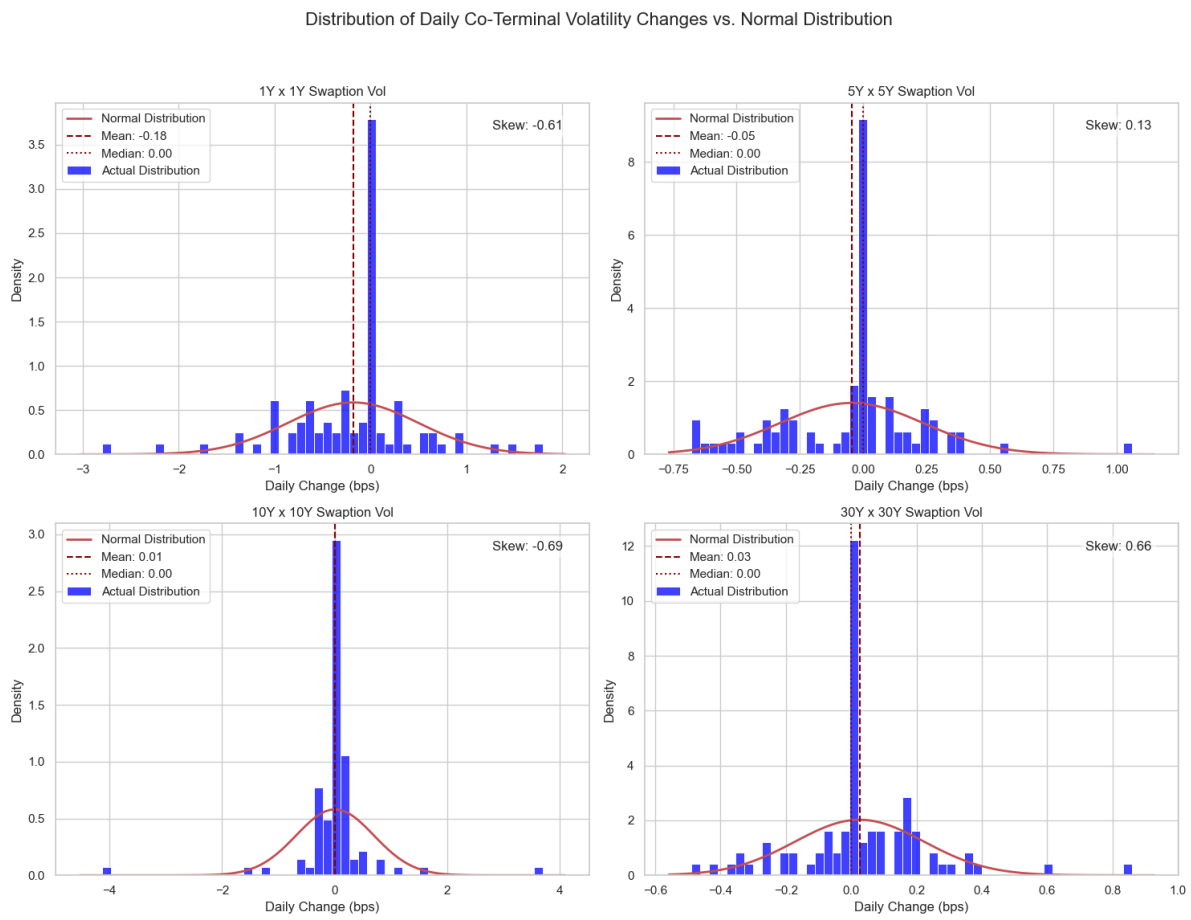

Figure 9: Histograms of Daily Swaption Volatility Changes

Figure 9 presents the distribution of daily changes in swaption volatilities. Similar to yield changes, the data exhibit significant leptokurtosis and fat tails, confirming the non-normal nature of the underlying process. A mild positive skew is particularly evident in the 5-year tenor (skew = 0.50), indicating that large volatility spikes are more frequent than declines. Together with the findings from Figure 8, this evidence supports the use of non-parametric, data-driven models—such as neural networks—that can flexibly capture these empirical deviations from normality.

3.3 Data Preprocessing

3.3.1 Bootstrapping Zero-Curves

The implementation of the one-factor Hull–White model in `QuantLib` requires zero-coupon curves as input for both discounting and forward-rate generation. Consequently, the EUR swap curves obtained from Bloomberg were bootstrapped into continuous zero-rate term structures following the procedure outlined in subsection 2.2. This ensures

compatibility with the `QuantLib` framework and provides a consistent foundation for pricing swaptions and calibrating the model parameters.

3.3.2 Handling Non-Trading Days in External Data

The external market data obtained from Yahoo Finance, including the MOVE index, VIX index, and EUR/USD exchange rate, is only available for trading days. In contrast, the swaption and swap curve datasets contain values for all calendar days, including weekends and public holidays. To ensure temporal consistency across all features used for Hull–White model calibration, the external data were reindexed to cover the entire daily date range of the analysis period.

Missing values corresponding to non-trading days were filled using a combination of forward-filling and backward-filling. Forward-filling propagates the most recent available value to subsequent missing days, while backward-filling assigns the nearest subsequent available value to preceding missing days. This procedure ensures that every day within the observation window has a corresponding value for each external variable, resulting in a continuous dataset that can be used by the neural network without introducing gaps or inconsistencies.

3.3.3 Feature Engineering for the Neural Network

In addition to the raw market data, several engineered features were created to enhance the neural network's ability to predict Hull–White model parameters. These features capture the shape of the yield curve, relationships between external market indicators, and underlying latent structures via dimensionality reduction. Each feature was designed with a clear economic or statistical justification for its relevance to the model parameters: the volatility σ and the mean-reversion speed a.

Features Engineered from the Yield Curve These features explicitly describe the shape and dynamics of the zero-coupon interest rate term structure. Let $\text{Rate}(T)$ denote the zero-coupon rate for a given tenor T (e.g., Rate(10Y) is the 10-year rate).

Slope Features:

$$\text{slope\_3m10y} = \text{Rate(10Y)} - \text{Rate(3M)} \tag{54}$$

$$\text{slope\_2y10y} = \text{Rate(10Y)} - \text{Rate(2Y)} \tag{55}$$

$$\text{slope\_5y30y} = \text{Rate(30Y)} - \text{Rate(5Y)} \tag{56}$$

A steep positive slope typically signals expectations of economic growth and future rate hikes, implying higher expected interest rate volatility (σ). Conversely, a flat or inverted

curve suggests a more stable rate environment and lower σ. Regarding mean reversion (a), a steep curve indicates strong anchoring to the long-term average, implying higher a, while an inverted curve signals weaker anchoring and lower a.

Curvature Features:

$$\text{curvature\_2y5y10y} = 2 \cdot \text{Rate(5Y)} - \text{Rate(2Y)} - \text{Rate(10Y)} \tag{57}$$

$$\text{curvature\_1y2y5y} = 2 \cdot \text{Rate(2Y)} - \text{Rate(1Y)} - \text{Rate(5Y)} \tag{58}$$

$$\text{curvature\_10y20y30y} = 2 \cdot \text{Rate(20Y)} - \text{Rate(10Y)} - \text{Rate(30Y)} \tag{59}$$

High curvature indicates significant uncertainty in medium-term rates, aiding prediction of the term structure of σ. Pronounced curvature suggests rates are expected to revert to a mean, implying higher a. Flat curvature indicates weaker mean reversion and lower a.

Features Engineered from External Market Data
Curvature-to-Slope Ratio:

$$\text{curvature\_slope\_ratio} = \frac{\text{curvature\_2y5y10y}}{\text{slope\_2y10y}} \tag{60}$$

This feature captures nuanced market regimes where slope and curvature provide complementary information. It allows the network to distinguish between different economic environments and predict both σ and a more accurately.

MOVE-to-VIX Ratio:

$$\text{MOVE\_VIX\_Ratio} = \frac{\text{MOVE\_Open}}{\text{VIX\_Open}} \tag{61}$$

A high ratio indicates stress concentrated in the bond market, signaling a higher σ specific to interest rates. It also implies uncertainty about central bank policy, potentially reducing a. This feature helps the network separate general market panics from fixed-income-specific volatility.

Features Engineered via Dimensionality Reduction Instead of feeding all individual zero-coupon rates directly into the neural network, the yield curve was transformed using Principal Component Analysis (PCA). The reason for this choice lies in the fact that individual rates on the curve are highly correlated — a move in the 5-year rate is almost always accompanied by similar movements in the 7-year and 10-year rates as visible in figure 10. Using these raw, correlated rates introduces several problems: multicollinearity, redundant information, and an increased risk of overfitting.

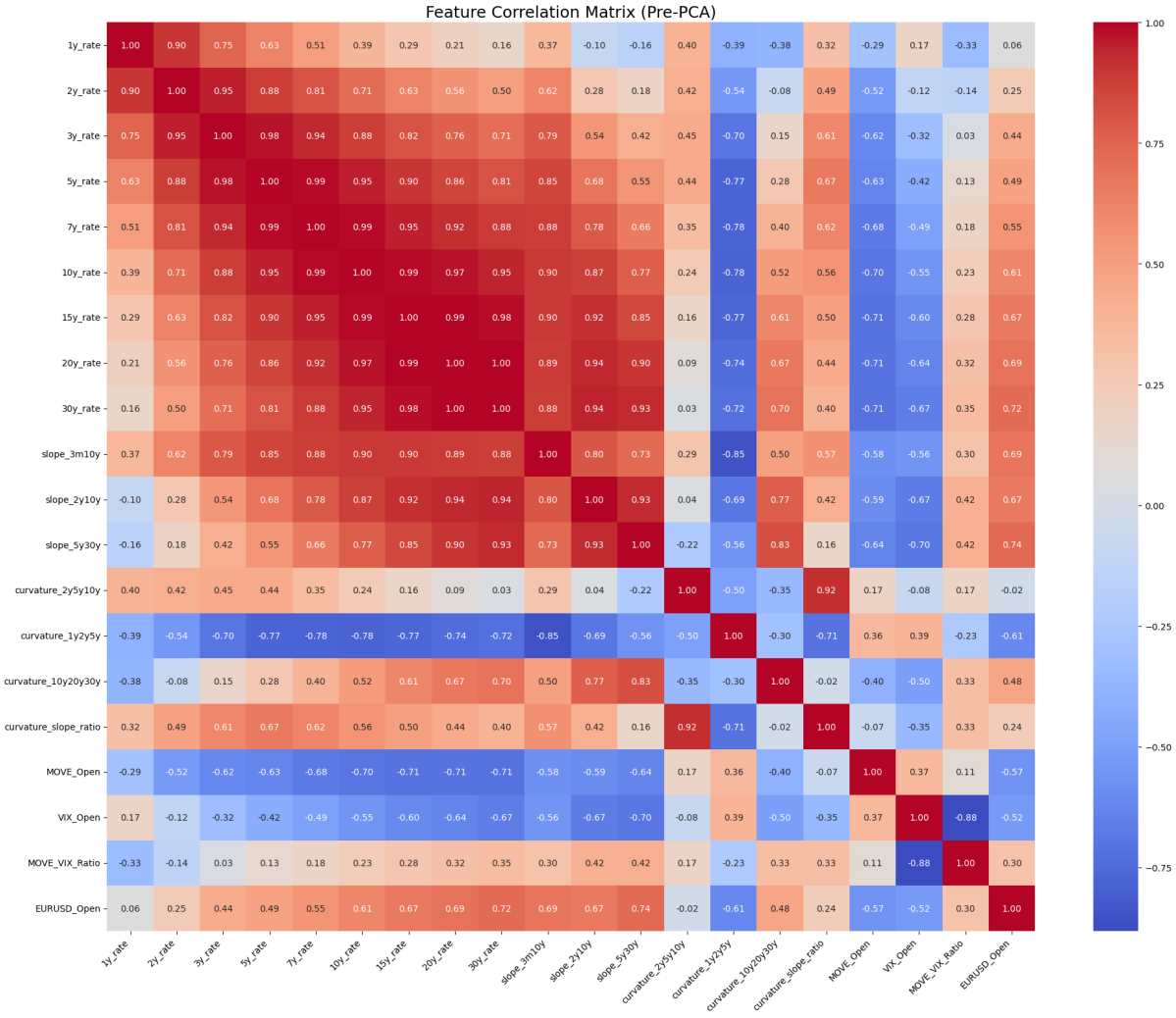

Figure 10: Correlation matrix of the engineered features before applying PCA.

Multicollinearity causes instability in the learned weights of the neural network. When input features contain overlapping information, the model struggles to assign stable importance to each feature. It may assign large, offsetting weights to features that move together, resulting in unstable and unreliable predictions (Chan et al., 2022, p. 2).

Redundant information makes the learning process inefficient and typically leads to poor test performance (Sildir et al., 2020, p. 3). Most daily movements in the yield curve are dominated by a single pattern — a parallel shift. Feeding the model all the raw rates forces it to re-learn this shared structure repeatedly instead of focusing on economically meaningful dynamics such as slope or curvature changes.

High dimensionality further increases the complexity of the model, encouraging overfitting. With too many correlated features, the neural network risks memorizing noise instead of generalizable relationships, which harms out-of-sample performance (Sildir et al., 2020, p. 3).

PCA directly resolves these issues by transforming the vector of rates

$$R = [\text{Rate}(1\text{Y}), \text{Rate}(2\text{Y}), \ldots, \text{Rate}(30\text{Y})]$$

into a set of uncorrelated and economically interpretable components:

$$\text{PC\_Level} = \mathbf{w}_1 \cdot \mathbf{R} \tag{62}$$

$$\text{PC\_Slope} = \mathbf{w}_2 \cdot \mathbf{R} \tag{63}$$

$$\text{PC\_Curvature} = \mathbf{w}_3 \cdot \mathbf{R} \tag{64}$$

These three "super-features" capture nearly all the meaningful variation of the yield curve and can be interpreted as follows (Rebonato, 2018, pp. 98–107):

- PC_Level represents the overall interest rate level and captures parallel shifts. Empirically, higher rate levels are associated with higher volatility (σ).
- PC_Slope measures the steepness of the curve and reflects economic expectations, influencing both σ and a.
- PC_Curvature captures the non-linear "bow" of the curve, aiding the model in learning term-structure effects in the piecewise σ and mean-reversion dynamics α.

After applying PCA to the engineered features, the resulting correlation matrix exhibits a significant reduction in the highest correlations. The originally strong dependencies between individual yield curve rates and other features are largely removed as visible in figure 11, demonstrating that the principal components provide a set of largely uncorrelated, independent features for the neural network.

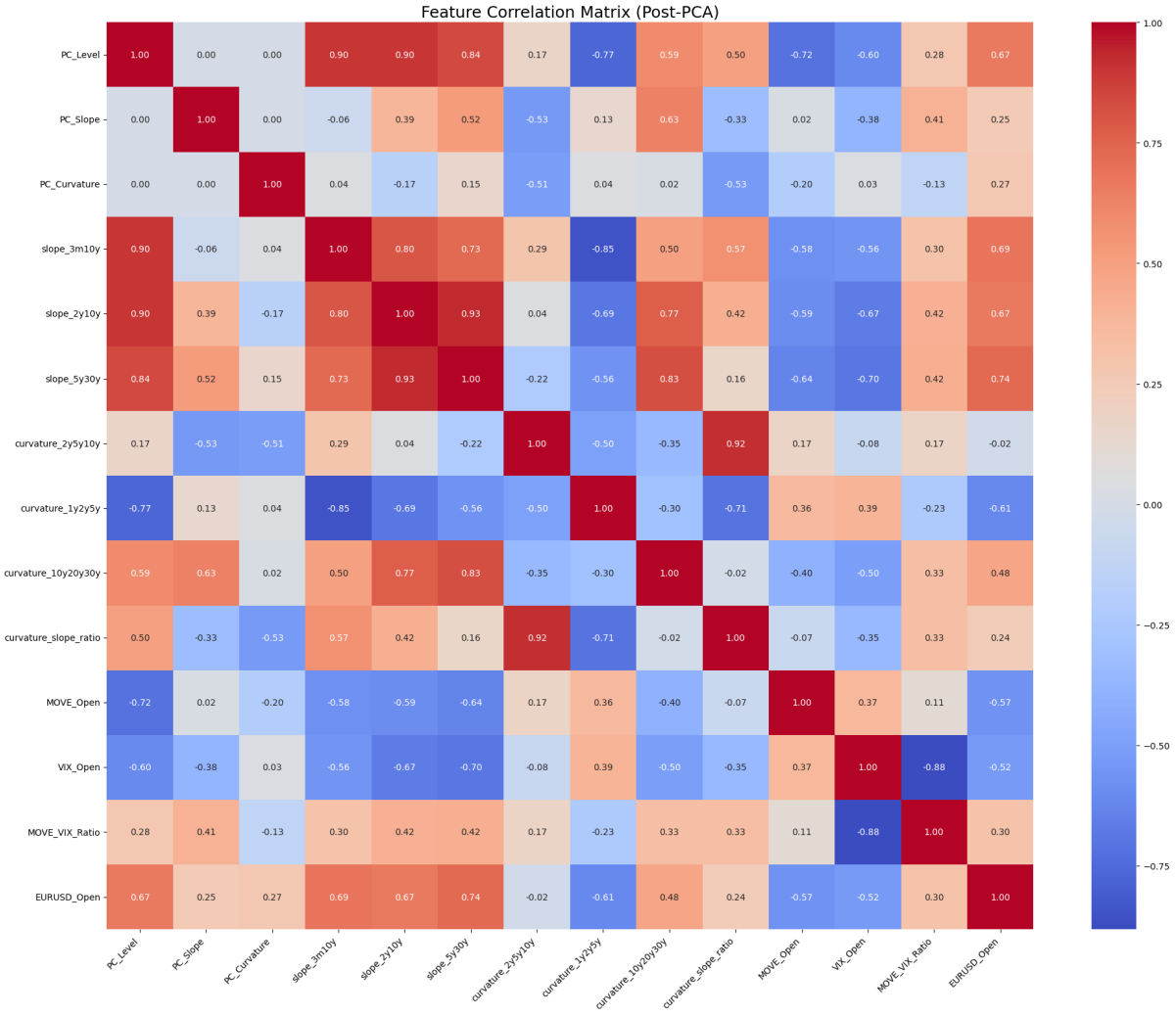

Figure 11: Correlation matrix of the engineered features after applying PCA.

By reducing the dimensionality from roughly ten correlated rate inputs to three independent components, PCA provides a stable, interpretable, and compact representation of the yield curve. This significantly improves the neural network's training efficiency, robustness, and generalization performance.

3.3.4 Feature Scaling

Prior to being fed into the neural network, all input features are scaled using the `StandardScaler` from the `scikit-learn` library. This process, also known as standardization or Z-score normalization, ensures that each feature has a mean of zero and a standard deviation of one (Zheng & Casari, 2018, pp. 31–32).

The scaling process consists of two main steps:

Step 1: Fit Phase (Learning from Training Data) The scaler analyzes the training data to learn the distribution of each feature. For each feature column, it calculates:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{65}$$

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2} \tag{66}$$

where x_i are the individual values of the feature and n is the number of training samples. The computed mean μ and standard deviation σ are stored in the scaler object.

Step 2: Transform Phase (Applying the Scaling) The learned parameters are used to transform all values of that feature in the training, validation, and test datasets:

$$z = \frac{x - \mu}{\sigma} \tag{67}$$

This centers the data around zero and scales it to unit variance.

From a computational perspective, normalization is necessary to manage issues arising from the inherent structure of raw data inputs. It is a technique widely used in statistics for making the units of variables comparable, which is critical when input data are expressed in very different units and exhibit disparate orders of magnitude. Without scaling, features with very large numerical values can dominate the error calculation, causing the error reduction algorithm to focus primarily on these variables while neglecting the information contained in smaller-valued features. Furthermore, when raw numerical values are extremely large, such as on the order of 10^6, their processing can generate outputs that exceed the capacity of standard computing hardware, making scaling essential to control for calculation and roundoff errors (Shanker et al., 1996).

The application of scaling directly addresses the optimization landscape of the training problem, leading to more efficient convergence. Normalizing the input is known to accelerate convergence during the optimization of linear models. Linear scale transformations, especially those that compress the search space, reduce the distance the backpropagation algorithm must cover in each iteration (Sola & Sevilla, 1997). This improved conditioning, where normalization helps ensure the covariance matrix of the layer input $\sum x$ is well-conditioned, is crucial for faster learning and aids the optimization algorithm in locating local minima (Huang et al., 2020). Consequently, an adequate normalization protocol can significantly reduce calculation and training time, with studies showing that estimation errors can be reduced by a factor of 5 to 10 and the time required to achieve such results reduced by an order of magnitude (Sola & Sevilla, 1997).

Ultimately, the primary result of standardization is an improvement in the neural network's quality and training stability. Data transformation is frequently used for improved learning, and normalization techniques are considered essential for accelerating the training and enhancing the generalization of deep neural networks (DNNs). Networks trained on standardized data generally yield better overall results, leading to a higher classification rate and a smaller Mean Square Error (MSE) in regression problems, where a smaller MSE indicates a higher quality solution (Shanker et al., 1996). The performance of a network, including the number of training iterations required and the final error

attained, is enhanced as the input variable ranges are 'equalized' by the normalization process. This contributes to scale-invariant learning, which is important for stabilizing training and has been shown to be useful for adaptively adjusting the learning rate. In some cases, standardization is not merely a best practice but a formal prerequisite due to specific algorithm requirements (Huang et al., 2020).

By applying `StandardScaler`, the neural network receives a well-conditioned, standardized input, which improves training stability, and convergence speed of the resulting model.

3.4 Hyperparameter Optimization

The predictive performance and stability of neural networks depend critically on an appropriate choice of hyperparameters. To identify an optimal configuration, a structured hyperparameter optimization process was conducted using the Hyperband algorithm, as described in Subsection 2.5.4. This algorithm efficiently allocates computational resources to explore a broad hyperparameter space while focusing on the most promising configurations.

The core settings for the Hyperband tuner were configured as follows:

- Maximum Epochs per Trial (`max_epochs`): A total of 1000 epochs were allocated as the maximum budget for training any single hyperparameter configuration.
- Reduction Factor (`factor`): A factor of 3 was used, meaning that after each round of training within a bracket, the number of configurations is reduced by a factor of 3, retaining only the top-performing third.
- Objective: The optimization objective was the minimization of the root mean squared error on the validation set.

The hyperparameters tuned in this study can be divided into two categories: those defining the neural network architecture and those related to the training process and loss function. Hyperparameters related to the architecture are:

- Number of Hidden Layers: This parameter determines the depth of the neural network. A deeper network has greater representational power and can capture more complex nonlinear relationships in the data, but it also increases the risk of overfitting and computational cost.
  Search Space: Integer between 1 and 5
- Number of Neurons per Layer: The width of each hidden layer, determined by the number of neurons, controls the capacity of the model. Larger layers allow for more detailed pattern recognition but can lead to overfitting if excessive.
  Search Space: Integer between 16 and 128, in steps of 16, for each hidden layer

- The activation function introduces non-linearity into the network, enabling it to model complex relationships between input features and outputs.
  Search Space: Choice between `relu` and `tanh`.

- Dropout is a regularization technique that randomly deactivates a fraction of neurons during training to prevent overfitting by discouraging co-adaptation among neurons.
  Search Space: Boolean choice between `True` and `False`.

- Dropout Rate: If dropout is used, this parameter determines the fraction of neurons dropped during each training iteration.
  Search Space: Floating-point value between 0.1 and 0.5 (only active if `use_dropout = True`) in steps of 0.1.

Tuneable hyperparameters which cover the training and the loss functions are:

- Learning Rate: The learning rate governs the step size used by the optimizer when updating the model's weights. A high learning rate may lead to instability or divergence, whereas a very low one results in slow convergence. To capture effective values across orders of magnitude, the search was conducted on a logarithmic scale.
  Search Space: Floating-point value between 0.0001 and 0.01, sampled logarithmically.

- Underestimation Penalty: This custom hyperparameter is designed to address the asymmetric cost of forecasting errors in financial volatility modeling. Underestimating volatility poses a greater financial risk than overestimating it. Therefore, this penalty increases the loss when the model underpredicts volatility, encouraging more conservative estimates.
  Search Space: Floating-point value between 0.5 (penalty for overestimation) and 4.0 (strong penalty) in steps of 0.1.

3.5 Loss Function and Error Metric

The objective of the neural network is to determine the optimal set of weights W and biases b such that the predicted Hull–White parameters minimize the discrepancy between model-implied swaption volatilities and their market-observed counterparts. The neural network acts as a function

$$\mathrm{NN} : x \mapsto \theta(x; W, b), \tag{68}$$

where $x \in \mathbb{R}^d$ is the input feature vector containing scaled and PCA-transformed market data (yield curve levels, slopes, curvatures, etc.) for a given day, and θ denotes the vector of Hull–White model parameters to be predicted. The network employs a residual architecture: instead of predicting θ directly, it predicts a deviation Δz from a fixed initial guess z_0:

$$\Delta z = \mathrm{NN}(x; W, b), \quad z = z_0 + \Delta z. \tag{69}$$

The predicted parameters are obtained by applying a scaled sigmoid to ensure they remain within a predefined range [0, U]:

$$\theta(x; W, b) = U \cdot \sigma(z) = U \cdot \frac{1}{1 + e^{-z}}. \tag{70}$$

The predicted parameters θ are subsequently used within the QuantLib library as a black-box function V_{QL}, which computes model-implied volatilities $\hat{\sigma}$ for a portfolio of n swaptions given the additional market data M_j on day j:

$$\hat{\sigma} = V_{\mathrm{QL}}(\theta, M_j) = \{\hat{\sigma}_1, \hat{\sigma}_2, \ldots, \hat{\sigma}_n\}. \tag{71}$$

The predictive accuracy of the network is quantified using the root mean squared error (RMSE), which measures the average magnitude of deviation between model-implied and market-observed volatilities, expressed in basis points (bps):

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{\sigma}_i - \sigma_i)^2} \times 10000. \tag{72}$$

Table 6: Notation used in the RMSE formula

Symbol	Meaning
n	Total number of swaptions in the evaluation dataset
$\hat{\sigma}_i$	Implied volatility predicted by the Hull–White model using the parameters generated by the neural network for the i-th swaption
σ_i	Market-observed implied volatility for the i-th swaption

To align the optimization with practical calibration objectives, an asymmetric loss function is employed during training. For each day j, the loss is computed as the mean of weighted squared errors between predicted and observed volatilities:

$$L_j(W, b) = \frac{1}{n}\sum_{i=1}^{n} w_i(\hat{\sigma}_i - \sigma_{j,i})^2, \tag{73}$$

where the weight w_i accounts for underestimation penalties:

$$w_i = \begin{cases} P, & \text{if } \hat{\sigma}_i < \sigma_{j,i} \quad \text{(underestimation)}, \\ 1, & \text{if } \hat{\sigma}_i \geq \sigma_{j,i} \quad \text{(overestimation)}. \end{cases} \tag{74}$$

Here, $P > 1$ is the underestimation penalty, treated as a hyperparameter optimized using the Hyperband algorithm. This asymmetric weighting is applied exclusively during training to encourage conservative predictions, while evaluation on validation and test sets uses the standard (unweighted) RMSE to assess predictive accuracy objectively.

The overall training objective is to identify network parameters (W^*, b^*) that minimize the expected daily loss over the distribution of training data D:

$$(W^*, b^*) = \arg\min_{W,b} \; \mathbb{E}_{(x_j, \sigma_j, M_j)\sim D}\left[L_j(W, b)\right]. \tag{75}$$

In practice, this expectation is approximated by iterating over the training set day by day (batches) using the stochastic optimizer Adam:

$$W_{t+1} = W_t - \eta \nabla_W L_j(W_t, b_t), \tag{76}$$

where η is the learning rate. Since V_{QL} is a non-differentiable black-box function, gradients are computed numerically via finite differences using a tf.custom_gradient implementation.

3.6 Methodology Workflow

The subsequent section delineates the comprehensive, step-by-step workflow implemented for the calibration of the Hull-White model, as depicted by the provided source code. The process is designed as an end-to-end pipeline, commencing with raw data ingestion and culminating in model evaluation, adaptive retraining, and interpretability analysis.

1. **Data Ingestion and Preprocessing:** The initial phase involves loading raw EUR swap curve data, provided in Excel format. For each historical date, the mid-market rate is computed from the bid and ask quotes. These raw swap curves are then processed and transformed into a standardized format. Subsequently, the `QuantLib` library is employed to bootstrap these swap curves, yielding the corresponding zero-coupon yield curves for each valuation date. This bootstrapping process utilizes a `PiecewiseLogLinearDiscount` methodology to construct a term structure of zero rates from the observed market swap rates.

2. **External Market Data Integration:** To enrich the feature set, external market data is loaded. This includes the ICE BofA MOVE Index (MOVE), which measures US Treasury rate volatility; the CBOE Volatility Index (VIX), which measures equity market volatility; and the EUR/USD foreign exchange rate. This data is sourced via the `yfinance` library. A complete time series is ensured by forward- and back-filling any missing daily values, providing a consistent set of external features for every yield curve date.

3. **Chronological Data Splitting:** Given the time-series nature of the financial data, a strict chronological split is enforced to prevent look-ahead bias. The dataset of available daily yield curves is divided into a training set (the initial 50%), a validation set (the following 20%), and a final out-of-sample testing set (the remaining 30%). This ensures that the model is trained on past data, validated on more recent data, and finally tested on unseen future data.

4. **Feature Engineering and Dimensionality Reduction:** For each yield curve, a comprehensive feature vector is engineered. This vector includes not only the zero rates at key tenors (1Y to 30Y) but also derived features such as various yield curve slopes (e.g., 10Y-2Y), curvatures (e.g., $(2 \times 5\mathrm{Y}) - 2\mathrm{Y} - 10\mathrm{Y}$), and the external market indicators. To address the high multicollinearity inherent in yield curve rates, Principal Component Analysis (PCA) is performed exclusively on the rate-based features of the training data. The first three principal components, which typically

correspond to the level, slope, and curvature of the yield curve, are retained. This reduces dimensionality while preserving the most significant sources of variance. The complete feature set, comprising these principal components and the other engineered features, is then standardized using Scikit-learn's `StandardScaler`, which is also fitted solely on the training data.

5. **Hyperparameter Optimization:** A rigorous hyperparameter tuning process is conducted using the Hyperband algorithm, implemented via the `Keras Tuner` library. The search space includes critical model architecture parameters (number of layers, neurons per layer, activation function, dropout usage and rate), optimization parameters (learning rate), and a custom hyperparameter for an asymmetric loss function (underestimation penalty). The objective function for the tuner is the minimization of the root mean squared error (RMSE) on the validation set.

6. **Model Training and Early Stopping:** Upon identifying the optimal set of hyperparameters, the final model is retrained from scratch on the entire training set. A key architectural choice is the `ResidualParameterModel`, which does not predict the Hull-White parameters directly but rather learns a residual correction to a predefined initial guess. Training is performed over a set number of epochs, but an early stopping mechanism is employed. The model's performance on the validation set is monitored after each epoch. If the validation RMSE does not improve for a predefined number of consecutive epochs (patience), the training is halted, and the model weights from the epoch with the lowest validation RMSE are restored. This prevents overfitting to the training data.

7. **Out-of-Sample Evaluation and Adaptive Retraining:** The trained model is evaluated on the unseen testing set in a forward-moving, chronological sequence. For each day in the test set, the model predicts the Hull-White parameters, and the resulting calibration error (RMSE) is calculated. This out-of-sample error stream is monitored by the ADWIN (ADaptive WINdowing) algorithm to detect concept drift—a statistically significant degradation in model performance. If ADWIN detects a drift, or if the daily error exceeds a predefined absolute threshold, a retraining event is triggered. In this event, all historical data up to the point of the trigger (original training, validation, and the processed portion of the test set) is used to form a new, larger training set, and the entire workflow from step 4 onwards is repeated. This rolling approach allows the model to adapt to changing market regimes.

8. **Model Explainability with SHAP:** To ensure transparency and interpretability of the final trained model, a post-hoc analysis is conducted using SHAP (SHapley Additive exPlanations). A `DeepExplainer` is utilized to compute SHAP values for the model's predictions on the test set. This analysis quantifies the contribution of each input feature (e.g., PC Level, MOVE Index) to the prediction of each output Hull-White parameter. The results are visualized through summary and feature importance plots, providing crucial insights into the model's decision-making process.

3.7 Used Model Architecture & Setup

- Description of how the actual used model is setup, trained and other technical details
- Model does not directly predict the parameters of the Hull-White model but predicts the correction of an initial guess e.g. the parameters from the last calibration (for example yesterday) or the parameters determined by the traditional calibration (hybrid approach)

3.8 Calibration Strategy

The calibration methodology implemented in this thesis aligns with the *simultaneous optimization* strategy discussed in section 2.4.5 This choice is evident from the design of the neural network and its training process. The model is configured to concurrently optimize all dynamic parameters of the Hull-White model within a single, unified optimization routine.

Specifically, the approach treats the mean-reversion parameter, α, as a single constant value and the volatility parameter, $\sigma(t)$, as a piecewise-constant function with seven distinct time segments. The neural network's output layer predicts the complete set of these eight parameters. During the training loop, the custom gradient calculation within the `_perform_training_step` function computes the gradients of the loss function with respect to all eight parameters simultaneously. The optimizer then updates all model weights to jointly minimize the pricing error, thereby fitting both α and $\sigma(t)$ at the same time. This is in contrast to a two-step approach where parameters are estimated sequentially or a fixed mean-reversion approach where α is not optimized at all.

Furthermore, the calibration employs a *global (full matrix) calibration* methodology with the exception of swaptions which have a maturity and tenor of less than 2 years. This global approach provides the model with a comprehensive view of the market, enabling it to achieve a more robust and stable fit across the entire volatility surface, as recommended for empirical analyses in section 2.4.5.

3.9 Comparison of Neural Network and Levenberg-Marquardt Calibration

A robust experimental framework was designed and implemented to enable a scientifically valid comparison between the traditional Levenberg-Marquardt (LM) optimization and the Neural Network (NN) predictor. The primary objective of this framework was to address the fundamental methodological challenge of fairly comparing an in-sample fitter with an out-of-sample predictor.

The core issue arises from the differing nature of the two models. The LM algorithm operates as an in-sample fitter: for a given day, it receives the complete set of that

day's market-observed swaption volatilities and determines the Hull-White parameters (a, σ) that minimize the pricing error for this exact set. Its performance metric, typically the Root Mean Squared Error (RMSE), is computed against the same data used for calibration, reflecting the model's goodness-of-fit. This approach is prone to overfitting, as the resulting parameters may capture noise and idiosyncrasies specific to the input data rather than generalizable features. In contrast, the NN is designed as an out-of-sample predictor. It is trained on historical data to learn a mapping from observable market features, such as yield curve shapes and volatility indices, to the corresponding optimal Hull-White parameters. On a test day, it receives only these market features and predicts parameters that are evaluated against the previously unseen swaption volatilities, making its error a measure of generalization ability.

To reconcile this methodological inconsistency, an *Intra-Day Hold-Out Set* framework was employed. This protocol enforces a uniform out-of-sample evaluation for both models. Each day within the test period was processed as follows. First, all valid swaptions were assembled into a master list and partitioned into calibration and hold-out sets using stratified random sampling, with the swaption expiry date as the stratification variable. Within each expiry stratum, 80% of the instruments were assigned to the calibration set and the remaining 20% to the hold-out set. Single-instrument strata were assigned to the calibration set to ensure sufficient data for LM optimization, and the random sampling was controlled by a fixed seed to guarantee reproducibility.

Once the data partitioning was completed, both models generated their Hull-White parameters under controlled conditions. The pre-trained NN produced its parameters via a forward pass using only the day's market features, whereas the LM algorithm was restricted to the 80% calibration set and iteratively optimized the parameters to fit this subset. Wall-clock times were recorded for both processes.

Finally, both sets of parameters were evaluated on the identical 20% hold-out set. Each model was used to price the hold-out swaptions, and the RMSE between model-implied volatilities and observed market volatilities was computed. This uniform evaluation enabled a direct and fair comparison of the NN's predictive capabilities against the LM's in-sample fitting performance, effectively quantifying their relative generalization power in unseen market conditions.

3.10 Technical Setup and Computational Optimization

The implementation of the neural network–based Hull–White calibration framework was carried out in Python 3.12.3, employing TensorFlow 2.15 for the neural network architecture and QuantLib 1.34 for the financial modeling components. All experiments were conducted on a commercially available laptop equipped with 32 GB of RAM and an AMD Ryzen 7 7840HS processor (16 cores, approximately 3.8 GHz), operating under Windows 11.

3.10.1 Integration of TensorFlow and QuantLib

A central technical challenge of this research lies in the hybrid nature of the model architecture, which combines the differentiable TensorFlow framework with the non-differentiable pricing engines of QuantLib. Since QuantLib's pricing routines are implemented in C++ and do not expose analytical gradients, the neural network cannot rely on TensorFlow's automatic differentiation. To overcome this limitation, the QuantLib-based loss function was implemented within a function decorated by TensorFlow's `@tf.custom_gradient`. This mechanism allows the explicit definition of a custom backward pass, where the gradients are computed manually rather than automatically propagated by TensorFlow.

The gradients with respect to the neural network outputs are approximated numerically using finite difference methods. For each model parameter, the loss function is evaluated twice — once with a positive and once with a negative perturbation — yielding an approximation of the partial derivatives. Consequently, each gradient step requires $2 \times$ (number of parameters) complete QuantLib evaluations. This approach is computationally demanding but necessary to correctly propagate gradients through the non-differentiable pricing engine.

3.10.2 Parallelization and Hardware Considerations

To mitigate the computational cost associated with the numerical gradient computation, the implementation leverages Python's `concurrent.futures.ThreadPoolExecutor` to parallelize function evaluations across the 16 available CPU cores. Thread-based parallelization is preferred over process-based parallelization because it introduces less communication overhead and efficiently shares memory among threads, which is particularly advantageous for memory-intensive but moderately CPU-bound pricing routines. In practice, this parallelization leads to a near-linear reduction in computation time with respect to the number of threads, thereby rendering the training process computationally feasible.

All training was executed on the CPU. This decision was motivated by the observation that the primary computational bottleneck lies not in the neural network's forward propagation—which would benefit from GPU acceleration—but in the iterative, conditional, and sequential nature of QuantLib's pricing algorithms. Such algorithms are inherently unsuited to GPU architectures, which excel in parallel matrix operations but perform poorly with branching control flow. Re-implementing QuantLib's pricing models in a GPU-compatible framework would require prohibitive effort while yielding minimal performance benefits. Although frameworks such as *TF-Quant-Finance* aim to provide GPU-compatible financial modeling tools, they currently lack active maintenance and do not support Gaussian short-rate (GSR) models with piecewise constant volatility—an essential feature for this research.

3.10.3 Algorithmic and Numerical Efficiency

Further acceleration was achieved through algorithmic refinements and numerical efficiency improvements. The numerical derivative used in gradient computation can be approximated either by the forward difference or the central difference scheme. The forward difference requires only one additional model evaluation per parameter and is therefore approximately twice as fast as the central difference method, albeit with slightly higher numerical noise. A careful balance between computational efficiency and numerical precision was thus sought in practice.

Additionally, the precision of numerical integration within the QuantLib pricing engine was adjusted through the `pricing_engine_integration_points` parameter, which controls the number of Gaussian quadrature points used by the `Gaussian1dSwaptionEngine`. Reducing the number of integration points lowers computational cost per pricing call but may introduce marginally higher numerical error. The chosen configuration represents a compromise between computational tractability and pricing stability.

3.10.4 Optimization of the Training Process

Several optimization strategies were applied to enhance convergence speed and stability. An *early stopping* criterion monitors the validation root mean squared error (RMSE) after each epoch and terminates training once no improvement is observed over a predefined patience period. This approach prevents overfitting and avoids unnecessary computation.

In addition, the concept of *instrument batching* was introduced to reduce the computational burden per training iteration. Rather than evaluating all available swaptions at once, the loss function is computed on randomly selected subsets of instruments, controlled by the parameter `instrument_batch_size_percentage`. This stochasticity not only accelerates each iteration but also introduces beneficial noise, which helps the optimizer escape local minima and improves generalization.

Hyperparameter optimization was conducted using the Hyperband algorithm (see Subsection 2.5.4). Hyperband allocates computational resources adaptively by evaluating a large number of configurations for a limited number of epochs and retaining only the most promising candidates for further training. This method significantly reduces the computational time required for hyperparameter tuning while maintaining robustness in identifying high-performing parameter configurations.

3.10.5 Workflow-Level Enhancements and Model Design

To further enhance efficiency, all input data—such as bootstrapped zero curves and volatility cubes—were preloaded into system memory prior to model training. This design choice eliminates repeated disk input/output operations, minimizing latency and enabling smoother training iterations.

Moreover, the preprocessing pipeline was designed modularly, allowing selective execu-

tion of computationally expensive routines. Boolean flags such as `PREPROCESS_CURVES` and `BOOTSTRAP_CURVES` allow bypassing redundant computations once intermediate data has been stored, thereby facilitating efficient experimentation and reproducibility during hyperparameter searches.

Finally, the neural network was implemented as a residual model, predicting adjustments to an initial parameter estimate rather than the parameters directly. If the initial parameter vector is denoted by θ_{initial} and the predicted correction by $\Delta\theta_{\text{predicted}}$, the final parameter estimate is given by

$$\theta_{\text{final}} = \theta_{\text{initial}} + \Delta\theta_{\text{predicted}}. \tag{77}$$

This residual formulation simplifies the learning task, as the network learns to approximate local corrections rather than the entire nonlinear mapping between market features and model parameters. Empirically, this leads to smoother loss surfaces, faster convergence, and improved numerical stability.

In summary, the proposed hybrid framework integrates TensorFlow's differentiable learning capabilities with QuantLib's high-fidelity financial pricing models through custom gradient definitions and parallelized numerical differentiation. Combined with algorithmic, workflow, and training-level optimizations, these measures ensure that the training process remains computationally feasible, stable, and reproducible on commercially available hardware.

3.11 Use of AI

To fulfil the requirements set by the Zürcher Hochschule für Angewandte Wissenschaften (Zürcher Hochschule für Angewandte Wissenschaften, 2024), in the following I will critically reflect the use of generative AI tools.

4 Results

Following the methodology outlined in the previous chapter, this section presents the empirical findings of the comparative analysis. The results are structured around the core evaluation criteria: calibration accuracy, computational speed, and the robustness of the derived Hull-White parameters. Through a series of tables, figures, and quantitative metrics, this chapter provides an objective assessment of how well each method replicates observed market prices and how they perform under different conditions. The aim is to deliver a clear, evidence-based comparison that highlights the respective strengths and weaknesses of the traditional and machine learning-based approaches.

- **Calibration Performance Metrics:** Define and present the metrics used to quantify the "goodness of fit" for both traditional and ML-based calibration methods.

  - Common metrics include Mean Squared Error (MSE), Root Mean Squared Error (RMSE), or Mean Absolute Error (MAE) between model-implied and market-observed swaption prices or implied volatilities.
  - Discuss any percentage errors or relative errors used.

- **Comparison of Calibration Speed:** Quantitatively compare the computational efficiency of each method.
  - Provide average calibration times for the traditional algorithms (e.g., Levenberg-Marquardt) per market snapshot.
  - Detail the training time required for the neural network model.
  - Report the inference or prediction time for the neural network once trained, when applied to new market data.
  - Discuss the trade-offs, such as the initial higher computational cost of training the NN versus its potentially faster inference speed for subsequent calibrations.

- **Comparison of Accuracy and Robustness:** Analyze how accurately each method calibrated the Hull-White parameters and replicated observed market prices/volatilities.
  - Present tables and graphs comparing model-implied swaption volatilities/prices against market values.
  - Discuss how well each method captures the various features of the volatility surface (e.g., skew, smile, term structure).
  - Evaluate the robustness of each method to noisy or incomplete market data, as well as its stability under different market regimes (e.g., high vs. low volatility periods).

- **Analysis of Hull-White Parameters:** Examine the derived mean reversion ‘a(t)‘ and volatility ‘$\sigma(t)$‘ parameters from both approaches.
  - Use visualizations (e.g., time series plots or surface plots) to show how these parameters vary across time, option maturities, or swap tenors for each method.
  - Offer an interpretation of these differences, linking them to the underlying assumptions and characteristics of each calibration technique and their financial implications.

- **Sensitivity Analysis (Optional):** Investigate how sensitive each model's calibration results are to changes in input data, initial parameter guesses, or hyperparameter choices.

5 Conclusion

This final chapter moves from the presentation of results to their interpretation and synthesis, drawing together the key insights from the research. It begins by summarizing the principal findings of the comparative study, directly addressing the research questions

posed in the introduction. The discussion then explores the broader implications of these findings for financial practitioners and academic researchers, identifying scenarios where each calibration method may be preferred. The chapter also acknowledges the inherent limitations of this study and, based on the outcomes and unanswered questions, proposes promising directions for future research in the field of model calibration and machine learning in finance.

- **Restate Thesis Aim:** Briefly reiterate the primary objective of the research.
- **Summarize Key Findings:** Condense the most significant results from the "Results" section regarding the comparative performance (speed, accuracy, robustness) of traditional versus ML-based calibration.
- **Discuss Implications:** Elaborate on what the findings mean for financial practitioners and academic researchers.
  - Identify scenarios or contexts where machine learning approaches might offer distinct advantages over traditional methods, and vice-versa.
  - Discuss whether specific market conditions favor one method over the other.
  - Highlight the practical benefits, such as improved risk management, faster pricing of complex derivatives, or more robust model fitting.
- **Limitations:** Acknowledge any constraints or limitations of the study (e.g., specific dataset size, focus on a one-factor model, inherent simplifications, computational resources).
- **Future Work:** Propose potential directions for further research. This could include exploring multi-factor Hull-White models, testing different machine learning architectures, investigating real-time calibration techniques, or extending the comparison to other financial instruments or markets.

References

Alaya, M. B., Kebaier, A., & Sarr, D. (2021). Deep calibration of interest rates model. *arXiv preprint arXiv:2110.15133*.

Aljalbout, E., Golkov, V., Siddiqui, Y., & Cremers, D. (2018). Clustering with deep learning: Taxonomy and new methods. *arXiv preprint arXiv:1801.07648*. https://doi.org/10.48550/arxiv.1801.07648

Alvarez, H., Sacchi, R., & Señaris, T. (2022). *Hull-white calibration for swaptions using nns* (tech. rep.). Barcelona School of Economics. https://repositori-api.upf.edu/api/core/bitstreams/8bae6b11-7b9e-44a8-8b8e-6ac7ec51c920/content

Baaquie, B. E. (2010). Interest rates in quantum finance: Caps, swaptions and bond options. *Physica A: Statistical Mechanics and its Applications*, *389*(2), 296–314. https://doi.org/https://doi.org/10.1016/j.physa.2009.09.031

Black, F. (1976). The pricing of commodity contracts. *Journal of Financial Economics*, *3*(1-2), 167–179. https://doi.org/10.1016/0304-405X(76)90024-6

Brigo, D., & Mercurio, F. (2006). *Interest rate models: Theory and practice* (2nd) [1014 pages]. Springer.

Chan, J. Y.-L., Leow, S. M. H., Bea, K. T., Cheng, W. K., Phoong, S. W., Hong, Z.-W., & Chen, Y.-L. (2022). Mitigating the multicollinearity problem and its machine learning approach: A review. *Mathematics*, *10*(8). https://doi.org/10.3390/math10081283

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning* [http://www.deeplearningbook.org]. MIT Press.

Gurrieri, S., Nakabayashi, M., & Wong, T. (2009). Calibration methods of hull-white model [Available at SSRN: https://ssrn.com/abstract=1514192]. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.1514192

Hernandez, A. (2016). Model calibration with nns. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.2812140

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, *313*(5786), 504–507. https://doi.org/10.1126/science.1127647

Hohmann, D., Hörig, M., Ketterer, F., Murray, K., & Ward, R. (2015, September). The new normal: Using the right volatility quote in times of low interest rates for solvency ii risk factor modelling [Accessed: 2025-10-19]. https://web.actuaries.ie/sites/default/files/erm-resources/2079LDP_20150911.pdf

Huang, L., Qin, J., Zhou, Y., Zhu, F., Liu, L., & Shao, L. (2020). Normalization techniques in training dnns: Methodology, analysis and application. https://arxiv.org/abs/2009.12836

Hull, J., & White, A. (1990). Pricing interest-rate derivative securities. *The Review of Financial Studies*, *3*(4), 573–592. https://doi.org/10.1093/rfs/3.4.573

Hull, J. C. (2015). *Options, futures and other derivatives* (9nth). Pearson Education.

Jamshidian, F. (1989). An exact bond option formula. *The Journal of Finance*, *44*(1), 205–209. https://doi.org/10.1111/j.1540-6261.1989.tb02414.x

Karlsson, P., Jain, S., & Oosterlee, C. W. (2016). Fast and accurate exercise policies for bermudan swaptions in the libor market model. *International Journal of Financial Engineering*, *03*(01), 1650005. https://doi.org/10.1142/S2424786316500055

Kladivko, K., & Rusy, T. (2023). Maximum likelihood estimation of the hull–white model. *Journal of Empirical Finance*, *70*, 227–247. https://doi.org/https://doi.org/10.1016/j.jempfin.2022.12.002

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., & Talwalkar, A. (2017). Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, *18*(1), 6765–6816. https://jmlr.org/papers/volume18/16-558/16-558.pdf

Longstaff, F. A., & Schwartz, E. S. (2001). Valuing american options by simulation: A simple least-squares approach. *The Review of Financial Studies*, *14*(1), 113–147. https://doi.org/None

Marquardt, D. W. (1963). An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Mathematics*, *11*(2), 431–441. https://doi.org/10.1137/0111030

Mienye, I. D., & Swart, T. G. (2024). A comprehensive review of deep learning: Architectures, recent advances, and applications. *Information*, *15*(12). https://doi.org/10.3390/info15120755

Moysiadis, G., Anagnostou, I., & Kandhai, D. (2019). Calibrating the mean-reversion parameter in the hull-white model using nns. In *Lecture notes in computer science* (pp. 23–36). https://doi.org/10.1007/978-3-030-13463-1_2

Rebonato, R. (2004). *Interest rate option models: Understanding, analysing and using models for exotic interest rate options* [First published: 27 August 2004]. John Wiley & Sons. https://doi.org/10.1002/9781118673539

Rebonato, R. (2018). Principal components: Theory. In *Bond pricing and yield curve modeling: A structural approach* (pp. 98–107). Cambridge University Press. https://doi.org/https://doi.org/10.1017/9781316694169.006

Sazli, M. H. (2006). A brief review of feed-forward neural networks. *Communications Faculty of Sciences University of Ankara Series A2-A3 Physical Sciences and Engineering*, *50*(01). https://doi.org/10.1501/commua1-2_0000000026

Shanker, M., Hu, M., & Hung, M. (1996). Effect of data standardization on neural network training. *Omega*, *24*(4), 385–397. https://doi.org/https://doi.org/10.1016/0305-0483(96)00010-2

Sildir, H., Aydin, E., & Kavzoglu, T. (2020). Design of feedforward neural networks in the classification of hyperspectral imagery using superstructural optimization. *Remote Sensing*, *12*(6). https://doi.org/10.3390/rs12060956

Sola, J., & Sevilla, J. (1997). Importance of input data normalization for the application of neural networks to complex industrial problems. *Nuclear Science, IEEE Transactions on*, *44*, 1464–1468. https://doi.org/10.1109/23.589532

Suo, W., Hull, J. C., & Daglish, T. C. (2009). Volatility surfaces: Theory, rules of thumb and empirical evidence [Accessed: 2025-10-19]. https://ssrn.com/abstract=1521882

Sutton, R. S., & Barto, A. G. (2015). *Reinforcement learning: An introduction* (2nd). MIT Press.

Zheng, A., & Casari, A. (2018). *Feature engineering for machine learning: Principles and techniques for data scientists* (1st) [A PDF version is available at https://www.repath.in/gallery/feature_engineering_for_machine_learning.pdf]. O'Reilly Media.

A Appendix

- **Detailed Data Tables:** Include extensive tables of raw data, implied volatility surfaces, or specific swaption characteristics if they were too large or detailed for the main body of the text.
- **Code Snippets or Repository Reference:** Provide key code segments, especially for the custom gradient implementation in TensorFlow or the neural network architecture. Alternatively, a link to a public code repository (e.g., GitHub) is highly recommended.
- **Additional Figures/Graphs:** Present any extra visualizations of results, detailed calibration error plots, or parameter evolutions that support your main findings but were not critical for the core discussion.
- **Mathematical Derivations:** Include any lengthy or complex mathematical derivations that were summarized or omitted from the main text to improve readability.
- **Hyperparameter Details:** List specific hyperparameters used for your neural network model (e.g., number of hidden layers, number of neurons per layer, choice of activation functions, optimizer used, learning rate schedule, batch size, number of epochs, regularization techniques).
- **Extensive Sensitivity Analysis Results:** If you conducted a comprehensive sensitivity analysis, the detailed findings and plots can be placed here.