PAMI: Data Augmentation for Financial Reinforcement Learning

Étienne Künzel: 7477641; N. J.: xxxxxxx
Goethe Universität Frankfurt

January 10, 2025

Contents

1 Introduction

Investing is the process of allocating money or resources into assets or projects to generate a return or profit over time. One of the main ways to invest today is to buy stocks, real estate, cryptocurrencies, and other types of assets. While the overall market has been going up over the last years making long-term investing a good choice, outperforming this market over many years with short-term trading while still having low volatility is an achievement that only a few professionals achieve[.]
The hard to learn patterns that emerge in stock and cryptocurrency trading make it a good problem to utilize and test machine learning approaches. This field is called algorithmic trading and consists of an agent trying to buy an asset before it goes up and sell it before it goes down to maximize returns. In this context, reinforcement learning (RL) has gained traction as a powerful tool for developing adaptive trading algorithms capable of learning and evolving from market interactions. By continuously refining strategies through iterative feedback, RL methods have the potential to outperform traditional models in dynamic and unpredictable market conditions.
With RL profiting from curricula in other problems(a method that structures the training process to gradually introduce more complex scenarios) we will research the impact of different curricula in this domain and if they improve certain aspects and capabilities of our base model.
A crucial aspect of training effective RL models is the availability of high-quality data. The creation of diverse and representative training datasets is essential for robust model performance. We explore the use of diffusion techniques to generate synthetic training data, thereby expanding the dataset and ideally providing RL models with a more comprehensive understanding of market behaviors.
This project in the PAMI course aims to investigate the synergy between reinforcement learning, curriculum learning, and data augmentation through diffusion, focusing on their collective impact on enhancing the training process of reinforcement learning agents tested on historical Bitcoin data. By integrating these advanced techniques, we seek to contribute to the development of more effective and adaptive trading systems capable of navigating the complexities of the digital asset landscape.

2 Trademaster

2.1 A Holistic Quantitative Trading Platform

TradeMaster NTU[Sun+24] promises a framework developed to enhance financial trading strategies using reinforcement learning (RLFT). It provides an open-source platform designed to standardize the evaluation of AI-driven trading agents. The framework includes environments, premade algorithms, data from different stocks and cryptocurrencies, and evaluation metrics to help researchers develop and compare trading algorithms more effectively(Figure 1). TradeMaster NTU claims to be beneficial for studying deep reinforcement learning applications in finance, enabling more consistent testing and comparison across different models, and fostering innovation in AI-driven financial markets.

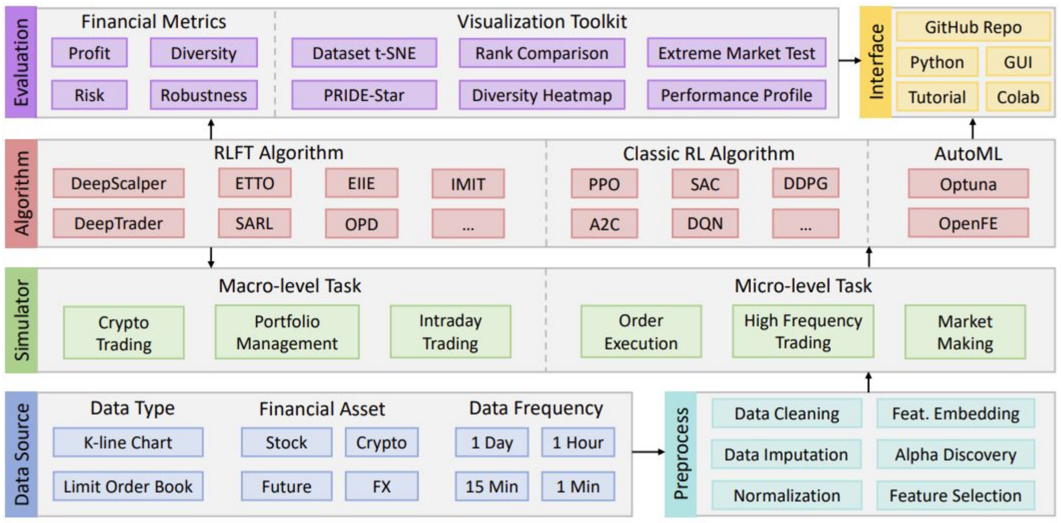


Figure 1: Overview of the TradeMaster platform[Sun+24].

2.2 Quantitative Trading as a Markov Decision Process

Quantitative trading is modeled as a Markov Decision Process (MDP) for reinforcement learning (RL), where an agent (investor) interacts with the financial market (environment) to make investment decisions (actions) and earn profits (rewards).
The MDP is defined by states, actions, transition probabilities, a reward function, a discount factor, and a time horizon. The agent aims to learn an optimal policy that maximizes expected rewards. States represent market data, which consists of the high, low, open, and close prices of the day, along with 12 additional pieces of information. Actions include buying, selling, or holding the current assets.

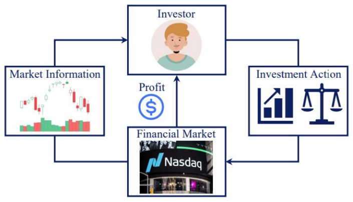


Figure 2: MDP formulation in RLFT[Sun+24]

2.3 The Problems

While Trademaster promises a range of capabilities, it suffers from several significant and minor issues. First, the installation process for Trademaster is problematic, with missing dependencies that render the RL environment non-functional when using the provided requirements.txt to create a Conda environment. Some package versions were incorrect, others were missing, and certain packages needed to be installed via mim rather than pip.
Second, after successfully setting up Trademaster on our local machine, we ran multiple tutorials provided in both GitHub and a Jupyter Notebook. Of the six pre-made tutorials, only three executed correctly, while the remaining three failed, generating various errors such as NotImplementedError and KeyError. Leaving us with just the DQN implemented for algorithmic
Third, there is a lack of comprehensive guidelines for using the platform. While the tutorials exist, they do not explain how to combine different algorithms effectively, leaving users to rely on trial and error due to the environment's obscure and unintuitive nature. Furthermore, some known issues have been left unresolved, despite being reported as early as last year.
Lastly, the code itself is inconsistent in both documentation and structure. While comments are mostly in English, some are in Chinese, and certain environments are implemented multiple times under different names with no apparent differences. Additionally, the code contains typos and spelling errors.

2.4 The Dataset

We decided to use the Bitcoin (BTC) dataset for this project for several key reasons that align with the goals of our study on improving trading performance using reinforcement learning.
BTC is the most widely traded and liquid cryptocurrency, offering us a lot of historical data from 2013 to 2021 (Figure 3), leaving us with 2990 days of information to train our RL agent. The volatility and frequency of price changes in the BTC market are quite varied and change massively from year to year, making it an interesting dataset to train and test on.

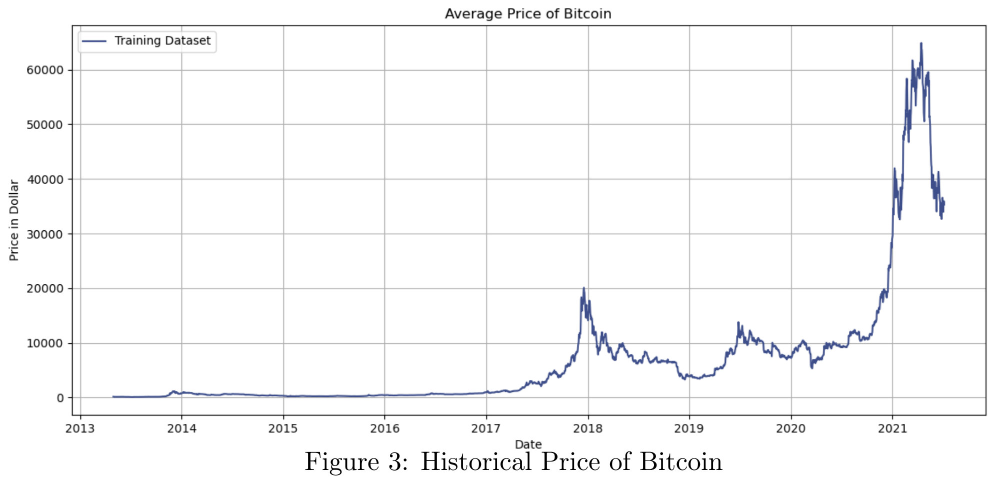


Figure 3: Historical Price of Bitcoin

3 Curriculum Learning

3.1 Introduction to Curriculum Learning

Both humans and animals demonstrate enhanced learning when examples are organized in a structured sequence, gradually introducing new concepts and increasing complexity over time, rather than confronting learners with randomly ordered or overly complex material from the beginning. In machine learning, this structured approach to sequencing training data is referred to as Curriculum Learning (CL) [Ben+09]. By mimicking this natural learning progression, CL enables models to build upon simpler tasks before advancing to more difficult ones, improving their ability to generalize and perform well on complex tasks.

There are numerous methods to implement curriculum learning (CL), which have been successfully applied across a variety of fields, including medical imaging [LSC17], speech processing [Amo+16], and navigation tasks [Flo+17], among many others [Sov+22]. These applications have demonstrated improvements in sample efficiency and better convergence of machine learning (ML) algorithms [Ben+09]. By considering the training process with CL as a continuation method, where the goal is to gradually find optimal parameters to achieve desired outputs from specific inputs, the earlier stages of training can be seen as dealing with smoother objectives. In these simpler scenarios, it is easier to identify the best behavior, and by progressively tracking this optimum through increasingly complex tasks, the model achieves better performance on evaluation data.

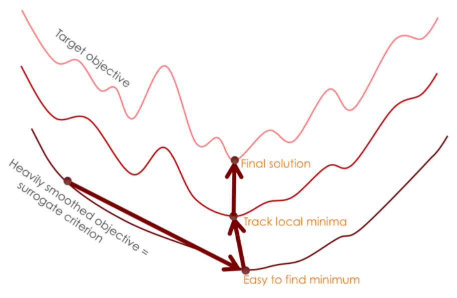


Figure 4: Schematic Display of Tracking the global optimum via CL.[WCZ21]

Figure 4 illustrates this kind of optimization, showing how a curriculum can guide the model towards the global optimum of the target objective. A similar effect is observed when dealing with noisy datasets, where the algorithm's learning capabilities can be enhanced by initially training on cleaner, less noisy parts of the dataset before gradually incorporating noisier data [WCZ21].

This perspective also highlights the potential problems of CL, as different stages in the curriculum could mislead the model's behavior, potentially trapping it in suboptimal regions of the parameter space. Thus, careful curriculum design is essential to avoid such scenarios.

Given CL's proven ability to enhance the performance of various ML models, it makes a natural complement to reinforcement learning (RL). In RL, agents often become stuck in suboptimal behaviors, a challenge that could be mitigated by leveraging CL. By introducing a curriculum that features environments with more limited state or action spaces compared to the target task, the agent can be guided towards the global optimum, as it becomes easier for the algorithm to distinguish between valuable and non-valuable inputs [Nar+20]. This structured progression provides a more effective learning pathway, reducing the likelihood of the agent becoming stuck in local optima and improving overall performance.

3.2 Creating a Curriculum for Trading

Designing an effective curriculum for our RL Agent is a complex challenge that requires a deep understanding of the domain and the behavior of the environment. In the context of trading, the environment itself does not change its future states dependent on the current actions, as is often the case in RL.

A key obstacle is determining what exactly makes an example more or less difficult in trading, especially since both of us are still relatively new to the field of trading and lack a deep understanding of its intricacy. Our approach is rather naive and based on the assumption that we can deconstruct the Bitcoin price(and its underlying data) into a base (long-term trend) and a noise(volatility) on top of the base, displayed in Figure 5 .

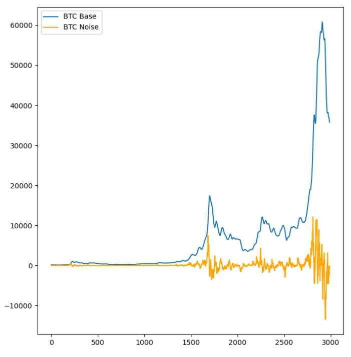


Figure 5: Deconstruction of the BTC price into its base and its noise

We, therefore, simplify the training data for the algorithm by reduc-

ing the noise to a minimum, leaving us with the base. This reduction is aimed at helping the model identify long-term patterns more easily at the start, laying a foundation before confronting it with the complexities of real market behavior.
As training progresses, this artificial smoothing is gradually lessened, allowing the prices to fluctuate more freely. This staged transition helps to ease the agent into increasingly complex and volatile environments that better resemble the actual, unpredictable dynamics of BTC prices.
We implement this noise reduction through two distinct strategies, and another counterproductive curriculum showing the possible adverse effects of poorly implemented curricula, resulting in three curricula, each derived from the original BTC dataset. These variations serve to guide our agent through a structured progression.
The first(Figure 6) and second (Figure 7)strategy consists of applying a moving/exponential average with an initial window size of 128, which is halved for each subsequent ninth of the dataset until we reach the raw, unaltered data.
The third strategie (Figure 8) inadequately simplifies the data by using the most frequent value with a decreasing window, possibly leading to worse performance than with standard training.

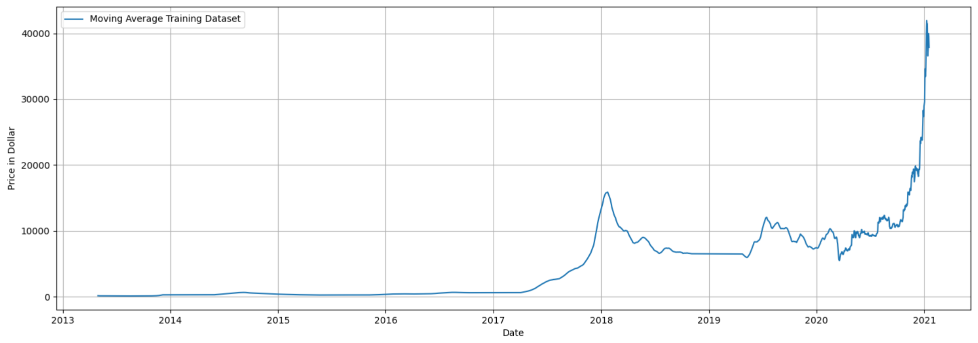


Figure 6: BTC Curriculum Dataset with gaining complexity via a decreasing Moving Average Window

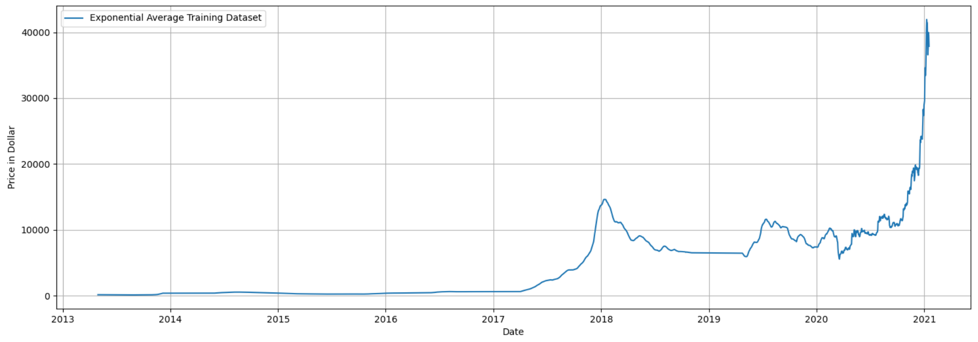


Figure 7: BTC Curriculum Dataset with gaining complexity via a decreasing Exponential Average Window

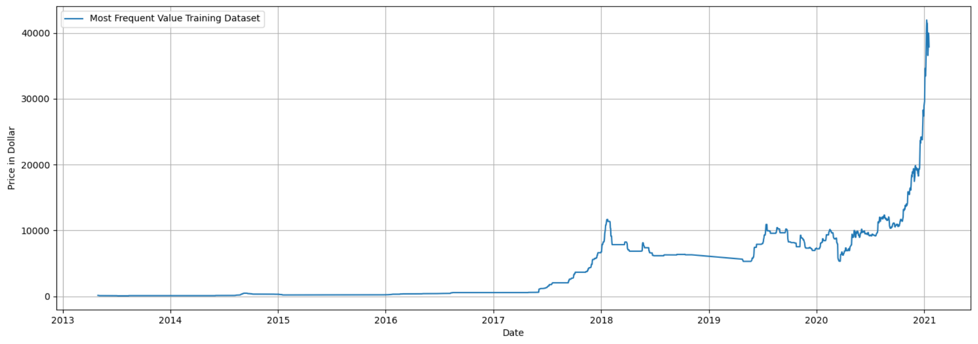


Figure 8: An adverse BTC Curriculum Dataset created with the mode operator(Most Frequent Value)

4 Diffusion Model

4.1 Introduction into Diffusion Models

In recent years, diffusion models [HJA20], [Soh+15] have gained more and more popularity when it came to image generation and image synthesis. This growing popularity was due to the fact that diffusion models include stable training, easy model scaling and good distribution coverage. Diffusion models belong to the family of generative models and can generate high-fidelity images that are akin to those generated by generative adversarial networks (GANs) [Goo+14]. To learn how to generate an image, diffusion models typically add Gaussian noise to it, gradually corrupting it. They then reverse this process and remove the noise, reconstructing the original image as seen in Figure 9. In essence, this means that the model learns to recover clean images from corrupted images. This entire process, consisting of the forward diffusion step (noise addition) and the reverse diffusion step (noise removal), help create high-resolution images of different quantities. In this paper we are going to use diffusion

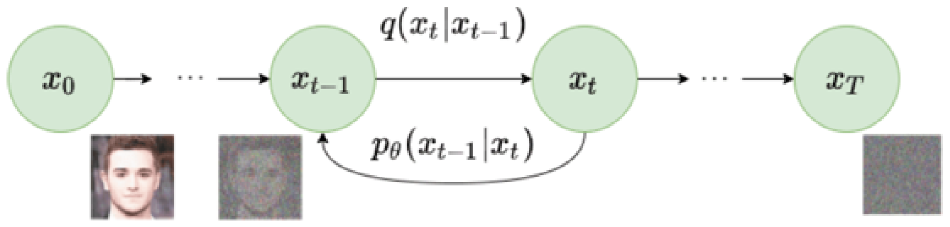


Figure 9: The Diffusion Process

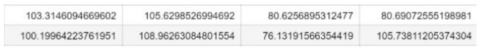
models to generate synthetic bitcoin market scenarios instead of images.

4.2 Roles of Diffusion Models in Reinforcement Learning

When it comes to diffusion models in reinforcement learning, there are three main roles the diffusion model can take on as described in [Zhu+24]. The first role is the planner role. In reinforcement learning, planning is the process of making decisions based on a dynamic model and choosing the best course of action to maximize cumulative rewards. By investigating different action and state patterns, this technique typically improves decisions over an extended period of time, but because the planning sequences are typically simulated autoregressively, there is a risk of serious compounding mistakes, particularly when the system is offline and there is insufficient data support. This is where diffusion models come in to solve this problem by generating multi-step planning sequences simultaneously. An example of this from [Lia+23], where it uses diffusion-based generative modeling to plan multiple tasks at once. This model adapts to novel surroundings more effectively than conventional autoregressive planners by utilizing diffusion models to tackle unseen tasks.
The second role is the policy role. In model-free RL, diffusion models act as policy measures. Diffusion models offer better richness and versatility than Gaussian policies since they may reflect any normalizable distribution, unlike Gaussian policies which have trouble with complex multi-modal datasets because of their unimodal character. This aids in resolving the issues with policy regularization and over-conservatism in offline reinforcement learning. An example was introduced in [Lu+23], where it presents contrastive energy prediction, which generates more accurate and varied policy actions than traditional methods by using diffusion models to drive action sampling in offline reinforcement learning challenges.
The third and final role, and that is the role we decided to use in our experiments, is the role of data synthesizer. By producing synthetic data that is consistent with the fundamental structure of the environment, diffusion models can enhance real-life datasets. This makes it possible for diffusion models to provide improved augmentation over conventional perturbation techniques in scenarios with sparse data or high-dimensional state spaces. In our case, where the bitcoin data was limited, it helped us overcome any overfitting problems that might occur.

4.3 Methodology: Diffusion Model Implementation and Design

As was stated earlier, the goal of this paper is to generate synthetic market scenarios using diffusion models as a data synthesizer for RL. The first approach we took was to implement the diffusion model using the well known U-Net architecture and try and adjust some factors to be able to generate time-series data instead of images by converting the 2D convolution layer into a 1D layer, since we only care about the temporal sequence and not the width and height of an image. The U-Net architecture consisted of two main sections, the encoder and decoder. The encoder was made up of four convolutional blocks, with each having a 1D convolutional layer (kernel size: 3 and padding: 1). In between each convolutional layer, there was a ReLU activation and dropout for regularization with a probability of 0.2 and finally in between each block there was a max pooling operation, which increased the number of feature channels from 32 to 512. The decoder had transposed convolutional layers to upsample the feature maps back to their original resolution. It used a sigmoid activation function to produce the output in the final layer. At first glance, looking at Figure 10, the results seem promising. However it was only after plotting the results in comparison to the original dataset, we were able to see that the generated data was not able to capture the pattern of the original dataset, it was not time dependent and was very volatile. Looking at Figure 11, we can see how the generated data is all squashed up in the first third of the plot, indicating no time dependency and no captured pattern from the original dataset.



103.3146094669602	105.6298526994692	80.6256895312477	80.69072555198981
100.19964223761951	108.96263084801554	76.13191566354419	105.73811205374304

Figure 10: Results Based on the U-Net Model Described

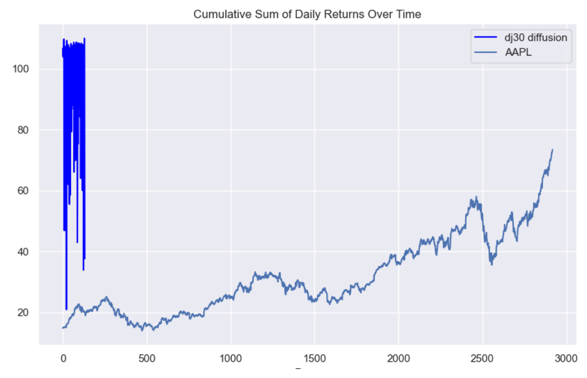
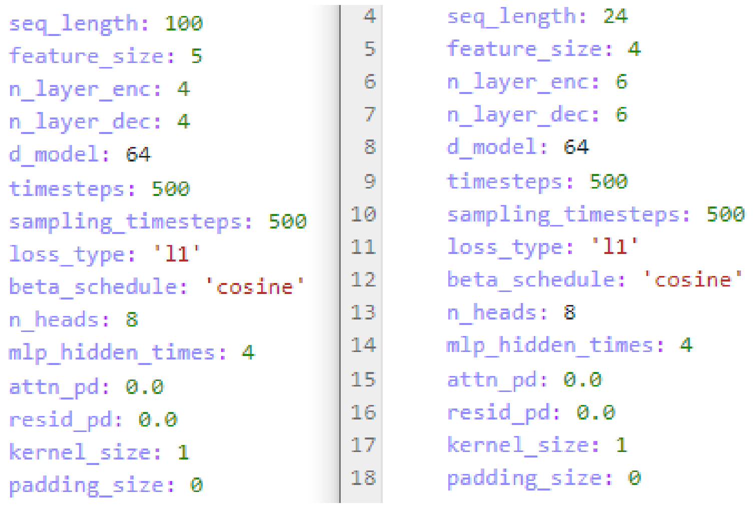


Figure 11: Comparison of Generated DJ30 Data and the Original

We then decided to shift our focus only on the bitcoin dataset instead of trying it out on the DJ30 dataset and to use the approach from [YQ24]. They use a denoising diffusion probabilistic model (DDPM), to propose Diffusion-TS, as they described it, "a novel diffusion-based framework that generates multivariate time series samples of high quality by using an encoder-decoder transformer with disentangled temporal representations, in which the decomposition technique guides Diffusion-TS to capture the semantic meaning of time series while transformers mine detailed sequential information from the noisy model input."

In this paper, we used two of the three sampling methods they provide. The first is the unconstrained method, where the model tries to generate synthetic data based on what it just learned from patterns from the original BTC dataset. The second method is forecasting, the model trains on part of the dataset and then tries to predict the next time steps, depending on the prediction length we provide. Looking at Figure 12, we see that the main difference when it came to the two different sampling tasks, was the amount of encoders and decoders used. After some trial-and-error, this yielded the best generated output for the different tasks. As for the training that lead to the unconstrained sampling, a sequence length of 24 and 50 were used, and those were the optimal length to look at it to capture the trends. Observing the first method, the unconstrained sampling, yielded the following results as seen on Figure 13. For a deeper dive into the model used, please refer to [Zhu+24]. We see that the training with sequence length of 50 had volatile results at the start but calmed down towards the end mimicking the pattern of the original dataset better, whereas with sequence length of 24, mimicked the original pattern from the start.



```
seq_length: 100
feature_size: 5
n_layer_enc: 4
n_layer_dec: 4
d_model: 64
timesteps: 500
sampling_timesteps: 500
loss_type: 'l1'
beta_schedule: 'cosine'
n_heads: 8
mlp_hidden_times: 4
attn_pd: 0.0
resid_pd: 0.0
kernel_size: 1
padding_size: 0
```

```
seq_length: 24
feature_size: 4
n_layer_enc: 6
n_layer_dec: 6
d_model: 64
timesteps: 500
sampling_timesteps: 500
loss_type: 'l1'
beta_schedule: 'cosine'
n_heads: 8
mlp_hidden_times: 4
attn_pd: 0.0
resid_pd: 0.0
kernel_size: 1
padding_size: 0
```

Figure 12: Optimal Training Parameters i) Left are for Forecasting Task ii) Right for Unconstrained Sampling Task

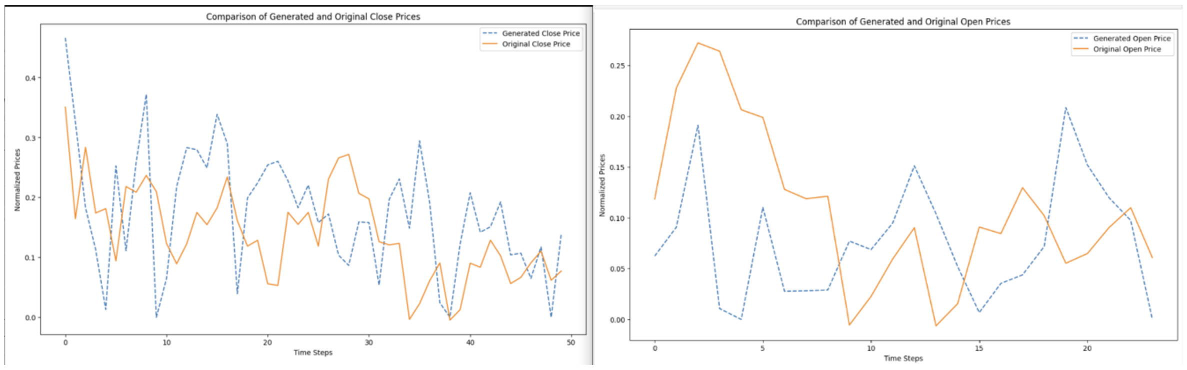


Figure 13: Generated Data on Different Sequence Lengths

Now that we tested the model to see how it performs and to find the optimal parameters, we moved on to the next task, which was to have a dataset with missing values and try to predict them. We took the original BTC dataset and removed 170 time-steps starting from the following dates 2013-12-05, 2016-09-03, 2018-11-03, 2021-01-16. Using the forecasting sampling method provided by [Zhu+24], we would train based on all the results before that specified date, and then let the model forecast the results for the next 170 time-steps. We would repeat that process for all the missing values. Figure 14 shows that the model generally predicted similar patterns to the original, with the exception of, the 2021 range mainly and the 2018 rang on a smaller scope. The reason for that is that the model did not train on a massive jump in values as seen in 2021 for example, it only saw the start of the rise and could therefore not predict that pattern properly causing in a volatile prediction.

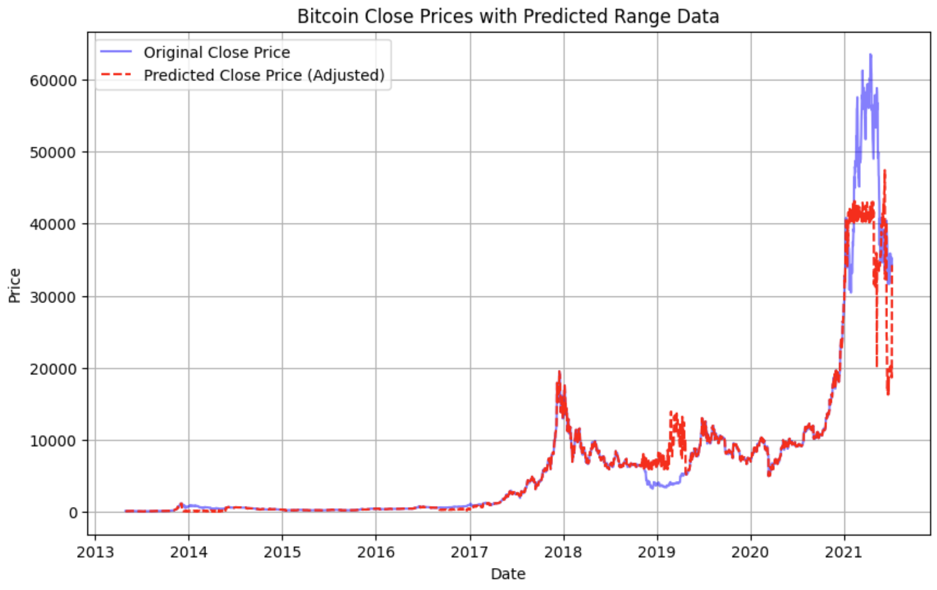


Figure 14: Predicted Data for the Missing Date Ranges

The third and final task we wanted to achieve, is to generate a BTC dataset, that is completely generated by the diffusion model. The initial idea was to train the model on the entire dataset and then generate the entire synthetic dataset using the unconstrained sampling method. However, that lead to some problems which we will discuss in the the following section. To counter this problem, we decided to use the forecasting method for sampling, by training the model on the previous year's data and then predict a period length of 365 representing the entire year. We did this process until we were able to generate a completely synthetic BTC dataset generated using the diffusion model, with the exception of 2013, since there were no previous data to train the year on. Figure 15 shows

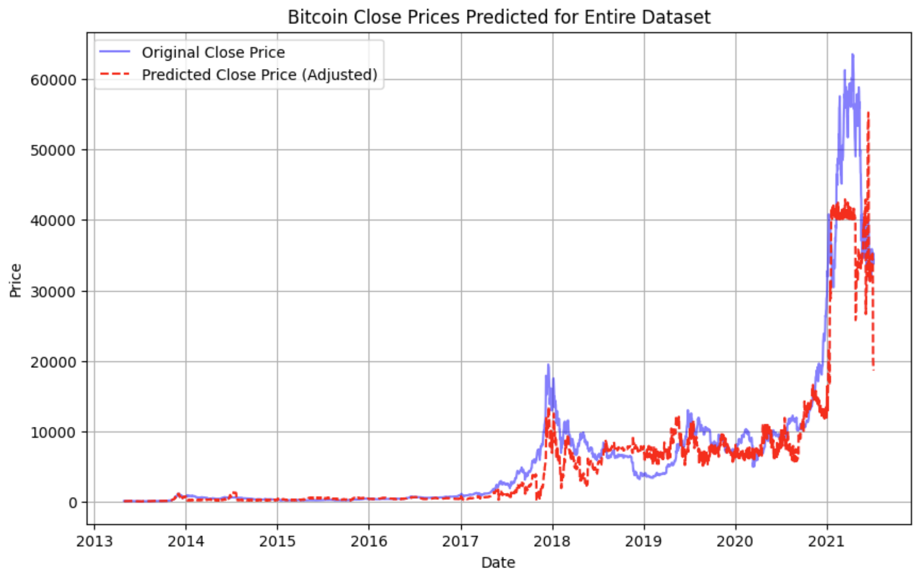


Figure 15: Predicted Data for all Years

the generated dataset using diffusion model and the forecasting method for sampling. The generated data showed promising similarities compared to the original, generally following the same pattern, but again with the exception of 2021.

4.4 The Problems

During the implementation of diffusion models, we faced a few problems. The first and main problem was when trying to train the model on the entire dataset and then generate a fully synthetic dataset, the model will barely learn any patterns and just end up with a very volatile random data points, that do not follow the overall trend of the original dataset. We tried changing multiple parameters, trying to train the model on a longer sequence length, hoping that it learns the pattern if exposed to a bigger window, but then we bumped into a hardware problem, where the GPU we are using could not handle a very long sequence length, so we decided to go with the solution mentioned in the previous section. Another problem is that the model did not recognize the dependency between the different columns, for example some data points would have the close price, outside the high and low price range. Unfortunately, we could not figure out what the cause for that was.

5 Experiments

5.1 Evaluation/Metrics

To evaluate our trained agent, we will use different parts of the BTC dataset that are not part of the training dataset to prevent the possibility of our agent overfitting to the test data. The test data we have chosen are from 4 distinct time frames(Figure 16), balancing each other out over the time frame to nearly be a neutral investment. We did this to evaluate our agent on a more objective basis. Otherwise, when the overall trend of the test dataset is going up or down, the performance of our agent tends to be alike.

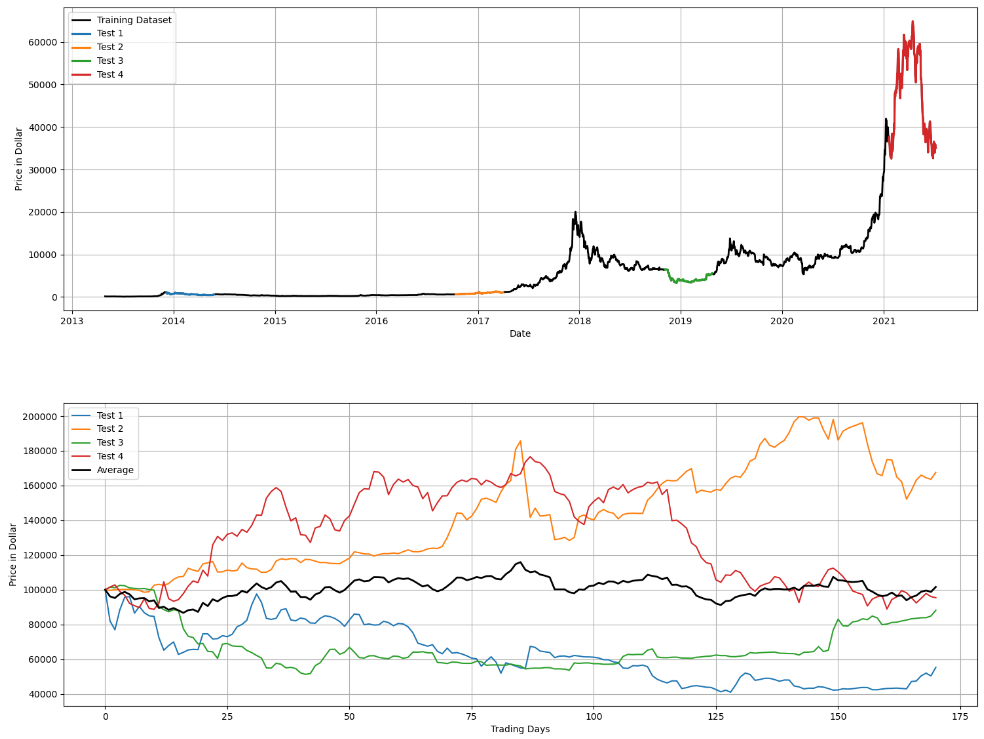


Figure 16: Test Timeframes taken from the BTC Dataset

As metrics for our evaluation, we use the Max Drawup: the highest point of the investment strategie between all four tests; Max Drawdown: the lowest point of the investment strategie between all four tests; Highest Volatility: the highest volatility between all of the tests; Overall Return: the average end return between all test.
We optimized the performance of the DQN agent by conducting a hyperparameter sweep using optuna[LINK], a widely used hyperparameter optimization framework. Optuna allowed us to efficiently explore the hyperparameter space and identify the best settings for maximizing the performance of our DQN agent. We have decided to sweep the following parameters across the specified ranges:

- **Network Dimensions:** (64, 64), (64, 32), (64, 64, 64)
- **Learning Rate :** Range: 1×10^{-5} to 1×10^{-1}
- **Discount Factor:** Range: 0.8 to 0.99
- **Soft Update Coefficient:** Range: 0.001 to 0.1
- **Horizon Length :** 64, 128, 256, 512
- **Batch Size:** 128, 256, 512, 1024

The best hyperparameters for the DQN agent are a network architecture with two hidden layers with 64 and 32 neurons respectively, a learning rate of 0.0306, a discount factor of 0.8804, a soft update coefficient of 0.0320, a horizon length of 128, and a batch size of 128.

5.2 Results

The chart and table display the average performance in the four tests of a DQN trained with different curriculum strategies. The graph illustrates the cumulative daily returns over time, while the table below summarizes key metrics: Max Drawup, Max Drawdown, Highest Volatility, and Overall Return. From the graph in Figure 17, we observe that the "Standard + Diffusion", "Smoothed Average", and "Exponential Average" approaches outperform the others in terms of cumulative returns by the end of the period. On the other hand, the "Standard" training, as well as the "Full Diffusion" and "Most Frequent Value" training, leads to our DQN experiencing significant underperformance, resulting in an overall loss by the end of the evaluation.
Looking at Table 1, the "Standard + Diffusion" approach achieves the highest Max Drawup (81.9%) and the best overall return (17.3%), indicating its superior ability to capitalize on upward market movements while controlling for drawdowns. This approach also experiences relatively moderate volatility (4.6%). On the other hand, the "Smoothed Average" strategy has a commendable Max Drawup (73.4%) and Max Drawdown (-45.8%), along with a substantial overall return of 16.5%, suggesting it performs well in most metrics, especially in comparison to the standard training. The results of the "Exponential Average" are similar.
The "Exponential Average" (purple line) shows an overall return of 12.4%, with an improvement over the baseline, but slightly higher volatility (4.4%) than some other approaches. As expected, the "Most Frequent Value" strategy, as a negative example of a curriculum, has the lowest overall return (-7.9%) despite showing slightly lower volatility (4.2%), indicating that this approach struggles to adapt to market trends.
The "Full Diffusion" strategy, despite being fully generated, displays better overall performance metrics than the "Standard" training. This is attributable to the fact that the "Full Diffusion" dataset is 30% larger than the others due to not being restricted to real data and the ability to generate training data. This could suggest that diffusion-generated data is a good enough substitute for real data to train RL agents.
In conclusion, deploying a sensible curriculum learning strategy seems to lead to an improvement in the performance of a DQN, in contrast to training with an unsuitable curriculum(Most Frequent Value). Diffusion can fill in missing real data, creating a mixed dataset that enables our agent to perform better than just being trained on the real data. The "Full Diffusion" approach, although worse than most of the other approaches, still performs similarly to being trained on real data, possibly indicating that it is a good substitute for real data.

Curriculum	**Max Drawup**	**Max Drawdown**	**Highest Volatility**	**Overall Return**
Standard	68,6%	-48,3%	4,3%	-2,6%
Smoothed Average	73,4%	-45,8%	4,3%	16,5%
Exponential Average	76,2%	-47,2%	4,4%	12,4%
Most Frequent Value	63,2%	-49,1%	**4,2%**	-7,9%
Standard + Diffusion	**81,9%**	-47,2%	4,6%	**17,3%**
Full Diffusion	68,6%	**-41,1%**	4,3%	-0,592%

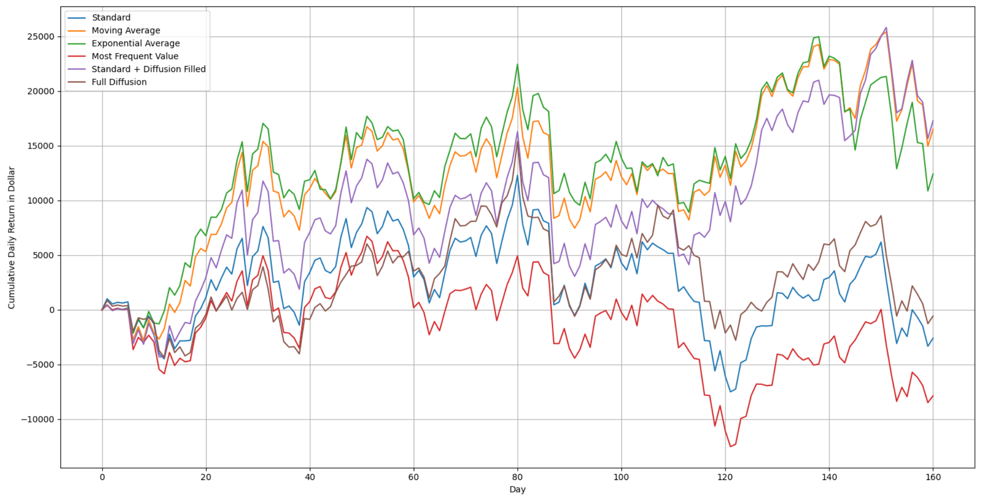


Figure 17: Comparison of six different Training Approaches, the results display the average of 4 different tests. The individual tests are displayed in the Appendix

6 Conclusion

In this work, we explored the integration of curriculum learning and diffusion models as data synthesizers to enhance the training of reinforcement learning (RL) agents in a trading environment, addressing key challenges such as sparse datasets and limited market conditions by generating synthetic market scenarios.
We demonstrated that curriculum learning can improve RL agent performance compared to training on unaltered data. Additionally, diffusion-generated data mitigated the sparse dataset problem by offering a larger, more varied dataset that enabled more robust training.
Initially, we adapted a classical U-Net architecture to generate time-series data, but this model struggled to capture the temporal patterns and trends of the original dataset effectively. Consequently, we transitioned to the diffusion-ts model, proposed by [YQ24], which uses an encoder-decoder transformer with disentangled temporal representations. This model provided significantly better results in generating realistic time-series data.
Two primary tasks were addressed: first, filling missing data in historical Bitcoin datasets, and second, generating a complete Bitcoin dataset from scratch using the diffusion model. The missing data approach performed comparably to curriculum learning methods, demonstrating that diffusion-based interpolation is a viable alternative for handling data gaps. The fully synthetic dataset generated using the diffusion model led to agent performance slightly better than standard training on real data, suggesting that the generated data may serve as a sufficient substitute in certain cases.
Despite these promising results, several limitations must be acknowledged. Our novice understanding of the financial domain could have influenced how we applied these strategies, and the limitations of the Trademaster environment may have impacted the generalizability of our findings. Moreover, our focus on Bitcoin data alone restricted the diversity of market conditions explored.
While diffusion models and curriculum learning show significant potential for improving RL agents in financial trading settings, further research is needed to validate these methods across more diverse datasets, broader market conditions, and different RL algorithms. Future work should also explore combining these two approaches to assess their synergistic effects on agent performance, alongside testing in more stable simulators to better measure their applicability in real-world trading environments.

References

[Sun+24] Shuo Sun et al. "TradeMaster: a holistic quantitative trading platform empowered by reinforcement learning". In: *Advances in Neural Information Processing Systems* 36 (2024).

[Ben+09] Yoshua Bengio et al. "Curriculum learning". In: *Proceedings of the 26th annual international conference on machine learning*. 2009, pp. 41–48.

[WCZ21] Xin Wang, Yudong Chen, and Wenwu Zhu. "A survey on curriculum learning". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.9 (2021), pp. 4555–4576.

[LSC17] William Lotter, Greg Sorensen, and David Cox. "A multi-scale CNN and curriculum learning strategy for mammogram classification". In: *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support: Third International Workshop, DLMIA 2017, and 7th International Workshop, ML-CDS 2017, Held in Conjunction with MICCAI 2017, Québec City, QC, Canada, September 14, Proceedings 3*. Springer. 2017, pp. 169–177.

[Amo+16] Dario Amodei et al. "Deep speech 2: End-to-end speech recognition in english and mandarin". In: *International conference on machine learning*. PMLR. 2016, pp. 173–182.

[Flo+17] Carlos Florensa et al. "Reverse curriculum generation for reinforcement learning". In: *Conference on robot learning*. PMLR. 2017, pp. 482–495.

[Sov+22] Petru Soviany et al. "Curriculum learning: A survey". In: *International Journal of Computer Vision* 130.6 (2022), pp. 1526–1565.

[Nar+20] Sanmit Narvekar et al. "Curriculum learning for reinforcement learning domains: A framework and survey". In: *The Journal of Machine Learning Research* 21.1 (2020), pp. 7382–7431.

[HJA20] Jonathan Ho, Ajay Jain, and Pieter Abbeel. "Denoising diffusion probabilistic models". In: *arXiv preprint arXiv:2006.11239* (2020).

[Soh+15] Jascha Sohl-Dickstein et al. "Deep unsupervised learning using nonequilibrium thermodynamics". In: *arXiv preprint arXiv:1503.03585* (2015).

[Goo+14] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems (NeurIPS)*. 2014, pp. 2672–2680.

[Zhu+24] Zhengbang Zhu et al. "Diffusion Models for Reinforcement Learning: A Survey". In: *arXiv preprint arXiv:2311.01223* (2024).

[Lia+23] Zhixuan Liang et al. "AdaptDiffuser: Diffusion models as adaptive self-evolving planners". In: *Proceedings of the 40th International Conference on Machine Learning*. PMLR. 2023, pp. 20725–20745.

[Lu+23] Cheng Lu et al. "Contrastive Energy Prediction for Exact Energy-Guided Diffusion Sampling in Offline Reinforcement Learning". In: *arXiv preprint arXiv:2303.01717* (2023).

[YQ24] Xinyu Yuan and Yan Qiao. "DIFFUSION-TS: Interpretable Diffusion for General Time Series Generation". In: *Hefei University of Technology* (2024).

List of Figures

A Appendix

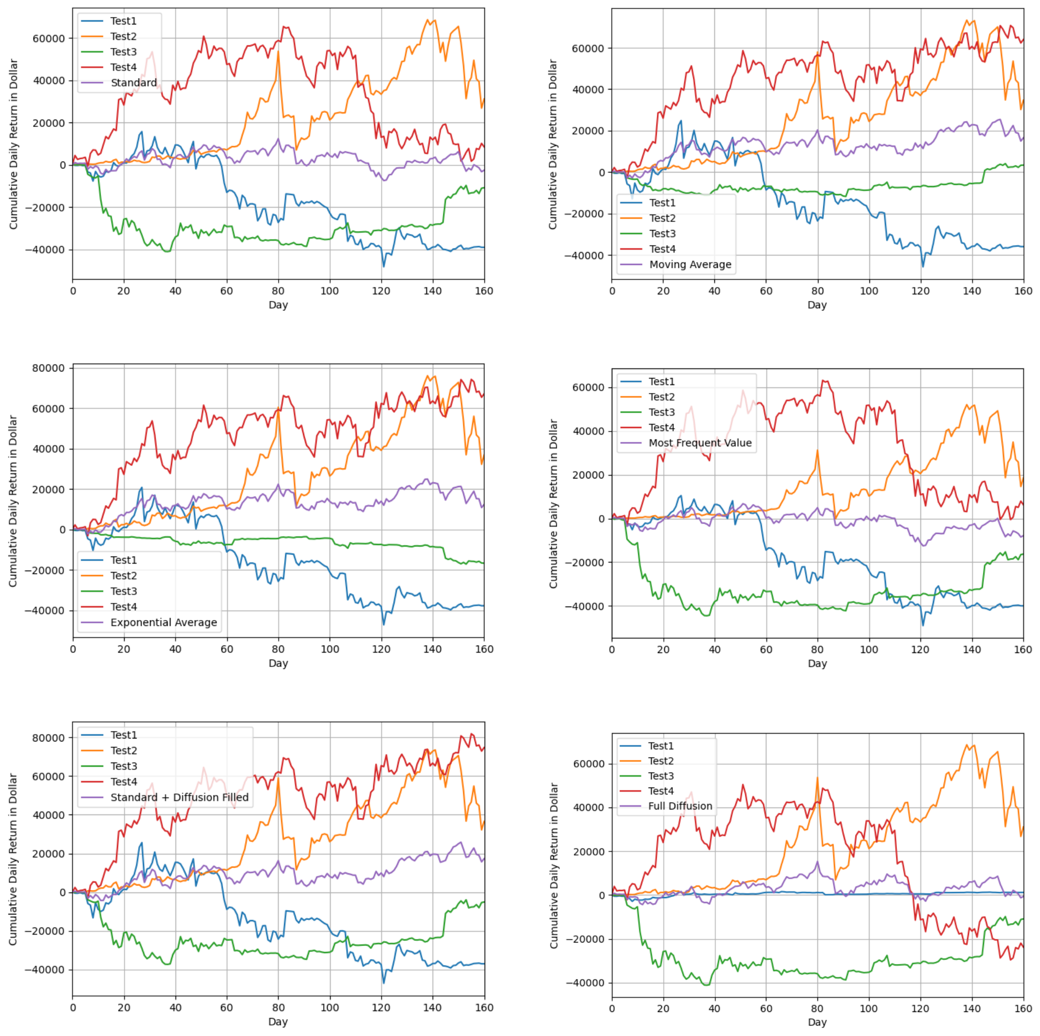


Figure 18: Different training strategies evaluated on four different tests