INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

ITESO

Universidad Jesuita de Guadalajara

PROJECT 4 - DEEP LEARNING TRADING WITH MLOPS

Microstructure and Trading Systems

Jeanette Valenzuela Gutiérrez

Paulina Elizabeth Mejia Hori

Professor: Luis Felipe Gómez Estrada

Tlaquepaque, Jalisco — November 13, 2025

Executive Summary

Using fifteen years of Walt Disney Co. (DIS) market data, we developed a complete deep-learning trading system built on a CNN trained with rolling time-window features engineered from momentum, trend, volatility, and volume indicators. Because market behavior changes over time, we monitored data drift using Kolmogorov–Smirnov and Chi-square tests, which showed significant distributional shifts across most indicators. These findings helped explain the model's difficulty generalizing to newer regimes, even after using weighted loss functions, MLflow experiment tracking, and a structured MLOps workflow. We also deployed the final model through an inference API and validated its reliability through unit tests.

When integrated into a realistic backtesting engine—including transaction costs and short-selling frictions—the model produced highly conservative predictions, dominated by "hold" signals, which limited exposure and kept the portfolio essentially flat. The strategy preserved the initial USD 1,000,000 with minimal drawdown but generated almost no return, reflecting how class imbalance and feature drift shaped the model's behavior. Although the system did not deliver strong profitability, it successfully demonstrates a fully operational, reproducible, and extensible machine-learning trading pipeline, establishing a solid base for future improvements in labeling, exposure, and adaptation to shifting market conditions.

Introduction

Financial markets are dynamic, noisy, and continuously shaped by new information. These characteristics make predicting market movements a complex challenge, one that traditional statistical techniques often struggle to address. In recent years, deep learning has emerged as a powerful tool capable of capturing nonlinear structures and short-term dependencies in financial time series. However, successful real-world application requires more than training a neural network—it demands a full pipeline that includes data preparation, drift analysis, model tracking, backtesting, and deployment.

In this project, we developed a deep learning-based trading system for Walt Disney Co. (DIS) built on a Convolutional Neural Network (CNN). Our objective was to design a model capable of generating long, hold, or short trading signals based on engineered time-series features. We constructed over thirty technical indicators across momentum, trend, volatility, and volume categories, and shaped them into rolling windows so the CNN could learn from temporal sequences rather than individual data points.

Given the non-stationary nature of financial markets, we implemented data drift monitoring to evaluate how feature distributions changed across time. This allowed us to interpret performance in the context of market regime shifts. Additionally, we integrated MLflow into the workflow to ensure full experiment traceability and version control during model development.

Beyond predictive modeling, we simulated a realistic trading environment through a backtesting engine that incorporated transaction costs and short-selling frictions. Finally, to approximate real-world deployment, we exposed the trained model through a prediction API and validated its behavior with unit tests. This project therefore combines deep learning, financial modeling, and MLOps to build a system that is not only theoretically grounded but also operationally viable.

Glossary

1. **Algorithmic trading**
   The use of computer algorithms to automatically execute trading strategies based on predefined rules, indicators, or model predictions.
2. **Backtesting**
   A process used to evaluate a trading strategy by simulating its performance on historical data to assess profitability, risk, and robustness before live deployment.
3. **Calmar Ratio**
   A risk-adjusted performance metric that compares a strategy's annualized return to its maximum drawdown, emphasizing capital preservation during downturns.
4. **Class imbalance**
   A situation in classification problems where some classes occur much more frequently than others, often causing models to favor the majority class.
5. **Class weighting**
   A technique used to compensate for class imbalance by assigning higher penalties to misclassifications of underrepresented classes during training.
6. **CNN (Convolutional Neural Network)**
   A type of deep learning architecture that uses convolutional filters to extract local and hierarchical patterns from structured data, such as time series or images.
7. **Concept drift**
   A change in the relationship between input variables and the target variable over time, causing a model's predictions to become less accurate.
8. **Data drift**
   A shift in the statistical distribution of input data compared to the training data, often leading to model performance degradation.
9. **Deep learning**
   A subfield of machine learning that uses multi-layered neural networks to automatically learn complex and nonlinear relationships from large datasets.

10. **Feature engineering**
    The process of creating, transforming, or selecting input variables (features) to improve a model’s ability to detect relevant patterns and relationships.
11. **Feature scaling**
    A preprocessing step that standardizes or normalizes input data so all features contribute equally to model training and gradient optimization.
12. **Forward testing (paper trading)**
    The simulation of a trading strategy in real market conditions using virtual capital to evaluate real-time performance without financial risk.
13. **Hyperparameters**
    External configuration settings of a machine learning model (e.g., learning rate, batch size, number of layers) that influence training dynamics and performance.
14. **Look-ahead bias**
    An error in backtesting that occurs when future information is accidentally included in the historical simulation, leading to unrealistically good results.
15. **MDD (Maximum Drawdown)**
    The largest peak-to-trough decline in portfolio value before reaching a new high, used to measure downside risk.
16. **MLflow**
    An open-source platform designed to manage the entire machine learning lifecycle, including experiment tracking, model versioning, and deployment.
17. **MLOps (Machine Learning Operations)**
    A set of practices that integrate machine learning with DevOps principles to automate model training, deployment, monitoring, and maintenance.
18. **Momentum indicator**
    A technical indicator that measures the speed or strength of price movements to identify overbought or oversold market conditions.
19. **Normalization**
    A scaling technique that adjusts numerical data to a common range—typically between 0 and 1—to improve neural network training stability.
20. **Overfitting**
    A modeling error where a machine learning algorithm fits training data too closely, capturing noise instead of underlying patterns, and performing poorly on new data.
21. **Sharpe Ratio**
    A risk-adjusted return metric that measures the excess return of an investment relative to its total volatility.
22. **Short / Long / Hold**
    Trading positions where *long* indicates buying an asset expecting its price to rise, *short* indicates selling it expecting a decline, and *hold* represents no position.

23. **Signal generation**
    The process of converting model outputs or indicator values into actionable trading decisions, such as buy, sell, or hold.
24. **Sortino Ratio**
    A refinement of the Sharpe Ratio that measures excess return per unit of downside risk, focusing only on negative volatility.
25. **Technical indicator**
    A mathematical formula based on price, volume, or volatility data used to identify market patterns and potential trading opportunities.
26. **Time series**
    A sequence of data points indexed in chronological order, often used to represent financial variables like prices, returns, or volumes.
27. **Transaction costs**
    All costs associated with executing trades, such as commissions, bid-ask spreads, or slippage, which can reduce net profitability.
28. **Volatility clustering**
    A characteristic of financial time series where high-volatility periods are followed by more high volatility, and low by low, forming distinct clusters.
29. **Walk-forward testing**
    An evaluation technique that retrains and tests a model over sequential rolling time windows to mimic live trading conditions.
30. **Z-score standardization**
    A data transformation method that rescales features to have a mean of zero and a standard deviation of one, ensuring consistent input scales.

Theoretical Framework

Deep Learning in Algorithmic Trading

Deep learning is a branch of machine learning that uses multilayered neural networks capable of recognizing complex and nonlinear relationships within data. Unlike traditional algorithms that depend on predefined rules or manually built features, deep neural networks adjust millions of internal parameters to approximate functions and automatically learn meaningful patterns. In finance, this is especially useful because financial time series tend to be noisy, unstable, and highly nonlinear. By recognizing relationships between technical indicators, price movements, and overall market behavior, deep learning models can generate trading signals—such as *long*, *hold*, or *short* positions—or estimate expected returns.

Convolutional Neural Networks (CNNs), one of the most popular architectures in deep learning, use mathematical filters that move across the data to detect local patterns. Although they were originally designed for image recognition, the same concept can be applied to one-dimensional temporal data like stock prices. In this case, CNNs can identify short-term dependencies and small recurring structures—such as momentum shifts or volatility spikes—within sliding time windows. As layers are added, the network learns increasingly complex patterns, from simple short-term fluctuations to long-term interactions between multiple indicators. This makes CNNs not only computationally efficient but also highly effective for identifying meaningful patterns in multivariate financial data.

Using deep learning for algorithmic trading brings several clear benefits. Neural networks can automatically extract relevant information from raw or lightly processed data, which reduces the need for manual feature engineering. They also capture complex, nonlinear relationships between variables and can integrate many technical indicators and timeframes at once. In addition, these models can adapt to changing market conditions through retraining and, when properly regularized, often achieve higher predictive accuracy and better risk-adjusted performance compared to more traditional techniques.

However, deep learning also has limitations that are important to consider. These models require large amounts of clean data and considerable computational power—often needing GPU acceleration—to work efficiently. They are also prone to overfitting, especially when the training data does not represent future market behavior well. Another common issue is their lack of interpretability, since neural networks tend to operate as “black boxes,” making it difficult to understand or justify their decisions. Moreover, because financial markets can shift abruptly, deep learning models may lose accuracy when those regime changes occur unless they are monitored and retrained regularly.

In short, deep learning—especially through CNN architectures—has become a powerful tool in algorithmic trading because it can capture complex, dynamic, and nonlinear patterns that traditional models often miss. By learning directly from historical market data and recognizing subtle temporal dependencies, CNNs help transform raw information into actionable trading signals. Still, their real success depends on good data management, solid validation processes, and continuous model supervision to ensure reliable and consistent results in practical trading scenarios.

Time Series and Financial Indicators

Financial time series represent the evolution of asset prices, returns, or volumes over time and exhibit several distinctive characteristics that make them challenging to model. These data are highly volatile and often display *volatility clustering*, meaning that large price fluctuations tend to be followed by other large movements, while calm periods tend to cluster together. They are also non-stationary, since their statistical properties—such as mean, variance, and autocorrelation—change over time, which complicates the use of traditional models that assume stable distributions. Another common feature is the presence of high noise levels, reflecting unpredictable market dynamics driven by external factors such as news or investor sentiment. Financial time series also tend to exhibit heavy tails (leptokurtosis), indicating that extreme price movements occur more frequently than what a normal distribution would predict. Finally, they often show second-order dependence, where the autocorrelation of squared or absolute returns decays slowly, revealing persistence in volatility and memory effects within markets.

To interpret and extract structure from such complex data, analysts rely on technical indicators, which are mathematical transformations of price and volume data designed to reveal market tendencies. These indicators fall mainly into three categories: momentum, trend, and volatility.

- **Momentum indicators**, such as the *Relative Strength Index (RSI)* or the *stochastic oscillator*, measure the speed or strength of price changes to detect overbought or oversold conditions.
- **Trend indicators**, including *moving averages* (simple, exponential, or weighted) and *Moving Average Convergence Divergence (MACD)*, help determine the prevailing market direction and potential reversals.
- **Volatility indicators**, such as *Bollinger Bands* or the *Average True Range (ATR)*, estimate the magnitude of price fluctuations and help traders gauge market risk.
- **Volume indicators**, on the other hand, focus on market activity itself. They reflect the number of transactions and show how buying and selling pressure evolves over time. Tools such as *vertical and horizontal volume, delta, or VWAP* help evaluate the balance between supply and demand and confirm the strength of trends or possible reversals.

Each indicator provides a different perspective, and their combination allows traders or models to capture complementary information—momentum strength, directional bias, and market uncertainty—essential for developing robust algorithmic strategies.

Before feeding these variables into a neural network, feature scaling or normalization is an essential preprocessing step. Neural networks are sensitive to input magnitude, and features measured on different scales can dominate others during training. Proper scaling ensures that all inputs contribute equally and that gradients remain stable throughout the optimization process. Common scaling methods include Min-Max normalization, which maps data to a [0, 1] range, and Z-score standardization, which centers data around zero with unit variance. According to Pinheiro et al. (2025), scaling significantly influences the performance of models such as multilayer perceptrons and support-vector machines, whereas ensemble methods like Random Forest are more robust to unscaled data. Therefore, consistent and well-documented normalization is crucial to ensure both model performance and reproducibility.

Financial time series are complex, noisy, and dynamic, requiring careful preprocessing and the use of technical indicators to uncover structure within apparent randomness. Combined with appropriate scaling, these transformations enable deep-learning models to interpret patterns more effectively and to generate more reliable trading signals.

Class Imbalance and Target Variable Definition

In classification problems, class imbalance occurs when one or more categories appear much more frequently than others within the training data. This unequal distribution often biases machine learning models toward predicting the majority class, neglecting the minority class and producing misleadingly high accuracy but poor generalization. For example, in a dataset where “hold” positions represent 85 % of all trading signals while “long” and “short” occur less often, a naive model might always predict “hold” and still achieve high accuracy, despite failing to capture the true market dynamics.

Imbalanced datasets make training difficult because models rely on sufficient examples from all classes to learn meaningful decision boundaries. When the minority class appears rarely, the model struggles to identify its underlying patterns, resulting in biased decision surfaces, high false negatives, and unstable training behavior. Therefore, accuracy alone becomes an unreliable performance metric; instead, measures such as precision, recall, macro-F1, or PR-AUC (average precision) are better suited to evaluate imbalanced classification models (Google Developers, 2025).

A common solution is to introduce class weighting, which adjusts the loss function so that misclassifying a minority-class example incurs a higher penalty than misclassifying a majority-class example. This method effectively rebalances the learning process without changing the data distribution. In practice, libraries such as scikit-learn implement this through the parameter *class_weight='balanced'*, which automatically assigns weights inversely proportional to class frequencies. In deep-learning models, this concept is similarly applied through weighted loss functions (e.g., weighted cross-entropy) to prevent the network from ignoring rare classes. The weighting factor is usually computed as:

$$w_j = \frac{N}{K \times N_j}$$

where w_j is the weight for class j, N is the total number of samples, K the number of classes, and N_j the number of samples in class j. Correctly tuning these weights helps the model pay balanced attention to each signal type.

In this trading context, the target variable represents the model's trading signal, typically defined as three classes:

- **Long (+1):** The model predicts a price increase and opens a buy position.
- **Hold (0):** The model expects no significant change, so it remains neutral.
- **Short (–1):** The model predicts a price decrease and opens a sell position.

These labels are generated from historical price data based on thresholds applied to future returns (for example, labeling long if the return $> \alpha$, short if $< -\alpha$, and hold otherwise). Because markets usually spend most of the time in neutral conditions, the resulting dataset is naturally imbalanced, with far more "hold" cases than "long" or "short." Applying class weights or resampling techniques ensures that the model can learn to recognize both upward and downward opportunities with similar importance.

Handling class imbalance through weighted losses or sampling strategies is essential to prevent bias and improve predictive reliability. Defining the target variable carefully—consistent with market behavior and labeling thresholds—ensures that the model's outputs (long, hold, short) reflect realistic trading actions and align with the ultimate goal of profitable and balanced signal generation.

MLOps and Model Tracking

As machine learning models evolve from experimentation to real-world deployment, maintaining consistency, scalability, and reliability across all stages of the workflow becomes increasingly complex. MLOps—short for *Machine Learning Operations*—emerges as the framework that bridges this gap. Inspired by DevOps principles, MLOps integrates continuous integration, continuous delivery (CI/CD), and automation into the ML lifecycle to streamline model development, deployment, and monitoring. It ensures collaboration among data scientists, engineers, and IT operations while reducing technical debt and improving reproducibility (Sculley et al., 2015).

MLOps focuses on creating a structured and repeatable pipeline that encompasses data collection, model training, validation, deployment, and ongoing monitoring. Unlike traditional ML workflows, where each step is executed manually, MLOps automates these processes, enabling faster iteration and real-time feedback when data distributions shift—a crucial feature for financial markets where patterns evolve constantly. This automation enhances model reliability, reduces human error, and supports continuous improvement as new data becomes available.

A key component of an MLOps environment is MLflow, an open-source platform developed to manage the complete machine learning lifecycle. MLflow simplifies experiment tracking, model versioning, and deployment, providing a unified interface for documenting metrics, parameters, and artifacts generated during experimentation. Through its core modules—Tracking, Projects, Models, and Model Registry—MLflow enables teams to log and compare multiple model runs, register and store models, and seamlessly transition from research notebooks to production pipelines.

In this project's context, MLflow plays a central role in experiment management and model traceability. Each model trained for trading signal prediction can be tracked with its corresponding features, hyperparameters, and performance metrics, ensuring full reproducibility and auditability. The Model Registry allows version control of the CNN architectures used, while CI/CD integration supports automatic deployment of the best-performing models. Monitoring components detect data drift or performance decay, triggering retraining procedures when the model's predictive accuracy drops below a predefined threshold.

The adoption of MLOps and MLflow provides several benefits:

- **Efficiency:** Automation frees data scientists from repetitive manual tasks, accelerating experimentation cycles.
- **Scalability:** Pipelines can be replicated across different assets or time frames with minimal adjustment.
- **Governance:** Versioning and documentation guarantee transparency and compliance with research and operational standards.
- **Resilience:** Continuous monitoring and rollback mechanisms mitigate risks from data anomalies or unexpected market shifts.

Overall, implementing MLOps practices with MLflow transforms the trading model lifecycle from a static experiment into a dynamic and adaptive system. This ensures that deep learning models for financial forecasting remain consistent, scalable, and responsive to new market information—bridging the gap between innovation and operational stability.

Data Drift in Financial Markets

In machine learning, data drift refers to the gradual change in the statistical properties of input data over time, which can lead to a decline in model performance. Closely related is concept drift, which occurs when the relationship between input variables (features) and the target variable changes. In simple terms, data drift alters *what* the data looks like, while concept drift alters *how* the data relates to predictions. Both phenomena are particularly relevant in financial markets, where conditions are inherently dynamic and influenced by constantly evolving external factors.

In mathematical terms, data drift can be described as a change in the probability distribution of the input features over time:

$$P_t(X) \neq P_{t+1}(X)$$

while concept drift is represented by a change in the conditional distribution:

$$P_t(Y|X) \neq P_{t+1}(Y|X)$$

These shifts make models trained on historical data less accurate when applied to new observations.

In financial contexts, drift often arises from regime changes, market crises, macroeconomic shocks, or evolving investor behavior. For instance, a trading model trained during a period of low volatility may underperform when sudden geopolitical tensions or inflationary pressures trigger new market patterns. Similarly, concept drift can occur when relationships between indicators and returns change—such as when central bank policies alter how interest rates impact asset prices. These dynamics make financial time series especially prone to drift, as their underlying data-generating processes are rarely stationary.

Detecting data drift is therefore critical for maintaining the reliability of predictive trading systems. Common statistical approaches include the Kolmogorov–Smirnov (K–S) test, which compares the cumulative distributions of two samples to detect significant shifts, and the Chi-square test, often used for categorical variables. Additional metrics like the Population Stability Index (PSI) or Kullback–Leibler (KL) divergence quantify how much feature distributions differ between training and production datasets. In practice, these methods are applied periodically or in near real time to monitor whether the incoming market data still resemble the conditions under which the model was trained.

To mitigate drift, several strategies can be implemented:

- **Continuous monitoring:** Regularly track model performance metrics (accuracy, precision, or Sharpe ratio) to identify degradation trends early.
- **Adaptive retraining:** Schedule retraining using recent data or trigger it automatically once drift metrics exceed defined thresholds.
- **Weighting and resampling:** Adjust training data distributions to reflect new market regimes more accurately.

- **Domain adaptation:** Employ transfer learning or fine-tuning techniques to help the model adapt to new conditions without losing prior knowledge.

From a financial standpoint, data drift represents the inevitable evolution of market behavior. Price dynamics, investor sentiment, and macroeconomic indicators are not static, and models must be designed to evolve alongside them. By integrating drift detection and monitoring tools within the MLOps pipeline—such as automated statistical tests or dashboards like EvidentlyAI—analysts can maintain model validity and ensure that predictions remain aligned with current market realities.

Ultimately, understanding and addressing data drift is essential to sustain robustness and profitability in algorithmic trading systems. Models that adapt to changing data distributions preserve their predictive edge, while those that ignore drift risk becoming outdated and misaligned with the ever-shifting nature of financial markets.

Backtesting and Performance Metrics

Backtesting is a fundamental process in quantitative finance that allows traders and researchers to assess how a trading strategy would have performed in the past by simulating its execution on historical market data. This procedure helps evaluate a model's profitability, robustness, and risk exposure before real capital is committed. The core assumption behind backtesting is that strategies that performed well under historical conditions may maintain similar behavior under comparable market dynamics. However, it is essential to recognize that past performance does not guarantee future results—backtesting serves as a diagnostic and validation tool rather than a predictive one.

When executed properly, backtesting provides valuable insights into how a strategy responds to various market regimes—bullish, bearish, and sideways—and enables systematic optimization through parameter tuning and sensitivity analysis. Nevertheless, the method also has inherent theoretical limitations. The most common issues include overfitting, which occurs when a model is excessively adapted to past noise instead of true patterns; look-ahead bias, when future information is inadvertently used in the simulation; and data snooping, where testing multiple variations on the same dataset produces artificially successful results by chance. Additionally, survivorship bias (excluding delisted or bankrupt assets) and neglecting transaction costs, spreads, or slippage can lead to overly optimistic outcomes.

To mitigate these problems, it is standard practice to divide the available data into in-sample (training), validation, and out-of-sample (testing) segments, ensuring that evaluation is performed on data unseen during model design. Complementary approaches such as walk-forward testing—which retrains the model over rolling time windows—and forward testing (or paper trading), which evaluates performance in real time using simulated trades, further strengthen the reliability of results.

Backtesting vs. Scenario and Forward Analysis

While backtesting uses actual historical data to evaluate performance, scenario analysis tests strategies under hypothetical or extreme market conditions, such as interest rate shocks or volatility spikes. Forward performance testing, also known as paper trading, simulates trades in live market conditions using virtual capital to verify that the model's real-time behavior aligns with its backtested results.

Main Performance Metrics

Once a strategy has been tested, quantitative metrics are employed to measure its efficiency and risk-adjusted return. The most relevant indicators include:

Sharpe Ratio
The Sharpe Ratio (Sharpe, 1966) measures the excess return per unit of total risk and is defined as:

$$Sharpe = \frac{R_p - R_f}{\sigma_p}$$

where R_p is the portfolio return, R_f the risk-free rate, and σ_p the standard deviation of portfolio returns. Higher Sharpe ratios indicate superior risk-adjusted performance, although this metric penalizes upside and downside volatility equally and assumes a normal distribution of returns.

Sortino Ratio
The Sortino Ratio refines the Sharpe Ratio by considering only downside deviation, thus isolating harmful volatility:

$$Sortino = \frac{R_p - R_f}{\sigma_d}$$

where σ_d represents the standard deviation of negative returns. This metric provides a more realistic assessment of performance when investors are primarily concerned with downside risk.

Calmar Ratio
The Calmar Ratio compares the annualized return of a portfolio to its maximum drawdown (MDD) over a given period:

$$Calmar = \frac{Annualized\ Return}{MDD}$$

Introduced by Terry W. Young in 1991, this measure focuses on long-term capital preservation and is particularly useful for evaluating strategies exposed to large downturns.

Maximum Drawdown (MDD)
The Maximum Drawdown quantifies the largest peak-to-trough decline in portfolio value before a new high is reached:

$$MDD = \frac{Through - Peak}{Peak}$$

This indicator reflects the worst historical loss experienced by a strategy, offering a clear measure of downside risk. A lower MDD denotes stronger capital preservation.

Interpretation and Practical Considerations

Evaluating trading systems requires combining several performance ratios rather than relying on a single metric. While the Sharpe Ratio captures total risk efficiency, the Sortino Ratio isolates downside exposure, and the Calmar Ratio directly relates returns to drawdowns. Consistency across different metrics, timeframes, and datasets generally indicates a robust strategy.

Additionally, realistic backtesting should incorporate transaction costs, spreads, slippage, and liquidity constraints to approximate live trading conditions. Comparing in-sample, out-of-sample, and forward testing results helps detect overfitting and ensures that the model's performance generalizes beyond historical data.

Ultimately, backtesting and performance metrics form the foundation for validating algorithmic trading systems. By systematically analyzing profitability, volatility, and drawdown behavior under diverse market regimes, researchers and practitioners can refine models, mitigate risk, and increase the likelihood that simulated success translates into sustainable real-world performance.

Methodology and Model Design

The development of the deep-learning trading system followed an end-to-end methodological framework that integrates principles from quantitative finance, time-series analysis, deep learning, and modern MLOps practices. Each component, from the initial construction of the dataset to the deployment of the predictive model, was designed to ensure reproducibility, statistical rigor, and alignment with the theoretical concepts presented earlier.

1. Data Preparation

The methodology begins with the preparation of a clean, chronological dataset containing fifteen years of daily market data from **Walt Disney Co (DIS)**. Because financial time series are prone to noise, missing values, and structural inconsistencies, the first step involved validating data integrity, repairing gaps, and ensuring strict temporal ordering to avoid look-ahead bias. After stabilizing the dataset, it was divided chronologically into training, validation, and testing subsets, preserving the natural time dependency of financial data and minimizing the risk of information leakage.

2. Feature Engineering

After data cleaning, the next step consisted of transforming the raw price and volume series into a structured set of technical indicators suitable for CNN-based learning. The feature space was intentionally designed as a balanced mixture of momentum, trend, volatility, and volume-derived indicators, resulting in approximately thirty variables that capture Disney's multi-scale market behavior.

From a **momentum** perspective, the model employs oscillators such as the 14-day and 30-day RSI, percentage returns over 1-, 5-, and 20-day horizons, and momentum-style differences and slopes that help detect trend acceleration or potential reversals.

To represent **trend dynamics**, the pipeline incorporates simple moving averages (5, 20, and 50 days), percentage distance from the moving averages, and the full MACD suite (line, signal, and histogram), which provides a smoothed representation of price convergence and divergence.

To model **volatility**, several metrics were included: rolling standard deviations, daily range and range percentage, normalized Bollinger Band positioning, and a 14-day ATR with its price-scaled variant, offering a robust measure of typical price dispersion.

Finally, **volume-based indicators**—including short- and medium-term moving averages and a volume ratio—capture levels of market participation, helping to distinguish high-commitment moves from low-liquidity noise.

All indicators were computed consistently across the training, validation, and test segments. Rows containing missing values resulting from rolling windows were removed to avoid distortions. This produced a compact, expressive feature set allowing the CNN to detect meaningful temporal patterns while controlling complexity and overfitting risk.

3. Data Drift Monitoring

Because financial markets are highly non-stationary, monitoring data drift was a crucial component of the methodology. The behavior of the variables used during model development does not necessarily remain stable over time—market shocks, policy changes, volatility spikes, and regime shifts can all alter the distribution of features. For that reason, I compared the distributions of the features across the **training**, **validation**, and **testing** periods to understand how much the data changed throughout the pipeline.

To quantify these changes, two statistical drift-detection methods were applied:

• Kolmogorov–Smirnov (KS) Test

The KS test measures differences between the cumulative distributions of two samples.
A high KS statistic or a very low p-value indicates that a feature's shape changed significantly from one period to another. This allowed me to identify variables whose dynamics shifted notably—for example, features affected by sudden volatility changes or macroeconomic announcements.

• Chi-Square Drift Metric

This metric was used when features were discretized into bins or when frequency changes were relevant. It highlights whether a feature shows unusual spikes in certain value ranges or if the distribution becomes more imbalanced over time.

Drift analysis plays a fundamental role in production-oriented machine learning.
By detecting when the new data no longer resembles the data used during training, it becomes possible to:

- anticipate performance degradation,
- understand the causes behind unexpected prediction behavior,
- and define **retraining triggers** to keep the model up-to-date.

In this project, drift monitoring provided essential context during evaluation and aligned with the MLOps objective of maintaining model reliability over time.

4. CNN Model Architecture

Once the feature engineering stage was completed and the time-windowed datasets were ready, I designed and implemented a Convolutional Neural Network (CNN) specifically tailored for this trading project. The goal was to build a model capable of recognizing short-term temporal structures in the data, since the input to the model consists of multivariate rolling windows rather than isolated observations. The idea was that by feeding the model sequences of technical indicators, returns, and volatility-based features, the CNN would learn to detect patterns that tend to precede upward or downward movements—patterns that simpler models might overlook.

The first major component was the convolutional layer, which was responsible for scanning across each time window and extracting local patterns. In practice, this meant applying several small filters along the time dimension to detect how the features behave in very short segments—for example, a sudden increase in volatility, a sharp momentum reversal, or a recurring micro-trend within the window. These filters slide step-by-step across the input sequence and produce activation maps that represent where meaningful structures were found. During training, the network automatically adjusted the filters to detect the patterns that were most predictive for the three trading classes.

After the convolution operations, the next step was pooling, which essentially condensed the information from the convolutional outputs while reducing noise. Financial data is notoriously noisy and full of random micro-movements that are not useful for prediction, so pooling played a crucial role in summarizing the strongest patterns while preventing the model from reacting to isolated spikes. This step also helped reduce the dimensionality of the representation, making the model more efficient and less prone to overfitting.

Once the series had been transformed by convolution and pooling, the output was flattened and passed into fully connected (dense) layers. These dense layers acted as the model's decision-making stage. At this point, the model was no longer focused on individual time steps but instead on the combined representation of the entire window. Here, the network learned how the extracted temporal patterns interacted with one another and how those interactions correlate with future market direction. The dense layers allowed the model to transform the convolutional features into a structured internal representation that captured both short-term and aggregated dependencies.

The final layer of the architecture used a softmax activation to output probabilities for the three trading actions: long, hold, and short. This means the model produced a normalized probability distribution, making it easier to interpret which action had the highest confidence. Framing the problem as a multi-class classification helped simplify the trading logic and aligned the model's output with the signals required for backtesting and evaluation.

Throughout the design process, I kept the architecture relatively lightweight on purpose. In financial modeling, deeper networks tend to overfit quickly because the data contains more noise than persistent structure. For that reason, I limited the number of convolutional filters, kept the kernel size moderate, and avoided stacking too many layers. The aim was to strike a balance: enough complexity to learn meaningful patterns, but not so much that the model would memorize noise from the training period. This design decision also made the training process more stable and reduced the risk of divergence across different market regimes.

The construction of the CNN involved designing a flow in which the model first captured short-term temporal behavior, then summarized that behavior into higher-level features, and finally used those features to classify the expected market direction. The final architecture reflects a practical balance between theoretical suitability for time-series data and the real constraints of financial modeling, such as noise, instability, and the need to generalize across shifting market conditions.

5. Model Training and Experiment Tracking

Model training prioritized both predictive accuracy and methodological robustness. Several best practices were incorporated to ensure reliable learning and reproducibility.

Weighted Cross-Entropy Loss
Addresses label imbalance by assigning higher importance to underrepresented trading signals.
This prevents the model from defaulting to the majority class (“hold”).

Adam Optimizer
Selected for its adaptive learning rate and strong performance in environments with noisy gradients—common in financial datasets.

Early Stopping
Monitors validation loss and halts training once improvements plateau, reducing overfitting and unnecessary computation.

MLflow Tracking
All experiments were logged using MLflow, including:

- hyperparameters,
- model metrics,
- learning curves,
- artifacts and checkpoints.

This ensures full auditability, supports versioned experimentation, and aligns the project with core MLOps practices around transparency, traceability, and long-term model management.

By integrating MLflow into the workflow, the project achieves a higher standard of operational discipline.

It ensures that model decisions can be traced, reproduced, and compared—key expectations in any production-oriented financial system.

6. Backtesting Engine

Once the model reached stable predictive performance, its outputs were integrated into a backtesting engine capable of simulating trading behavior under historical conditions. The system translated model-predicted signals into executed trades while incorporating realistic market frictions—such as transaction costs, short-selling borrow fees, slippage, and predefined stop-loss and take-profit thresholds.

The backtest reproduced different market environments, allowing the analysis of how the strategy behaved in bullish, bearish, and sideways regimes. This stage emphasized not only predictive accuracy but the economic viability of the trading strategy.

7. Performance Evaluation

Following backtesting, the strategy was assessed using industry-standard performance metrics.

- **Sharpe Ratio** measured risk-adjusted excess returns based on total volatility
- **Sortino Ratio** focused specifically on downside volatility
- **Calmar Ratio** compared annualized returns relative to maximum drawdown
- **Maximum drawdown (MDD)** quantified the worst historical peak-to-trough decline.

Together, these metrics offered a multidimensional view of profitability, risk exposure, and robustness—providing a more meaningful interpretation than classification accuracy alone.

8. Deployment and API Inference

For the deployment stage, **we focused on creating a simple but functional prediction API** that would simulate how the model could be used in a real trading environment. Our goal was not just to train a model, but to demonstrate that it could actually operate as part of an end-to-end system capable of receiving data and returning trading signals instantly.

To achieve this, we deployed the final CNN model through an API endpoint. The API loads the best model saved during training and exposes a predict route where external systems can send feature vectors (formatted just like the windows used during training). When a request arrives, the API preprocesses the input, runs it through the CNN, and returns the predicted probabilities for the three trading classes—long, hold, or short. This setup mirrors the type of real-time service that an automated trading system or a market-monitoring tool would use in production.

As part of making the deployment reliable, we implemented unit tests to validate the entire inference workflow. These tests ensured that:

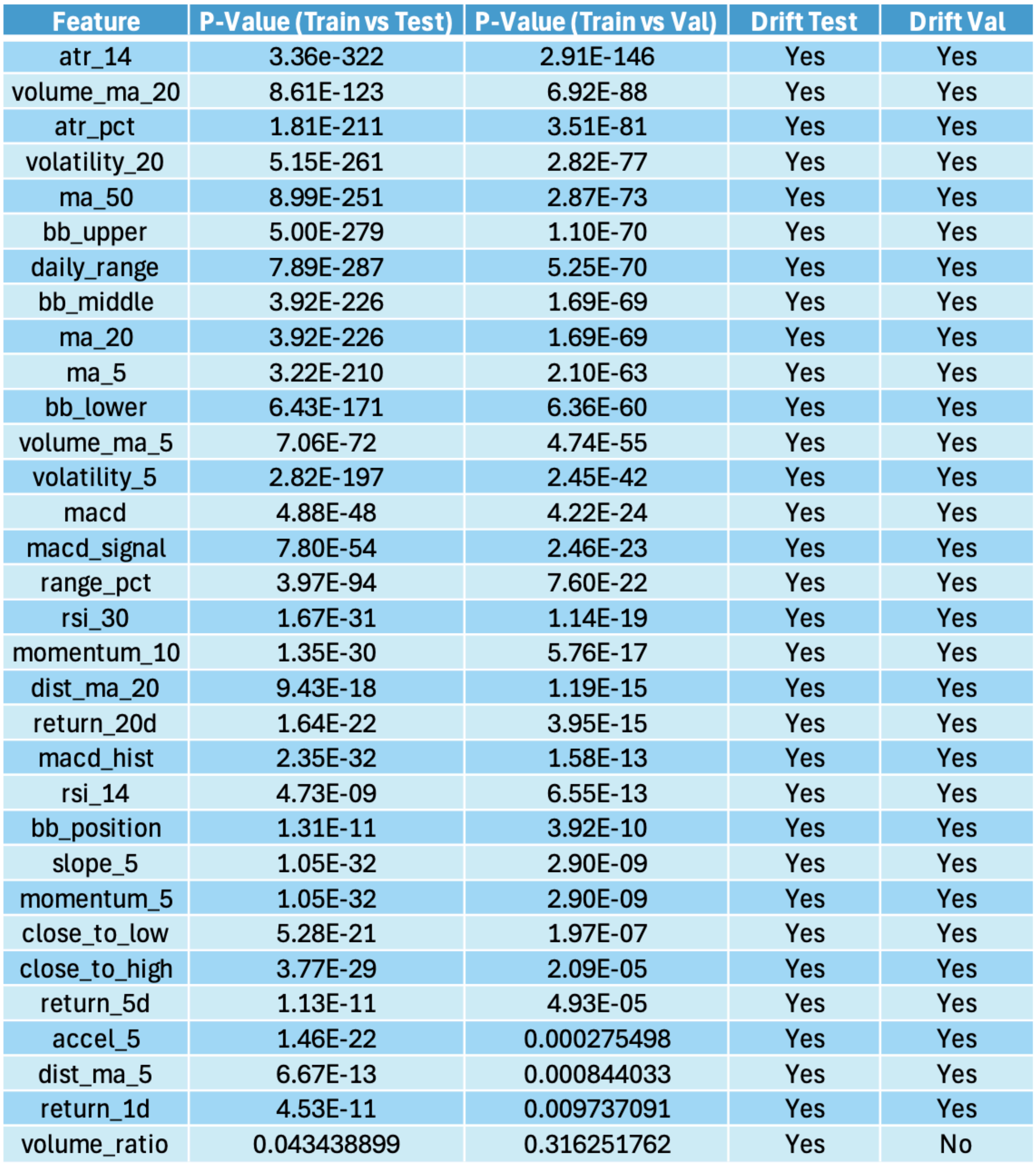
- the model loaded correctly every time the API started,
- the input format was handled safely,
- malformed or incomplete requests returned clear error messages instead of crashing the service,
- and the API always returned outputs in the expected structure.

By doing this, we reduced the operational risks that typically appear when machine-learning models are moved from notebooks into real systems.

Results and Performance Analysis

The final pipeline was evaluated on fifteen years of daily data for Walt Disney Co. (DIS), split chronologically into training (2010–2019), test (2019–2022) and validation (2022–2025) windows. After feature generation, the effective sample sizes were 2,214 rows for training and 706 rows for both test and validation, with 32 technical indicators per day.

From a statistical perspective, the drift analysis confirmed that Disney's feature distributions changed substantially across time. Using Kolmogorov–Smirnov tests, drift was detected in 28 out of 32 indicators (87.5 %) when comparing the training window against both the test and validation periods. The strongest shifts appeared in medium- and long-term trend indicators (MA(5), MA(20), MA(50)) and the Bollinger Band components (bb_middle, bb_upper, bb_lower), followed by volatility and range measures. This pattern is consistent with regime changes in Disney's price dynamics: the behavior of moving averages and volatility bands in the most recent years looks statistically different from the decade in which the model was trained. These results help interpret any performance deterioration as at least partly driven by non-stationarity rather than pure model error.



Feature	P-Value (Train vs Test)	P-Value (Train vs Val)	Drift Test	Drift Val
atr_14	3.36e-322	2.91E-146	Yes	Yes
volume_ma_20	8.61E-123	6.92E-88	Yes	Yes
atr_pct	1.81E-211	3.51E-81	Yes	Yes
volatility_20	5.15E-261	2.82E-77	Yes	Yes
ma_50	8.99E-251	2.87E-73	Yes	Yes
bb_upper	5.00E-279	1.10E-70	Yes	Yes
daily_range	7.89E-287	5.25E-70	Yes	Yes
bb_middle	3.92E-226	1.69E-69	Yes	Yes
ma_20	3.92E-226	1.69E-69	Yes	Yes
ma_5	3.22E-210	2.10E-63	Yes	Yes
bb_lower	6.43E-171	6.36E-60	Yes	Yes
volume_ma_5	7.06E-72	4.74E-55	Yes	Yes
volatility_5	2.82E-197	2.45E-42	Yes	Yes
macd	4.88E-48	4.22E-24	Yes	Yes
macd_signal	7.80E-54	2.46E-23	Yes	Yes
range_pct	3.97E-94	7.60E-22	Yes	Yes
rsi_30	1.67E-31	1.14E-19	Yes	Yes
momentum_10	1.35E-30	5.76E-17	Yes	Yes
dist_ma_20	9.43E-18	1.19E-15	Yes	Yes
return_20d	1.64E-22	3.95E-15	Yes	Yes
macd_hist	2.35E-32	1.58E-13	Yes	Yes
rsi_14	4.73E-09	6.55E-13	Yes	Yes
bb_position	1.31E-11	3.92E-10	Yes	Yes
slope_5	1.05E-32	2.90E-09	Yes	Yes
momentum_5	1.05E-32	2.90E-09	Yes	Yes
close_to_low	5.28E-21	1.97E-07	Yes	Yes
close_to_high	3.77E-29	2.09E-05	Yes	Yes
return_5d	1.13E-11	4.93E-05	Yes	Yes
accel_5	1.46E-22	0.000275498	Yes	Yes
dist_ma_5	6.67E-13	0.000844033	Yes	Yes
return_1d	4.53E-11	0.009737091	Yes	Yes
volume_ratio	0.043438899	0.316251762	Yes	No

Figure 1. Results for al Drift Metrics

At the classification level, the CNN was trained on sliding windows of 30 days and three target classes (short, hold, long). The label distribution in the training set was clearly imbalanced, with 66.1 % “hold”, 18.9 % “long” and 15.0 % “short”. Class weights were therefore applied in the loss function to penalize mistakes on minority classes more strongly. After 11 epochs and early stopping, the model achieved a training accuracy of 56.64 % with a loss of 0.8679, while validation accuracy remained at 33.73 % with a loss of 1.2266. This gap between train and validation performance suggests that the network learned patterns that do not generalize perfectly to newer market conditions, something that aligns with the strong drift detected in the indicators.

The prediction distribution on the validation window illustrates how the model translated these learned patterns into trading signals. Out of 676 sequences, 647 (95.7 %) were classified as hold, 27 (4.0 %) as short and only 2 (0.3 %) as long. In practice, this means the CNN behaves as a very conservative signal generator: it prefers to stay out of the market most of the time and only opens positions in a small fraction of days. From a risk-management perspective this reduces exposure, but it also limits the possibility of capturing meaningful trends.

These signals were then passed to the backtesting engine, starting from an initial capital of 1,000,000 USD and incorporating transaction costs and borrowing rates. Over the full validation period, the strategy executed 18 trades with a win rate of 44.44 %. The average gain per winning trade was 5.60 USD, while the average loss per losing trade was –2.72 USD. The final capital reached 1,000,019.95 USD, which corresponds to a net return that is effectively 0 % once rounded to two decimals. The equity curve remains almost flat around the initial capital, with very small oscillations and a maximum drawdown close to zero.

Because returns are so small and volatility of the equity curve is extremely low, the risk-adjusted metrics behave in a somewhat pathological way. The Sharpe Ratio is reported as -- 1196.309, not because the strategy is losing massive amounts of money, but because the denominator (standard deviation of returns) is near zero while the average excess return is also very close to zero and slightly negative. In contrast, the Calmar Ratio is 0.38, reflecting that even a tiny positive annualized return divided by a practically negligible drawdown produces a moderate value. Taken together, these numbers indicate that the strategy behaves more like a capital-preservation mechanism than an aggressive alpha-generating system: it almost never deviates from the starting capital, but it also does not generate economically interesting profits.

Comparing these results with the classification metrics highlights an important point from the theoretical framework: higher predictive accuracy does not automatically translate into better trading performance. Even though the model learns to distinguish some patterns in the training data, the combination of class imbalance, conservative decision boundaries, and strong data drift makes the live signal stream dominated by “hold” decisions. As a result, the portfolio spends most of the time uninvested, so its final value ends up very close to the initial 1,000,000 USD.

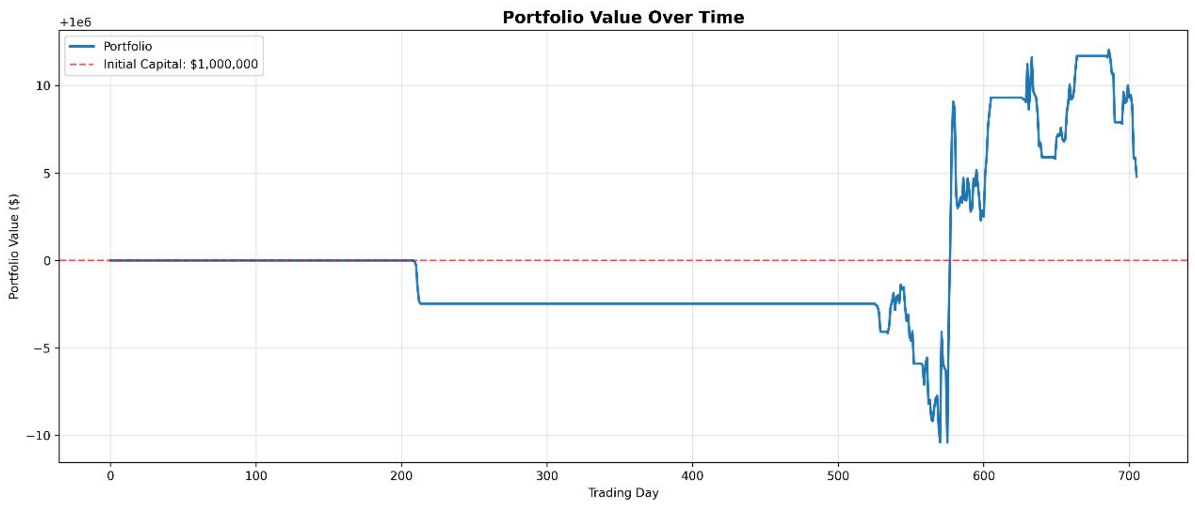


Figure 2. Equity curve of the strategy during the validation period (initial capital = 1,000,000 USD)

From an operational standpoint, the API deployment successfully loaded the trained CNN and the scaler, exposing endpoints for single and batch predictions as well as a health check. This confirms that, beyond the numerical results, the end-to-end pipeline—from data preparation and model training to signal generation and serving—works as intended and can be extended in future iterations.

Overall, the empirical evaluation shows a coherent picture: the model is stable, conservative, and robust in terms of capital preservation, but its profitability is limited under the current configuration. The strong feature drift across time, the imbalance toward hold signals, and the cautious exposure profile all help explain why the equity curve remains essentially flat. These findings point directly to the next steps for improvement, such as adjusting the labeling thresholds, relaxing the decision rule to increase exposure, or retraining the model on shorter, more recent windows to adapt more quickly to changing market regimes.

Conclusions

The results of the project highlight both the potential and the limitations of applying deep learning to financial time series. The CNN successfully learned patterns from historical data and behaved consistently across different stages of the pipeline. However, the strong dominance of “hold” predictions and the conservative behavior observed in backtesting indicate that the model remained cautious in taking directional positions. This outcome is aligned with the significant data drift detected between training and more recent periods, suggesting that the model encountered market conditions materially different from those it learned.

While the strategy achieved excellent capital preservation—maintaining the portfolio near its initial value with almost no drawdown—it did not generate meaningful returns during the evaluation window. This reinforces a key concept in quantitative finance: model accuracy alone does not guarantee profitable trading performance. The imbalance of labels, the structural drift in the indicators, and the cautious exposure profile all limited the model's ability to capture profitable opportunities.

On the operational side, the project successfully implemented the infrastructure expected in a real-world machine learning system. Experiment tracking, drift analysis, model deployment, and API-based inference were all completed and validated, demonstrating that the pipeline is functional and extendable.

These findings point toward several improvement directions: refining the labeling thresholds, adjusting class weights, increasing the model's exposure, and retraining on shorter, more recent windows to better adapt to current regimes. Despite its conservative behavior, the system built in this project forms a strong foundation for future iterations of deep learning-based trading models that integrate MLOps principles for reliability and scalability.

References

Ali, M. (2023, January 11). *Understanding data drift and model drift: Drift detection in Python.* DataCamp. https://www.datacamp.com/tutorial/understanding-data-drift-model-drift

Bergmann, D. (n.d.). *What is deep learning?* IBM Think. IBM. https://www.ibm.com/think/topics/deep-learning

Bhuiyan, M. S. M., Rafi, M. A., Rodrigues, G. N., Hossain Mir, M. N., Ishraq, A., Mridha, M. F., & Shin, J. (2025). *Deep learning for algorithmic trading: A systematic review of predictive models and optimization strategies. Journal of Computational Finance and Data Analytics, 5*(2). https://www.sciencedirect.com/science/article/pii/S2590005625000177

Canuma, P. (2025, 14 marzo). MLOps: What It Is, Why It Matters, and How to Implement It. neptune.ai. https://neptune.ai/blog/mlops

Chen, G., Chen, Y., & Fushimi, T. (2017). *Application of deep learning to algorithmic trading.* Stanford University. https://cs229.stanford.edu/proj2017/final-reports/5241098.pdf

Chen, J. (2025, 1 septiembre). *Backtesting in Trading: Definition, Benefits, and Limitations*. Investopedia. https://www.investopedia.com/terms/b/backtesting.asp
Chen, W., Yang, K., Yu, Z., Shi, Y., & Chen, C. L. P. (2024). A survey on imbalanced learning: latest research, applications and future directions. *Artificial Intelligence Review*, *57*(6). https://doi.org/10.1007/s10462-024-10759-6

Communications. (2024, 5 noviembre). «Deep learning» y «machine learning»: en qué se diferencian los dos grandes cerebros de la era digital. BBVA NOTICIAS. https://www.bbva.com/es/innovacion/deep-learning-y-machine-learning-en-que-se-diferencian-los-dos-grandes-cerebros-de-la-era-digital/

Data Drift vs. Concept Drift: What Is the Difference? - Dataversity. (2025, 15 septiembre). Dataversity. https://www.dataversity.net/articles/data-drift-vs-concept-drift-what-is-the-difference/

Datasets: Class-imbalanced datasets. (s. f.). Google For Developers. https://developers.google.com/machine-learning/crash-course/overfitting/imbalanced-datasets

Deepika. (2025, 25 agosto). *What is Portfolio Drift? The Silent Risk That Rebalancing Solves*. GIGAPRO. https://www.gwcindia.in/gigapro/blog/what-is-portfolio-drift-the-silent-risk-that-rebalancing-solves/

Fernando, J. (2025, 15 septiembre). *Sharpe Ratio: Definition, Formula, and Examples*. Investopedia. https://www.investopedia.com/terms/s/sharperatio.asp

Hantec Markets. (s. f.). https://hmarkets.com/es/blog/que-son-los-indicadores-de-trading/

Hayes, A. (2025, 1 julio). *Understanding Maximum Drawdown (MDD): Key Insights and Formula*. Investopedia. https://www.investopedia.com/terms/m/maximum-drawdown-mdd.asp#toc-deeper-insights-into-maximum-drawdown

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

IBM. (n.d.). *¿Cómo funcionan las redes neuronales convolucionales?* IBM Think. https://www.ibm.com/mx-es/think/topics/convolutional-neural-networks#:~:text=Ahora%2C%20las%20CNN%20proporcionan%20un,patrones%20dentro%20de%20una%20imagen

Kamaldeep. (2025, 1 mayo). How to Improve Class Imbalance using Class Weights in Machine Learning? Analytics Vidhya. https://www.analyticsvidhya.com/blog/2020/10/improve-class-imbalance-class-weights/

Kenton, W. (2025b, octubre 5). *Understanding the Calmar Ratio: Risk-Adjusted Returns for Hedge Funds*. Investopedia. https://www.investopedia.com/terms/c/calmarratio.asp

Kenton, W. (2025, 6 junio). *Sortino Ratio: Definition, Formula, Calculation, and Example*. Investopedia. https://www.investopedia.com/terms/s/sortinoratio.asp

Mattos, A. A. (2025, 17 abril). *¿Qué es el Backtesting y para qué sirve?* Rankia. https://www.rankia.mx/blog/forex-mexico/6791082-que-backtesting-para-sirve

MLFlow Tracking | MLFlow. (s. f.). https://mlflow.org/docs/latest/ml/tracking/

Mucci, T., & Stryker, C. (n.d.). *What is MLOps?* IBM Think. https://www.ibm.com/think/topics/mlops

Ngubeni, N. (2022, 27 julio). Backtesting: Cómo testear una estrategia de trading. *IG*. https://www.ig.com/es/estrategias-de-trading/backtesting-como-testear-una-estrategia-de-trading-220720

Pinheiro, J. M. H., de Oliveira, S. V. B., Silva, T. H. S., Saraiva, P. A. R., de Souza, E. F., Godoy, R. V., ... & Becker, M. (2025). The impact of feature scaling in machine learning: Effects on regression and classification tasks. arXiv preprint arXiv:2506.08274.

syedirfan@intellectyx.com. (2025, 15 julio). *Understanding Data Drift and Why It Happens*. DQLabs. https://www.dqlabs.ai/blog/understanding-data-drift-and-why-it-happens/

Vdk, S. (2025, 14 mayo). Indicadores de volumen en trading: qué son y cómo utilizarlos. Atas.net. https://atas.net/es/analisis-de-volumen/indicadores-de-volumen-en-trading-que-son-y-como-utilizarlos/

What is MLflow? (s. f.). https://www.mlflow.org/docs/2.9.2/introduction/index.html