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

Introduction

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 > α, short if < –α, 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.

The methodology begins with the preparation of a clean, chronological dataset containing fifteen years of daily market data from The Coca-Cola Company (KO). Since financial time series are prone to noise, missing values, and structural inconsistencies, the first step consisted of validating data integrity, correcting gaps, and ensuring a strictly sequential ordering to avoid any form of look-ahead bias. Once the dataset was stabilized, the data were divided chronologically into training, validation, and testing subsets, preserving the temporal structure of the series and reducing the risk of information leakage into future periods.

Once the raw data were cleaned and split, the next step consisted of engineering a structured set of technical indicators to transform the raw price series into a richer representation for CNN. The goal was to build a balanced feature set that captures momentum, trend, volatility, and volume dynamics of DIS price over time, while

keeping the total number of variables at a manageable scale for stable model training (around thirty indicators).

From a momentum perspective, the feature set includes classic oscillators such as the Relative Strength Index (RSI)computed over 14 and 30 days, which quantify the speed and magnitude of recent price movements. Short- and medium-term percentage returns (1, 5, and 20 days) complement this view by encoding directional changes directly in return space. Additional momentum-style features, such as price differences over 5 and 10 days and a simple slope and acceleration measure (first and second differences over a 5-day horizon), help the model detect changes in trend intensity and inflection points.

To capture the underlying trend structure, several simple moving averages (SMA) are added over different horizons (5, 20, and 50 days). These are combined with the distance from the moving averages in percentage terms, which indicates how extended the current price is relative to its recent baseline—useful for detecting potential mean-reversion or overextension regimes. On top of that, the MACD (Moving Average Convergence Divergence) and its signal and histogram components provide a smoother, multi-scale view of trend and momentum by comparing fast and slow exponential moving averages.

Volatility is modeled from different angles. First, the pipeline computes rolling standard deviations of returns over short (5-day) and medium (20-day) windows to approximate local volatility levels. Second, it derives the daily range(high–low) and its percentage relative to the close, which reflects intraday price dispersion. Third, Bollinger Bands (20-day moving average plus/minus two standard deviations) are used together with a band position indicator, which normalizes the current price between the upper and lower bands and helps identify whether the asset is trading in relatively extreme zones. In addition, the Average True Range (ATR) over 14 days and its percentage of price provide a more robust measure of typical price movement, taking into account gaps between sessions.

Volume-related features are included to incorporate information about market participation and pressure behind price moves. The model uses short- and medium-term moving averages of volume (5 and 20 days), together with a volume ratio that compares current volume to its 20-day baseline. These features help distinguish high-volume breakouts from low-interest fluctuations. Finally, simple relative price ratios, such as close-to-high and close-to-low, summarize where the closing price sits within the daily range, adding another microstructure perspective.

All indicators are computed consistently across the training, validation, and test splits. Rows containing initial missing values introduced by rolling windows are removed to avoid artifacts during training. The result is a compact yet expressive feature matrix that encodes KO’s price, volatility, and volume behavior over multiple

horizons, providing the CNN with enough structure to detect meaningful temporal patterns while controlling complexity and overfitting risk.

A key element of the methodology involved monitoring data drift. Since financial markets are non-stationary environments, the statistical properties of features may change significantly over time due to macroeconomic shocks, regime transitions, or volatility spikes. To quantify this phenomenon, distributional comparisons were performed between the training, validation, and testing windows using statistical measures such as the Kolmogorov–Smirnov test and Chi-square metrics. Identifying drift allowed us to understand when the model faced market conditions different from those it was originally trained on, providing context for evaluating performance degradation or shifts in predictive behavior.

After establishing a robust feature space, a convolutional neural network (CNN) architecture was selected for its ability to capture local temporal dependencies within sliding time windows. CNNs are particularly effective at identifying short-term structures in multivariate time series, such as momentum bursts or volatility transitions, and progressively integrating them through deeper layers into higher-level representations. The design of the architecture balanced model complexity with the risk of overfitting, incorporating convolutional layers, pooling operations, dense layers, and a final softmax activation for multi-class prediction.

Training the model required careful control over optimization dynamics, regularization, and experiment monitoring. The network was trained using cross-entropy loss with class weighting, and the Adam optimizer was selected for efficient gradient-based learning. Early stopping criteria were applied to prevent overfitting by halting training when validation performance no longer improved. Throughout the training process, MLflow was used to track parameters, metrics, and artifacts, enabling reproducibility and systematic comparison between experiments. This experiment-tracking component reflects standard MLOps practices and ensures transparency throughout the entire workflow.

Once the model achieved stable performance, it was integrated into a backtesting system designed to simulate real trading behavior under historical market conditions. The backtesting engine transformed model predictions into trading decisions, accounting for realistic frictions such as transaction costs, short-selling borrow rates, slippage, and position-management rules including stop-loss and take-profit thresholds. This simulation allowed us to examine not only whether the model was accurate in classification terms but—more importantly—whether its predictions translated into economically meaningful returns. The backtesting stage reproduced different market regimes, enabling a detailed evaluation of how the strategy behaved under bullish, bearish, and sideways conditions.

Following the backtesting analysis, the strategy was evaluated through industry-standard performance metrics. The Sharpe Ratio provided a measure of

risk-adjusted return based on overall volatility; the Sortino Ratio refined this perspective by focusing exclusively on downside volatility; the Calmar Ratio related annualized return to the maximum historical drawdown; and the maximum drawdown itself quantified the worst observed peak-to-trough loss. Together, these metrics provided a comprehensive assessment of profitability, risk, and robustness, offering a more meaningful interpretation than accuracy alone.

To approximate a production setting, the final model was deployed through a prediction API capable of serving live inference requests. This component allowed external applications to submit feature data and receive trading signals in real time, illustrating how the system would operate in a real-world environment. Automated tests ensured that the model loaded correctly, produced valid outputs, and handled malformed requests gracefully—reducing operational risk and confirming the stability of the deployment pipeline.

Finally, a drift-monitoring dashboard (under development) will complement the deployment process by visualizing distribution shifts across time, highlighting features with significant drift, and contextualizing these changes with market events such as regime transitions or volatility surges. This dashboard closes the MLOps loop by enabling continuous monitoring, early drift detection, and timely retraining decisions.

Overall, the methodology integrates financial theory, deep learning, and MLOps into a coherent and rigorous pipeline. Each stage contributes to the construction of a model that is not only predictive but also operationally viable, interpretable within financial contexts, and capable of adapting to the evolving nature of real markets.

Results and Performance Analysis

Conclusions

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