Supplementary Material for: TRADES: Generating Realistic Market Simulations with Diffusion Models

Paper #5552

1 Denoising diffusion probabilistic model

Diffusion models are latent variable models of the form $p_\theta(\mathbf{x}_0) := \int p_\theta(\mathbf{x}_{0:T})d\mathbf{x}_{1:T}$, where $\mathbf{x}_1, ..., \mathbf{x}_T$ are latents of the same dimensionality as the original data sample $x_0 \approx q(x_0)$. The objective of diffusion models is to learn a model distribution $p_\theta(x_0)$ that approximates the data distribution $q(x_0)$. Diffusion probabilistic models [10] are latent variable models composed of two Markov chain processes, i.e., the forward and reverse processes. The forward process is defined as in Eq. (1).

$$q(\mathbf{x}_{0:T}) := q(\mathbf{x}_0)\prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}), \tag{1}$$

where $q(\mathbf{x}_t|\mathbf{x}_{t-1}) := \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$. We start from an initial sample x_0 and add a small amount of Gaussian noise with every step for T steps, according to a variance schedule $\beta_1, ..., \beta_T$. The schedule is deterministic and defined, so $\mathbf{x}_T$ is pure Gaussian noise. Sampling of $\mathbf{x}_t$ can be define in a closed form $q(\mathbf{x}_t|\mathbf{x}_0) := \mathcal{N}(\mathbf{x}_t; \sqrt{\alpha_t}\mathbf{x}_{t-1}, (1-\alpha_t)\mathbf{I})$, where $\alpha_t := 1-\beta_t$ and $\alpha_t := \prod_{i=1}^{t}\alpha_i$. During the reverse process, $\mathbf{x}_T$ is denoised to recover $\mathbf{x}_0$ following the Markov chain process in Eq. (2).

$$\begin{aligned} p_\theta(\mathbf{x}_{0:T}) &:= p(\mathbf{x}_T)\prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t), \\ p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) &:= \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)) \end{aligned} \tag{2}$$

The reverse process model is trained with the variational lower bound of the likelihood of $\mathbf{x}_0$ as in Eq. (3).

$$\begin{aligned} \mathcal{L}_{vlb} := \mathbb{E}_q \Bigg[& \underbrace{-p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{L_0} \\ & + \sum_{t=2}^{T} \underbrace{D_{KL}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \,||\, p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_{t-1}} \\ & + \underbrace{D_{KL}(q(\mathbf{x}_T|\mathbf{x}_0) \,||\, p_\theta(\mathbf{x}_T)}_{L_T} \Bigg]. \end{aligned} \tag{3}$$

Since both q and p_θ are Gaussian, D_{KL} (the Kullback–Leibler divergence) can be evaluated in a closed form with only the mean and covariance of the two distributions. Ho et al. [4] propose the following reparametrization of $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$:

$$\boldsymbol{\mu}_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\alpha_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right), \tag{4}$$

where $\mathbf{x}_t = \sqrt{\alpha_t}\mathbf{x}_0 + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}$ s.t. $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and $\boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t) = \sigma_t^2\mathbf{I}$ where

$$\sigma_t^2 = \begin{cases} \beta_1 & t = 1 \\ \tilde{\beta}_t & 1 < t \leq T \end{cases} \quad \text{and } \tilde{\beta}_t = \frac{1-\alpha_{t-1}}{1-\alpha_t}\beta_t, \tag{5}$$

where $\boldsymbol{\epsilon}_\theta$ is the trainable function approximated by the neural network, intended to predict $\boldsymbol{\epsilon}$ from $\mathbf{x}_t$. As shown in [11], the denoising function given by Eq. (4) is equivalent to a score model rescaled for score-based generative models. Using this parameterization, Ho et al. [4] demonstrated that the inverse process can be learned by minimizing the simplified objective function in Eq. (6).

$$\mathcal{L}_{\text{simple}}(\theta) := \mathbb{E}_{t,\mathbf{x}_0,\boldsymbol{\epsilon}}\left[||\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)||^2\right] \tag{6}$$

The denoising function $\boldsymbol{\epsilon}_\theta$ aims to recover the noise vector $\boldsymbol{\epsilon}$ that corrupted its input $\mathbf{x}_t$. This training objective can be interpreted as a weighted variant of denoising score matching, a method for training score-based generative models [11, 12].

2 Overview of FI-2010 dataset

The FI-2010 dataset [8] is the most used LOB dataset in the field of the deep learning application to limit order book [17, 18, 13, 14], especially for forecasting tasks. It contains LOB data from five Finnish companies listed on the NASDAQ Nordic stock market: i.e., Kesko Oyj, Outokumpu Oyj, Sampo, Rautaruukki, and Wärtsilä Oyj. The data covers 10 trading days from June 1st to June 14th, 2010. It records about 4 million limit order snapshots for 10 levels of the LOB. The authors sample LOB observations every 10 events, totaling 394,337 events. The label of the data, representing the mid-price movement, depends on the percentage change between the actual price p_t and the average of the subsequent h (chosen horizon) mid-prices:

$$l_t = \frac{m_+(t) - p_t}{p_t}$$

The labels are then decided based on a threshold (θ) for the percentage change (l_t). If $l_t > \theta$ or $l_t < -\theta$, the label is an *up* in the former case, and a *down* in the latter. When $-\theta < l_t < \theta$, the label is *stationary*. The dataset provides the time series and the classes for five horizons $h \in H = \{1, 2, 3, 5, 10\}$. The authors of the dataset employed a single threshold $\theta = 2 \times 10^{-3}$ for all horizons, balancing only the case for $h = 5$.

Besides the absence of message files, FI-2010 comes already preprocessed such that the original LOB cannot be reconstructed, impeding comprehensive experimentation. Finally, as shown in [17], this method of labeling the data is susceptible to instability.

3 Implementation Details

All the experiments are run with an RTX 3090 and an A100. We implemented early stopping with a patience of 5 epochs and then used the checkpoint with the best validation loss for simulation. On average we trained each model for 70,000 steps until convergence. Lastly, we halve the learning rate each time the validation loss exceeds the previous one.

Pre-processing. To properly train the neural network, we preprocess the data. The data contains discrete and continuous features, so we rely on different preprocessing methods for each feature type. We replace the timestamp with the time distance from the previous order, which we normalize according to the z-score. We also perform a z-score normalization of message and order book file volume and prices. We encode the event type with an embedding layer. Finally, we remove the order ID and add the depth[1] to the orders feature vector.

Post-processing. The diffusion output is post-processed before being handled through the exchange simulation. First, we report the continuous features (offset, size, depth) to the original scale using the mean and standard deviation computed on the training set; the direction is discretized by simply checking if it is > 0 (buy) or < 0 (sell). Lastly, the order type is discretized between 0 (limit order), 1 (cancel order), and 2 (market order) based on the index of the nearest[2] embedding layer row. If the order is a limit order, the depth is utilized as the price. For instance, if the depth is 10 and the direction is "buy", the order will be positioned at a 10-cent difference from the best available bid price. Occasionally, it happens that the size is negative or the depth is over the first 10 levels; in that case, the order is discarded, and a new one is generated. Approximately 25% of the time, the generated cancel order does not directly correspond to an active limit order with the same depth. We identify the limit order with the closest depth and size in such instances.

Hyperparameters search. To find the best hyperparameters, we employ a grid search exploring different values as shown in Table 1. Furthermore, we set the number of diffusion steps to 100. Lastly, We set $\lambda = 0.01$ to prevent $\mathcal{L}_{\boldsymbol{\Sigma}}$ from overwhelming $\mathcal{L}_{\epsilon}$. We implement this mixed training by relying on the stop gradient functionality [9], in such a way that $\mathcal{L}_{\boldsymbol{\epsilon}}$ optimizes the error prediction and $\mathcal{L}_{\boldsymbol{\Sigma}}$ the standard deviation.

Table 1: The hyperparameter search spaces and best choice.

Hyperparameter	Search Space	Best Choice
Optimizer	{Adam, Lion}	Adam
Sequence size	{64, 128, 256, 512}	256
Learning Rate	$\{10^{-3}, 10^{-4}\}$	10^{-3}
TRADES Layers	$\{4, 6, 8, 16\}$	8
Dropout	$\{0, 0.1\}$	0.1
Attention Heads	$\{1, 2, 4\}$	2
Augmentation Dim.	$\{32, 64, 128, 256\}$	64
λ	$\{0.1, 0.01, 0.001\}$	0.01

Noise scheduler. As shown in [7], too much noise at the end of the forward noising process lowers the sample's quality. Hence, we rely on a non-linear noise scheduler as described in Eq. (7).

$$a_t = \frac{f(t)}{f(0)}, \quad f(t) = \cos\left(\frac{t/T + s}{1+s} \cdot \frac{\pi}{2}\right)^2 \tag{7}$$

[1] The depth is the difference between the order price and the current best available price in the LOB.

[2] The distance is computed using the Euclidean distance.

Importance Sampling. Because some diffusion steps contribute to most of the loss, we exploit importance sampling [7] to focus on these steps as in Eq. (8).

$$\mathcal{L}_{\boldsymbol{\Sigma}} = \mathbf{E}_{t\sim p_t}\left[\frac{\mathcal{L}_t}{p_t}\right], \quad \text{where } p_t \propto \sqrt{\mathbf{E}[\mathcal{L}_t^2]} \text{ and } \sum_t p_t = 1. \tag{8}$$

Diffusion step and positional embedding. We embed the diffusion step t and each vector's position in the sequence using sinusoidal embedding [15]. Obviously, the diffusion step t embedding is one for the whole sequence, while the position embedding is different for each element.

CGAN implementation. We implemented CGAN from scratch given that none of the implementations in [3, 2, 1] are available. Furthermore, we performed a hyperparameters search because the majority of the hyperparameters are not specified. The generator comprises an LSTM, a fully connected, and four 1D convolution layers with a final tanh activation function. Each convolution layer is interleaved with batch normalization and ReLUs. The kernel size of each convolution is 4, and the stride is 2. The optimizer, the learning rate, and the sequence size are the same as TRADES.

An important detail is how the discrete features are post-processed after the tanh function during the generation. We set the binomial feature (direction, quantity type) to -1 if the value is less than 0 and vice versa. While, for the order type, we suppose[3] that the authors search for a threshold that resembles the real order type distribution. Our final strategy for Tesla is: if the value is lower than 0.1, we assign -1 (limit order); if the value is between 0.1 and 0.25, we set it to 0 (cancel order) and with 1 (market order) otherwise. For Intel, we do smaller changes: if the value is lower than 0.15, we assign -1 (limit order); if the value is between 0.15 and 0.95, we set it to 0 (cancel order) and with 1 (market order) otherwise. The distribution of the generated data is similar to the real one when exploiting this heuristic. We did our best to implement CGAN most competitively. The full implementation is available in our framework.

4 Predictive Score Calculation

We rely on the predictive score [16] to measure how much the synthetic data is effective for the stock mid-price forecasting task. It is computed by training a predictive model Φ on synthetic data and measures the MAE on a real test set. The task considered is forecasting the mid-price with a horizon of 10, given in input the last 100 market observations. A market observation contains the last order and the first level of the LOB. We choose a 2-layered LSTM [5], standard architecture for time series forecasting [6], for Φ. We train a Φ on the generated market simulation for each generative method and each simulated day (29/1 and 30/01 for both stocks) for 100 epochs at maximum. We used early stopping with 5 patience. Next, we evaluate each Φ on the real test set extracted from the market replay.[4] In addition, a comparative Φ model was trained and tested exclusively on real market data to benchmark performance differences.

5 Additional Results

For completeness, we show the volume (see Fig. 1) and stylized facts (see Fig. 2) on 30/01. Notice how, as shown in Fig. **??**, the volume for 30/01 on Tesla follows realistic trends. Meanwhile, SoTA approaches

[3] The original paper lacks necessary details.

[4] The market replay is the simulation performed with the real orders of that trading day.

cannot seem to cope with the GT. Similar reasoning applies to the stylized facts discussed in the main paper (see Fig. **??**). Notice how all of them are satisfied, except returns and volatility negative correlation. Interestingly, the correlation between the volume and volatility (2) is slightly negative for the GT. Nevertheless, because TRADES observes a positive correlation during training, this is also reflected in the generated orders.

6 Responsiveness Experiment Settings

A δ-POV strategy is characterized by a percentage level $\delta \in (0, 1]$, a wake-up frequency Δt, a direction (buy or sell), and a target quantity of shares ϕ. The agent wakes up every Δt time unit and places buy or sell orders for several shares equal to δV_t. This process continues until either ϕ shares have been transacted or the temporal window ends. In our experiments, we use $\delta = 0.1$ and a wake-up frequency $\Delta t = 1$ min. The temporal window is set from 09:45 to 10:30, and we set $\phi = 10^5$.

References

[1] A. Coletta, M. Prata, M. Conti, E. Mercanti, N. Bartolini, A. Moulin, S. Vyetrenko, and T. Balch. Towards realistic market simulations: a generative adversarial networks approach. In *Proceedings of the Second ACM International Conference on AI in Finance*, pages 1–9, 2021.

[2] A. Coletta, A. Moulin, S. Vyetrenko, and T. Balch. Learning to simulate realistic limit order book markets from data as a world agent. In *Proceedings of the Third ACM International Conference on AI in Finance*, pages 428–436, 2022.

[3] A. Coletta, J. Jerome, R. Savani, and S. Vyetrenko. Conditional generators for limit order book environments: Explainability, challenges, and robustness. In *Proceedings of the Fourth ACM International Conference on AI in Finance*, pages 27–35, 2023.

[4] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[5] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[6] B. Lindemann, T. Müller, H. Vietz, N. Jazdi, and M. Weyrich. A survey on long short-term memory networks for time series prediction. *Procedia Cirp*, 99:650–655, 2021.

[7] A. Q. Nichol and P. Dhariwal. Improved denoising diffusion probabilistic models. In *International conference on machine learning*, pages 8162–8171. PMLR, 2021.

[8] A. Ntakaris, M. Magris, J. Kanniainen, M. Gabbouj, and A. Iosifidis. Benchmark dataset for mid-price forecasting of limit order book data with machine learning methods. http://urn.fi/urn:nbn:fi:csc-kata20170601153214969115. N/A.

[9] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[10] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pages 2256–2265. PMLR, 2015.

[11] Y. Song and S. Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32, 2019.

[12] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020.

[13] A. Tsantekidis, N. Passalis, A. Tefas, J. Kanniainen, M. Gabbouj, and A. Iosifidis. Forecasting stock prices from the limit order book using convolutional neural networks. In *2017 IEEE 19th conference on business informatics (CBI)*, volume 1, pages 7–12. IEEE, 2017.

[14] A. Tsantekidis, N. Passalis, A. Tefas, J. Kanniainen, M. Gabbouj, and A. Iosifidis. Using deep learning to detect price change indications in financial markets. In *2017 25th European Signal Processing Conference (EUSIPCO)*, pages 2511–2515. IEEE, 2017.

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[16] J. Yoon, D. Jarrett, and M. Van der Schaar. Time-series generative adversarial networks. *Advances in neural information processing systems*, 32, 2019.

[17] Z. Zhang, S. Zohren, and S. Roberts. Deeplob: Deep convolutional neural networks for limit order books. *IEEE Transactions on Signal Processing*, 67(11):3001–3012, 2019.

[18] Z. Zhang, S. Zohren, and S. Roberts. Deep reinforcement learning for trading. *The Journal of Financial Data Science*, 2(2):25–40, 2020.

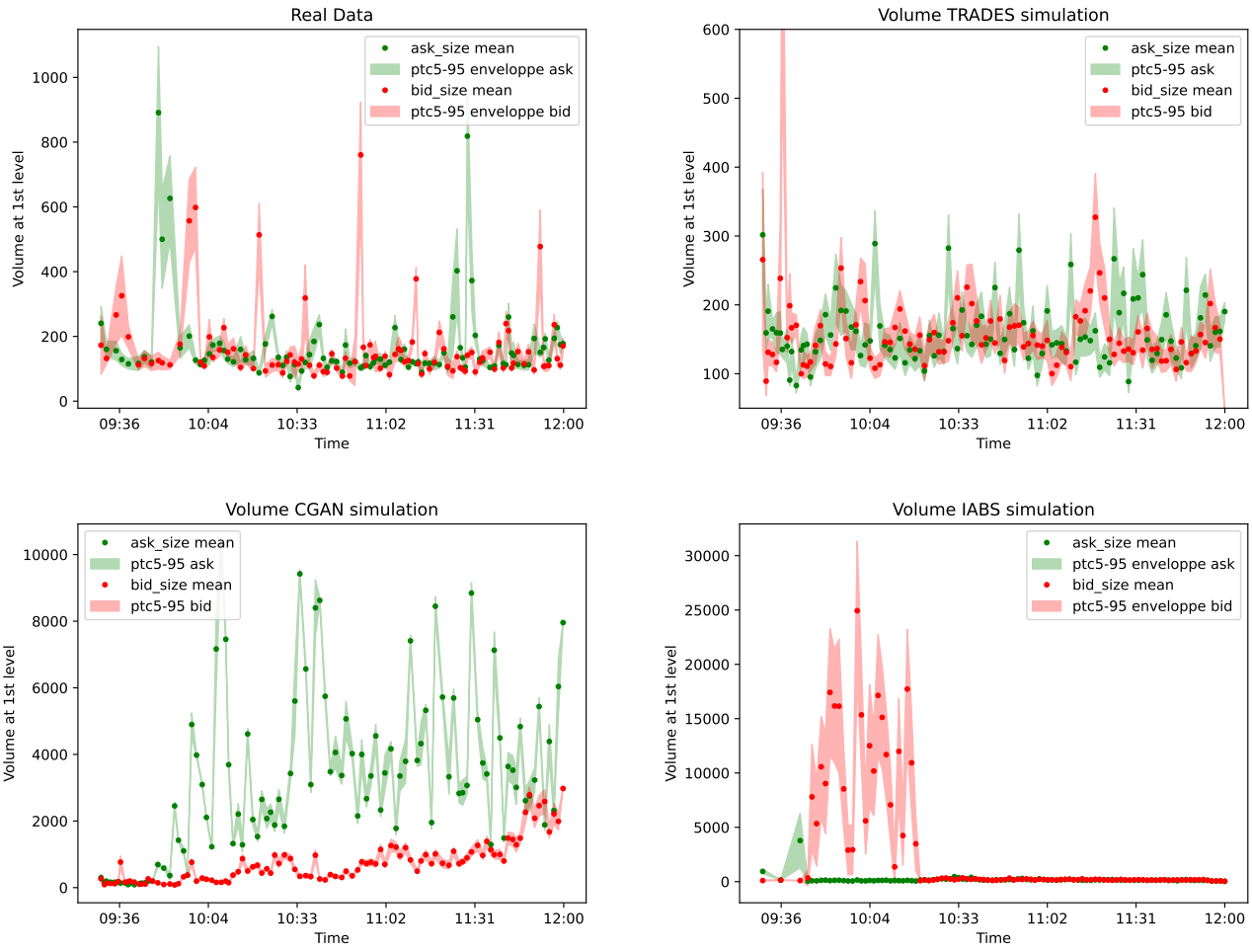

Figure 1: Volume at 1st LOB level extracted from the simulation of TSLA on 30/01

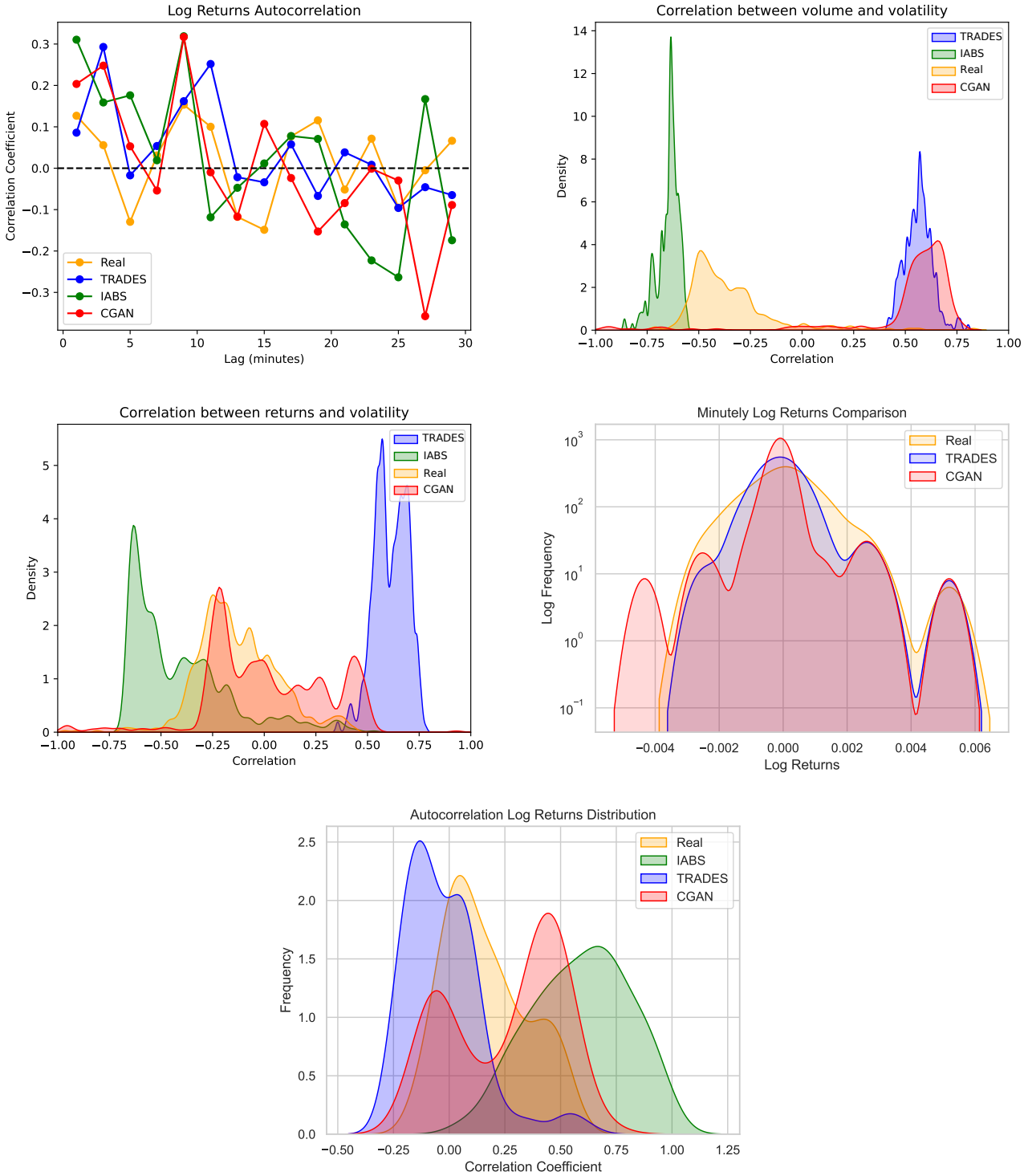

Figure 2: Stylized facts of Tesla on 30/01. (1) Log returns autocorrelation. (2) The correlation between volume and volatility, and (3) between returns and volatility. (4-5) Comparison of the minute Log Returns distribution and autocorrelation.