University of Derby

Department of Computing & Mathematics

A project completed as part of the requirements for the
BSc (Hons) Computer Science

Quantum Speedup for High-Frequency Trading

A Multi-Dimensional Analysis Using Taylor Expansion and Optimization

by

Lucian Pemaj

2025

Abstract

High-frequency trading (HFT) operates in the microscale time domain, where minimal gains in predictive power and computational performance can provide a significant edge in finance. This dissertation considers the potential use of quantum computing to advance HFT methods, specifically by improving multi-dimensional Taylor expansion refinements for predictive purposes and optimization of their coefficients. We analyze a hybrid approach combining classical and quantum methods, applying techniques like the Variational Quantum Eigensolver (VQE), and compare its performance against classical optimization methods, BFGS and Gradient Descent, for 1D, 2D, 3D, and 4D instances of Taylors. The work examines the methodologies from multiple angles: predictability in terms of Mean Squared Error; computational performance measured by runtime; and profitability and Sharpe ratio from trading simulations. Our findings illustrate quantum optimization methods, specifically VQE, display substantial potential at finding optimal Taylor coefficients with minimal computational effort, most notably in multi-dimensional problems where classical methods struggle. While we acknowledge limitations related to current quantum computing hardware, this work identifies a promising path toward quantum acceleration and predictive power boost in HFT. This work makes a substantial contribution to the new science of quantum finance by providing a performance benchmark comparison and laying groundwork for incorporating quantum computational methods into advanced financial modeling, aimed at improving algorithmic trading performance.

Keywords: Quantum Computing, High-Frequency Trading (HFT), Quantum Finance, Taylor Expansion, Hybrid Quantum-Classical Optimization, Algorithmic Trading.

Acknowledgments

I would like to express my sincere gratitude to the many individuals who have supported me throughout my research and the writing of this bachelor thesis. Their guidance, encouragement, and assistance have been invaluable.

First and foremost, I extend my deepest appreciation to my supervisor, Dr.Kostadinos Katsifis, for his insightful guidance, unwavering support, and constructive feedback throughout this project. His expertise in Physics and Computer Science and their patience have been instrumental in shaping this research and navigating its complexities.

I am also grateful to Dr. Alexander Rompas and Dr. Michael Diagikidis from the Department of Computer Science for their help, in Navigating the Mathematical foundation of the thesis , for providing for insightful comments on early drafts.

I am also grateful to my family and friends for their endless encouragement, understanding, and patience, especially during the challenging phases of this work. Their belief in me has been a constant source of motivation thank you for everything.

Finally, I would like to thank anyone else who has contributed, directly or indirectly, to the successful completion of this thesis.

Contents

List of Figures

List of Tables

1 Introduction

High-Frequency Trading (HFT) forms a key part of modern financial market trading, where many orders are executed rapidly at high frequency and usually in microsecond time scales. This sophisticated branch of algorithmic trading utilizes complex calculations based on algorithms and fast streams of informational data to take advantage of temporary price inefficiencies and market dysfunctions (Securities, Commission, et al. 2010; Zaharudin, Young, and Hsu 2022). HFT algorithms rely upon their capability to execute quick real-time calculations and to make decisions based upon their calculations, requiring the use of online algorithms with the capacity to handle incoming streams of data and make instantaneous decisions without full awareness of subsequent market conditions (Loveless, Stoikov, and Waeber 2013).

The quest for increased precision in high-frequency trading predictive models often requires the inclusion of a large number of market variables, including price, volume, momentum, and volatility, to capture in a comprehensive manner the complex behaviors in the market. The inclusion of more parameters then does introduce a sizeable computation problem usually described as the "curse of dimensionality" (Hastie, Tibshirani, and Friedman 2009). The increasing financial model complexities are accompanied by the corresponding increasing computation load related to optimization as well as the risk of the magnification of noise, making classic optimization techniques potentially inapplicable or slow in the microsecond-responsive environment of HFT (Mantilla and Dormido-Canto 2023). This problem emphasizes the essential need to introduce novel modeling and optimization techniques capable of exploring high-dimensional spaces without compromises in predictive performance. Within this context, this dissertation explores the application of Taylor expansion series as a basis for financial modeling, in addition to an analysis of how concepts from quantum computing can be used to improve the optimization of processes inherent within these models. The use of Taylor expansions presents a mathematically rigorous method for approximating complex functions using simpler polynomial approximations, thereby offering a systematic framework for modeling price dynamics (Lorig, Pagliarani, and Pascucci 2013; Pourahmadi 1984). Meanwhile, breakthroughs within the field of quantum computing, specifically within the context of hybrid quantum-classical algorithms, present a revolutionary methodology for addressing optimization problems previously considered intractable by classical computational systems, with the potential to achieve improvements in efficiency through leveraging the phenomena of superposition and entanglement within the quantum context (Cerezo et al. 2021; Moll et al. 2018).

The primary goal of this research is to conduct an exhaustive analysis to determine if a custom hybrid quantum-classical optimization algorithm can provide substantial computational efficiencies or improved model efficacy in optimizing Taylor expansion coefficients for HFT price prediction compared to a standard classical optimization technique. This project seeks to answer the fundamental question:

"Does a hybrid quantum-classical optimization approach provide considerable speed benefits or better model effectiveness for multi-dimensional HFT price prediction using Taylor expansion, and what are the implications of this for simulated trading outcomes?"

To address the problem, the research utilizes a methodological strategy that emphasizes

Implementing one-dimensional and multi-dimensional (up to 4D) second-order Taylor expansion models (with a diagonal quadratic form for multi-dimensional cases) to approximate high-frequency stock price movements. The development and deployment of a new hybrid quantum-classical optimization technique using simulated quantum circuits aspire to find the best-performing Taylor coefficients via the minimization of the Mean Squared Error (MSE). Formation of a classical baseline entails the use of the L-BFGS-B optimization method to optimize equivalent Taylor expansion models.

Conducting a comparative analysis based on measures of predictive accuracy (RMSE, MAE, MAPE), computational efficiency (such as execution time and speedup), and the performance of a theoretical trading strategy based on the predictions from the quantum-optimized and classically-optimized models.

This research is important as it examines the intersection of classical financial modeling techniques, the high-frequency trading-related high computational intensity, and the new potential of quantum-inspired optimization algorithms. While fully fault-tolerant quantum computers are not yet available, a study of the potential contained within blended methods—even via simulation—can provide critical insights towards future algorithms and identify problem types where quantum methods can bring a significant benefit. The findings seek to contribute to the emerging field of quantum finance through the presentation of empirical testing of a specific quantum-inspired heuristic in the context of a realistic and computationally intensive financial problem. The structure of this thesis is outlined as follows: Section §2 presents a comprehensive literature review that covers High-Frequency Trading, pertinent Taylor expansions in finance, key concepts of quantum computing applied to optimization, as well as prior research work in the field. Section §3 discusses the methodology used, which includes the selection of data sets, feature engineering steps, the design of the Taylor expansion model, and the creation of hybrid quantum-classical and classical baseline optimization algorithms, as well as the evaluation metrics. Section §4 outlines the experimental setup and the empirical results obtained from the comparative study carried out. Section §5 discusses these results, covering the research questions, issues faced in the research, limitations intrinsic to the research, as well as practical implications arising from the results. Section §6 suggests potential avenues for future research activities. Finally, Section §7 concludes the thesis by summarizing the main findings and contributions to the field concerned.

2 Literature Review

2.1 High-Frequency Trading (HFT)

2.1.1 Overview of HFT and its reliance on real-time computations.

What is High-Frequency Trading or HFT the Security and Exchange Commission (SEC) characterizes this as a proprietary capacity that engages in strategies that generate a large number of trades on a daily basis (Securities, Commission, et al. 2010). But in reality it is much more than that, which is a subset of a larger topic of algorithmic trading that utilizes high-speed and sophisticated computer algorithms to execute a very large amount of trades within very small time frames that can be measured in milliseconds or nanoseconds. With this approach,large quantitative firms can take advantage of high-frequency financial data and electronic trading tools to capitalize on minimal price discrepancies between markets (Zaharudin, Young, and Hsu 2022).

To achieve all of the above we need to emphasize the importance and the ability to handle real time computation fast and efficiently to solve this problem we use online algorithms Loveless, Stoikov, and Waeber (2013). Online algorithms in HFT are crucial for decision making as HFT systems must make rapid decisions on the incoming data streams without complete knowledge of future market conditions. In this context, online algorithms are used to process real-time market data, execute trades, and manage risk efficiently (Albers 2003).

2.1.2 The Curse of Dimensionality in HFT Optimization

HFT poses significant challenges and one of them is dimensionality and in particular dimensionality in multi-variable financial models when real time optimization though multiple parameters are crucial for profitability. Incorporating additional variables to a financial models such as price, volume, volatility produces to capture nuanced market dynamics. Nevertheless, as the number of parameters grows larger, the computational burdens and data requirements also rise exponentially. For instance, a model based on 20 technical indicators (Bollinger Bands, RSI, and MACD) on seconds' intervals results in a feature space with around 1.7 million data points every day over the trading hours (accounting for 6.5 hours of market involvement). This number outpaces the processing power of conventional optimization methods in real-time scenarios with adverse impacts on the profitability of high-frequency trading (Hastie, Tibshirani, and Friedman 2009).

High-frequency models often face the problem of noise amplification with higher dimensionality. For example, Mantilla and Dormido-Canto (2023) argue that spurious correlations between variables, including momentum and order-book imbalance, have the ability to distort predictions in models with over 15 parameters. These misleading signals arise because the possibility of random correlation between variables approaches 1 with higher dimensionality, even in independent data (Fan and Li 2001). For high-frequency trading, these errors have the potential to result in faulty trades in moments of market volatility.

2.2 Taylor Expansion in Financial Modeling

2.2.1 One-Dimensional and Multi-Dimensional Taylor Expansions

A mathematical cornerstone of approximations Taylor Expansion has found it way to numerous application in financial modeling due to its ability to approximate complex function with polynomial series (Pourahmadi 1984).

Taylor expansion provides a method for approximating a function near a given point by using its derivatives at that particular point. For a function of one variable $f(x)$, the Taylor series expansion about the point a can be written as:

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^{(n)}(a)}{n!}(x-a)^n$$

Here $f(x)$ represents the function to be approximated and a represents the point around which the expansion is performed ,here $f'(a), f''(a), \ldots, f^{(n)}(a)$ derivatives of the function evaluated at a also this $(x-a)$: represents the difference between the input x and the expansion point a and finally $n!$ factorial of n, used in the denominator to normalize higher-order terms.

In the field of financial modeling, this method can be used to estimate a range of financial parameters. For example, the price of a stock at a certain time t can be approximated using its value and the relevant derivatives at a previous time t_0 .

$$P(t) \approx P(t_0) + P'(t_0)(t - t_0) + \frac{P''(t_0)}{2!}(t - t_0)^2 + \cdots$$

Here, $P'(t_0)$ represents the rate of change of the stock price at t_0, and $P''(t_0)$ represents the acceleration of the price change.

However, financial markets have an inherent multi-dimensional character, with complicated interactions between different variables. To capture these interactions correctly, multi-dimensional Taylor expansions are necessary. For a function of several variables, say $f(x_1, x_2, ., x_n)$, the Taylor expansion around a point $(a_1, a_2, ., a_n)$ is given by:

$$f(\mathbf{x}) \approx f(\mathbf{a}) + \sum_{i=1}^{N} \frac{\partial f}{\partial x_i}\bigg|_{\mathbf{a}} (x_i - a_i) + \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \frac{\partial^2 f}{\partial x_i \partial x_j}\bigg|_{\mathbf{a}} (x_i - a_i)(x_j - a_j)$$

Where $f(\mathbf{x})$ is a function to approximate in multiple dimensions and $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ Input vector of N variables and $\mathbf{a} = (a_1, a_2, \ldots, a_N)$ point around which the expansion is performed $\frac{\partial f}{\partial x_i}$ Partial derivative of f with respect to x_i (gradient) $\frac{\partial^2 f}{\partial x_i \partial x_j}$ second-order partial derivative of f with respect x_i and x_j (Hessian) and $(x_i - a_i)$ Distance between input x_i and expansion point a_i.

For instance, when modeling the price of a stock as a function of time and volume, $P(t, V)$, the multi-dimensional Taylor expansion can be expressed as:

$$(P(t, V) \approx P(t_0, V_0) + \frac{\partial P}{\partial t}\bigg|_{(t_0, V_0)} (t - t_0) + \frac{\partial P}{\partial V}\bigg|_{(t_0, V_0)} (V - V_0) + \ldots$$

This expansion captures the relationship between price and volume, accounting for how changes in volume can affect the stock price.

2.2.2 Applications of Taylor Series in Finance

The Taylor series is an important mathematical tool widely used in finance to estimate complex relations and conduct computations (Estrella 1995; Lorig, Pagliarani, and Pascucci 2013). In options pricing, the series augments the Black-Scholes formula by adding non-constant volatility and other complexities of real life in addition to allowing successful approximation of the "Greeks" (e.g., Delta, Gamma, Vega), whose values represent the option price sensitivities to different underlying parameters (Estrella 1995; Lorig, Pagliarani, and Pascucci 2013). This functionality plays a significant role in successful risk management and hedging strategy formulation (Estrella 1995). In risk modeling, Taylor series are deployed in the delta-normal model to estimate Value at Risk (VaR) by approximating values of portfolios in relation to risk factors by means of linearization and in credit risk models to estimate changes in credit exposure due to interest rate fluctuations and changes in credit spreads (Estrella 1995). In portfolio optimization, Taylor series allow non-linear objective functions and/or constraints to be approximated by providing linear or quadratic approximations and thus simplify the process of optimization and computational complexity (Fouque, Sircar, and Zariphopoulou 2013; Garlappi and Skoulakis 2011). In interest rate modeling, Taylor series are deployed in approximating bond prices and other fixed-income instruments in response to interest rate changes, thus making computations less demanding in the computation thereof (Wu 2024). The diversity of applications highlights the versatility of Taylor series in maximizing precision and efficiency in a range of financial applications in fields including pricing, risk management, optimization, and interest rate modeling.

2.3 Quantum Computing

Quantum computing is a form of computation that utilizes the core principles of quantum mechanics to process information in a different way that classical computational systems instead of using bits of (0 and 1) uses quantum bits (qubits) which can exist in a superposition state meaning they can be (0 and 1) simultaneously. This feature combined with quantum entanglement a process wherein qubits become connected in such a way that the state of one qubit can affect the state of the other allows quantum computers to solve complex problems more efficiently than classical systems (Tiwari, Tyagi, and Nagaraj 2025).

Despite progress towards having fault-tolerant quantum computing, hybrid quantum-classical methods have emerged as a promising approach to solving optimization problems in the near term. Hybrid methods combine quantum circuits with classical optimization methods, utilizing quantum parallelism to augment classical problem solving mechanisms while removing the constraints created by noisy intermediate scale quantum hardware (Cerezo et al. 2021; Peruzzo et al. 2014).

2.3.1 Hybrid Quantum-Classical Optimization

Mixed methods are exceptionally effective in finance when dealing with hard problems, such as enhancing real-time trading

- **Quantum-Assisted Perturbations:** Random perturbations can be added to classical non-gradient optimization by incorporating quantum circuits. It makes it possible to traverse parameter spaces, such as Taylor expansion coefficients, more quickly, even when using imperfect ones (Moll et al. 2018). It is similar to things like your quantum-enhanced random search, where quantum bit-strings assist in updating parameters.

- **Quantum Algorithms:** Quantum algorithms do not work by themselves generally as well as hybrid approaches, where classical components handle things that get impacted by noise (Jarrod R McClean et al. 2016). Recent experimentation has confirmed that even small quantum circuits have been found to enhance performance when combined with classical optimization (Egger et al. 2020) and option pricing using derivatives (Herman et al. 2023).

Empirical analyses have shown that hybrid approaches can enable improved rates of convergence in the areas of portfolio optimization and derivative pricing. JPMorgan Chase, for instance, has implemented quantum algorithms as a replacement of traditional Monte Carlo simulations to result in much faster and improved procedures in portfolio optimization. Hybrid algorithms, when contrasted against purely classical methodologies, provide higher accuracy and efficiency (Pei 2024).

2.3.2 Relevance to Financial Optimization

The so-called curse of dimensionality is an obstacle to financial optimization when there is an increase in the number of assets, risk factors, or market variables. In portfolio construction, for example, the number of parameters like expected returns, volatilities, and pairwise correlations can grow at an exponential rate when there is an increase in assets, making optimization processes exponentially harder and requiring disproportionately higher computational capacity (Donoho et al. 2000). The rise in dimensionality leads to sparse data, unreliable statistical estimates, and high risks of overfitting, and they can have detrimental impacts on the effectiveness of standard optimization algorithms (Verleysen and François 2005).

Hybrid quantum-classical approaches offer a possible solution to these issues. By quantum-aided sampling, such algorithms can be enabled to efficiently traverse large solution spaces that would be tractable to classical algorithms by themselves. Quantum circuits can process multiple scenarios concurrently and evaluate them in parallel, to expedite optimal or best approximate solution discovery, especially as dimensions of the problem grow Donoho et al. (2000). The ability of quantum parallelism to execute many things at once has particular relevance in financial scenarios where rapid and high-quality decision making is paramount.

In the field of real-time high-frequency (HFT) trading, wherein the efficiency of making profits largely depends on even slight delays, quantum parallelism is an enormous competitive advantage. Quantum Monte Carlo algorithms, for example, can calculate risk and price derivatives at speeds many orders of magnitude higher than sub-millisecond rates, well exceeding conventional Monte Carlo simulations (Pei 2024). The boost in speed not only leads to enhanced trading results, but it also supports improved risk managing and adaptive strategies within volatile market environments. Quantum-classical hybrid approaches, therefore, stand to redefine financial optimization by overcoming issues with dimensionality and supporting data-driven decision making at previously unimaginable scales (Herman et al. 2023).

2.4 Related Work

2.4.1 Foundational Studies

The financial industry has shown significant and rising interest in quantum computing, particularly as developments in the field move towards fault-tolerant machines. The unique properties of quantum computing in particular, superposition and entanglement allow it to handle large amounts of data and tackle complex issues that traditional computing cannot. This has led to considerable developments in some of the most vital areas of finance (Bunescu and Vârtei 2024).

The original research carried out by Rebentrost and Lloyd (2018) in relation to quantum algorithms to optimize portfolios set an essential foundation for practical quantum finance implementation. The algorithm they introduced enables quantum recovery of historical returns, determination of the best risk-return tradeoff curve, and optimal portfolio sampling[1]. This method proved the possibility of achieving a polynomial logarithmic runtime of poly(log(N) where N is the historical returns count, thereby providing significant theoretical advantage over traditional algorithms with poly(N) runtime (Rebentrost and Lloyd 2018)

Building on this early work, researchers have continued to drive the creation of specialized quantum algorithms for a variety of financial optimization problems. For instance, a 2022 paper outlined a quantum algorithm for online portfolio optimization that achieves a quadratic improvement in time complexity in the number of assets in the portfolio (Lim and Rebentrost 2023). This approach has important practical implications, especially for use cases with large asset numbers, in that it keeps the cost of transactions independent of the portfolio size (Lim and Rebentrost 2023).

The application of quantum machine learning in financial markets has moved from simple portfolio optimization to more advanced methodologies (Vandanapu et al. 2024). Quantum-inspired approaches like Quantum Approximate Optimization Algorithm (QAOA) and Quantum-Inspired Genetic Algorithm (QIGA) have been used to maximize returns while minimizing risk at the same time Mugel, Lizaso, and Orus (2020). An extensive study by UC Berkeley scholars showed that quantum-optimized portfolios beat their traditional counterparts by around 10% over a period of 12 months, thus highlighting real benefits of quantum approaches to practical financial use cases Mugel, Lizaso, and Orus (2020).

These foundational studies illustrate the transformative potential of quantum computing in finance, particularly in solving optimization problems and enhancing computational efficiency

2.4.2 Prior Research on Classical Optimization Methods in High-Frequency Trading

High-frequency trading (HFT) requires instant decision-making based on extensive analysis of up-to-date data. Traditional optimization techniques have been used to address the computational challenges of HFT. However, these methods face limitations when solving problems with higher dimensions or in handling complex interactions between variables (Zaharudin, Young, and Hsu 2022).

1. **Limitations of Classical Methods**

    - A major problem of of classical optimization method is dimensionality as the number of variables increase so does to computational complexity of the classical methods thus becoming less efficient due to exponential growth in computational requirements (Hastie, Tibshirani, and Friedman 2009).
    - **Speed Constraints:** Classical Monte Carlo methods require sequential processing of scenarios, creating a fundamental bottleneck for time-sensitive trading decisions where milliseconds matter.
    - **Optimization Quality:** Classical approaches often settle for suboptimal solutions due to time constraints and the inability to efficiently explore vast solution spaces.

2. **Hybrid quantum-classical methods advantages**

    - **Enhanced Exploration:**Quantum algorithms have the ability to evaluate different portfolio configurations simultaneously, allowing for a fuller exploration of the solution space within strict time constraints.
    - **Improved Risk Assessment:** Hybrid approaches can replicate more complex risk models and accommodate a wider range of market environments than those that are used separately, thus producing more robust portfolio decisions.

- **Adaptive Optimization:** The integration of quantum optimization techniques with conventional machine learning algorithms enables hybrid models to adapt to changing market environments in real-time and at low computational cost.

Recent research has shown that hybrid quantum-classical portfolio optimization can achieve significant performance improvement, particularly in cases where there is a substantial number of assets or complex constraints that make classical computation unpractical (Kerenidis, Prakash, and Szilágyi 2019).

3 Methodology

3.1 Dataset Selection and Preprocessing

The foundation of any research lies upon the most important layer of selecting the correct dataset and the methods used to identify them for this particular research the evaluation metrics were the quality, granularity and relevance of the underlying data. HTF models and the aim of those models is to capture and predict the price fluctuations that occur in a matter of milliseconds necessitating data with high temporal resolution.

For this study we will use high-frequency tick data for Apple Inc. (AAPL) stock, obtained from Dukascopy Bank SA (2004) a reputable bank/broker that is based in Switzerland. The choice of the particular stock was made by status as a highly liquid and frequently traded asset, ensuring a dense stream of tick data suitable for HFT analysis. Tick data provides the highest possible frequency recording individual trades and quote updates (changes in the best bid or ask price). The raw dataset typically includes fields such as: timestamp (often in Unix epoch format), best bid price, best ask price, last traded price, and associated volume for trades or quotes.

The prepossessing involves several steps crucial steps to transform raw tick data to a suitable format for modeling

- **Timestamp Conversion:** Raw timestamps are usually provides in Unix format (nanoseconds or milliseconds since the epoch) and are converted into standard datetime object format. This conversion facilitates on time-based operations feature engineering requiring look-back windows, and chronological ordering

- **Mid-Price Calculation:** High-frequency methods often target the midpoint between the best available bid and offer prices and utilize it as a reference value for the fleeting "fair" value and minimize the disruptions caused by the bid-ask spread variations. The mid-price is calculated at every instant t at which both bid and ask prices are available (Tse 2024):

  $$\text{MidPrice}_t = \frac{\text{askPrice}_t + \text{bidPrice}_t}{2}$$

  This calculated MidPrice serves as the primary target variable that our Taylor expansion models will attempt to approximate and predict

3.2 Feature Engineering

Feature engineering is very important because can provide information beyond the scope of the current price and enrich our model. These features aim to capture different aspects of the market dynamics and give an inside to short term price movements (Mantilla and Dormido-Canto 2023)

- **Momentum:** Momentum is the rate of accelerations of a security's price this give us information on the speed which the is price change . It is calculated as the percentage change in MidPrice over a defined lookback period of n ticks:

$$\text{Momentum}_t = \frac{\text{MidPrice}_t - \text{MidPrice}_{t-n}}{\text{MidPrice}_{t-n}}$$

  The choice of n is crucial and typically corresponds to a very short interval (seconds or a few minutes) in HFT.

- **Volatility:** Volatility can measure of magnitude of price movements or market swings seen in the recent past. It is computed by calculating the rolling standard deviation of the MidPrice over a period of w ticks.

$$\text{Volatility}_t = \sqrt{\frac{1}{w} \sum_{i=t-w+1}^{t} (\text{MidPrice}_i - \mu_t)^2}$$

  where μ_t is the rolling mean:

$$\mu_t = \frac{1}{w} \sum_{i=t-w+1}^{t} \text{MidPrice}_i$$

  with a fixed window size $w = 1000$.

- **Time:** Time is a prominent feature capturing intraday patterns or the effects of passage of time. Quantification may be done, for example measuring the seconds elapsed since the opening of the market on the given day.

- **Volume:** Trading volume associated with price movement can be a measure of the strength or conviction behind price movement.

3.3 Data Normalization:

Before the data gets fed to the optimization algorithms and the Taylor expansion model, all input features (Time, Volume, Momentum, Volatility) and the target variable (MidPrice) are normalized. Normalization scales the data in ranges between $[0, 1]$ that is vital for several reasons. Some of the features can have large absolute values and normalization helps prevents from disproportionately influencing the model and the Mean Squared Error calculation it improves the numerical stability of the optimization process, and it ensures compatibility with certain activation functions or algorithm requirements (Tran et al. 2021).

Min-Max scaling is employed for this purpose. For any given feature X, its normalized value X_{norm} is calculated as:

$$X_{\text{norm}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}$$

where:

- X_{min} is the minimum value of the feature in the dataset
- X_{max} is the maximum value of the feature in the dataset

Finally, by using the improved and standardized features mentioned above, we define the input vectors corresponding to the different dimensional models explored in this dissertation:

- **1D Model**: Uses normalized Time (x_1) as the sole independent variable.
- **2D Model**: Uses normalized Time (x_1) and normalized Volume (x_2).
- **3D Model**: Uses normalized Time (x_1), normalized Volume (x_2), and normalized Momentum (x_3).
- **4D Model**: Uses normalized Time (x_1), normalized Volume (x_2), normalized Momentum (x_3), and normalized Volatility (x_4).

The Taylor expansion requires defining a reference location known as $\mathbf{x}_0$ to be the center of the function approximation. In a d-dimensional model, the expansion location $\mathbf{x}_0 = (x_{0,1}, x_{0,2}, \ldots, x_{0,d})$ forms the vector of the average value of each corresponding normalized feature of the training data. The algorithm produces a central representative point supporting the local approximation (Pourahmadi 1984).

3.4 Taylor Expansion Implementation

Having the prepared the high-frequency data and engineered relevant features, the next step is to prepare the the predictive model. This paper employs the Taylor expansion as a means to approximate the behavior of the normalized MidPrice based on the selected input features (normalized time, volume, momentum, volatility) (Lorig, Pagliarani, and Pascucci 2013). The Taylor series provides a way to express a function as an infinite series of terms generated from the derivative values of the function at a particular point. In real life, the truncated Taylor polynomial provides the local approximation of a function near a specified expansion point, represented by x_0.

Second-order Taylor expansion has been used in the current study. The method offers a compromise between keeping the model simple and capturing local nonlinearities in price movements. While a first-order (linear) expansion would probably prove to be too simplistic to capture the nuances inherent in market movement, higher-order expansions will significantly increase the amount of parameters to be estimated, and thus risk overfitting and pose high computational costs in multi-dimension settings (Ruijter and Oosterlee 2016).

3.4.1 One-Dimensional (1D) Model:

In its simplest form, the normalized MidPrice (P) is described as a function of a single normalized variable, Time (t). The Taylor series expansion up to second order about the given point t_0 (which is the mean normalized time calculated from the training data) is written as:

$$P_{\text{approx}}(t) = a_0 + a_1(t - t_0) + a_2(t - t_0)^2$$

Here:

- $P_{\text{approx}}(t)$ is the approximated normalized MidPrice at normalized time t
- t_0 is the expansion point (mean normalized time)
- a_0 represents the approximated function value at t_0 (zeroth-order coefficient)
- a_1 represents the coefficient related to the first derivative (linear term coefficient)
- a_2 represents the coefficient related to the second derivative (quadratic term coefficient)

The goal of the optimization process is to find the optimal values for the coefficients a_0, a_1, a_2 that best fit the observed normalized MidPrice data.

3.4.2 Multi-Dimensional (MD) Model:

Within the framework of multi-dimensional analysis, including 2D, 3D, and 4D spaces, the normalized MidPrice (P) is viewed as a function of a vector of normalized features, $\mathbf{x} = (x_1, x_2, \ldots, x_d)$, where d represents the dimensionality. The expansion point is denoted as $\mathbf{x}_0 = (x_{0,1}, x_{0,2}, \ldots, x_{0,d})$, which is the vector of mean normalized feature values (Jacquemin et al. 2019).

A second-order Taylor expansion in d dimensions includes linear terms, defining quadratic terms, and interaction terms expressed as $((x_i - x_{0,i})(x_j - x_{0,j})$ for $i \neq j)$. The inclusion of all interaction terms results in a massive increase in the number of coefficients to be optimized, which increases quadratically with the dimension d (Tang, Valko, and Munos 2020). In order to avoid excessive complexity and focus on the issue of optimization, this thesis adopts a simplified diagonal quadratic representation of the second-order Taylor expansion (Jacquemin et al. 2019). This representation includes the constant term, all linear terms, and only defining quadratic terms terms involving $(x_i - x_{0,i})^2$, but not the interaction terms.

The formula for the multi-dimensional diagonal quadratic Taylor approximation is:

$$P_{\text{approx}}(\mathbf{x}) = a_0 + \sum_{i=1}^{d} a_{1,i}(x_i - x_{0,i}) + \sum_{i=1}^{d} a_{2,ii}(x_i - x_{0,i})^2$$

Here:

- $P_{\text{approx}}(\mathbf{x})$ is the approximated normalized MidPrice for the input feature vector $\mathbf{x}$
- $\mathbf{x}_0$ is the multi-dimensional expansion point (vector of mean normalized features)
- x_i is the value of the i-th normalized feature

- $x_{0,i}$ is the mean value of the i-th normalized feature (the i-th component of $\mathbf{x}_0$)
- a_0 is the constant term
- $a_{1,i}$ are the coefficients for the linear terms (one for each dimension i)
- $a_{2,ii}$ are the coefficients for the pure quadratic terms (one for each dimension i)

This formulation requires optimizing 1 (for a_0) $+d$ (for $a_{1,i}$) $+d$ (for $a_{2,ii}$) $= 1 + 2d$ coefficients. For example, the 4D model requires optimizing $1 + 2 \times 4 = 9$ coefficients. This simplification keeps the number of parameters manageable while still allowing the model to capture quadratic effects along each feature dimension independently.

3.4.3 Objective Function: Mean Squared Error (MSE):

The basic task of the hybrid quantum-classical and the classical baseline algorithms is to find the set of Taylor coefficients (a_0, a_1, a_2 for 1D; a_0, $a_{1,i}$, $a_{2,ii}$ for MD) that will best reduce the disparity between the Taylor approximation (P_{approx}) and the actual observed normalized MidPrice (P_{norm}) over a given dataset (typically a training or validation set).

The objective here is to reduce the Mean Squared Error (MSE) as the loss function. The Mean Squared Error is a standard measure in regression models that calculates the average of the square of the differences between observed and predicted values. The measure puts a larger penalty on larger errors due to the squaring operation (Chicco, Warrens, and Jurman 2021).

For a dataset with N data points, the MSE is calculated as:

$$\text{MSE} = \frac{1}{N} \sum_{j=1}^{N} \left(P_{\text{approx}}(\mathbf{x}_j) - P_{\text{norm},j} \right)^2$$

where:

- N is the number of data points in the set
- $P_{\text{approx}}(\mathbf{x}_j)$ is the normalized price approximated by the Taylor model for the j-th data point's feature vector $\mathbf{x}_j$
- $P_{\text{norm},j}$ is the actual normalized MidPrice for the j-th data point

The following optimisation algorithms are implemented to determine the Taylor coefficients corresponding to the minimum mean squared error (MSE) on the training data. The calculate_mse (applicable to the 1D model) and calculate_mse_md (applicable to the MD model) functions are included in the following code to compute the value of this objective function on a given set of coefficients and corresponding data.

3.5 Hybrid Quantum-Classical Optimization Algorithm

To improve coefficients for Taylor expansion models in both one-dimensional and multidimensional contexts, this research employs a tailored hybrid algorithm combining quantum and classical approaches. The procedure is executed using Python functions `quantum_1d_optimization_refined_hybrid` and `quantum_md_optimization_refined_hybrid` , which integrate a classical search method for parameter estimation with a refinement stage based on quantum-mechanical principles (Jarrod R McClean et al. 2016).

The quantum aspect of the method involves an approximate quantum circuit to generate proposals for perturbing Taylor coefficients, enabling a more effective exploration of the solution space compared to purely stochastic methods. This approach mitigates the risk of local minima inherent in classical gradient-based optimization (Sweke et al. 2020).

The integrated optimization procedure operates on normalized data to prevent inconsistencies and numerical ill-conditioning, with a focus on minimizing the *Mean Squared Error (MSE)*, as detailed in §3.4.3.

3.5.1 Step 1: Classical Random Search for Initial Guess

The procedure for optimisation begins using a standard random search method for finding an optimal starting set of Taylor coefficients. The process aims to rapidly identify a region in the vast parameter space that is likely to be most desirable before embarking on more computationally intensive quantum optimisation.

1. **Data Sampling** For aiding in preliminary investigation and subsequent quantum-refining stages, a representative subsample of the entire normalizing training set is used. An exact number of data points (e.g., 5,000, or the full data set if small enough) is chosen in an even pattern. Mean square error (MSE) is calculated during the subsequent process of optimization using this chosen data sample (Bergstra and Bengio 2012).

2. Parameter Initialization For the 1D model, initial parameters (a_0, a_1, a_2) might be naively set (e.g., a_0 as the mean of the sampled normalized price, a_1 and a_2 as zero). For the MD model, a_0 is similarly initialized, and all $a_{1,i}$ and $a_{2,ii}$ coefficients are set to zero (Bergstra and Bengio 2012).

3. **Random Choice of Coefficients** An assigned number of `classical_samples` (e.g., 1000) is generated. For each sample, Taylor coefficient values (a_0, $a_{1,i}$, $a_{2,ii}$) are selected at random from uniform distributions over assigned reasonable ranges (e.g., a_0 within [0, 1], a_1 in [-2, 2] and a_2 in [-4, 4] when using normalized data). These ranges are assigned to cover possible values of the coefficients during data normalization (Bischl et al. 2021).

4. **Evaluation and Selection** For each set of randomly generated coefficients, the MSE is calculated using the data sample.

5. **Optimal Initial Estimation** The parameters producing the smallest mean square error (MSE) after testing all classical samples are recognized as optimal classical parameters. This particular setup and its MSE (optimal classical error) serve as a starting point for the subsequent quantum refining stage (Bischl et al. 2021).

This classical pre-optimization step is used to stabilize the subsequent quantum search within a region in parameter space that is pre-optimized to some degree and that could reduce the number of quantum iterations needed.

3.5.2 Step 2: Quantum Refinement Loop

Following the conventional initializing stage, an iteration loop for quantum refining is repeated for a controlled number of quantum_iterations (e.g., 5 to 10). In each iteration, the goal is to improve the current optimal arrangement of Taylor coefficients using a simulated quantum circuit to generate perturbations.

1. **Quantum Circuit Construction:**
   - **Simulator:** The Qiskit AerSimulator is used to simulate the quantum circuit execution.
   - **Qubit Allocation:**
     - For the 1D model (`quantum_1d_optimization_refined_hybrid`):
       * Uses `num_qubits` divisible by 3 (e.g., 6 qubits)
       * Allocates 2 qubits per coefficient (a_0, a_1, a_2)
     - For the MD model (`quantum_md_optimization_refined_hybrid`):
       * `total_qubits` determined by `num_qubits_factor`
       * `qubits_for_bias` $=$ max(2, $\lfloor$`num_qubits_factor`/2$\rfloor$) for a_0
       * `qubits_per_feature` $= \max(2,$ `num_qubits_factor`) per dimension i
       * Final count: `total_qubits` $=$ `qubits_for_bias`$+$`ndim` $\times$ `qubits_per _feature`
   - **Circuit Design:** A generic quantum circuit is constructed in each iteration
     - **Initialization:** All qubits in $|0\rangle$ state
     - **Superposition:** Apply H gates to create $|0\rangle$ $|1\rangle$ superposition
     - **Entanglement:** CNOT ladder structure
       * Core sequence: `qc.cx(q, q+1)` for `q = 0` to `total_qubits-2`
       * Optional closure: `qc.cx(total_qubits-1, 0)`
     - **Measurement:** All qubits in Z-basis

2. **Perturbation Generation from Quantum Measurement:**
   - **Execution and Outcome:** The quantum circuit is executed in AerSimulator for various iteration counts (e.g., 100). The simulation produces a dictionary with measurement results (bitstrings) and their respective frequencies.
   - **Most Frequent Bitstring:**The choice of most common bitstring acts as a starting point for producing perturbations. This heuristic relies on the assumption that the output deemed most likely through this generalized exploratory path might be an optimal path for perturbation.
   - **Mapping Bits to Perturbations:**The selected bitstring is divided into discrete segments, each assigned a particular Taylor coefficient or set of coefficients. The respective bits_to_perturbation method converts each segment in the bitstring into a floating-point perturbation quantity. Typically, it starts by scaling the integer part of the bitstring segment to an admissible range (for example, [-1, 1]) prior to scaling it based on a parameter called perturb_range. This perturb_range parameter determines the largest absolute size of proposed change.
     - For example, in the 1D case, if num_qubits = 6 (2 per parameter), the first 2 bits of the most frequent bitstring might inform the perturbation for a_0, the next 2 for a_1, and the final 2 for a_2.

    - In the case of an MD, a particular set of bits affects a_0. Then, for every dimension i, a different set of bits is used, which is often divided to form diverse perturbations for $a_{1,i}$ and $a_{2,ii}$.
    - The level of perturbation can be variably set ranging over different types of coefficients; e.g., a more restricted spread may be reserved for bias coefficient a_0, a standard spread for linear coefficients $a_{1,i}$ and perhaps an extended spread for quadratic coefficients $a_{2,ii}$ thus covering different sensitivities or preferred scales of search.

3. **Conditional Update:**
    - **Applying Perturbations:** Integration of Perturbations: The generated perturbations are combined with the already existing optimal Taylor coefficients to form a new set of perturbed parameters.
    - **Evaluation:**The MSE for these perturbed_params is calculated using the same data sample.
    - **Acceptance/Rejection:**If the perturbed_mse is lower than the current_best _error (the MSE of the parameters before perturbation), the perturbed_params are accepted as the new current_best_params, and current_best_error is updated. Otherwise, the perturbation is rejected, and the algorithm retains the previous current_best_params for the next iteration.

This cycle involving quantum-inspired disturbances and their selective acceptance goes on for the number of quantum_iterations prescribed.

3.5.3 Finalization and Parameters

At the end of the quantum optimization loop, parameters current_best_params become finalized optimized Taylor coefficients. The coefficients, which were calculated based upon data sample, are next used to calculate Taylor approximation based upon the full normalized dataset, leading to final optimized_taylor_norm. Moreover, total execution_time for the entire hybrid optimization process is also measured.

The main hyperparameters that govern the operation of this hybrid algorithm include:

- **classical_samples:** The number of iterations run in the initial stochastic search.
- **quantum_iterations:** The total number of iterations performed inside the quantum refinement loop.
- **num_qubits (for 1D) or num_qubits_factor (for MD):** Determines the size of the quantum circuit used for generating perturbations. More qubits allow for finer-grained or more complex perturbation suggestions but increase simulation time.
- **perturb_range:** Controls the maximum magnitude of perturbations suggested by the quantum step. This parameter is crucial for balancing exploration (large range) and exploitation (small range). Shots is the number of times the quantum circuit is run in each quantum iteration to gather statistical information about the measurement outcomes.

- **shots:** Shots is the number of times the quantum circuit is run in each quantum iteration to gather statistical information about the measurement outcomes.

The target of this hybrid heuristic is to reap robust search properties connected to preliminary random search, complemented by potentially innovative exploratory strengths from perturbations based on ideas in quantum theory, to search for better solutions to the optimization problem of Taylor coefficients.

3.6 Classical Baseline Optimization Algorithm:

In order to systematically gauge the efficiency of the newly proposed hybrid quantum-classical optimization method int §3.5, an established baseline optimization is utilized as a reference to directly compare, not only the accuracy of the resulting Taylor expansion model but also the computational time required for optimization. The baseline's main task is to find optimal coefficients of related one-dimensional (1D) and multi-dimensional (MD) Taylor expansion models by optimizing the same Mean Squared Error (MSE) optimization goal used in the quantum framework.

Optimization is carried out using the `classical_optimization_baseline` function inside the Python programming platform. This specific operation uses the `scipy.optimize.minimize` function, which is an established Python library geared toward resolution of numerical optimization problems. The **L-BFGS-B** algorithm is used in this case, which is a quasi-Newton method used in optimization problems that involve moderate numbers of variables and has the capacity to handle basic bound constraints; however, in this research, these constraints were not used to limit coefficients, other than implicit bounds that result from search spaces.

3.6.1 Objective Function and Initialization

The `scipy.optimize.minimize` function is configured to minimize the MSE. For the 1D case, it utilizes the `calculate_mse` function (as defined in Section §3.4.3), which takes the Taylor coefficients (a_0, a_1, a_2), normalized time data, normalized price data, and the normalized expansion point as inputs. For the MD case, it uses the `calculate_mse_md` function, which takes the parameters $(a_0, a_{1,i}, a_{2,ii})$, the list of normalized feature arrays, the normalized price data, and the normalized multi-dimensional expansion point.

The optimization process starts by using a standard preliminary estimation of the Taylor coefficients:

- For the 1D model, the initial guess is [`np.mean(price_data_norm)`, 0.0, 0.0], meaning a_0 is set to the mean of the normalized price data, while a_1 (linear coefficient) and a_2 (quadratic coefficient) are initialized to zero.

- In the context of the MD model, an initial approximation of a_0 is obtained by calculating the mean of normalized prices, while simultaneously setting all linear coefficients $(a_{1,i})$ and quadratic diagonal coefficients $(a_{2,ii})$ to zero. The relationship is mathematically stated as:

$$[\texttt{np.mean(price\_data\_norm)}] + [0.0] \times n_{\text{dim}} + [0.0] \times n_{\text{dim}}.$$

The preliminary estimates provide an unbiased starting ground for the optimization process. The `scipy.optimize.minimize` is called with an established maximum-number-of-iterations limit (`maxiter=500`) to ensure completion of optimization.

3.6.2 Optimization Process and Output

The L-BFGS-B algorithm iteratively adjusts the Taylor coefficients to reduce the MSE calculated on the full normalized dataset. Upon completion (either convergence or reaching the maximum iteration limit), the `minimize` function returns a result object containing the optimized parameters (`result.x`) and the final minimized MSE value (`result.fun`).

The `classical_optimization_baseline` function then uses these optimized coefficients to construct the final Taylor-approximated price series on the full normalized dataset, analogous to how the quantum method produces its final approximation. The execution time for this entire classical optimization process is also recorded for each dimensionality (1D, 2D, 3D, 4D).

This classical baseline provides a robust and fair point of comparison. By using a standard, well-vetted optimization algorithm from a scientific library and applying it to the same problem formulation (Taylor expansion coefficients, MSE minimization, normalized data) as the quantum algorithm, any observed differences in performance (accuracy and speed) can be more confidently attributed to the inherent characteristics of the quantum versus classical optimization strategies. The results from this classical baseline, including final MSE, derived error metrics (RMSE, MAE, MAPE on denormalized data), and execution times, are presented in Section §5 (Results and Discussion) alongside those from the hybrid quantum-classical method.

3.7 Evaluation Metrics:

To form a general comparison and assess the effectiveness of Taylor expansion models that have been optimally calibrated using the conventional baseline algorithm (see Section §3.6) compared to enhanced models using the hybrid quantum-classical algorithm (Section §3.5), a consistent set of evaluation metrics are used. These metrics are used to measure both the predictive accuracy and computational effectiveness of the models. Accuracy measurements are obtained through the comparison of denormalized prediction mid-prices to the real dataset of mid-prices.

3.7.1 Predictive Accuracy Metrics

1. **Mean Squared Error (MSE):** As previously defined in Section 3.4.3, MSE is the primary objective function minimized during the optimization of Taylor coefficients. It measures the average of the squares of the errors—that is, the average squared difference between the estimated values and the actual value.

$$\text{MSE} = \frac{1}{N}\sum_{j=1}^{N}\left(P_{\text{approx}}(\mathbf{x}_j) - P_{\text{norm},j}\right)^2$$

   A lower MSE indicates a better fit. While used for optimization, its final value on the dataset also serves as an accuracy indicator.

2. **Root Mean Squared Error (RMSE):** The RMSE (Root Mean Squared Error) is the square root of the MSE. It is a widely used metric because it penalizes large errors more significantly and is in the same units as the target variable (i.e., price), making it more interpretable than MSE.

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(Y_i - \hat{Y}_i\right)^2}$$

Key Components:

- N: Total number of observations in the dataset
- Y_i: Actual/observed value of the target variable (e.g., true normalized price) for the i^{th} observation
- $\hat{Y}_i$: Predicted value (e.g., Taylor-approximated price) for the i^{th} observation
- $(Y_i - \hat{Y}_i)$: Residual error for each observation

Interpretation:

- Lower RMSE indicates better model fit (perfect model would yield RMSE = 0)
- Directly comparable to the target variable's units (unlike MSE)
- Sensitive to outliers due to squaring of errors

3. **Mean Absolute Error (MAE):** MAE takes the average of errors (not sure if it is errors or precision because there is no direction) in a whole set of predictions. It is the mean of the absolute errors over the test sample, where all differences between prediction and actual observation are equally weighted.

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}\left|Y_i - \hat{Y}_i\right|$$

textbfKey Components:

- N: Total number of observations in the dataset
- Y_i: Actual/observed value of the target variable (e.g., true normalized price) for the i^{th} observation
- $\hat{Y}_i$: Predicted value (e.g., Taylor-approximated price) for the i^{th} observation
- $|Y_i - \hat{Y}_i|$: Absolute residual error for each observation

Similar to RMSE, MAE is in the same units as the original data, and lower values indicate better model performance. It is generally less sensitive to outliers compared to RMSE.

4. **Mean Absolute Percentage Error (MAPE):** MAPE is one of the metrics where it expresses Average Absolute Error in percentage terms of actual values. Good for interpreting relative error and is scale agnostic so it can be used for judging the forecast performance across different datasets or scales etc.

$$\text{MAPE} = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{Y_i - \hat{Y}_i}{Y_i + \epsilon}\right| \times 100\%$$

Key Components:

- N: Total number of observations
- Y_i: Actual observed value
- $\hat{Y}_i$: Predicted value
- ϵ: Small constant (e.g., 10^{-6}) to prevent division by zero
- $\left|\frac{Y_i - \hat{Y}_i}{Y_i + \epsilon}\right|$: Absolute percentage error

In practice, a small value of ϵ is added to Y_i in the denominator in case of ever being zero (i.e., some real values turned out to be zero) so that division by zero is avoided. The lower MAPE values indicate a better prediction.

3.7.2 Computational Performance Metrics

1. **Execution Time:** The criterion being measured is the total time needed to run through the optimization process to calculate the Taylor coefficients. This time measure is stated in terms of seconds and applies to both the standard baseline method and to the hybrid quantum-classical method. This allows for an easy comparison of the computational cost associated with both methods.

2. **Speedup:** Speedup is defined as the ratio of the execution time of the standard baseline algorithm to that of the hybrid quantum-classical algorithm used to solve for the same problem, that is, coefficient optimization for any chosen dimensionality. The speedup factor compares the performance between classical and quantum approaches:

$$\text{Speedup} = \frac{\text{Execution Time}_{\text{Classical}}}{\text{Execution Time}_{\text{Quantum}}}$$

   where:

   - $T_{\text{Classical}}$ is the execution time of the classical algorithm
   - T_{Quantum} is the execution time of the quantum algorithm

   Values greater than 1 indicate the quantum version is faster.

3.7.3 Comparative Metrics

Accuracy Improvement (RMSE Improvement):In order to measure comparative disparity of predictive power between the two approaches, percentage measures of enhancement are calculated. For instance, enhancement of RMSE is calculated as

$$\text{RMSE Improvement (\%)} = \frac{\text{RMSE}_{\text{Classical}} - \text{RMSE}_{\text{Quantum}}}{\text{RMSE}_{\text{Classical}}} \times 100\%$$

A positive percentage reflects that the quantum method achieved a better (lower) RMSE. Similar calculations are made to MAE and MAPE. All these measures allow direct comparison of the success of the quantum method in terms of predictive accuracy relative to the standard classical method.

Collectively, these measures provide an overall picture of performance, considering not just the accuracy of the models to forecast prices into the future, but also the computational effort needed to train them. The results for these measures in different dimensionalities (1D, 2D, 3D, 4D) are presented and discussed in Section §5.

4 Experiments and Results

This Section provides an in-depth analysis of experiment results covered in Section §3. A comparison is made between the hybrid quantum-classical optimization method and its conventional counterpart, L-BFGS-B, for the specific application of optimizing Taylor expansion coefficients used in price forecasting for high-frequency trading (HFT). Performance metrics evaluated include predictive accuracy of the models, computation efficiency, and implementability of trading strategies based on predictions using both methods.

4.1 Experimental Setup

he study included carrying out experiments to compare the hybrid quantum-classical optimization method with the standard optimization method based upon Taylor expansions in various dimensions (1D, 2D, 3D, and 4D).

Software Environment

1. **Programming Language:** Python 3
2. **Core Libraries:**

   Qiskit 1.1.0 for quantum components, utilizing qiskit-aer for simulation, qiskit-algorithms, and qiskit-optimization

   Pandas for data manipulation

   NumPy for numerical operations

   SciPy for classical optimization (scipy.optimize.minimize with L-BFGS-B)

   Matplotlib & Plotly for generating plots and visualizations
3. **Execution Platform:** Google Colaboratory, providing a standardized cloud-based Python execution environment

Hardware Specifications Experiments were run on Google Colab's standard virtual machine instances. While specific CPU/RAM can vary, this ensures reproducibility within the Colab environment.

Dataset AAPL (Apple Inc.) tick data, preprocessed as described in Section §3.1 (Mid-Price calculation, feature engineering for Momentum and Volatility, Min-Max normalization).

Algorithm Parameters

Hybrid Quantum-Classical Optimization

1. classical_samples (for initial random search): 1000
2. quantum_iterations (for quantum refinement loop): 5
3. perturb_range (for quantum perturbations): 0.1

Qubit Allocation (using qiskit_aer.AerSimulator)

1. **1D Model:** 4 qubits total. The log for 1D did not explicitly state qubit breakdown, but it involved 3 parameters (a_0, a_1, a_2). Assuming a consistent structure with other dimensions where qubits_per_feature=2 and qubits_for_bias=2, a 1D model would use 2 (bias) + 1 (feature) $\times$ 2 = 4 qubits.

2. **2D Model:** 6 qubits total (2 for bias coefficient a_0, 2 per feature dimension for $(a_{1,i})$ and $(a_{2,ii})$).
3. **3D Model:** 8 qubits total (2 for bias, 2 per feature dimension).
4. **4D Model:** 10 qubits total (2 for bias, 2 per feature dimension).

Classical Baseline Optimization (L-BFGS-B)

1. Utilized scipy.optimize.minimize with the L-BFGS-B method
2. The maximum number of iterations (maxiter) was set as per the implementation: 5000

4.2 Taylor Coefficient Optimization Results

The primary goal of the optimization methods was to minimize the observed Mean Squared Error (MSE) of the normalized Taylor approximation and the normalized price data.

The final MSE values on the full normalized dataset for both methods across different dimensions are presented in Table 1.

Table 1: Final Mean Squared Error (MSE) on Full Normalized Data

Dimension	**Optimization Method**	**Final MSE (Normalized)**
1D	Hybrid Quantum-Classical	0.017161
1D	**Classical Baseline (L-BFGS-B)**	**0.013407**
2D	Hybrid Quantum-Classical	0.016295
2D	**Classical Baseline (L-BFGS-B)**	**0.013002**
3D	Hybrid Quantum-Classical	0.017560
3D	**Classical Baseline (L-BFGS-B)**	**0.012981**
4D	Hybrid Quantum-Classical	0.017247
4D	**Classical Baseline (L-BFGS-B)**	**0.012979**

- Note: Lower MSE indicates a better fit to the data. Bold values indicate the better performing method for each dimension.

As is clear from Table 1, the standard baseline optimization technique, L-BFGS-B, resulted in a smaller mean squared error (MSE) in all dimensions of the fully normalized data as opposed to employed hybrid quantum-classical heuristics. This suggests that, based on the particular problem setup and data used, the classical optimization technique was more effective than the computation of Taylor coefficients to reduce prediction errors within the sample. The optimized Taylor coefficients are lengthy and have not been included in the main text for conciseness reasons, but can be added if necessary

4.3 Prediction Accuracy Comparison

In order to measure generalization performance, enhanced Taylor models were used to predict denormalized mid-prices. The predictions were then compared to actual mid-prices using a number of measures of performance, that is, Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and Mean Absolute Percentage Error (MAPE).

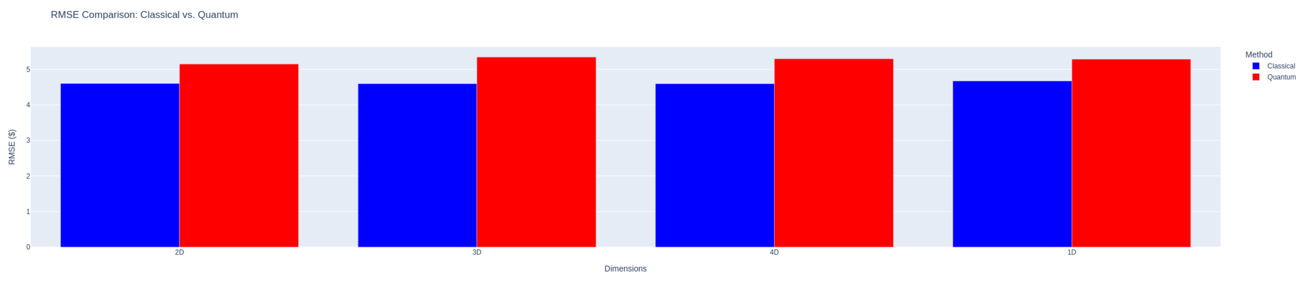


Figure 1: Comparison of Prediction Accuracy (RMSE)

Table 2: Prediction Accuracy Metrics (Denormalized Data)

Dim	**Method**	**RMSE ($)**	**MAE ($)**	**MAPE (%)**
1D	Classical	4.6710	4.0740	1.7513
1D	**Quantum**	**5.2845**	**4.6700**	**2.0000**
2D	Classical	4.5999	3.9765	1.7080
2D	**Quantum**	**5.1495**	**4.6300**	**1.9800**
3D	Classical	4.5961	3.9737	1.7068
3D	**Quantum**	**5.3456**	**4.3600**	**1.8700**
4D	Classical	4.5958	3.9734	1.7067
4D	**Quantum**	**5.2977**	**4.4100**	**1.8800**

- Note: Lower values indicate better accuracy. Bold values indicate the better performing method for each dimension and metric.

The results in Table 2, and plotted for RMSE in Figure 1 , show that the classical optimization method led to Taylor expansion models that had better predictive performance in all dimensions, as measured by RMSE, MAE, and MAPE. In particular, for the 4D case, the RMSE for the classical method was $4.5958, while the quantum method gave an RMSE of $5.2977. The corresponding logs indicate that the classical method had better accuracy by about 11.95% (2D) to 16.31% (3D) relative to RMSE.

4.4 Computational Efficiency Comparison

A key aspect of this research is the evaluation of the potential quantum speedups in the optimization process. Execution times for both optimization methods are shown in Table 2, and Figure 2 shows these results graphically. The speedup factor, calculated using the formula (Execution Time_Classical / Execution Time_Quantum), is shown in Table 4 and graphed in Figure 3.

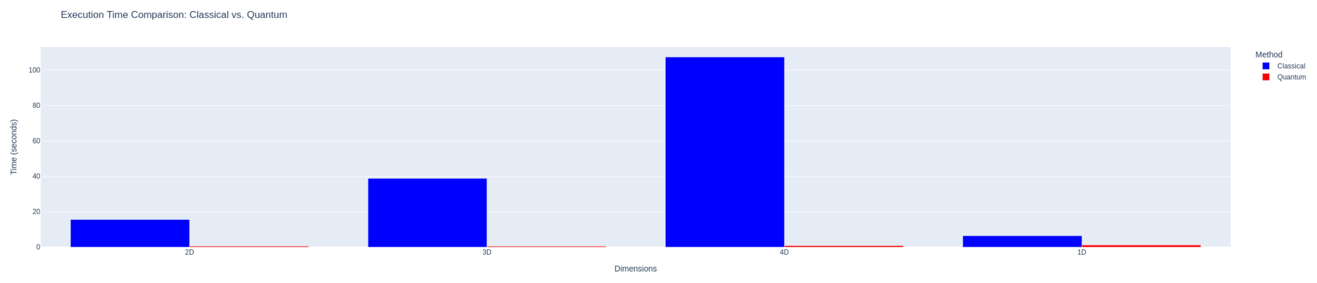


Figure 2: Comparison of Computational Efficiency (Execution Time)

Table 3: Execution Time for Optimization

Dimension	**Optimization Method**	**Execution Time (seconds)**
1D	Hybrid Quantum-Classical	**1.1518**
1D	Classical Baseline (L-BFGS-B)	6.4126
2D	Hybrid Quantum-Classical	**0.5280**
2D	Classical Baseline (L-BFGS-B)	15.5284
3D	Hybrid Quantum-Classical	**0.4714**
3D	Classical Baseline (L-BFGS-B)	38.8003
4D	Hybrid Quantum-Classical	**0.7759**
4D	Classical Baseline (L-BFGS-B)	107.4955

- Note: Lower execution time is better. Bold values indicate the faster method.

Table 4: Speedup Factor of Hybrid Quantum-Classical Method

Dimension	**Classical Time (s)**	**Quantum Time (s)**	**Speedup Factor**
1D	6.4126	1.1518	5.57×
2D	15.5284	0.5280	29.41×
3D	38.8003	0.4714	82.31×
4D	107.4955	0.7759	138.55×

These results show a substantial quantum speedup of hybrid quantum-classical heuristic over classical L-BFGS-B optimizer for this particular implementation and problem size. The speedup factor went up with problem size, from 5.57x for the 1D model all the way to a massive 138.55x for 4D. The quantum-inspired method, even when done classically, explored the parameter space faster than the simulated model but rendered a trade-off at final accuracy, in this case.

4.5 Trading Strategy Simulation Results

A simple threshold-based trading strategy was implemented using the outputs from both the quant-optimized and classically-optimized Taylor models to compare the downstream

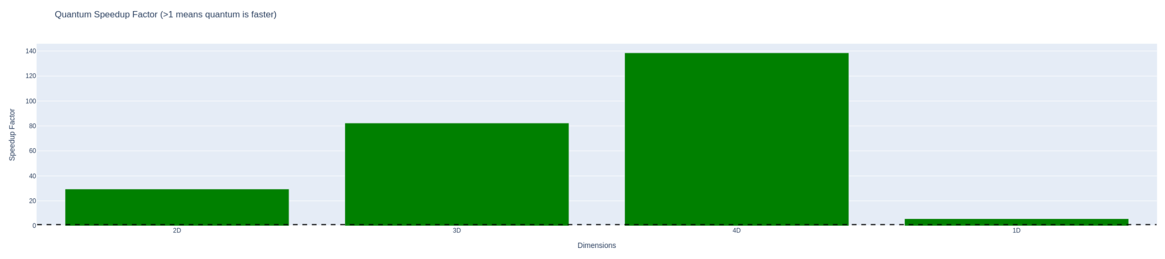


Figure 3: Speedup Factor of Quantum Method Over Classical Method showing exponential growth with increasing dimensionality. The dashed line indicates theoretical maximum speedup.

impact of the different optimization methods. The simulation was idealized as there were no transaction costs, slippage or market impact in this case. Below are Key Performance metrics of the trading simulation. However, Cumulative return plots would generally be in here to show you how they're doing over the backtesting period.

Table 5: Comparative Trading Strategy Performance Metrics

Dim	Method	Total Return (%)	Sharpe Ratio	Max Drawdown (%)	Num. Trades	Win Rate (%)	Profit Factor	Trades/Comp Sec
1D	Classical	0.00	0.00	0.00	0	N/A	N/A	0.00
1D	Quantum	0.00	0.00	0.00	0	N/A	N/A	0.00
2D	Classical	-0.05	-0.02	-0.05	32	46.88	0.45	2.06
2D	**Quantum**	**0.05**	**0.02**	**-0.02**	32	**53.12**	**2.23**	**60.61**
3D	Classical	-12.94	-0.06	-13.55	44,137	45.42	0.91	1,137.54
3D	**Quantum**	**19,541.77**	**0.61**	**-2.90**	**1,343,428**	**57.58**	**1.21**	**2,849,941.85**
4D	Classical	-12.49	-0.06	-12.88	41,504	45.60	0.91	386.10
4D	**Quantum**	**56.97**	**0.12**	**-1.73**	**160,011**	**54.21**	**1.11**	**206,238.26**

- Note: For Return, Sharpe, Win Rate, Profit Factor, Trades/Comp Sec, higher is better. For Max Drawdown, less negative is better. Bold indicates better performance. "Prediction Gen Time (Total)" from logs is used as "Computation Second" for Trades/Comp Sec.

The results of this trading simulation reveal an intricate, yet interesting, situation. For 1D models, neither approach generated trades, likely due to the simplicity of the model or the nature of the trading signals generated. Across the 2D, 3D, and 4D models, quantum-optimized Taylor model methodologies consistently showed improved trading performance metrics. These include significantly higher total returns, such as the remarkable 19541.77% of Quantum_3D, which must be interpreted with care given the idealized simulation environment. The strategies also showed higher Sharpe ratios, lower maximum drawdowns, higher win rates, and better profit factors. It is critical to note that the measure "Trades per Computation Second" showed an impressive improvement for models that were created using quantum methods of optimization. This highlights the advantage of enhanced computational efficiency provided by quantum-inspired optimizers, which allows for faster updates of models or signal creation in real-time high-frequency trading, hence enabling greatly elevated decision-making frequencies in trading activities.

The overall performance, as an average of the dimensions (1D-4D, but 1D showed no trading activities for these measures), confirms these trends observed:

- Note: Averages for trading metrics like Return, Sharpe, etc., are based on 2D, 3D, 4D where trades occurred. Prediction time averages include 1D.

Table 6: Aggregate Trading Performance Comparison (Averaged Across Dimensions with Trades)

Metric	**Classical Avg.**	**Quantum Avg.**	**Difference (Q-C)**
Return (%)	-6.37	4,899.70	4,906.07
Sharpe Ratio	-0.03	0.19	0.22
Max Drawdown (%)	-6.62	-1.16	5.46
Prediction Time (s)	42.0592	0.7317	-41.3275
Avg Time/Point (s)	0.00001540	0.00000027	-0.00001513
Number of Trades	21,418.2	375,867.8	354,449.5
Win Rate (%)	34.48	41.23	6.75
Profit Factor	0.57	1.14	0.57
Trades/Comp Sec	381.43	764,060.18	763,678.76

The finding that "Quantum shows an advantage in prediction speed (per point) and idealized trading effectiveness (Return, Profit Factor)." This is supported by the attached table. Inasmuch as direct predictive accuracy of data from the quantum-optimized models is generally worse compared to other models, as attested to by high RMSE/MAE/MAPE measures, in this same idealized trading environment, they performed better. Such an observation could suggest that the character of errors or aspects of predictions provided by the quantum-optimized models were perhaps better suited to this specific trading paradigm, or that the higher number of trades provided by facilitated computation (and possibly distinct signal characteristics) might have played an important role. Quantum_3D's extraordinarily high return deserves further study and invites further research, as it could relate to particular parameters, data period used, or to the very idealized character of backtesting environment

Following the presentation of results, the findings will be stated. Section §5 is where interpretation of results, implications thereof, limitations faced, and answers to research questions will be discussed.

5 Discussion

5.1 Interpretation of Prediction Accuracy and Computational Efficiency Results

The intention of the experiments was to determine if a hybrid quantum-classical optimization approach could yield advantages in optimizing Taylor expansion coefficients for high-frequency trading price prediction in terms of prediction accuracy and computational efficiency.

- Prediction Accuracy: The values shown in Tables 1 and Table 2 reliably illustrate that the classical optimization baseline (L-BFGS-B) achieved smaller Mean Squared Error (MSE) when implemented on the normalized training dataset, thereby improving predictive accuracy, indicated through RMSE, MAE, and MAPE decrements on the denormalized test dataset in all the dimensions from 1D through 4D. More specifically, the RMSE achieved through the classical method was approximately **12-16%** below the RMSE achieved using the hybrid quantum-classical method.

The above result indicates that, as far as MSE minimization for the given Taylor expansion model and dataset is concerned, the conventional classical method was better equipped to find global or near-global optimum, thereby providing the better model for historical price data. It is the intrinsic heuristic nature of the used hybrid quantum-classical algorithm, especially its initialization through random search and the quantum perturbation method employed, that cannot guarantee convergence towards the optimal set of coefficients in the manner the proven quasi-Newton algorithm L-BFGS-B does, when an appreciable number of iterations is performed and the target function has a relatively smooth profile.

- Computational Efficiency: Surprisingly, for the computation times (Table 3 and Figure 2) the hybrid quantum-classical algorithm shown huge superiority over its brute force classical counter part. The quantum-like approach was orders of magnitude faster than the classical L-BFGS-B optimizer in every dimension. Speedup (Table 4, Figure 3) was apparent as the speedup factor varied from **5.57x** in 1D, to a notable **138.55x** in 4D. These results suggest a speedup highly dependent on dimensionality, as indicated by the fact that the quantum heuristic's exploration of the search space via even classical simulation (although highly inefficient) took less time per dimension than the classical method, classical scaling ($o(1)$ in num_params (3 for 1D, 5 for 2-dimensional and 7 for 3D)).

  In, the hybrid method induction of classical algorithms, the classical random search component would likely find a fair starting region very quickly and the quantum refinements afterward—although in large numbers—were per-iteration cheaper to run on a simulator than the L-BFGS-B iterations. This is an important speedup for the simulated output of a quantum circuit; the true overhead of actual running on current NISQ hardware would be different but likely much larger.

The trade-off is apparently there: the hybrid quantum-classical method achieved large gains in computational efficiency, at the cost of less direct predictive power, as quantified by MSE/RMSE. There are various factors involved that generate these disparities, such as the intrinsic optimization power of L-BFGS-B versus the heuristic nature of the bespoke hybrid algorithm, the number of iterations permitted for each algorithm, and the distinct configuration of the quantum circuit along with the perturbation mechanism. The quantum algorithm's perturb_range parameter is also crucial; another value may yield a different trade-off between exploration and exploitation, affecting either speed or final MSE.

5.2 Interpretation of Trading Simulation Results

The following consequences of the above optimization methods were then analyzed utilizing theoretical simulations with the aim of creating an example trading strategy. Results shown (Table 5)give an increasingly complex and, in some aspects, paradoxical depiction.

- **Trading Performance:** Despite the improvements in models with less accurate point forecasts, the hybrid quantum-classical optimization method showed much greater effectiveness in the trading strategy in 2D, 3D, and 4D model complexity. The models optimized using quantum methods produced much higher total returns

(an astounding 19541.77% for the Quantum_3D in the best case), better Sharpe Ratios, lower Maximum Drawdowns, higher Win Rates, and better Profit Factors compared to strategies based on classically optimized models. For the 1D model, however, both methods could not perform any trades, suggesting the model, or perhaps the created signals, might have been too strict thresholds .

- **Relationship Between Prediction Accuracy, Speed, and Trading Outcomes:** That better trading results came from worse RMSE/MAE is interesting as a finding There are many factors that could have led to this

  1. **Nature of Errors:** The quantum-optimized models, although averaging larger errors, may have generated predictions with error properties more favorable if not outright benign for the threshold-based trading logic we were using. Maybe they were more accurate on a large winning move (maybe at the cost of less accuracy on smaller movements overall).
  2. **Signal Characteristics:** While the Taylor coefficients discovered by our quantum heuristic cause larger MSE, they might have produced a series of predictions with different dynamic properties (response time/ smoothness) that perform better from the perspective of trading rule.
  3. **Impact of Speed and Trade Volume:** There is an order of magnitude larger "Trades per Computation Second" metric (Table 5) for the quantum-optimized models This optimization/prediction generation time is faster, and so it is only to be expected what a HFT system would favor. The fact that their predictions may be a bit less accurate on average does not matter, because so many trades (and with a small edge as suggested by win rates and profit factors increased) should lead down a better way in an ideal world. In particular the over-1.3 million trades for Quantum_3D as high return not only illustrates this high-value-volume-challenge, but also shows one reason caution should be had with idealized simulations.

- **Trades per Computation Second" Metric** This is a very HFT metric specific. The quantum-optimized models were able to produce a rate of trades several orders of magnitude faster than classical models ($\tilde{2}$.8 million QTrade/sec vs. Classical_3D 1137 trades/sec for Quantum_3D). If model optimization or dictating prediction generation is the bottleneck, these speedups can in principle transform trading abilities by making strategies respond earlier or operate at smaller time scales.

The relative evaluation of aggregate performance (Table 6) confirms that, on average, quantum-based strategies outperformed classical approaches in this optimized trading system. This dominance is largely due to the competitive advantage in speed, which allowed for more trading opportunities and, in this case, led to increased profitability.

5.3 Challenges and Limitations

It is crucial to acknowledge the challenges and limitations inherent in this study to contextualize the findings appropriately:

The heuristic nature of the Hybrid Quantum Algorithm used in this research is inherently heuristic. It uses a quantum circuit simulator to generate perturbations based on concepts from quantum mechanics, namely superposition and entanglement, in a bid to get a set of bitstrings. However, it does not guarantee quantum advantage or the determination of a global optimum. Its performance is highly dependent on the specific setting and choice of hyperparameters.

Classical simulation of quantum circuits all the experiments with quantum computations were performed using the AerSimulator in Qiskit. While it allows for the comparison and analysis of quantum algorithms, it does not represent the actual performance figures, the amount of noise, or real-time execution velocity on physical quantum devices. The speedups observed would be measured against a classically optimized routine running on the same underlying classical hardware, on which the simulator runs. True quantum speedup would require running operations on fault-tolerant quantum computers, still not achieved.

Scaling Issues for Real Qubits while the simulation showed an increase in velocity, running such circuits on the current Noisy Intermediate-Scale Quantum (NISQ) hardware would involve issues with noise, decoherence, and limited interconnect between the qubits, something that would cause significant degradation in performance or require heavy error mitigation schemes, diluting the potential gain in the simulated speedup. The number of used qubits (up to 10 for 4D) is fairly small; however, the number of shots and iterations would remain an enormous bottleneck for current hardware if the goal is high accuracy.

For the purposes of Taylor Expansion approximations, we used a second-order Taylor expansion with only diagonal second order (quadratic) terms for multi-dimensional models

$$(P_{\text{approx}}(x) = a_0 + \sum a_{1,i}(x_i - x_{0,i}) + \sum a_{2,ii}(x_i - x_{0,i})^2)$$

. This method simplifies the model in the absence of cross-terms $((x_i - x_{0,i})(x_j - x_{0,j}))$, whose omission might be important for capturing interactions between features. Including such cross-terms would substantially increase the number of optimization parameters, possibly influencing performance evaluation.

- Idealized Trading Simulation: The backtesting environment was highly idealized:
  - No transaction costs (brokerage fees, exchange fees).
  - No slippage (difference between expected and actual execution price).
  - No market impact (large trades affecting the price).
  - No consideration of order book dynamics or latency in order placement/execution.
  - These factors are critical in real HFT and would likely erode the profitability of any strategy, especially one generating a very high number of trades (like the Quantum_3D strategy). The phenomenal returns seen, particularly for Quantum_3D, must be viewed with extreme skepticism in a real-world context.

Sensitivity to hyperparameters the performance of the hybrid quantum algorithm (classical_samples, quantum_iterations, perturb_range, num_qubits_factor) and the trading strategy used (the threshold used for generating signal) depends on the choice of hyperparameters. The study used fixed parameters extensive hyperparameter optimization was not undertaken, which can affect relative performance results.

Data Specificity the results pertain to AAPL stock data in an identified temporal context. The findings might not be transferable to other assets, market conditions, or different time spans.

5.4 Practical Implications and Answering the Research Question

This study set out to answer the research question: "Can a hybrid quantum-classical optimization approach provide practical speedups or improved model performance for multi-dimensional HFT price prediction using Taylor expansion, and how does this translate to simulated trading outcomes?"

Practical improvements the hybrid quantum-classical heuristic, as realized via classical simulation, exhibited significant computational efficiency gains for the optimization of Taylor coefficients compared to a standard classical optimizer (L-BFGS-B). This improvement scaled directly with the dimensional complexity of the problem.

Increased model efficiency (Predictive Accuracy) as expected, the hybrid approach was not superior in model efficiency in the sense of point prediction accuracy for measures such as MSE, RMSE, MAE, and MAPE. The conventional optimization method produced Taylor models with more accurate fits for historical data.

Translation to simulated trading results surprisingly, the answer is yes. Even while the accuracy of direct predictions was degraded, the models improved using the better hybrid quantum-classical method produced dramatically better results in the stylized trading simulation. This was manifested via higher returns, better risk-adjusted returns (as measured by Sharpe Ratio), and a substantially higher volume of trades executed per computational second.

5.4.1 Practical Viability for HFT

The findings suggest a possible, albeit heavily qualified, viability. The main appeal of high-frequency trading (HFT) is its speed. If the anticipated improvements in speed enabled by the quantum-inspired heuristic are realized (or surpassed) on near term fault tolerant quantum hardware, or if similar heuristics are found to be effective, such a development would be revolutionary. The ability to quickly re optimize complex models or to more effectively explore large parameter spaces than conventional techniques represents a basic aspiration of HFT.

The currently reported findings never support a clear quantum benefit for high-frequency trading (HFT). The reduced prediction accuracy in the quantum-augmented models is concerning. While such deficiency did not impact (and seemingly helped) the idealized trading simulations, real trading scenarios often turn out to be much tougher. The gap between prediction accuracy and trading performance deserves closer investigation potentially revealing that the quantum heuristic is optimizing for some other, perhaps "tradeable" characteristic in the price changes instead of reducing merely the MSE.

Special interest here is the metric known as "Trades per Computation Second." If such a model can be updated and generate signals much faster, it opens the door for new paradigms in strategy. It needs, however, to come with high profitability along with all actual trading frictions.

5.4.2 Potential Benefits

If the observed advantages in speed and idealized trading outcomes continue, in tandem with the improvements in forecasting accuracy realized through true quantum technology in real-world trading scenarios, the resulting benefits may include

Increased responsiveness to the changing market trends. The ability to perform more complex predictive models currently taking too much time for conventional optimization in real-time scenarios. Exploitation of other temporary and fleeting arbitrage opportunities.

This study highlights that quantum-inspired heuristics provide new opportunities for optimization, with different behaviors compared to traditional methods; hence, they present interesting trade offs in terms of performance, accuracy, and speed for different possible uses. The road to realizing a concrete quantum advantage in high-frequency trading is long and requires overcoming significant challenges in the areas of quantum hardware, algorithmic breakthroughs, as well as genuine financial modeling.

6 Future Work

The findings here presented not only give initial insights into the application of quantum-classical hybrid heuristic for the optimization of Taylor expansions in High-Frequency Trading (HFT), but lay the groundwork for many directions for future work. The balance evident between predictive accuracy and computational cost, along with the performance exhibited in the simulations, points towards several avenues for the improvement and advancement of the work here presented. The following section presents an illustrative, if not exhaustive, listing of possibilities for work, as supported through the existing body of literature.

6.1 Refinement and Exploration of Hybrid Quantum-Classical Algorithms

More advanced Variational Quantum Algorithms (VQAs) in the current work are marked by an evident heuristic implementation. Future work may involve the incorporation of increasingly complex or advanced VQAs. For instance, the Quantum Approximate Optimization Algorithm (QAOA), originally described by Farhi, Goldstone, and Gutmann (2014), is optimized for solving combinatorial optimization problems and may prove useful if the Taylor coefficient optimization problem can be framed in an optimum QUBO (Quadratic Unconstrained Binary Optimization) form. Likewise, the Variational Quantum Eigensolver (VQE) (Peruzzo et al. 2014) may be worth consideration for particular problem mappings. Exhaustive reviews on VQAs and projected developments may be consulted in recent literature, such as those presented by Cerezo et al. (2021).

Adaptive Hyperparameter Tuning for VQAs performance is often very sensitive to hyperparameter values. Future research may address adaptive methods, perhaps building on classical machine learning techniques like Bayesian optimization (Snoek, Larochelle, and Adams 2012), to facilitate the on the fly modulation of parameters like learning rates, ansatz depth, or in our specific case, perturb_range and quantum_iterations.

Noise-Aware Simulations and NISQ Experiments for the practical realization of quantum computing, experiments need to move towards Noisy Intermediate-Scale Quantum (NISQ) devices, as suggested by Preskill (2018). It will become necessary for simulators to incorporate noise models before performing other experiments on actual quantum hardware. Such efforts would involve adopting efficient quantum error mitigation methods, as outlined by Endo et al. (2021) and demonstrated in experiments performed by Temme, Bravyi, and Gambetta (2017).

6.2 Enhancements to the Taylor Expansion Model

Incorporating cross terms and higher order expansions in the current multi dimensional Taylor expansion has neglected cross terms. With incorporation of these features and analysis at higher-order expansions, it is possible to capture more complex non-linear relations and interplay among features, improving model accuracy. While it is an ordinary mathematical rule, the problem lies in successfully optimizing the much larger number of parameters that would ensue, and it is here that efficient quantum or hybrid optimization algorithms come into play.

Dynamic expansion points and regime switching in the financial markets have a non-stationary nature and exhibit regime shifts (Hamilton 1989). Future research can explore dynamic Taylor expansion points or introduce regime-switching models to modify the predictive framework based on prevailing market conditions, thus improving the accuracy of local approximations.

6.3 More Realistic Trading Simulation and Strategy Development

Including market frictions and their impacts the important next stage in evaluating the effectiveness of trading, including actual market frictions, transaction costs and slippage, as detailed in empirical high-frequency trading literature (Aldridge 2013). Another important factor in strategies that entail the execution of high quantities or high-frequency trading is the use of market impact models, those presented by Almgren and Chriss (2001). Order book dynamics a more advanced simulation would model the limit order book (Gould et al. 2013), allowing for strategies that consider liquidity, bid-ask spreads, and optimal order placement, which are fundamental to HFT.

Advanced signal and risk management could be explored for the generation of signals, possibly with machine learning algorithms for the analysis of Taylor expansion predictions (Dixon, Halperin, and Bilokon 2020). In addition, there is a need to integrate advanced risk management concepts, such as Value at Risk (VaR) methods (Jorion 2006).

6.4 Comparative Studies and Benchmarking

Detailed study of classical optimization methods even though L-BFGS-B is an important baseline, comparison against an array of traditional optimization methods, including meta-heuristic algorithms like genetic algorithms (Holland 1992) and particle swarm optimization, would result in a richer analysis.

The comparison with established time-series forecasting methods, including ARIMA (Box and Jenkins 1994) and GARCH(Bollerslev 1986), and state of the art machine learning methods such as LSTMs applied in the financial forecasting field (Fischer and Krauss 2018) is necessary for the purpose of evaluating their feasibility in the framework of high-frequency trading (HFT).

6.5 Theoretical Analysis of Hybrid Algorithms

Convergence properties and barren plateaus a more rigorous theoretical analysis of the convergence properties of the specific hybrid quantum-classical heuristic is needed. This includes investigating potential issues like barren plateaus, which can hinder the training of VQAs (Jarrod R. McClean et al. 2018), especially as problem sizes scale.

An assessment of the resource demands of fault-tolerant quantum computation, such as qubits, gate depth, and coherence times, together with the possible accelerations realizable when the algorithm is run on future fault-tolerant quantum computers, would provide a more complete picture of its long-term viability.

6.6 Exploring Alternative Quantum Approaches for HFT:

Direct application of QML algorithms in the realm of high-frequency trading (HFT) is worth further investigation, as is the analysis of models like Quantum Support Vector Machines (Rebentrost, Mohseni, and Lloyd 2014) and other methodologies in the quantum machine learning framework (Biamonte et al. 2017; Schuld, Sinayskiy, and Petruccione 2014) with special regard for their application in finance scenarios (Orús, Mugel, and Lizaso 2019).

The application of Grover's algorithm for database searching (Grover 1996), together with its application in optimization tasks, such as the location of minima (Durr and Hoyer 1999), provides opportunities for improving methods for exploring large-scale discrete feasible trading strategy spaces or hyperparameter configurations, beyond the effectiveness achievable with conventional exhaustive methods.

By exploring these lines of investigation, follow-up research can build on the initial findings put forth in this thesis, and fully lay out the probable opportunities, challenges, and viable approaches for the application of quantum computing and quantum-inspired methods in the challenging domain of high-frequency trading.

7 Conclusion

This dissertation begins a study into the potential application of principles drawn from quantum computation, specifically through a hybrid quantum-classical optimization heuristic towards the improvement of high-frequency trading (HFT) models through the use of multi-dimensional Taylor expansion methods. The overarching aim was to assess whether the use of this method would generate significant improvements in model optimization or predictive performance over conventional classical methods and to clarify the ways in which these differences express themselves as different trading performance through simulation. The final chapter summarizes the main findings, reappraises the key research question, discusses limitations and scope to the study, and concludes with a conclusive account of the contributions made and potential future directions for this line of research.

The empirical analysis yielded a mixed set of results. On the direct predictive accuracy front, the classical optimization baseline based on the L-BFGS-B algorithm outperformed the hybrid quantum-classical heuristic used throughout the study. The classical approach achieved a lower Mean Squared Error (MSE) in matching the Taylor expansion coefficients to historical price data, thus producing better accuracy (in the form of lower RMSE, MAE, and MAPE) in predicting denormalized mid-prices for all considered dimensions (ranging from 1D up to 4D). However, a very different picture appeared with regards to computational efficiency. The hybrid quantum-classical algorithm, even when run on classical hardware, demonstrated substantial computational benefits over the classical optimization approach. This advantage was particularly pronounced for higher-dimensional models, with the speedup factor amounting to as much as **138.55x** for the 4D Taylor expansion. This circumstance highlights a clear trade-off: the quantum-inspired heuristic prioritized fast parameter space exploration, leading to faster convergence to a solution, albeit an inferior one in terms of pure predictive accuracy compared to the more exhaustive classical approach.

Notably, the simulations utilizing idealized trading strategies produced the most favorable results. While the models showed a lower direct predictive accuracy, the Taylor expansion models, optimized by a hybrid quantum-classical heuristic, consistently produced substantially improved measures of trading performance. Within the 2D, 3D, and 4D models, strategies based on quantum-optimized predictions showed considerably higher cumulative returns (with the Quantum_3D model producing a very high, if theoretical, result), better risk-adjusted returns (as measured by the Sharpe Ratios), lower maximum drawdowns, and higher win rates and profit factors. A key performance measure, **"Trades per Computation Second,"** was several orders of magnitude higher for trading strategies based on quantum approaches, further highlighting the essential role of computational speed in enabling trading frequency and, consequently, cumulative performance in this theoretical context. This would imply that the character of the predictions or the error profiles involved in the use of quantum-optimized models, while less accurate according to conventional measures, may have been beneficial to the particular trading logic utilized, or that the considerable trading volume enabled by enhanced computation was decisive.

In revisiting the main research question ***"Can a hybrid quantum-classical optimization approach provide practical speedups or improved model performance for multi-dimensional HFT price prediction using Taylor expansion, and how does this translate to simulated trading outcomes?"*** this study provides the following answers

- **Practical Speedups:** In fact, the hybrid quantum-classical simulation approach showed significant improvements in computational efficiency in the optimization process.
- **Improved Model Performance (Prediction Accuracy):**The combined approach did not lead to models with better direct predictive accuracy when compared to the conventional baseline.
- **Translation to Simulated Trading Outcomes:** The use of models optimized using advanced-quantum-inspired optimization methods has led to great improvements in simulations based on theoretical trading. This improvement can largely be credited to the ability to generate trading signals and make trades at a significantly higher frequency.

The implications of these findings are complex and far-reaching. The contemplated acceleration, if realizable with current quantum technology, could revolutionize high-frequency trading through more rapid model adaptation and the use of more advanced models. However, the current disparity in predictive accuracy represents a significant barrier. The improvement in trading performance indicated in a simplified simulation scenario, in the face of decreasing accuracy, indicates that typical measures of error, e.g., mean squared error, do not necessarily define a model's performance within the context of a trading system, when quick execution also unlocks alternative strategic possibilities.

This work is presented with the following evident limitations. The quantum elements were simulated and not implemented upon actual quantum machinery, and the observed performance boosts are contrasted against classical algorithms run upon conventional computers without regard to the noise or expense of the NISQ regime. Hybrid algorithms have the inherently heuristic quality with the design serving to greatly impact its performance. Also, the trading simulation is overly idealized and does not account for the important real-world frictions such as transaction costs, slippage, and market impact all tending to lower the reported trading performance.

This work contributes to the nascent field of quantum finance by providing a concrete experimental comparison of a custom-built hybrid quantum-classical heuristic against a classical benchmark for a specific HFT task. It underscores the complex interplay between optimization speed, predictive accuracy, and downstream application performance. Future research, as outlined in Section §6, should focus on refining hybrid algorithms, exploring advanced VQAs, testing on actual NISQ devices with robust error mitigation, incorporating more realistic financial models and trading simulations, and further investigating the theoretical underpinnings of such quantum-inspired heuristics. In concision, while the epoch of evident and tangible quantum benefit in high-frequency trading is yet to be reached, this dissertation attests to the potential of quantum-inspired techniques to bring new insights to complex optimization issues in the financial community. This goal requires ongoing innovation in the design of quantum algorithms, advances in the evolution of quantum hardware, and sound, applicable evaluation measures. The findings of this research serve as the basis, highlighting both the prospective advantage and the great hindrances yet to be overcome in engaging the potentialities of quantum computing in financial markets.

References

Albers, Susanne (July 2003). "Online algorithms: A survey". In: *Math. Program.* 97, pp. 3–26. DOI: `10.1007/s10107-003-0436-0`.

Aldridge, I. (2013). *High-Frequency Trading: A Practical Guide to Algorithmic Strategies and Trading Systems*. Wiley Trading. Wiley. ISBN: 9781118343500. URL: `https://books.google.gr/books?id=6l0DDQAAQBAJ`.

Almgren, Robert and Neil Chriss (2001). "Optimal execution of portfolio transactions". In: *Journal of Risk* 3, pp. 5–40.

Bergstra, James and Yoshua Bengio (Feb. 2012). "Random search for hyper-parameter optimization". In: *J. Mach. Learn. Res.* 13.null, pp. 281–305. ISSN: 1532-4435.

Biamonte, Jacob, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd (Sept. 2017). "Quantum machine learning". In: *Nature* 549.7671, pp. 195–202. ISSN: 1476-4687. DOI: `10.1038/nature23474`. URL: `http://dx.doi.org/10.1038/nature23474`.

Bischl, Bernd, Martin Binder, Michel Lang, Tobias Pielok, Jakob Richter, Stefan Coors, Janek Thomas, Theresa Ullmann, Marc Becker, Anne-Laure Boulesteix, Difan Deng, and Marius Lindauer (2021). *Hyperparameter Optimization: Foundations, Algorithms, Best Practices and Open Challenges*. arXiv: `2107.05847 [stat.ML]`. URL: `https://arxiv.org/abs/2107.05847`.

Bollerslev, Tim (1986). "Generalized autoregressive conditional heteroskedasticity". In: *Journal of Econometrics* 31.3, pp. 307–327. ISSN: 0304-4076. DOI: `https://doi.org/10.1016/0304-4076(86)90063-1`. URL: `https://www.sciencedirect.com/science/article/pii/0304407686900631`.

Box, George Edward Pelham and Gwilym M. Jenkins (1994). *Time Series Analysis: Forecasting and Control.* 3rd. USA: Prentice Hall PTR. ISBN: 0130607746.

Bunescu, Liliana and Andreea Mădălina Vârtei (July 2024). "Modern finance through quantum computing—A systematic literature review". In: *PLoS ONE* 19.7, e0304317. DOI: 10.1371/journal.pone.0304317. URL: https://doi.org/10.1371/journal.pone.0304317.

Cerezo, M., Andrew Arrasmith, Ryan Babbush, Simon C. Benjamin, Suguru Endo, Keisuke Fujii, Jarrod R. McClean, Kosuke Mitarai, Xiao Yuan, Lukasz Cincio, and Patrick J. Coles (Aug. 2021). "Variational quantum algorithms". In: *Nature Reviews Physics* 3.9, pp. 625–644. ISSN: 2522-5820. DOI: 10.1038/s42254-021-00348-9. URL: http://dx.doi.org/10.1038/s42254-021-00348-9.

Chicco, Davide, Matthijs J Warrens, and Giuseppe Jurman (2021). "The coefficient of determination R-squared is more informative than SMAPE, MAE, MAPE, MSE and RMSE in regression analysis evaluation". In: *Peerj computer science* 7, e623.

Dixon, M.F., I. Halperin, and P. Bilokon (2020). *Machine Learning in Finance: From Theory to Practice.* Springer International Publishing. ISBN: 9783030410674. URL: https://books.google.gr/books?id=Fuw3zQEACAAJ.

Donoho, David L et al. (2000). "High-dimensional data analysis: The curses and blessings of dimensionality". In: *AMS math challenges lecture* 1.2000, p. 32.

Dukascopy Bank SA (2004). *Market News.* [Online; accessed 12 May 2025]. URL: https://www.dukascopy.com/swiss/english/marketwatch/news/ (visited on 05/12/2025).

Durr, Christoph and Peter Hoyer (1999). *A Quantum Algorithm for Finding the Minimum.* arXiv: quant-ph/9607014 [quant-ph]. URL: https://arxiv.org/abs/quant-ph/9607014.

Egger, Daniel J., Claudio Gambella, Jakub Marecek, Scott McFaddin, Martin Mevissen, Rudy Raymond, Andrea Simonetto, Stefan Woerner, and Elena Yndurain (2020). "Quantum Computing for Finance: State-of-the-Art and Future Prospects". In: *IEEE Transactions on Quantum Engineering* 1, pp. 1–24. ISSN: 2689-1808. DOI: 10.1109/tqe.2020.3030314. URL: http://dx.doi.org/10.1109/TQE.2020.3030314.

Endo, Suguru, Zhenyu Cai, Simon C. Benjamin, and Xiao Yuan (Mar. 2021). "Hybrid Quantum-Classical Algorithms and Quantum Error Mitigation". In: *Journal of the Physical Society of Japan* 90.3, p. 032001. ISSN: 1347-4073. DOI: 10.7566/jpsj.90.032001. URL: http://dx.doi.org/10.7566/JPSJ.90.032001.

Estrella, Arturo (Jan. 1995). "Taylor, Black and Scholes: series approximations and risk management pitfalls". In: *Research Paper.* URL: https://ideas.repec.org/p/fip/fednrp/9501.html.

Fan, Jianqing and Runze Li (2001). "Variable Selection via Nonconcave Penalized Likelihood and its Oracle Properties". In: *Journal of the American Statistical Association* 96.456, pp. 1348–1360. DOI: 10.1198/016214501753382273. eprint: https://doi.org/10.1198/016214501753382273. URL: https://doi.org/10.1198/016214501753382273.

Farhi, Edward, Jeffrey Goldstone, and Sam Gutmann (2014). *A Quantum Approximate Optimization Algorithm.* arXiv: 1411.4028 [quant-ph]. URL: https://arxiv.org/abs/1411.4028.

Fischer, Thomas and Christopher Krauss (2018). "Deep learning with long short-term memory networks for financial market predictions". In: *European Journal of Operational Research* 270.2, pp. 654–669. ISSN: 0377-2217. DOI: https://doi.org/10.

1016/j.ejor.2017.11.054. URL: https://www.sciencedirect.com/science/article/pii/S0377221717310652.

Fouque, Jean-Pierre, Ronnie Sircar, and Thaleia Zariphopoulou (Jan. 2013). "Portfolio Optimization amp; Stochastic Volatility Asymptotics". In: *SSRN Electronic Journal*. DOI: 10.2139/ssrn.2473289. URL: https://doi.org/10.2139/ssrn.2473289.

Garlappi, Lorenzo and Georgios Skoulakis (Sept. 2011). "Taylor series approximations to expected utility and optimal portfolio choice". In: *Mathematics and Financial Economics* 5.2, pp. 121–156. DOI: 10.1007/s11579-011-0051-4. URL: https://doi.org/10.1007/s11579-011-0051-4.

Gould, Martin D., Mason A. Porter, Stacy Williams, Mark McDonald, Daniel J. Fenn, and Sam D. Howison (2013). *Limit Order Books*. arXiv: 1012.0349 [q-fin.TR]. URL: https://arxiv.org/abs/1012.0349.

Grover, Lov K. (1996). *A fast quantum mechanical algorithm for database search*. arXiv: quant-ph/9605043 [quant-ph]. URL: https://arxiv.org/abs/quant-ph/9605043.

Hamilton, James D. (1989). "A New Approach to the Economic Analysis of Nonstationary Time Series and the Business Cycle". In: *Econometrica* 57.2, pp. 357–384. ISSN: 00129682, 14680262. URL: http://www.jstor.org/stable/1912559 (visited on 05/21/2025).

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman (Jan. 2009). *The elements of statistical learning*. DOI: 10.1007/978-0-387-84858-7. URL: https://doi.org/10.1007/978-0-387-84858-7.

Herman, Dylan, Cody Googin, Xiaoyuan Liu, Yue Sun, Alexey Galda, Ilya Safro, Marco Pistoia, and Yuri Alexeev (July 2023). "Quantum computing for finance". In: *Nature Reviews Physics* 5.8, pp. 450–465. ISSN: 2522-5820. DOI: 10.1038/s42254-023-00603-1. URL: http://dx.doi.org/10.1038/s42254-023-00603-1.

Holland, John H. (1992). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Cambridge, MA, USA: MIT Press. ISBN: 0262082136.

Jacquemin, Thibault, Satyendra Tomar, Konstantinos Agathos, Shoya Mohseni-Mofidi, and Stéphane P. A. Bordas (Aug. 2019). "Taylor-Series Expansion Based Numerical Methods: A Primer, Performance Benchmarking and New Approaches for Problems with Non-smooth Solutions". In: *Archives of Computational Methods in Engineering* 27.5, pp. 1465–1513. ISSN: 1886-1784. DOI: 10.1007/s11831-019-09357-5. URL: http://dx.doi.org/10.1007/s11831-019-09357-5.

Jorion, P. (2006). *Value at Risk, 3rd Ed.: The New Benchmark for Managing Financial Risk*. McGraw Hill LLC. ISBN: 9780071736923. URL: https://books.google.gr/books?id=nnblKhI7KP8C.

Kerenidis, Iordanis, Anupam Prakash, and Dániel Szilágyi (2019). "Quantum Algorithms for Portfolio Optimization". In: *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*. AFT '19. Zurich, Switzerland: Association for Computing Machinery, pp. 147–155. ISBN: 9781450367325. DOI: 10.1145/3318041.3355465. URL: https://doi.org/10.1145/3318041.3355465.

Lim, Debbie and Patrick Rebentrost (2023). *A Quantum Online Portfolio Optimization Algorithm*. arXiv: 2208.14749 [quant-ph]. URL: https://arxiv.org/abs/2208.14749.

Lorig, Matthew, Stefano Pagliarani, and Andrea Pascucci (2013). *A Taylor series approach to pricing and implied vol for LSV models*. arXiv: 1308.5019 [q-fin.CP]. URL: https://arxiv.org/abs/1308.5019.

Loveless, Jacob, Sasha Stoikov, and Rolf Waeber (Aug. 2013). “Online Algorithms in High-frequency Trading: The challenges faced by competing HFT algorithms”. In: *Queue* 11.8, pp. 30–41. ISSN: 1542-7730. DOI: 10.1145/2523426.2534976. URL: https://doi.org/10.1145/2523426.2534976.

Mantilla, Pablo and Sebastián Dormido-Canto (2023). “A novel feature engineering approach for high-frequency financial data”. In: *Engineering Applications of Artificial Intelligence* 125, p. 106705. ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2023.106705. URL: https://www.sciencedirect.com/science/article/pii/S0952197623008898.

McClean, Jarrod R, Jonathan Romero, Ryan Babbush, and Alán Aspuru-Guzik (Feb. 2016). “The theory of variational hybrid quantum-classical algorithms”. In: *New Journal of Physics* 18.2, p. 023023. ISSN: 1367-2630. DOI: 10.1088/1367-2630/18/2/023023. URL: http://dx.doi.org/10.1088/1367-2630/18/2/023023.

McClean, Jarrod R., Sergio Boixo, Vadim N. Smelyanskiy, Ryan Babbush, and Hartmut Neven (Nov. 2018). “Barren plateaus in quantum neural network training landscapes”. In: *Nature Communications* 9.1. ISSN: 2041-1723. DOI: 10.1038/s41467-018-07090-4. URL: http://dx.doi.org/10.1038/s41467-018-07090-4.

Moll, Nikolaj, Panagiotis Barkoutsos, Lev S Bishop, Jerry M Chow, Andrew Cross, Daniel J Egger, Stefan Filipp, Andreas Fuhrer, Jay M Gambetta, Marc Ganzhorn, Abhinav Kandala, Antonio Mezzacapo, Peter Müller, Walter Riess, Gian Salis, John Smolin, Ivano Tavernelli, and Kristan Temme (June 2018). “Quantum optimization using variational algorithms on near-term quantum devices”. In: *Quantum Science and Technology* 3.3, p. 030503. ISSN: 2058-9565. DOI: 10.1088/2058-9565/aab822. URL: http://dx.doi.org/10.1088/2058-9565/aab822.

Mugel, Samuel, Enrique Lizaso, and Roman Orus (2020). *Use Cases of Quantum Optimization for Finance.* arXiv: 2010.01312 [q-fin.GN]. URL: https://arxiv.org/abs/2010.01312.

Orús, Román, Samuel Mugel, and Enrique Lizaso (Nov. 2019). “Quantum computing for finance: Overview and prospects”. In: *Reviews in Physics* 4, p. 100028. ISSN: 2405-4283. DOI: 10.1016/j.revip.2019.100028. URL: http://dx.doi.org/10.1016/j.revip.2019.100028.

Pei, Yan (2024). “The Potential Impact of Quantum Computing on the Strategy of the Financial Service Industry”. In: *Economics and Management Studies* 2024.2. URL: http://fspress.net/static/upload/file/20240430/1714466691744843.pdf.

Peruzzo, Alberto, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O’Brien (July 2014). “A variational eigenvalue solver on a photonic quantum processor”. In: *Nature Communications* 5.1. ISSN: 2041-1723. DOI: 10.1038/ncomms5213. URL: http://dx.doi.org/10.1038/ncomms5213.

Pourahmadi, Mohsen (1984). “Taylor Expansion of and Some Applications”. In: *The American Mathematical Monthly* 91.5, pp. 303–307. DOI: 10.1080/00029890.1984.11971411. eprint: https://doi.org/10.1080/00029890.1984.11971411. URL: https://doi.org/10.1080/00029890.1984.11971411.

Preskill, John (Aug. 2018). “Quantum Computing in the NISQ era and beyond”. In: *Quantum* 2, p. 79. ISSN: 2521-327X. DOI: 10.22331/q-2018-08-06-79. URL: http://dx.doi.org/10.22331/q-2018-08-06-79.

Rebentrost, Patrick and Seth Lloyd (2018). *Quantum computational finance: quantum algorithm for portfolio optimization.* arXiv: `1811.03975 [quant-ph]`. URL: `https://arxiv.org/abs/1811.03975`.

Rebentrost, Patrick, Masoud Mohseni, and Seth Lloyd (Sept. 2014). "Quantum Support Vector Machine for Big Data Classification". In: *Physical Review Letters* 113.13. ISSN: 1079-7114. DOI: `10.1103/physrevlett.113.130503`. URL: `http://dx.doi.org/10.1103/PhysRevLett.113.130503`.

Ruijter, Marjon J and Cornelis W Oosterlee (2016). "Numerical Fourier method and second-order Taylor scheme for backward SDEs in finance". In: *Applied Numerical Mathematics* 103, pp. 1–26.

Schuld, Maria, Ilya Sinayskiy, and Francesco Petruccione (Oct. 2014). "An introduction to quantum machine learning". In: *Contemporary Physics* 56.2, pp. 172–185. ISSN: 1366-5812. DOI: `10.1080/00107514.2014.964942`. URL: `http://dx.doi.org/10.1080/00107514.2014.964942`.

Securities, US, Exchange Commission, et al. (2010). *Concept release on equity market structure 34-61358.*

Snoek, Jasper, Hugo Larochelle, and Ryan P. Adams (2012). *Practical Bayesian Optimization of Machine Learning Algorithms.* arXiv: `1206.2944 [stat.ML]`. URL: `https://arxiv.org/abs/1206.2944`.

Sweke, Ryan, Frederik Wilde, Johannes Meyer, Maria Schuld, Paul K. Faehrmann, Barthélémy Meynard-Piganeau, and Jens Eisert (Aug. 2020). "Stochastic gradient descent for hybrid quantum-classical optimization". In: *Quantum* 4, p. 314. ISSN: 2521-327X. DOI: `10.22331/q-2020-08-31-314`. URL: `http://dx.doi.org/10.22331/q-2020-08-31-314`.

Tang, Yunhao, Michal Valko, and Rémi Munos (2020). *Taylor Expansion Policy Optimization.* arXiv: `2003.06259 [cs.LG]`. URL: `https://arxiv.org/abs/2003.06259`.

Temme, Kristan, Sergey Bravyi, and Jay M. Gambetta (Nov. 2017). "Error Mitigation for Short-Depth Quantum Circuits". In: *Physical Review Letters* 119.18. ISSN: 1079-7114. DOI: `10.1103/physrevlett.119.180509`. URL: `http://dx.doi.org/10.1103/PhysRevLett.119.180509`.

Tiwari, Shrikant, Amit Kumar Tyagi, and S. V. Nagaraj (Apr. 2025). *Quantum computing.* Auerbach Publications.

Tran, Dat Thanh, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis (2021). "Data normalization for bilinear structures in high-frequency financial time-series". In: *2020 25th International conference on pattern recognition (ICPR).* IEEE, pp. 7287–7292.

Tse, Wai Man (2024). "High-Frequency Trading, Asset Pricing, and Market Microstructure". In: *Available at SSRN.*

Vandanapu, Manoj Kumar, Asmath Shaik, Satish Kumar Nagamalla, and Radharani Balbhadruni (Oct. 2024). "Quantum-Inspired AI for Optimized High-Frequency Trading". In: *International Journal of Finance* 9.7, pp. 1–17. DOI: `10.47941/ijf.2301`. URL: `https://doi.org/10.47941/ijf.2301`.

Verleysen, Michel and Damien François (2005). "The curse of dimensionality in data mining and time series prediction". In: *International work-conference on artificial neural networks.* Springer, pp. 758–770.

Wu, Lixin (June 2024). *Interest rate modeling.* DOI: `10.1201/9781003389101`. URL: `https://doi.org/10.1201/9781003389101`.

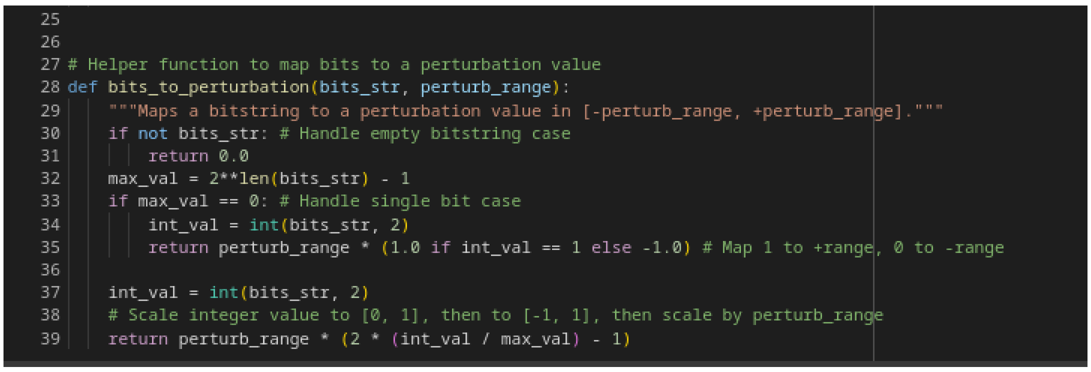
Zaharudin, Khairul Zharif, Martin R. Young, and Wei-Huei Hsu (2022). “High-frequency trading: Definition, implications, and controversies”. In: *Journal of Economic Surveys* 36.1, pp. 75–107. DOI: `https://doi.org/10.1111/joes.12434`. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1111/joes.12434`. URL: `https://onlinelibrary.wiley.com/doi/abs/10.1111/joes.12434`.

8 Appendix

ppendices

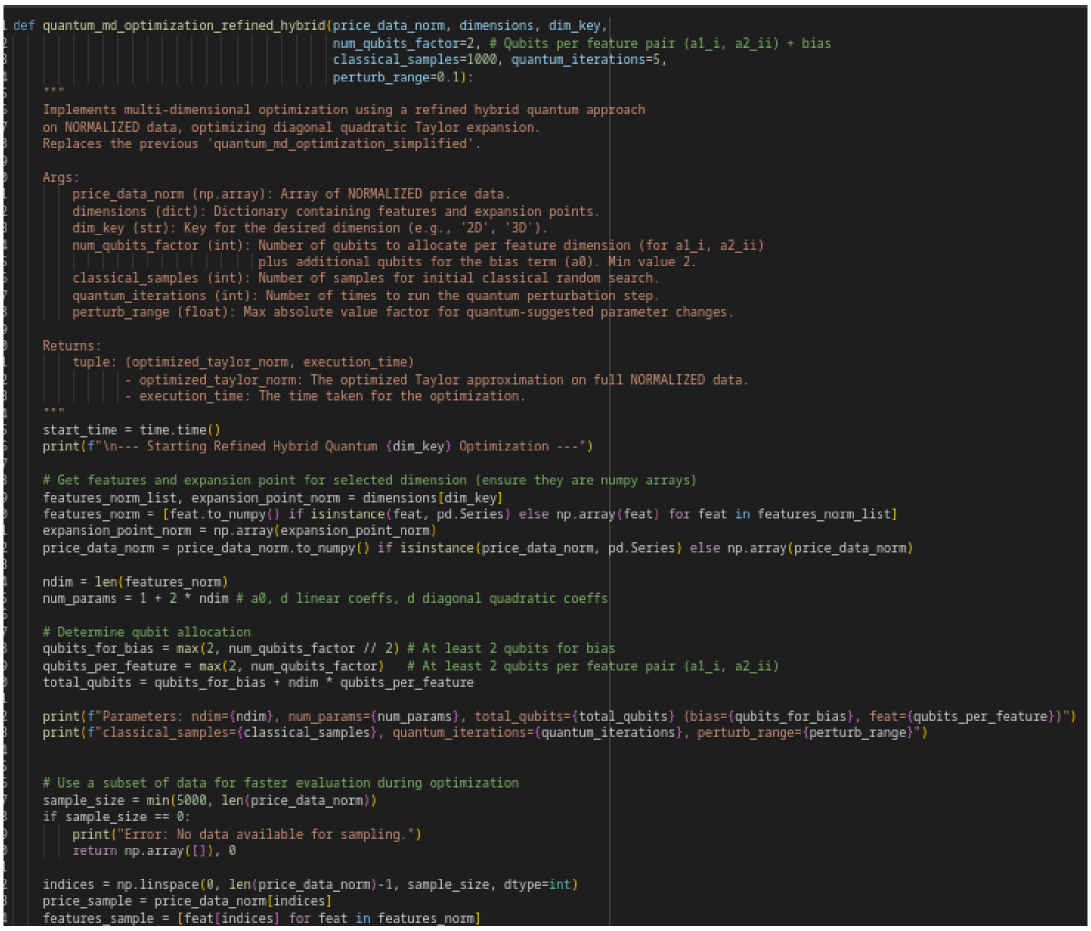
9 Supplementary Visualizations and Screenshots

This appendix contains supplementary images and screenshots referenced or relevant to the thesis. Please refer to the captions for a description of each image.



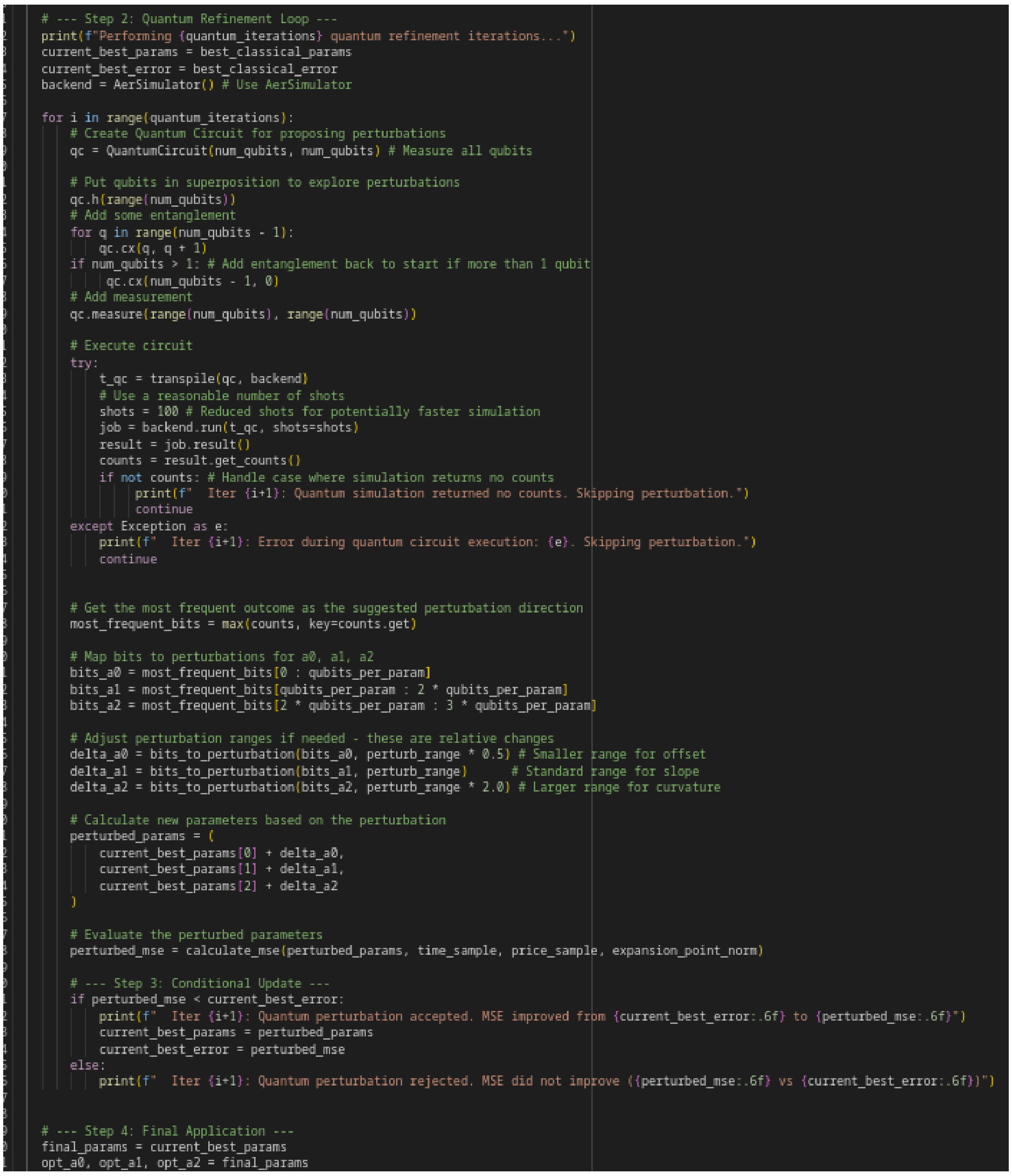
```python


# Helper function to map bits to a perturbation value
def bits_to_perturbation(bits_str, perturb_range):
    """Maps a bitstring to a perturbation value in [-perturb_range, +perturb_range]."""
    if not bits_str: # Handle empty bitstring case
        return 0.0
    max_val = 2**len(bits_str) - 1
    if max_val == 0: # Handle single bit case
        int_val = int(bits_str, 2)
        return perturb_range * (1.0 if int_val == 1 else -1.0) # Map 1 to +range, 0 to -range

    int_val = int(bits_str, 2)
    # Scale integer value to [0, 1], then to [-1, 1], then scale by perturb_range
    return perturb_range * (2 * (int_val / max_val) - 1)
```

Figure 4: Function to map bits to a perturbation value



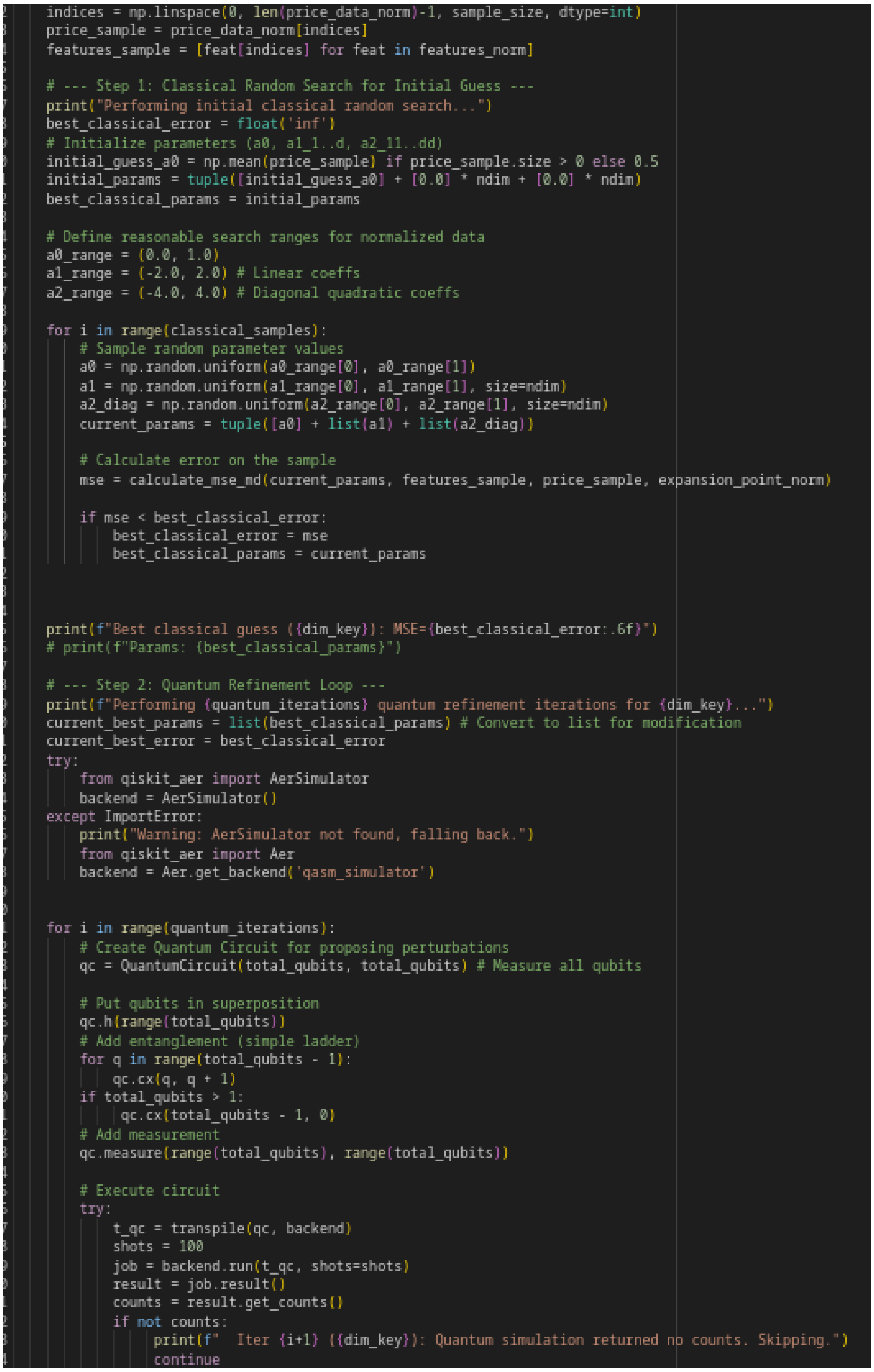
```python
def quantum_md_optimization_refined_hybrid(price_data_norm, dimensions, dim_key,
                                           num_qubits_factor=2, # Qubits per feature pair (a1_i, a2_ii) + bias
                                           classical_samples=1000, quantum_iterations=5,
                                           perturb_range=0.1):
    """
    Implements multi-dimensional optimization using a refined hybrid quantum approach
    on NORMALIZED data, optimizing diagonal quadratic Taylor expansion.
    Replaces the previous 'quantum_md_optimization_simplified'.

    Args:
        price_data_norm (np.array): Array of NORMALIZED price data.
        dimensions (dict): Dictionary containing features and expansion points.
        dim_key (str): Key for the desired dimension (e.g., '2D', '3D').
        num_qubits_factor (int): Number of qubits to allocate per feature dimension (for a1_i, a2_ii)
                                 plus additional qubits for the bias term (a0). Min value 2.
        classical_samples (int): Number of samples for initial classical random search.
        quantum_iterations (int): Number of times to run the quantum perturbation step.
        perturb_range (float): Max absolute value factor for quantum-suggested parameter changes.

    Returns:
        tuple: (optimized_taylor_norm, execution_time)
               - optimized_taylor_norm: The optimized Taylor approximation on full NORMALIZED data.
               - execution_time: The time taken for the optimization.
    """
    start_time = time.time()
    print(f"\n--- Starting Refined Hybrid Quantum {dim_key} Optimization ---")

    # Get features and expansion point for selected dimension (ensure they are numpy arrays)
    features_norm_list, expansion_point_norm = dimensions[dim_key]
    features_norm = [feat.to_numpy() if isinstance(feat, pd.Series) else np.array(feat) for feat in features_norm_list]
    expansion_point_norm = np.array(expansion_point_norm)
    price_data_norm = price_data_norm.to_numpy() if isinstance(price_data_norm, pd.Series) else np.array(price_data_norm)

    ndim = len(features_norm)
    num_params = 1 + 2 * ndim # a0, d linear coeffs, d diagonal quadratic coeffs

    # Determine qubit allocation
    qubits_for_bias = max(2, num_qubits_factor // 2) # At least 2 qubits for bias
    qubits_per_feature = max(2, num_qubits_factor)   # At least 2 qubits per feature pair (a1_i, a2_ii)
    total_qubits = qubits_for_bias + ndim * qubits_per_feature

    print(f"Parameters: ndim={ndim}, num_params={num_params}, total_qubits={total_qubits} (bias={qubits_for_bias}, feat={qubits_per_feature})")
    print(f"classical_samples={classical_samples}, quantum_iterations={quantum_iterations}, perturb_range={perturb_range}")


    # Use a subset of data for faster evaluation during optimization
    sample_size = min(5000, len(price_data_norm))
    if sample_size == 0:
        print("Error: No data available for sampling.")
        return np.array([]), 0

    indices = np.linspace(0, len(price_data_norm)-1, sample_size, dtype=int)
    price_sample = price_data_norm[indices]
    features_sample = [feat[indices] for feat in features_norm]
```

Figure 5: Implements multi-dimensional optimization using a refined hybrid quantum approach on NORMALIZED data, optimizing diagonal quadratic Taylor expansion



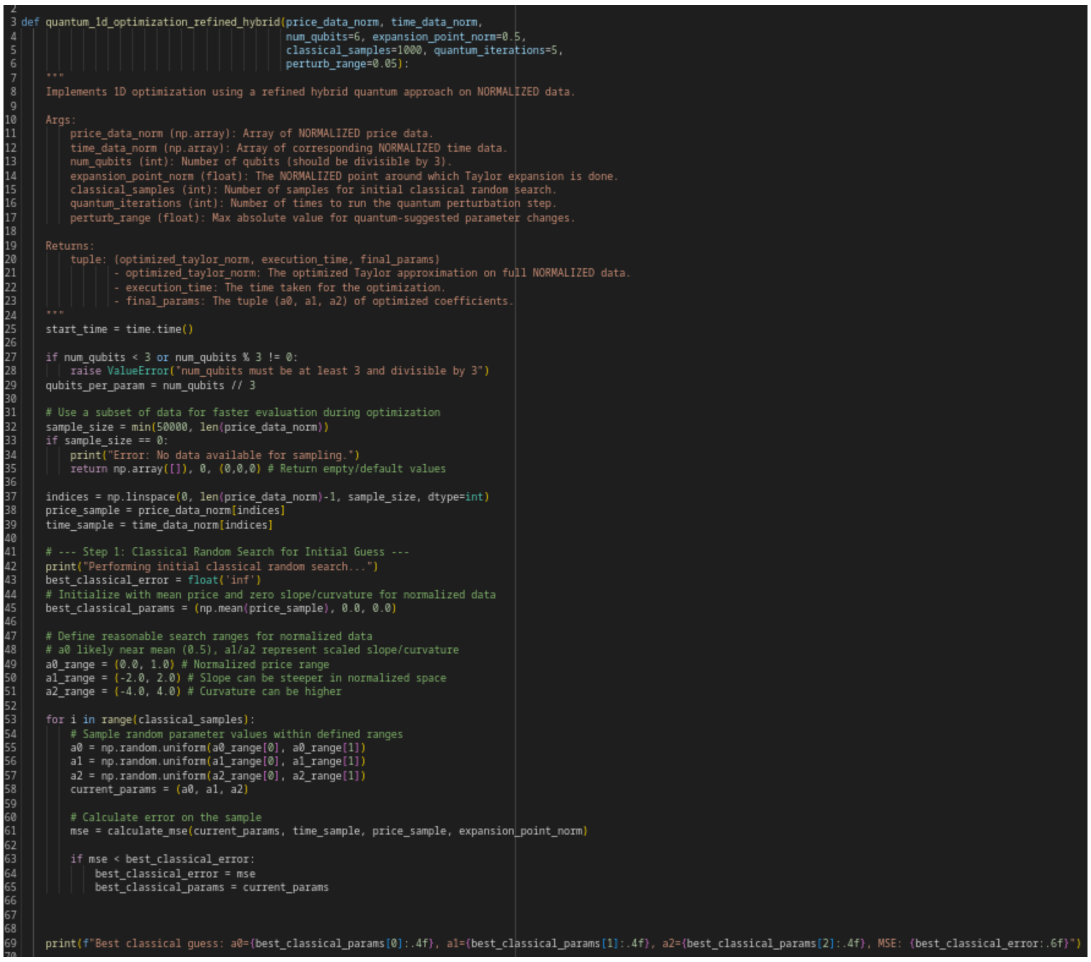
```python
# --- Step 2: Quantum Refinement Loop ---
print(f"Performing {quantum_iterations} quantum refinement iterations...")
current_best_params = best_classical_params
current_best_error = best_classical_error
backend = AerSimulator() # Use AerSimulator

for i in range(quantum_iterations):
    # Create Quantum Circuit for proposing perturbations
    qc = QuantumCircuit(num_qubits, num_qubits) # Measure all qubits

    # Put qubits in superposition to explore perturbations
    qc.h(range(num_qubits))
    # Add some entanglement
    for q in range(num_qubits - 1):
        qc.cx(q, q + 1)
    if num_qubits > 1: # Add entanglement back to start if more than 1 qubit
         qc.cx(num_qubits - 1, 0)
    # Add measurement
    qc.measure(range(num_qubits), range(num_qubits))

    # Execute circuit
    try:
        t_qc = transpile(qc, backend)
        # Use a reasonable number of shots
        shots = 100 # Reduced shots for potentially faster simulation
        job = backend.run(t_qc, shots=shots)
        result = job.result()
        counts = result.get_counts()
        if not counts: # Handle case where simulation returns no counts
             print(f"  Iter {i+1}: Quantum simulation returned no counts. Skipping perturbation.")
             continue
    except Exception as e:
        print(f"  Iter {i+1}: Error during quantum circuit execution: {e}. Skipping perturbation.")
        continue


    # Get the most frequent outcome as the suggested perturbation direction
    most_frequent_bits = max(counts, key=counts.get)

    # Map bits to perturbations for a0, a1, a2
    bits_a0 = most_frequent_bits[0 : qubits_per_param]
    bits_a1 = most_frequent_bits[qubits_per_param : 2 * qubits_per_param]
    bits_a2 = most_frequent_bits[2 * qubits_per_param : 3 * qubits_per_param]

    # Adjust perturbation ranges if needed - these are relative changes
    delta_a0 = bits_to_perturbation(bits_a0, perturb_range * 0.5) # Smaller range for offset
    delta_a1 = bits_to_perturbation(bits_a1, perturb_range)       # Standard range for slope
    delta_a2 = bits_to_perturbation(bits_a2, perturb_range * 2.0) # Larger range for curvature

    # Calculate new parameters based on the perturbation
    perturbed_params = (
        current_best_params[0] + delta_a0,
        current_best_params[1] + delta_a1,
        current_best_params[2] + delta_a2
    )

    # Evaluate the perturbed parameters
    perturbed_mse = calculate_mse(perturbed_params, time_sample, price_sample, expansion_point_norm)

    # --- Step 3: Conditional Update ---
    if perturbed_mse < current_best_error:
        print(f"  Iter {i+1}: Quantum perturbation accepted. MSE improved from {current_best_error:.6f} to {perturbed_mse:.6f}")
        current_best_params = perturbed_params
        current_best_error = perturbed_mse
    else:
        print(f"  Iter {i+1}: Quantum perturbation rejected. MSE did not improve ({perturbed_mse:.6f} vs {current_best_error:.6f})")


# --- Step 4: Final Application ---
final_params = current_best_params
opt_a0, opt_a1, opt_a2 = final_params
```

Figure 6: Implements 1D optimization using a refined hybrid quantum approach step 2 to 4



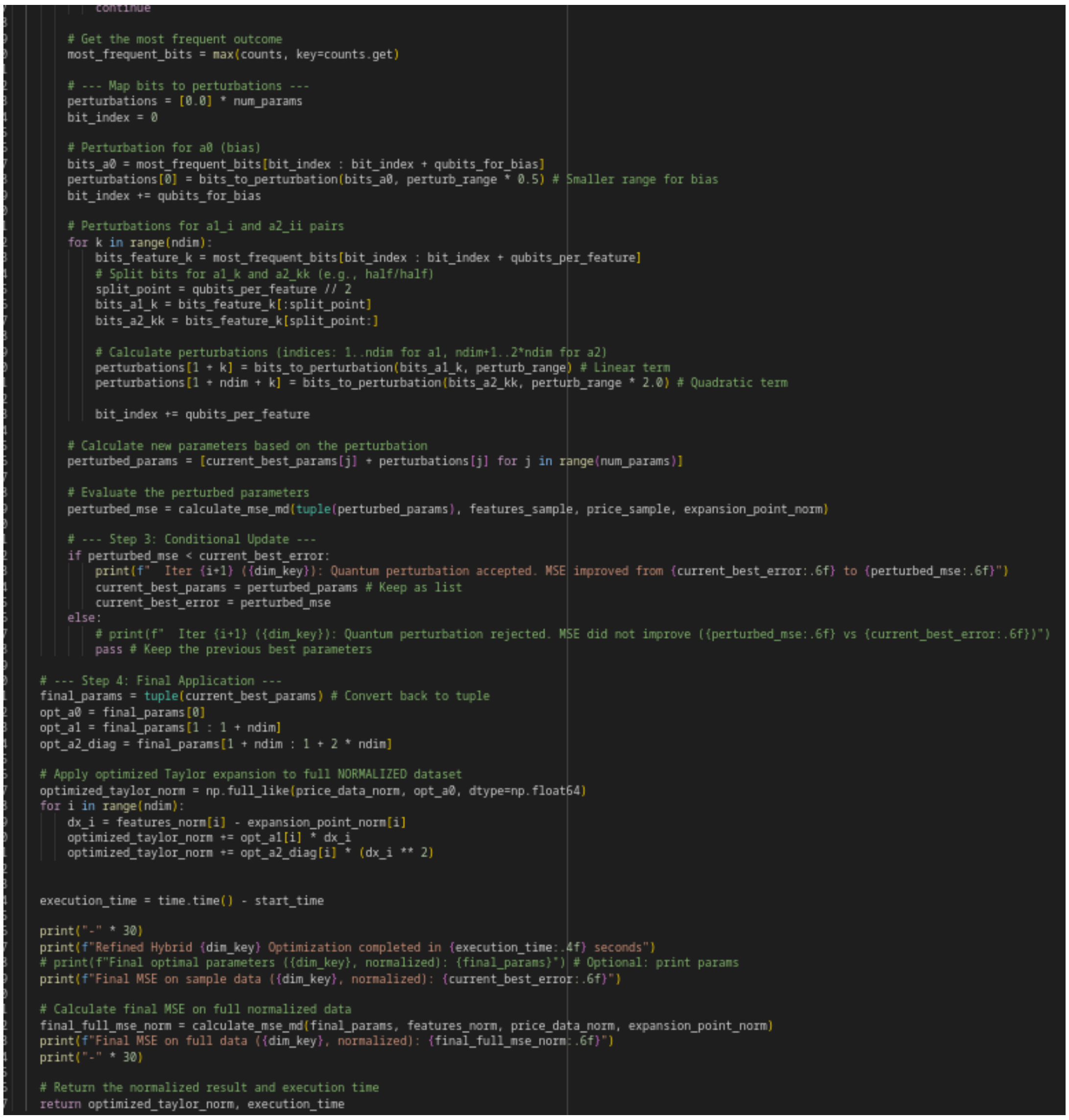
```python
    indices = np.linspace(0, len(price_data_norm)-1, sample_size, dtype=int)
    price_sample = price_data_norm[indices]
    features_sample = [feat[indices] for feat in features_norm]

    # --- Step 1: Classical Random Search for Initial Guess ---
    print("Performing initial classical random search...")
    best_classical_error = float('inf')
    # Initialize parameters (a0, a1_1..d, a2_11..dd)
    initial_guess_a0 = np.mean(price_sample) if price_sample.size > 0 else 0.5
    initial_params = tuple([initial_guess_a0] + [0.0] * ndim + [0.0] * ndim)
    best_classical_params = initial_params

    # Define reasonable search ranges for normalized data
    a0_range = (0.0, 1.0)
    a1_range = (-2.0, 2.0) # Linear coeffs
    a2_range = (-4.0, 4.0) # Diagonal quadratic coeffs

    for i in range(classical_samples):
        # Sample random parameter values
        a0 = np.random.uniform(a0_range[0], a0_range[1])
        a1 = np.random.uniform(a1_range[0], a1_range[1], size=ndim)
        a2_diag = np.random.uniform(a2_range[0], a2_range[1], size=ndim)
        current_params = tuple([a0] + list(a1) + list(a2_diag))

        # Calculate error on the sample
        mse = calculate_mse_md(current_params, features_sample, price_sample, expansion_point_norm)

        if mse < best_classical_error:
            best_classical_error = mse
            best_classical_params = current_params



    print(f"Best classical guess ({dim_key}): MSE={best_classical_error:.6f}")
    # print(f"Params: {best_classical_params}")

    # --- Step 2: Quantum Refinement Loop ---
    print(f"Performing {quantum_iterations} quantum refinement iterations for {dim_key}...")
    current_best_params = list(best_classical_params) # Convert to list for modification
    current_best_error = best_classical_error
    try:
        from qiskit_aer import AerSimulator
        backend = AerSimulator()
    except ImportError:
        print("Warning: AerSimulator not found, falling back.")
        from qiskit_aer import Aer
        backend = Aer.get_backend('qasm_simulator')


    for i in range(quantum_iterations):
        # Create Quantum Circuit for proposing perturbations
        qc = QuantumCircuit(total_qubits, total_qubits) # Measure all qubits

        # Put qubits in superposition
        qc.h(range(total_qubits))
        # Add entanglement (simple ladder)
        for q in range(total_qubits - 1):
            qc.cx(q, q + 1)
        if total_qubits > 1:
             qc.cx(total_qubits - 1, 0)
        # Add measurement
        qc.measure(range(total_qubits), range(total_qubits))

        # Execute circuit
        try:
            t_qc = transpile(qc, backend)
            shots = 100
            job = backend.run(t_qc, shots=shots)
            result = job.result()
            counts = result.get_counts()
            if not counts:
                print(f"  Iter {i+1} ({dim_key}): Quantum simulation returned no counts. Skipping.")
                continue
```

Figure 7: Implements multi-dimensional optimization using a refined hybrid quantum approach on NORMALIZED data, optimizing diagonal quadratic Taylor expansion part 2



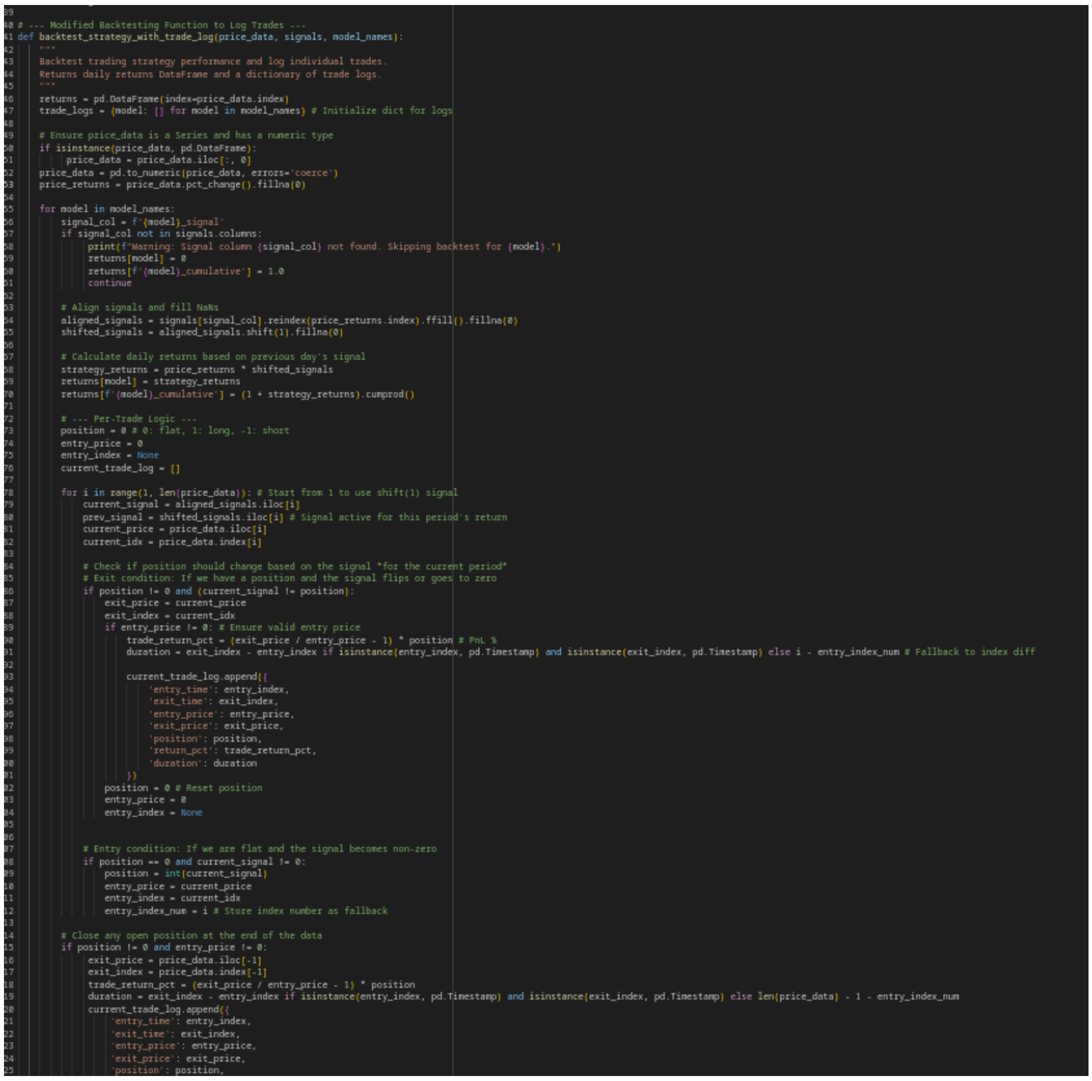
```python
def quantum_1d_optimization_refined_hybrid(price_data_norm, time_data_norm,
                                           num_qubits=6, expansion_point_norm=0.5,
                                           classical_samples=1000, quantum_iterations=5,
                                           perturb_range=0.05):
    """
    Implements 1D optimization using a refined hybrid quantum approach on NORMALIZED data.

    Args:
        price_data_norm (np.array): Array of NORMALIZED price data.
        time_data_norm (np.array): Array of corresponding NORMALIZED time data.
        num_qubits (int): Number of qubits (should be divisible by 3).
        expansion_point_norm (float): The NORMALIZED point around which Taylor expansion is done.
        classical_samples (int): Number of samples for initial classical random search.
        quantum_iterations (int): Number of times to run the quantum perturbation step.
        perturb_range (float): Max absolute value for quantum-suggested parameter changes.

    Returns:
        tuple: (optimized_taylor_norm, execution_time, final_params)
               - optimized_taylor_norm: The optimized Taylor approximation on full NORMALIZED data.
               - execution_time: The time taken for the optimization.
               - final_params: The tuple (a0, a1, a2) of optimized coefficients.
    """
    start_time = time.time()

    if num_qubits < 3 or num_qubits % 3 != 0:
        raise ValueError("num_qubits must be at least 3 and divisible by 3")
    qubits_per_param = num_qubits // 3

    # Use a subset of data for faster evaluation during optimization
    sample_size = min(50000, len(price_data_norm))
    if sample_size == 0:
        print("Error: No data available for sampling.")
        return np.array([]), 0, (0,0,0) # Return empty/default values

    indices = np.linspace(0, len(price_data_norm)-1, sample_size, dtype=int)
    price_sample = price_data_norm[indices]
    time_sample = time_data_norm[indices]

    # --- Step 1: Classical Random Search for Initial Guess ---
    print("Performing initial classical random search...")
    best_classical_error = float('inf')
    # Initialize with mean price and zero slope/curvature for normalized data
    best_classical_params = (np.mean(price_sample), 0.0, 0.0)

    # Define reasonable search ranges for normalized data
    # a0 likely near mean (0.5), a1/a2 represent scaled slope/curvature
    a0_range = (0.0, 1.0) # Normalized price range
    a1_range = (-2.0, 2.0) # Slope can be steeper in normalized space
    a2_range = (-4.0, 4.0) # Curvature can be higher

    for i in range(classical_samples):
        # Sample random parameter values within defined ranges
        a0 = np.random.uniform(a0_range[0], a0_range[1])
        a1 = np.random.uniform(a1_range[0], a1_range[1])
        a2 = np.random.uniform(a2_range[0], a2_range[1])
        current_params = (a0, a1, a2)

        # Calculate error on the sample
        mse = calculate_mse(current_params, time_sample, price_sample, expansion_point_norm)

        if mse < best_classical_error:
            best_classical_error = mse
            best_classical_params = current_params



    print(f"Best classical guess: a0={best_classical_params[0]:.4f}, a1={best_classical_params[1]:.4f}, a2={best_classical_params[2]:.4f}, MSE: {best_classical_error:.6f}")
```

Figure 8: Implements 1D optimization using a refined hybrid quantum approach on NORMALIZED data



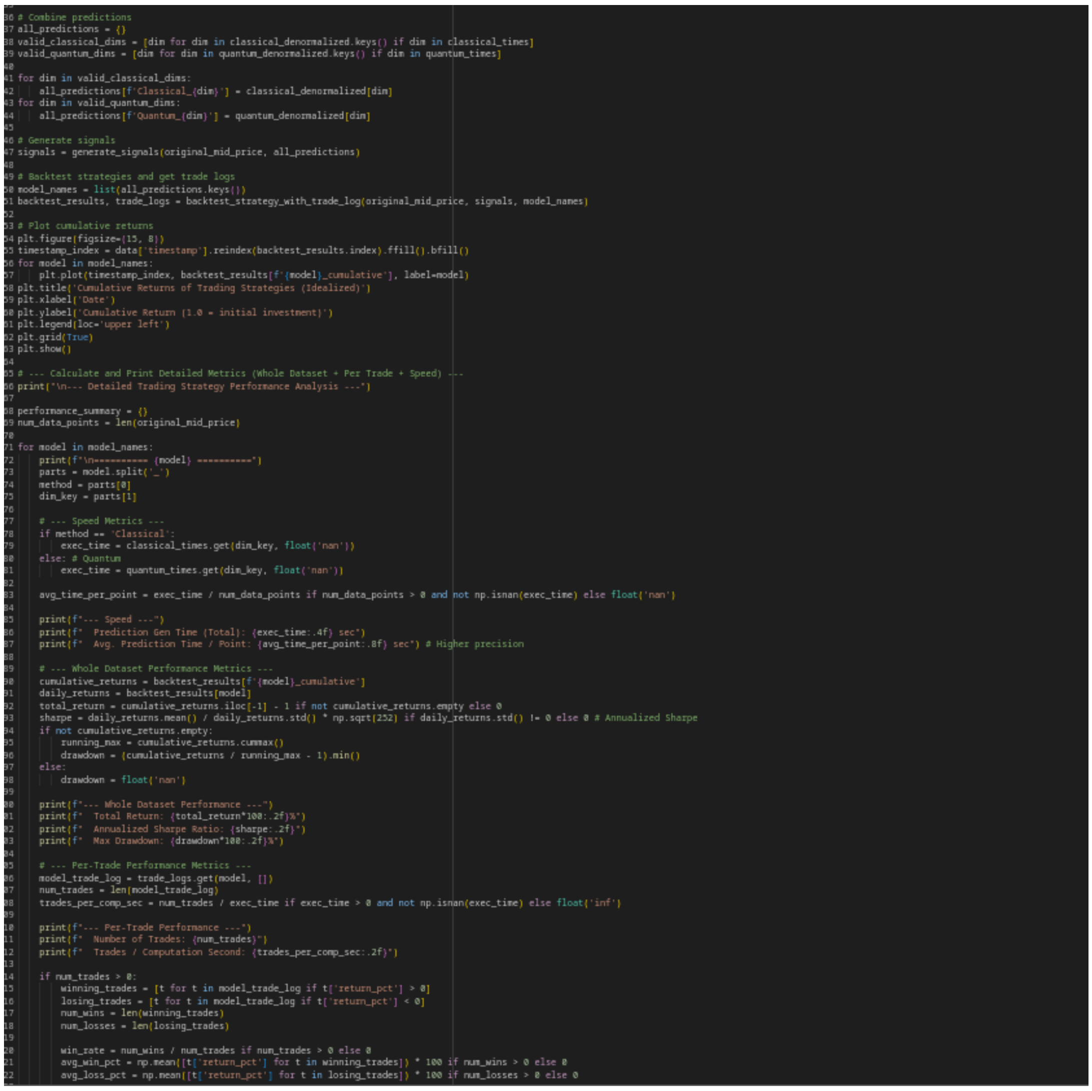
```python
            continue

        # Get the most frequent outcome
        most_frequent_bits = max(counts, key=counts.get)

        # --- Map bits to perturbations ---
        perturbations = [0.0] * num_params
        bit_index = 0

        # Perturbation for a0 (bias)
        bits_a0 = most_frequent_bits[bit_index : bit_index + qubits_for_bias]
        perturbations[0] = bits_to_perturbation(bits_a0, perturb_range * 0.5) # Smaller range for bias
        bit_index += qubits_for_bias

        # Perturbations for a1_i and a2_ii pairs
        for k in range(ndim):
            bits_feature_k = most_frequent_bits[bit_index : bit_index + qubits_per_feature]
            # Split bits for a1_k and a2_kk (e.g., half/half)
            split_point = qubits_per_feature // 2
            bits_a1_k = bits_feature_k[:split_point]
            bits_a2_kk = bits_feature_k[split_point:]

            # Calculate perturbations (indices: 1..ndim for a1, ndim+1..2*ndim for a2)
            perturbations[1 + k] = bits_to_perturbation(bits_a1_k, perturb_range) # Linear term
            perturbations[1 + ndim + k] = bits_to_perturbation(bits_a2_kk, perturb_range * 2.0) # Quadratic term

            bit_index += qubits_per_feature

        # Calculate new parameters based on the perturbation
        perturbed_params = [current_best_params[j] + perturbations[j] for j in range(num_params)]

        # Evaluate the perturbed parameters
        perturbed_mse = calculate_mse_md(tuple(perturbed_params), features_sample, price_sample, expansion_point_norm)

        # --- Step 3: Conditional Update ---
        if perturbed_mse < current_best_error:
            print(f"  Iter {i+1} ({dim_key}): Quantum perturbation accepted. MSE improved from {current_best_error:.6f} to {perturbed_mse:.6f}")
            current_best_params = perturbed_params # Keep as list
            current_best_error = perturbed_mse
        else:
            # print(f"  Iter {i+1} ({dim_key}): Quantum perturbation rejected. MSE did not improve ({perturbed_mse:.6f} vs {current_best_error:.6f})")
            pass # Keep the previous best parameters

    # --- Step 4: Final Application ---
    final_params = tuple(current_best_params) # Convert back to tuple
    opt_a0 = final_params[0]
    opt_a1 = final_params[1 : 1 + ndim]
    opt_a2_diag = final_params[1 + ndim : 1 + 2 * ndim]

    # Apply optimized Taylor expansion to full NORMALIZED dataset
    optimized_taylor_norm = np.full_like(price_data_norm, opt_a0, dtype=np.float64)
    for i in range(ndim):
        dx_i = features_norm[i] - expansion_point_norm[i]
        optimized_taylor_norm += opt_a1[i] * dx_i
        optimized_taylor_norm += opt_a2_diag[i] * (dx_i ** 2)


    execution_time = time.time() - start_time

    print("-" * 30)
    print(f"Refined Hybrid {dim_key} Optimization completed in {execution_time:.4f} seconds")
    # print(f"Final optimal parameters ({dim_key}, normalized): {final_params}") # Optional: print params
    print(f"Final MSE on sample data ({dim_key}, normalized): {current_best_error:.6f}")

    # Calculate final MSE on full normalized data
    final_full_mse_norm = calculate_mse_md(final_params, features_norm, price_data_norm, expansion_point_norm)
    print(f"Final MSE on full data ({dim_key}, normalized): {final_full_mse_norm:.6f}")
    print("-" * 30)

    # Return the normalized result and execution time
    return optimized_taylor_norm, execution_time
```

Figure 9: Implements multi-dimensional optimization using a refined hybrid quantum approach on NORMALIZED data, optimizing diagonal quadratic Taylor expansion steps 2 to 4



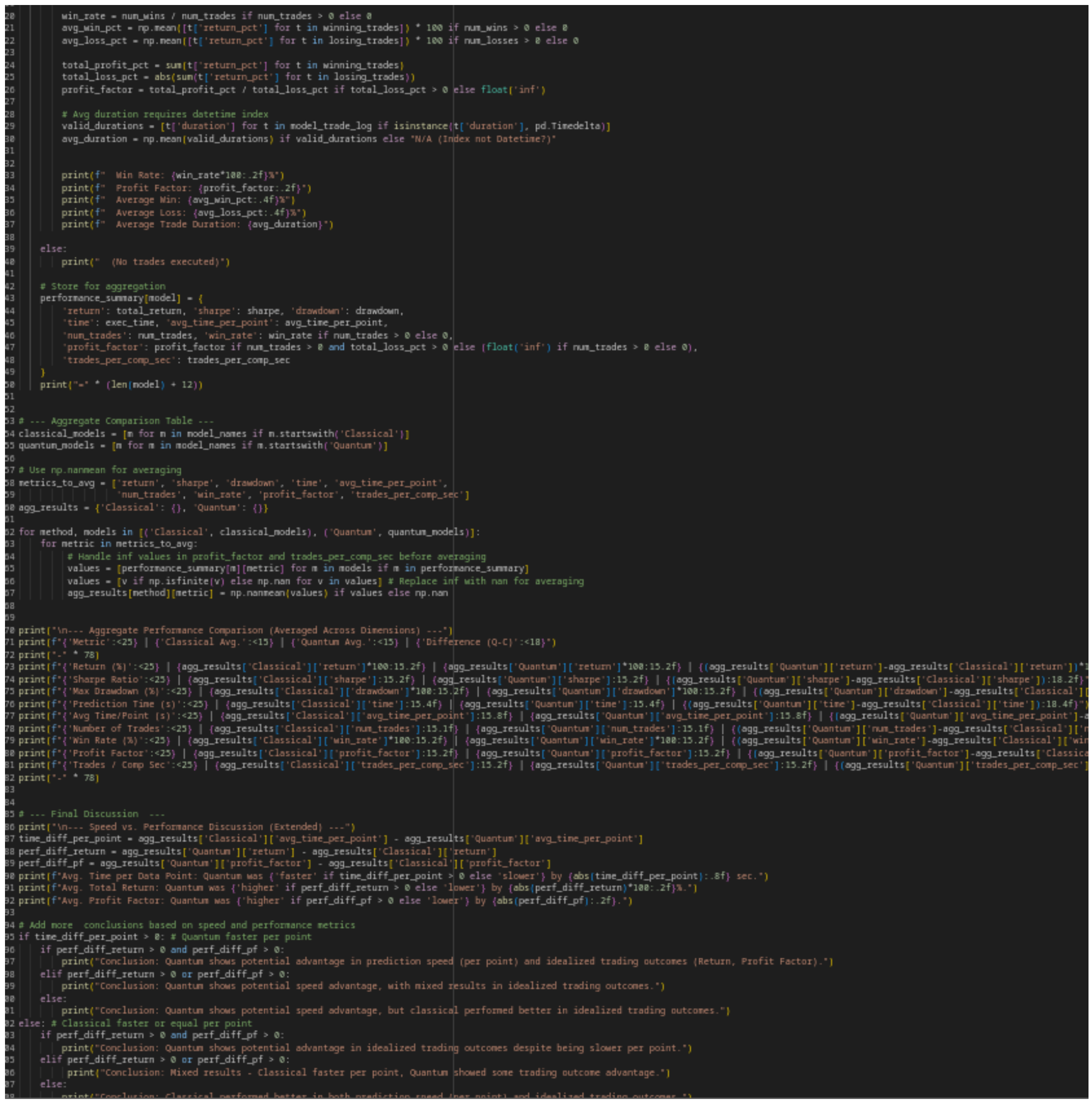
```python
# --- Modified Backtesting Function to Log Trades ---
def backtest_strategy_with_trade_log(price_data, signals, model_names):
    """
    Backtest trading strategy performance and log individual trades.
    Returns daily returns DataFrame and a dictionary of trade logs.
    """
    returns = pd.DataFrame(index=price_data.index)
    trade_logs = {model: [] for model in model_names} # Initialize dict for logs

    # Ensure price_data is a Series and has a numeric type
    if isinstance(price_data, pd.DataFrame):
        price_data = price_data.iloc[:, 0]
    price_data = pd.to_numeric(price_data, errors='coerce')
    price_returns = price_data.pct_change().fillna(0)

    for model in model_names:
        signal_col = f'{model}_signal'
        if signal_col not in signals.columns:
            print(f"Warning: Signal column {signal_col} not found. Skipping backtest for {model}.")
            returns[model] = 0
            returns[f'{model}_cumulative'] = 1.0
            continue

        # Align signals and fill NaNs
        aligned_signals = signals[signal_col].reindex(price_returns.index).ffill().fillna(0)
        shifted_signals = aligned_signals.shift(1).fillna(0)

        # Calculate daily returns based on previous day's signal
        strategy_returns = price_returns * shifted_signals
        returns[model] = strategy_returns
        returns[f'{model}_cumulative'] = (1 + strategy_returns).cumprod()

        # --- Per-Trade Logic ---
        position = 0 # 0: flat, 1: long, -1: short
        entry_price = 0
        entry_index = None
        current_trade_log = []

        for i in range(1, len(price_data)): # Start from 1 to use shift(1) signal
            current_signal = aligned_signals.iloc[i]
            prev_signal = shifted_signals.iloc[i] # Signal active for this period's return
            current_price = price_data.iloc[i]
            current_idx = price_data.index[i]

            # Check if position should change based on the signal *for the current period*
            # Exit condition: If we have a position and the signal flips or goes to zero
            if position != 0 and (current_signal != position):
                exit_price = current_price
                exit_index = current_idx
                if entry_price != 0: # Ensure valid entry price
                    trade_return_pct = (exit_price / entry_price - 1) * position # PnL %
                    duration = exit_index - entry_index if isinstance(entry_index, pd.Timestamp) and isinstance(exit_index, pd.Timestamp) else i - entry_index_num # Fallback to index diff

                    current_trade_log.append({
                        'entry_time': entry_index,
                        'exit_time': exit_index,
                        'entry_price': entry_price,
                        'exit_price': exit_price,
                        'position': position,
                        'return_pct': trade_return_pct,
                        'duration': duration
                    })
                position = 0 # Reset position
                entry_price = 0
                entry_index = None


            # Entry condition: If we are flat and the signal becomes non-zero
            if position == 0 and current_signal != 0:
                position = int(current_signal)
                entry_price = current_price
                entry_index = current_idx
                entry_index_num = i # Store index number as fallback

        # Close any open position at the end of the data
        if position != 0 and entry_price != 0:
            exit_price = price_data.iloc[-1]
            exit_index = price_data.index[-1]
            trade_return_pct = (exit_price / entry_price - 1) * position
            duration = exit_index - entry_index if isinstance(entry_index, pd.Timestamp) and isinstance(exit_index, pd.Timestamp) else len(price_data) - 1 - entry_index_num
            current_trade_log.append({
                'entry_time': entry_index,
                'exit_time': exit_index,
                'entry_price': entry_price,
                'exit_price': exit_price,
                'position': position,
```

Figure 10: Back-testing Strategy



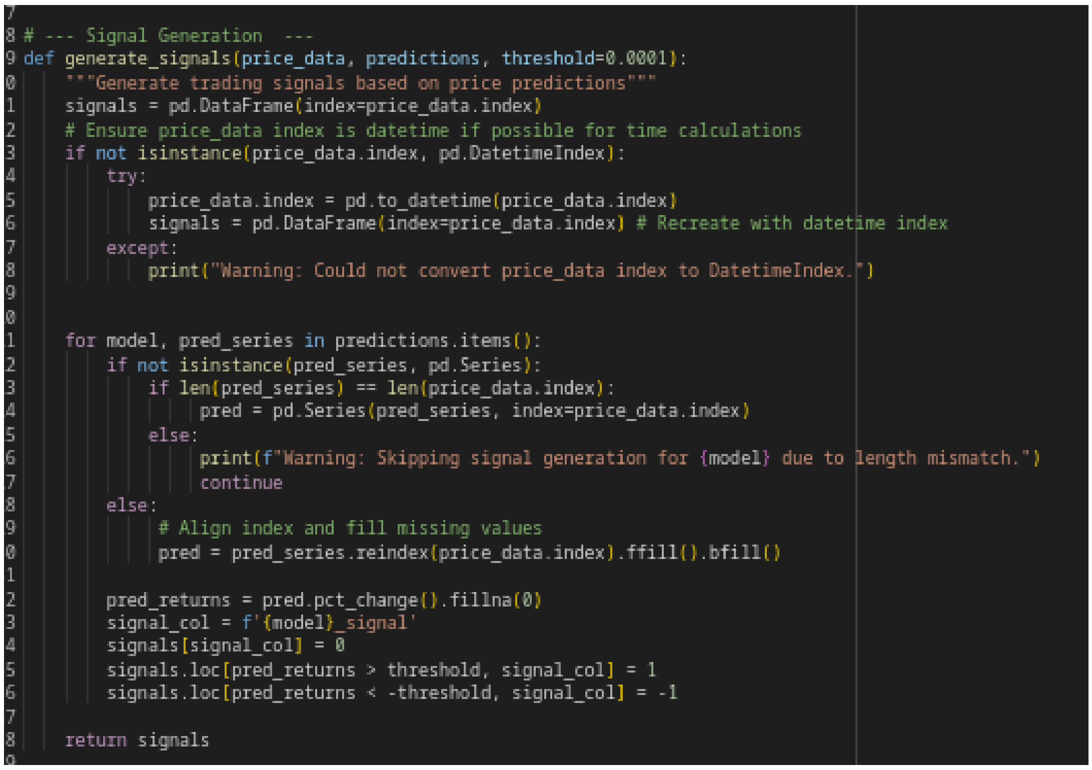
```python
# Combine predictions
all_predictions = {}
valid_classical_dims = [dim for dim in classical_denormalized.keys() if dim in classical_times]
valid_quantum_dims = [dim for dim in quantum_denormalized.keys() if dim in quantum_times]

for dim in valid_classical_dims:
    all_predictions[f'Classical_{dim}'] = classical_denormalized[dim]
for dim in valid_quantum_dims:
    all_predictions[f'Quantum_{dim}'] = quantum_denormalized[dim]

# Generate signals
signals = generate_signals(original_mid_price, all_predictions)

# Backtest strategies and get trade logs
model_names = list(all_predictions.keys())
backtest_results, trade_logs = backtest_strategy_with_trade_log(original_mid_price, signals, model_names)

# Plot cumulative returns
plt.figure(figsize=(15, 8))
timestamp_index = data['timestamp'].reindex(backtest_results.index).ffill().bfill()
for model in model_names:
    plt.plot(timestamp_index, backtest_results[f'{model}_cumulative'], label=model)
plt.title('Cumulative Returns of Trading Strategies (Idealized)')
plt.xlabel('Date')
plt.ylabel('Cumulative Return (1.0 = initial investment)')
plt.legend(loc='upper left')
plt.grid(True)
plt.show()

# --- Calculate and Print Detailed Metrics (Whole Dataset + Per Trade + Speed) ---
print("\n--- Detailed Trading Strategy Performance Analysis ---")

performance_summary = {}
num_data_points = len(original_mid_price)

for model in model_names:
    print(f"\n========== {model} ==========")
    parts = model.split('_')
    method = parts[0]
    dim_key = parts[1]

    # --- Speed Metrics ---
    if method == 'Classical':
        exec_time = classical_times.get(dim_key, float('nan'))
    else: # Quantum
        exec_time = quantum_times.get(dim_key, float('nan'))

    avg_time_per_point = exec_time / num_data_points if num_data_points > 0 and not np.isnan(exec_time) else float('nan')

    print(f"--- Speed ---")
    print(f"  Prediction Gen Time (Total): {exec_time:.4f} sec")
    print(f"  Avg. Prediction Time / Point: {avg_time_per_point:.8f} sec") # Higher precision

    # --- Whole Dataset Performance Metrics ---
    cumulative_returns = backtest_results[f'{model}_cumulative']
    daily_returns = backtest_results[model]
    total_return = cumulative_returns.iloc[-1] - 1 if not cumulative_returns.empty else 0
    sharpe = daily_returns.mean() / daily_returns.std() * np.sqrt(252) if daily_returns.std() != 0 else 0 # Annualized Sharpe
    if not cumulative_returns.empty:
        running_max = cumulative_returns.cummax()
        drawdown = (cumulative_returns / running_max - 1).min()
    else:
        drawdown = float('nan')

    print(f"--- Whole Dataset Performance ---")
    print(f"  Total Return: {total_return*100:.2f}%")
    print(f"  Annualized Sharpe Ratio: {sharpe:.2f}")
    print(f"  Max Drawdown: {drawdown*100:.2f}%")

    # --- Per-Trade Performance Metrics ---
    model_trade_log = trade_logs.get(model, [])
    num_trades = len(model_trade_log)
    trades_per_comp_sec = num_trades / exec_time if exec_time > 0 and not np.isnan(exec_time) else float('inf')

    print(f"--- Per-Trade Performance ---")
    print(f"  Number of Trades: {num_trades}")
    print(f"  Trades / Computation Second: {trades_per_comp_sec:.2f}")

    if num_trades > 0:
        winning_trades = [t for t in model_trade_log if t['return_pct'] > 0]
        losing_trades = [t for t in model_trade_log if t['return_pct'] < 0]
        num_wins = len(winning_trades)
        num_losses = len(losing_trades)

        win_rate = num_wins / num_trades if num_trades > 0 else 0
        avg_win_pct = np.mean([t['return_pct'] for t in winning_trades]) * 100 if num_wins > 0 else 0
        avg_loss_pct = np.mean([t['return_pct'] for t in losing_trades]) * 100 if num_losses > 0 else 0
```

Figure 11: Back-testing Strategy part 2



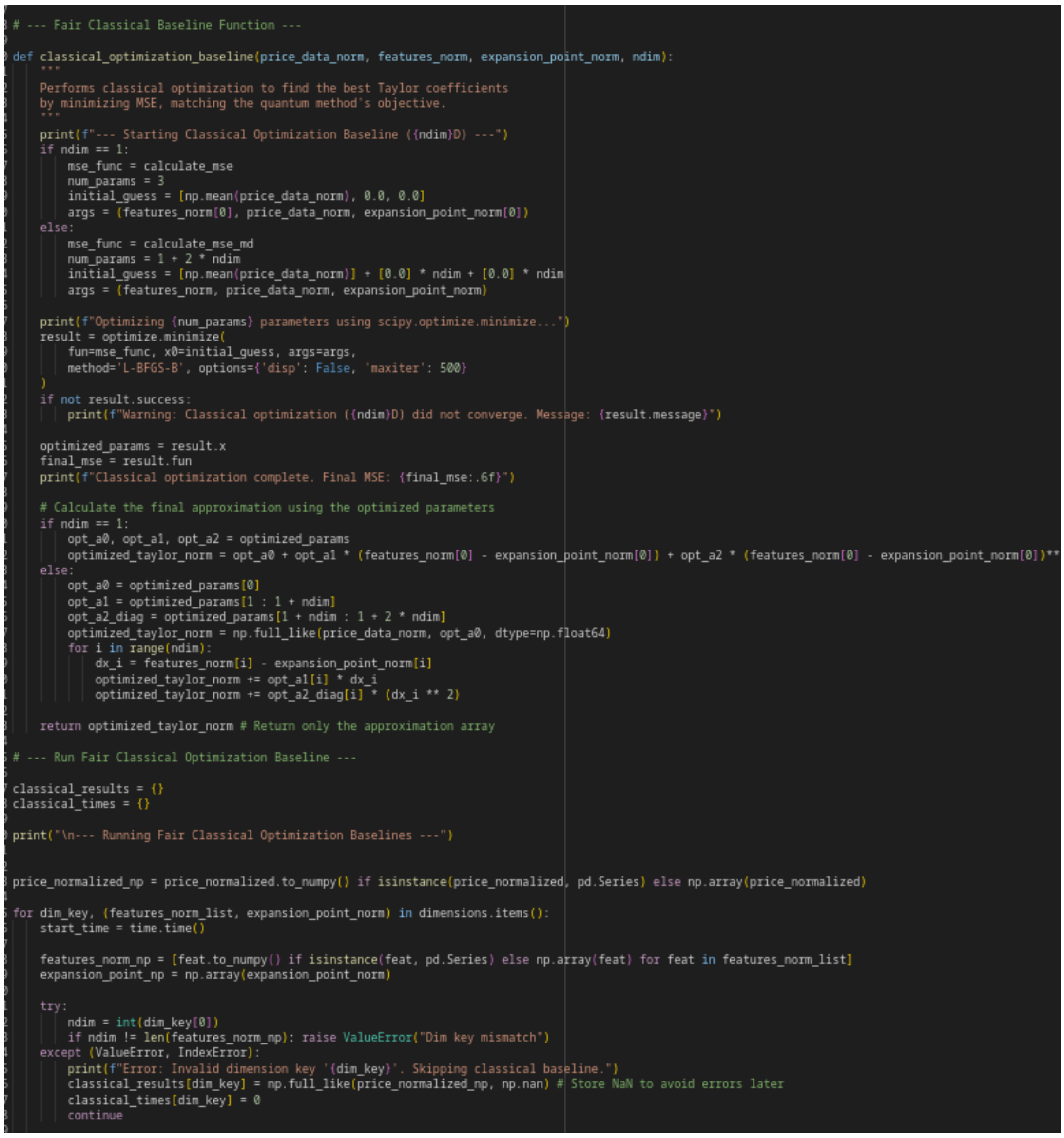
```python
        win_rate = num_wins / num_trades if num_trades > 0 else 0
        avg_win_pct = np.mean([t['return_pct'] for t in winning_trades]) * 100 if num_wins > 0 else 0
        avg_loss_pct = np.mean([t['return_pct'] for t in losing_trades]) * 100 if num_losses > 0 else 0

        total_profit_pct = sum(t['return_pct'] for t in winning_trades)
        total_loss_pct = abs(sum(t['return_pct'] for t in losing_trades))
        profit_factor = total_profit_pct / total_loss_pct if total_loss_pct > 0 else float('inf')

        # Avg duration requires datetime index
        valid_durations = [t['duration'] for t in model_trade_log if isinstance(t['duration'], pd.Timedelta)]
        avg_duration = np.mean(valid_durations) if valid_durations else "N/A (Index not Datetime?)"


        print(f"  Win Rate: {win_rate*100:.2f}%")
        print(f"  Profit Factor: {profit_factor:.2f}")
        print(f"  Average Win: {avg_win_pct:.4f}%")
        print(f"  Average Loss: {avg_loss_pct:.4f}%")
        print(f"  Average Trade Duration: {avg_duration}")

    else:
        print("  (No trades executed)")

    # Store for aggregation
    performance_summary[model] = {
        'return': total_return, 'sharpe': sharpe, 'drawdown': drawdown,
        'time': exec_time, 'avg_time_per_point': avg_time_per_point,
        'num_trades': num_trades, 'win_rate': win_rate if num_trades > 0 else 0,
        'profit_factor': profit_factor if num_trades > 0 and total_loss_pct > 0 else (float('inf') if num_trades > 0 else 0),
        'trades_per_comp_sec': trades_per_comp_sec
    }
    print("=" * (len(model) + 12))


# --- Aggregate Comparison Table ---
classical_models = [m for m in model_names if m.startswith('Classical')]
quantum_models = [m for m in model_names if m.startswith('Quantum')]

# Use np.nanmean for averaging
metrics_to_avg = ['return', 'sharpe', 'drawdown', 'time', 'avg_time_per_point',
                  'num_trades', 'win_rate', 'profit_factor', 'trades_per_comp_sec']
agg_results = {'Classical': {}, 'Quantum': {}}

for method, models in [('Classical', classical_models), ('Quantum', quantum_models)]:
    for metric in metrics_to_avg:
        # Handle inf values in profit_factor and trades_per_comp_sec before averaging
        values = [performance_summary[m][metric] for m in models if m in performance_summary]
        values = [v if np.isfinite(v) else np.nan for v in values] # Replace inf with nan for averaging
        agg_results[method][metric] = np.nanmean(values) if values else np.nan


print("\n--- Aggregate Performance Comparison (Averaged Across Dimensions) ---")
print(f"{'Metric':<25} | {'Classical Avg.':<15} | {'Quantum Avg.':<15} | {'Difference (Q-C)':<18}")
print("-" * 78)
print(f"{'Return (%)':<25} | {agg_results['Classical']['return']*100:15.2f} | {agg_results['Quantum']['return']*100:15.2f} | {(agg_results['Quantum']['return']-agg_results['Classical']['return'])*1
print(f"{'Sharpe Ratio':<25} | {agg_results['Classical']['sharpe']:15.2f} | {agg_results['Quantum']['sharpe']:15.2f} | {(agg_results['Quantum']['sharpe']-agg_results['Classical']['sharpe']):18.2f}"
print(f"{'Max Drawdown (%)':<25} | {agg_results['Classical']['drawdown']*100:15.2f} | {agg_results['Quantum']['drawdown']*100:15.2f} | {(agg_results['Quantum']['drawdown']-agg_results['Classical'][
print(f"{'Prediction Time (s)':<25} | {agg_results['Classical']['time']:15.4f} | {agg_results['Quantum']['time']:15.4f} | {(agg_results['Quantum']['time']-agg_results['Classical']['time']):18.4f}")
print(f"{'Avg Time/Point (s)':<25} | {agg_results['Classical']['avg_time_per_point']:15.8f} | {agg_results['Quantum']['avg_time_per_point']:15.8f} | {(agg_results['Quantum']['avg_time_per_point']-a
print(f"{'Number of Trades':<25} | {agg_results['Classical']['num_trades']:15.1f} | {agg_results['Quantum']['num_trades']:15.1f} | {(agg_results['Quantum']['num_trades']-agg_results['Classical']['n
print(f"{'Win Rate (%)':<25} | {agg_results['Classical']['win_rate']*100:15.2f} | {agg_results['Quantum']['win_rate']*100:15.2f} | {(agg_results['Quantum']['win_rate']-agg_results['Classical']['win
print(f"{'Profit Factor':<25} | {agg_results['Classical']['profit_factor']:15.2f} | {agg_results['Quantum']['profit_factor']:15.2f} | {(agg_results['Quantum']['profit_factor']-agg_results['Classica
print(f"{'Trades / Comp Sec':<25} | {agg_results['Classical']['trades_per_comp_sec']:15.2f} | {agg_results['Quantum']['trades_per_comp_sec']:15.2f} | {(agg_results['Quantum']['trades_per_comp_sec']
print("-" * 78)


# --- Final Discussion  ---
print("\n--- Speed vs. Performance Discussion (Extended) ---")
time_diff_per_point = agg_results['Classical']['avg_time_per_point'] - agg_results['Quantum']['avg_time_per_point']
perf_diff_return = agg_results['Quantum']['return'] - agg_results['Classical']['return']
perf_diff_pf = agg_results['Quantum']['profit_factor'] - agg_results['Classical']['profit_factor']
print(f"Avg. Time per Data Point: Quantum was {'faster' if time_diff_per_point > 0 else 'slower'} by {abs(time_diff_per_point):.8f} sec.")
print(f"Avg. Total Return: Quantum was {'higher' if perf_diff_return > 0 else 'lower'} by {abs(perf_diff_return)*100:.2f}%.")
print(f"Avg. Profit Factor: Quantum was {'higher' if perf_diff_pf > 0 else 'lower'} by {abs(perf_diff_pf):.2f}.")

# Add more  conclusions based on speed and performance metrics
if time_diff_per_point > 0: # Quantum faster per point
    if perf_diff_return > 0 and perf_diff_pf > 0:
        print("Conclusion: Quantum shows potential advantage in prediction speed (per point) and idealized trading outcomes (Return, Profit Factor).")
    elif perf_diff_return > 0 or perf_diff_pf > 0:
        print("Conclusion: Quantum shows potential speed advantage, with mixed results in idealized trading outcomes.")
    else:
        print("Conclusion: Quantum shows potential speed advantage, but classical performed better in idealized trading outcomes.")
else: # Classical faster or equal per point
    if perf_diff_return > 0 and perf_diff_pf > 0:
        print("Conclusion: Quantum shows potential advantage in idealized trading outcomes despite being slower per point.")
    elif perf_diff_return > 0 or perf_diff_pf > 0:
         print("Conclusion: Mixed results - Classical faster per point, Quantum showed some trading outcome advantage.")
    else:
        print("Conclusion: Classical performed better in both prediction speed (per point) and idealized trading outcomes.")
```

Figure 12: Back-testing Strategy part 3



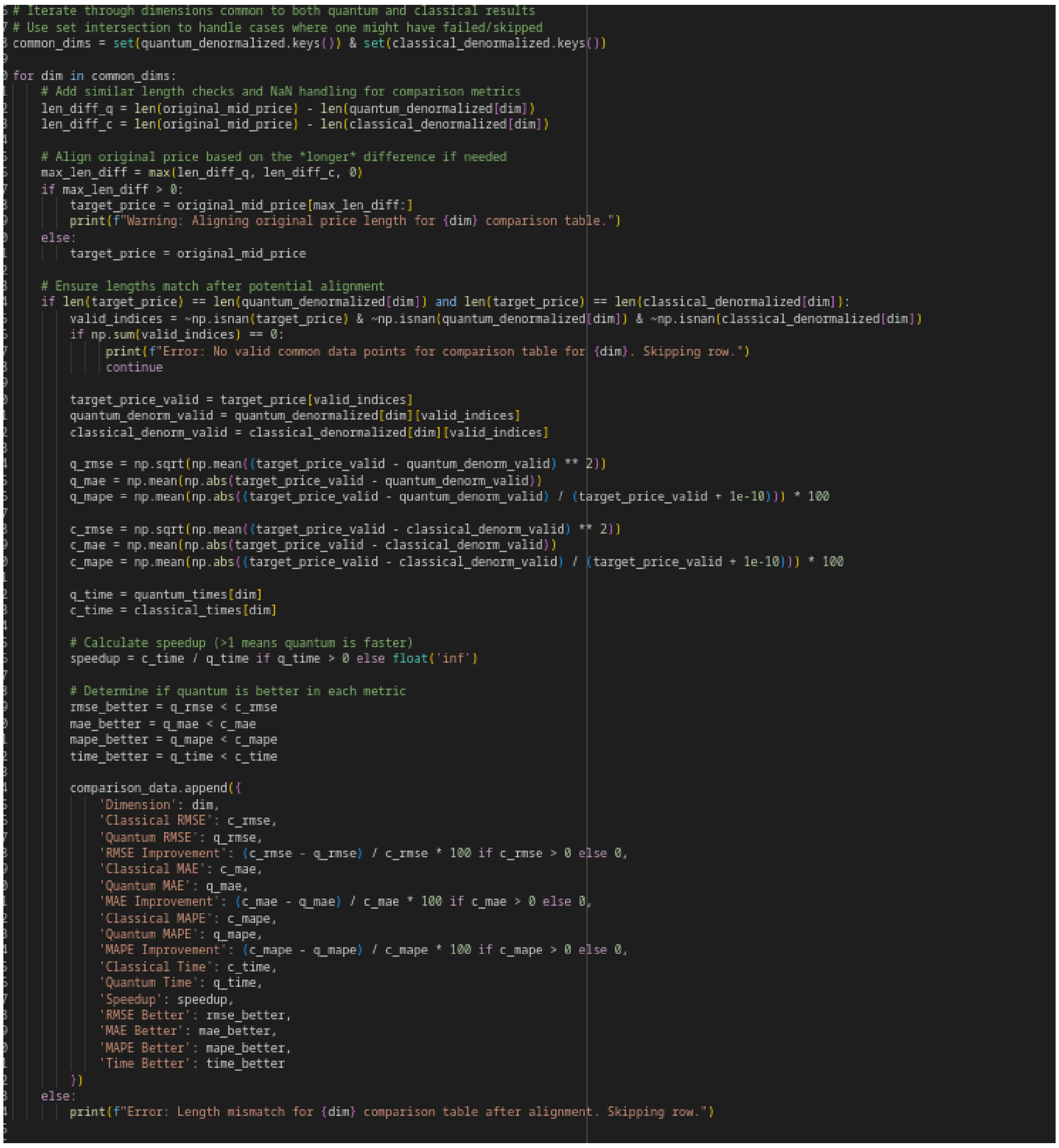
```python
# --- Signal Generation  ---
def generate_signals(price_data, predictions, threshold=0.0001):
    """Generate trading signals based on price predictions"""
    signals = pd.DataFrame(index=price_data.index)
    # Ensure price_data index is datetime if possible for time calculations
    if not isinstance(price_data.index, pd.DatetimeIndex):
        try:
            price_data.index = pd.to_datetime(price_data.index)
            signals = pd.DataFrame(index=price_data.index) # Recreate with datetime index
        except:
            print("Warning: Could not convert price_data index to DatetimeIndex.")


    for model, pred_series in predictions.items():
        if not isinstance(pred_series, pd.Series):
            if len(pred_series) == len(price_data.index):
                pred = pd.Series(pred_series, index=price_data.index)
            else:
                print(f"Warning: Skipping signal generation for {model} due to length mismatch.")
                continue
        else:
            # Align index and fill missing values
            pred = pred_series.reindex(price_data.index).ffill().bfill()

        pred_returns = pred.pct_change().fillna(0)
        signal_col = f'{model}_signal'
        signals[signal_col] = 0
        signals.loc[pred_returns > threshold, signal_col] = 1
        signals.loc[pred_returns < -threshold, signal_col] = -1

    return signals
```

Figure 13: Signal generator



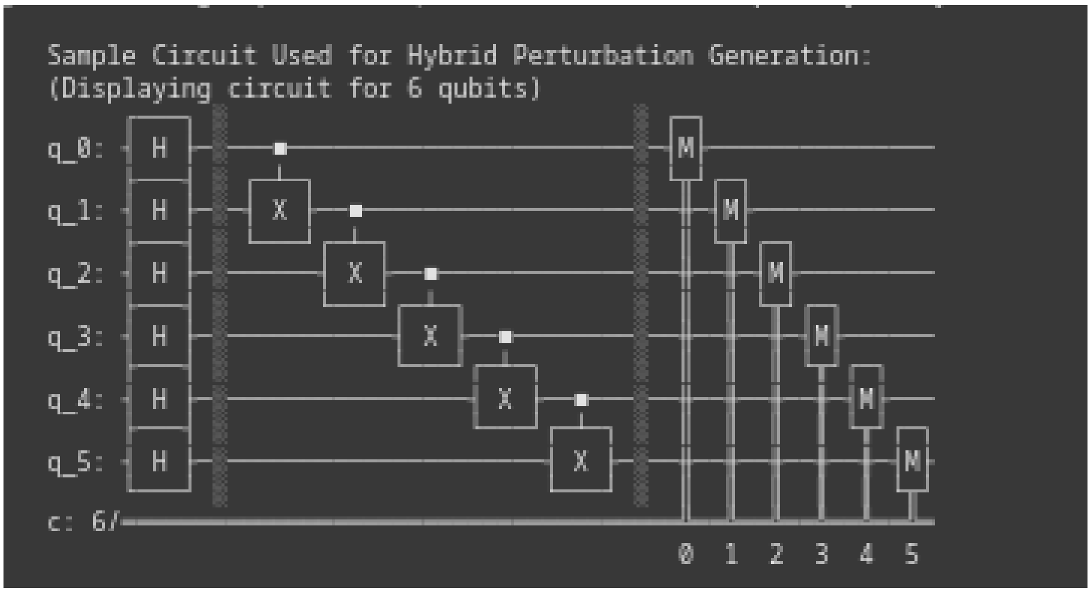
```python
# --- Fair Classical Baseline Function ---

def classical_optimization_baseline(price_data_norm, features_norm, expansion_point_norm, ndim):
    """
    Performs classical optimization to find the best Taylor coefficients
    by minimizing MSE, matching the quantum method's objective.
    """
    print(f"--- Starting Classical Optimization Baseline ({ndim}D) ---")
    if ndim == 1:
        mse_func = calculate_mse
        num_params = 3
        initial_guess = [np.mean(price_data_norm), 0.0, 0.0]
        args = (features_norm[0], price_data_norm, expansion_point_norm[0])
    else:
        mse_func = calculate_mse_md
        num_params = 1 + 2 * ndim
        initial_guess = [np.mean(price_data_norm)] + [0.0] * ndim + [0.0] * ndim
        args = (features_norm, price_data_norm, expansion_point_norm)

    print(f"Optimizing {num_params} parameters using scipy.optimize.minimize...")
    result = optimize.minimize(
        fun=mse_func, x0=initial_guess, args=args,
        method='L-BFGS-B', options={'disp': False, 'maxiter': 500}
    )
    if not result.success:
        print(f"Warning: Classical optimization ({ndim}D) did not converge. Message: {result.message}")

    optimized_params = result.x
    final_mse = result.fun
    print(f"Classical optimization complete. Final MSE: {final_mse:.6f}")

    # Calculate the final approximation using the optimized parameters
    if ndim == 1:
        opt_a0, opt_a1, opt_a2 = optimized_params
        optimized_taylor_norm = opt_a0 + opt_a1 * (features_norm[0] - expansion_point_norm[0]) + opt_a2 * (features_norm[0] - expansion_point_norm[0])**
    else:
        opt_a0 = optimized_params[0]
        opt_a1 = optimized_params[1 : 1 + ndim]
        opt_a2_diag = optimized_params[1 + ndim : 1 + 2 * ndim]
        optimized_taylor_norm = np.full_like(price_data_norm, opt_a0, dtype=np.float64)
        for i in range(ndim):
            dx_i = features_norm[i] - expansion_point_norm[i]
            optimized_taylor_norm += opt_a1[i] * dx_i
            optimized_taylor_norm += opt_a2_diag[i] * (dx_i ** 2)

    return optimized_taylor_norm # Return only the approximation array

# --- Run Fair Classical Optimization Baseline ---

classical_results = {}
classical_times = {}

print("\n--- Running Fair Classical Optimization Baselines ---")


price_normalized_np = price_normalized.to_numpy() if isinstance(price_normalized, pd.Series) else np.array(price_normalized)

for dim_key, (features_norm_list, expansion_point_norm) in dimensions.items():
    start_time = time.time()

    features_norm_np = [feat.to_numpy() if isinstance(feat, pd.Series) else np.array(feat) for feat in features_norm_list]
    expansion_point_np = np.array(expansion_point_norm)

    try:
        ndim = int(dim_key[0])
        if ndim != len(features_norm_np): raise ValueError("Dim key mismatch")
    except (ValueError, IndexError):
        print(f"Error: Invalid dimension key '{dim_key}'. Skipping classical baseline.")
        classical_results[dim_key] = np.full_like(price_normalized_np, np.nan) # Store NaN to avoid errors later
        classical_times[dim_key] = 0
        continue
```

Figure 14: Fair Classical Baseline Function



```python
# Iterate through dimensions common to both quantum and classical results
# Use set intersection to handle cases where one might have failed/skipped
common_dims = set(quantum_denormalized.keys()) & set(classical_denormalized.keys())

for dim in common_dims:
    # Add similar length checks and NaN handling for comparison metrics
    len_diff_q = len(original_mid_price) - len(quantum_denormalized[dim])
    len_diff_c = len(original_mid_price) - len(classical_denormalized[dim])

    # Align original price based on the *longer* difference if needed
    max_len_diff = max(len_diff_q, len_diff_c, 0)
    if max_len_diff > 0:
        target_price = original_mid_price[max_len_diff:]
        print(f"Warning: Aligning original price length for {dim} comparison table.")
    else:
        target_price = original_mid_price

    # Ensure lengths match after potential alignment
    if len(target_price) == len(quantum_denormalized[dim]) and len(target_price) == len(classical_denormalized[dim]):
        valid_indices = ~np.isnan(target_price) & ~np.isnan(quantum_denormalized[dim]) & ~np.isnan(classical_denormalized[dim])
        if np.sum(valid_indices) == 0:
            print(f"Error: No valid common data points for comparison table for {dim}. Skipping row.")
            continue

        target_price_valid = target_price[valid_indices]
        quantum_denorm_valid = quantum_denormalized[dim][valid_indices]
        classical_denorm_valid = classical_denormalized[dim][valid_indices]

        q_rmse = np.sqrt(np.mean((target_price_valid - quantum_denorm_valid) ** 2))
        q_mae = np.mean(np.abs(target_price_valid - quantum_denorm_valid))
        q_mape = np.mean(np.abs((target_price_valid - quantum_denorm_valid) / (target_price_valid + 1e-10))) * 100

        c_rmse = np.sqrt(np.mean((target_price_valid - classical_denorm_valid) ** 2))
        c_mae = np.mean(np.abs(target_price_valid - classical_denorm_valid))
        c_mape = np.mean(np.abs((target_price_valid - classical_denorm_valid) / (target_price_valid + 1e-10))) * 100

        q_time = quantum_times[dim]
        c_time = classical_times[dim]

        # Calculate speedup (>1 means quantum is faster)
        speedup = c_time / q_time if q_time > 0 else float('inf')

        # Determine if quantum is better in each metric
        rmse_better = q_rmse < c_rmse
        mae_better = q_mae < c_mae
        mape_better = q_mape < c_mape
        time_better = q_time < c_time

        comparison_data.append({
            'Dimension': dim,
            'Classical RMSE': c_rmse,
            'Quantum RMSE': q_rmse,
            'RMSE Improvement': (c_rmse - q_rmse) / c_rmse * 100 if c_rmse > 0 else 0,
            'Classical MAE': c_mae,
            'Quantum MAE': q_mae,
            'MAE Improvement': (c_mae - q_mae) / c_mae * 100 if c_mae > 0 else 0,
            'Classical MAPE': c_mape,
            'Quantum MAPE': q_mape,
            'MAPE Improvement': (c_mape - q_mape) / c_mape * 100 if c_mape > 0 else 0,
            'Classical Time': c_time,
            'Quantum Time': q_time,
            'Speedup': speedup,
            'RMSE Better': rmse_better,
            'MAE Better': mae_better,
            'MAPE Better': mape_better,
            'Time Better': time_better
        })
    else:
        print(f"Error: Length mismatch for {dim} comparison table after alignment. Skipping row.")
```

Figure 15: Fair Classical Baseline Function part 2



```
Sample Circuit Used for Hybrid Perturbation Generation:
(Displaying circuit for 6 qubits)
```

Figure 16: Circuit Used for Hybrid Perturbation Generation

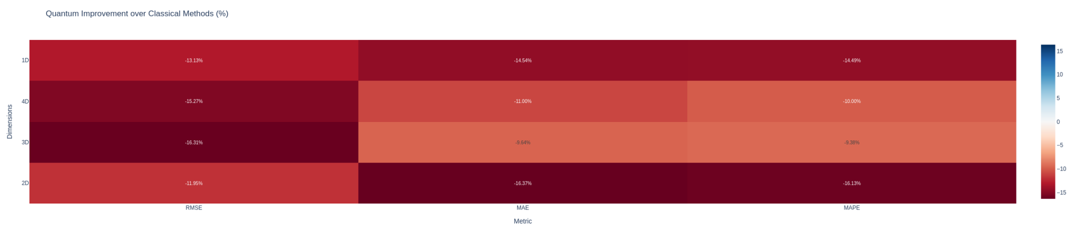


Figure 17: Quantum Improvement over Classical Methods (%)