Imperial College London

Imperial College London

Department of Mathematics

Solving the Collatz conjecture

Author: Ilia Sobakinskikh (CID: 00000000)

A thesis submitted for the degree of

MSc in Mathematics and Finance, 2022-2023

Declaration

The work contained in this thesis is my own work unless otherwise stated.

Acknowledgements

This is where you usually thank people who have provided useful assistance, feedback,...., during your project.

Abstract

The abstract is a short summary of the thesis' contents. It should be about half a page long and be accessible by someone not familiar with the project. The goal of the abstract is also to tease the reader and make him want to read the whole thesis.

Contents

1 Methodology **6**

1.1 Problem Formulation . . . 6

1.2 Transformers . . . 6

1.2.1 Attention mechanism . . . 7

1.2.2 Transformers for time series modelling . . . 9

1.3 FPGA design . . . 9

1.3.1 Introduction to FPGA . . . 9

1.3.2 FPGA development and HLS . . . 9

1.3.3 Common optimizations . . . 9

2 Experiments **11**

2.1 Architecture and hyperparameters . . . 11

2.2 Datasets . . . 11

2.2.1 Numenta Anomaly Benchmark (NAB) . . . 11

2.2.2 KPI Anomaly Detection Dataset . . . 11

2.2.3 FI2010 . . . 12

2.3 Accuracy . . . 13

2.4 Performance/Speed . . . 13

2.5 Resource utilization on FPGA . . . 13

2.6 Future work . . . 14

A Code **15**

A.1 Efficient matrix multiplication . . . 15

Bibliography **16**

List of Figures

1.1 Model architecture of the Transformer [1] 7
1.2 Scaled Dot-Product Attention and Multi-Head Attention [1] 8

2.1 NYC Taxi demand - anomalies highlighted in red 11
2.2 Sensor data from a machine in a data center. The red dots indicate the anomalies. 12
2.3 Example of the injected outliers in the FI2010 dataset. 13

List of Tables

Introduction

In this thesis, we explore how the inference time of a Transformer Neural Network can be efficiently optimized with applications to real-time anomaly detection in financial time series. The financial time series are price series such as asset prices. Unfortunately, the data is often with errors or outliers that make the downstream data processing tasks useless, unstable or even harmful [2] [3]. Moreover, the amount of financial time-series data has been significantly increasing [4]. Hence, there is a need for better data-cleaning methods in terms of accuracy and in terms of processing speed.

Transformers as a neural network architecture have achieved superior performances in many tasks such as Natural Language Processing and Computer Vision [5]. Time series modelling and especially anomaly detection tasks can benefit from the features of transformers architecture in multiple ways, including the capacity to capture long-range dependencies and interactions [6].

Increasingly powerful hardware, such as field-programmable gate arrays (FPGAs), have seen increasing usage in recent years due to their reconfigurability and high performance [7].

We explore different Transformer architectures for time series modelling and how they can be efficiently implemented on an FPGA board (PYNQ-Z2). In particular, we examine the application of Transformers to detect anomalies in time series and we show how they can be efficiently implemented on an FPGA board to minimize latency or to maximize throughput.

Chapter 1

Methodology

In this chapter, we will describe the main concepts and ideas used in this work. The reader will be introduced to the main concepts of the Transformer architecture and how it can be used for anomaly detection in time series. Finally, the main concepts of programming an FPGA will be introduced and the specific optimizations that can be applied to speed up the computations.

1.1 Problem Formulation

Definition 1.1.1. We consider a **time-series** $\mathcal{T}$ which is simply a timestamped sequence of observations $x_i \in R^n$.

Remark 1.1.2. Most of the times we will consider univariate case, i.e. $n = 1$. An example of this is a price time-series of a single stock. However, the multivariate case is also important and we will consider it in the experiments. For example, one can consider a time-series of prices of multiple stocks to get a multivariate time series.

Definition 1.1.3. The **Anomaly Detection** task: for any time-series $\hat{\mathcal{T}}$ of length n, we need to predict $\mathcal{Y} = \{y_1, ..., y_n\}, y_i \in \{0, 1\}$, whether the datapoint at the i-th timestamp anomalous (where by convention we will use 1 as anomaly and 0 as not an anomaly).

In this work, we will restrict ourselves to the **supervised case** where the labels y_i are known for the *seen* (or training) part of the dataset.

Remark 1.1.4. One can also consider an unsupervised task. However, one issue with the unsupervised task is that it is hard to evaluate the performance (i.e., accuracy) of the model's predictions [8].

1.2 Transformers

In this section, we will describe the main concepts of the Transformer architecture. We will describe the main building blocks of the Transformer architecture and will give a special treatment to the attention mechanism firstly introduced in [9].

General architecture

In [1], authors introduced the Transformer architecture which a neural network architecture which is the architecture that is dominantly used in Natural Language Processing tasks. The architecture's main feature was reliance on the attention mechanism and the complete elimination of recurrent and convolutional layers.

Figure 1.1 presents the main architecture of the transformer. The architecture consists of an **encoder** and a **decoder**. For the purpose of the thesis we will only consider the **encoder** part of the architecture. The **encoder** is preceded by a **positional encoding** layer which is used to *inject* the positional information to the input vectors x_i because the attention mechanism is permutation invariant, this will be explained in Section 1.2.1.

The **encoder** consists of N identical layers. Each layer has two sub-layers which are a **multi-head self-attention** layer and a **feed-forward** layer. The **feed-forward** layer $\mathrm{FFN}(\cdot)$ is a simple

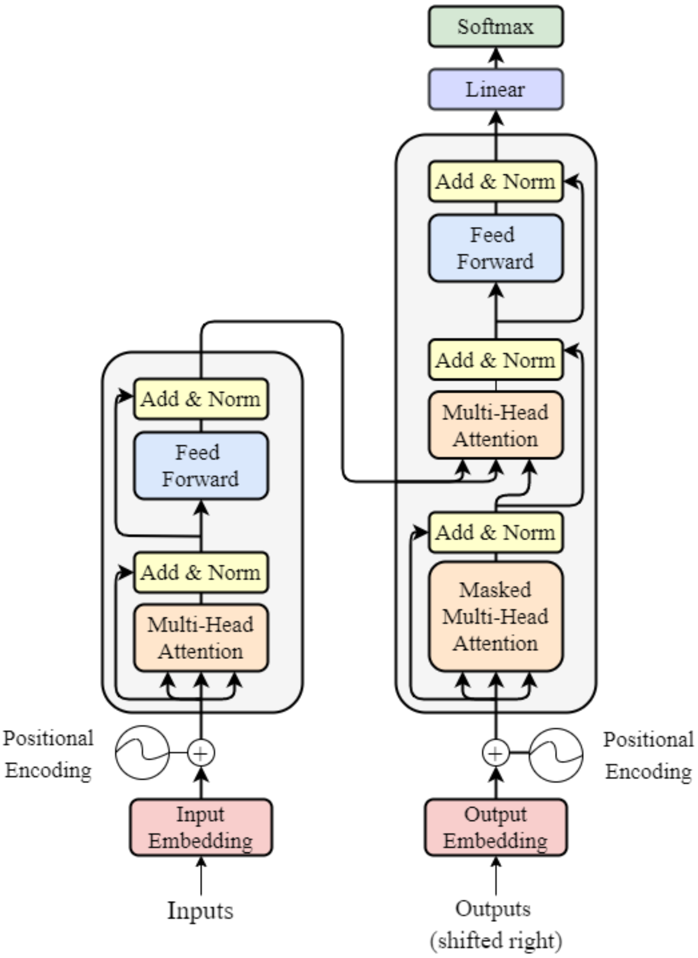

Figure 1.1: Model architecture of the Transformer [1]

fully-connected layer with a non-linear activation function applied element-wise to the result[1]. Specifically, authors of [1] used an FC layer with the ReLU activation function

$$\mathrm{ReLU}(x) = \max(0, x)$$

followed by another FC layer without activation function, i.e.

$$\mathrm{FFN}(x) = W_2 \cdot ReLU(W_1 \cdot x + b_1) + b_2$$

The **Add & Norm** layer is a residual connection [10] followed by a layer normalization layers [11]. Those layers are not essential for this work and will not be described in detail.

This constitutes the main building block of the Transformer encoder architecture. The next section will describe the attention mechanism in detail.

1.2.1 Attention mechanism

This section will describe the attention mechanism, its variations and the intuition behind it. Moreover, we will compare different attention mechanism implementations in terms of their computational complexity and their ability to capture long-range dependencies.

Dot-Product Attention and Multi-Head Attention

The attention mechanism introduced in the Transformer architecture [1] used a **scaled dot-product attention**.

[1] In general, a **fully-connected** layer $\mathrm{FC}(x)$ is simply a linear transformation inputs X (i.e., a matrix multiplication) with the activation function applied element-wise to the result. That is, $\mathrm{FC}(x) = f(W \cdot x + b)$ where W is a weight matrix and b is a bias vector and $f(\cdot)$ is the activation function. Authors of [1] used the ReLU activation function

The main idea of the **dot-product attention** mechanism is to compute the mapping of a query q_i for each input vector x_i to a set of key-value pairs (k_j, v_j). The query q_i, key k_i and value v_i vectors are simply linear transformations of the input vectors x_i, i.e., $q_i = W_Q \cdot x_i$, $k_i = W_K \cdot x_i$, $v_i = W_V \cdot x_i$ where W_Q, W_K and W_V are the weight matrices. The attention mechanism is a weighted sum of the values v_j where the weights are computed as a function of the query q_i and the key k_j. That is, $Attention(x_i) = \sum_j \alpha_{ij} v_j$ where $\alpha_{ij} = \text{softmax}(q_i \cdot v_i)$ is the weight of the j-th value v_j. In practice, the attention mechanism is computed for all the queries q_i at the same time by utilizing the following expression in matrix form:

$$\begin{aligned} Q &= W_Q \cdot X, \\ K &= W_K \cdot X, \\ V &= W_V \cdot X \end{aligned} \tag{1.2.1}$$

$$\text{Attention}(Q, K, V) = \text{softmax}(QK^T)V \tag{1.2.2}$$

Remark 1.2.1. In [1], authors additionally scaled the weights α_i by the square root of the dimension of the key vectors d_k. This is, however, not strictly necessary and is done for numerical stability reasons.

The **multi-head attention** mechanism is simply a concatenation of multiple attention mechanisms. That is, we can compute h different attention mechanisms in parallel and then concatenate the results. The main idea behind this is that different attention mechanisms can learn different features of the input vectors.

Figure 1.1 visualizes the attention mechanism introduced in [1].

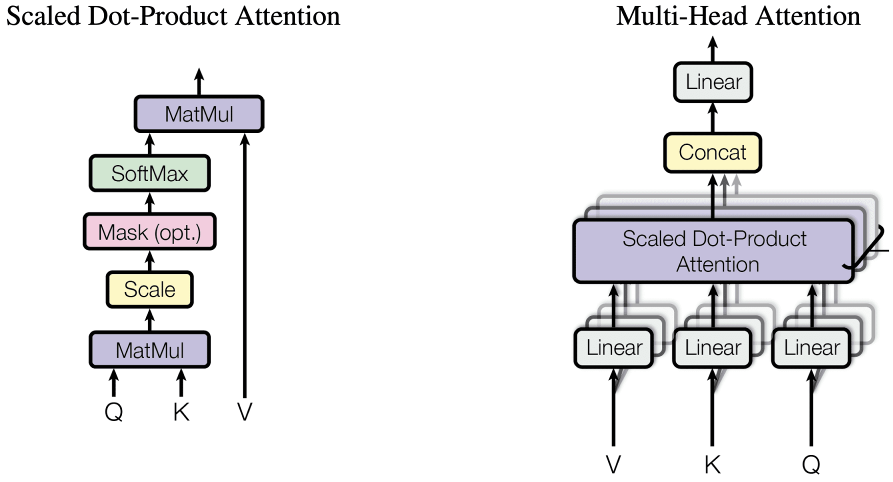

Figure 1.2: Scaled Dot-Product Attention and Multi-Head Attention [1]

Linear attention

In [12], authors propose an extension to the dot-product attention mechanism called **linear attention**. This significantly reduces the computational complexity of the attention mechanism by eliminating the need to compute the softmax function.

Notice that in 1.2.2, the softmax function is applied rowwise to the matrix QK^T. The softmax function can be substituted with a general similarity function $\text{sim}(\cdot, \cdot)$ between a query q_i and a key k_j. The equation 1.2.2 for output value v_i' can then be rewritten as follows:

$$v_i' = \text{Attention}(Q, K, V)_i = \frac{\sum_j \text{sim}(q_i, k_j) v_j}{\sum_j \text{sim}(q_i, k_j)} \tag{1.2.3}$$

The only constrained imposed on the similarity function $\text{sim}(\cdot, \cdot)$ is that it should be non-negative for it to define an attention function. This conveniently includes all kernels. That is $\text{sim}(q_i, k_j) = \phi(q_i)^T \phi(k_j)$ where $\phi(\cdot)$ is a feature map.

So that given a kernel with a feature map $\phi(\cdot)$, the attention mechanism can be computed as follows:

$$v_i' = \text{Attention}(Q, K, V)_i = \frac{\sum_j \phi(q_i)^T \phi(k_j) v_j}{\sum_j \phi(q_i)^T \phi(k_j)} \tag{1.2.4}$$

And we can rewrite the attention mechanism in matrix form as follows:

$$\text{Attention}(Q, K, V) = \frac{\phi(Q)^T \phi(K) V}{\phi(Q)^T \phi(K)} \tag{1.2.5}$$

Regrouping the terms, we get the following expression for the attention mechanism:

$$\text{Attention}(Q, K, V) = \phi(Q)^T \frac{\phi(K) V}{\phi(Q)^T \phi(K)} \tag{1.2.6}$$

which makes it evident that we can compute $\sum_j \phi(k_j) v_j$ once and reuse them for all the queries q_i which reduces the computational complexity from $O(N^2)$ to $O(N)$ where N is the number of input vectors in the attention layer.

Remark 1.2.2. In [12], authors used the $\phi(x) = elu(x)+1$ feature map where $elu(x) = \max(0, x) + \min(0, \alpha(\exp(x) - 1))$ is the exponential linear unit activation function. This feature map is used to ensure that the attention mechanism is non-negative and hence defines a valid attention function. Moreover, $elu(\cdot)$ is used instead of $ReLU(\cdot)$ to ensure the differentiability when x is negative.

1.2.2 Transformers for time series modelling

TODO: provide the main overview of the papers that use transformers for time series modelling, the comparison to other methods, the comparison of different transformer architectures specifically tailored for time-series

1.3 FPGA design

In this section, the main design principles of programming an FPGA board will be described. Readers will be introduced to the common optimization techniques and how they are achieved. The FPGA programming will be done using C++ HLS which is converted to verilog code.

1.3.1 Introduction to FPGA

The progress of hardware acceleration devices like field-programmable gate arrays (FPGAs) enables the achievement of high component density and low power consumption, all the while minimizing latency [7]. They are commonly used to accelerate high-performance, computationally intensive systems (for example, data centers) or to minimize the latency of execution (for example, in high-frequency trading).

1.3.2 FPGA development and HLS

Common Terms

Simulation, Cosimulation

A way to design and debug the solution without running it on the board.

HLS synthesis

In this section, HLS synthesis will be described [13]. It is now the common workflow in the FPGA development because it significantly improves the productivity when working with design.

1.3.3 Common optimizations

In this section, common optimization techniques and how they are achieved will be introduced.

Pipelining

Example code:

```c
void toplevel(din_t* a, din_t* b, dout_t* c, int len) {
  vadd: for(int i = 0; i < len; i++) {
#pragma HLS PIPELINE
     c[i] = a[i] + b[i];
  }
}
```

Loop Unrolling

Arrays

TODO: Partitioning

Streams

TODO:

Chapter 2

Experiments

2.1 Architecture and hyperparameters

use simple gridsearch + plot evaluation metrics

2.2 Datasets

In this section, the datasets used for evaluation will be described.

2.2.1 Numenta Anomaly Benchmark (NAB)

To assess the accuracy of predictions, we use the Numenta Anomaly Benchmark [14] dataset, which contains various real-world labeled time series of temperature sensor readings, CPU utilization of cloud machines, service request latencies, and taxi demands in New York City. It is commonly used to assess the performance of anomaly detection models on time-series data.

The reason why we use this dataset is that it is a standard benchmark dataset for anomaly detection in time series and because it has a large number of labeled time series.

A sample time series of NYC taxi demand is presented in Figure 2.1.

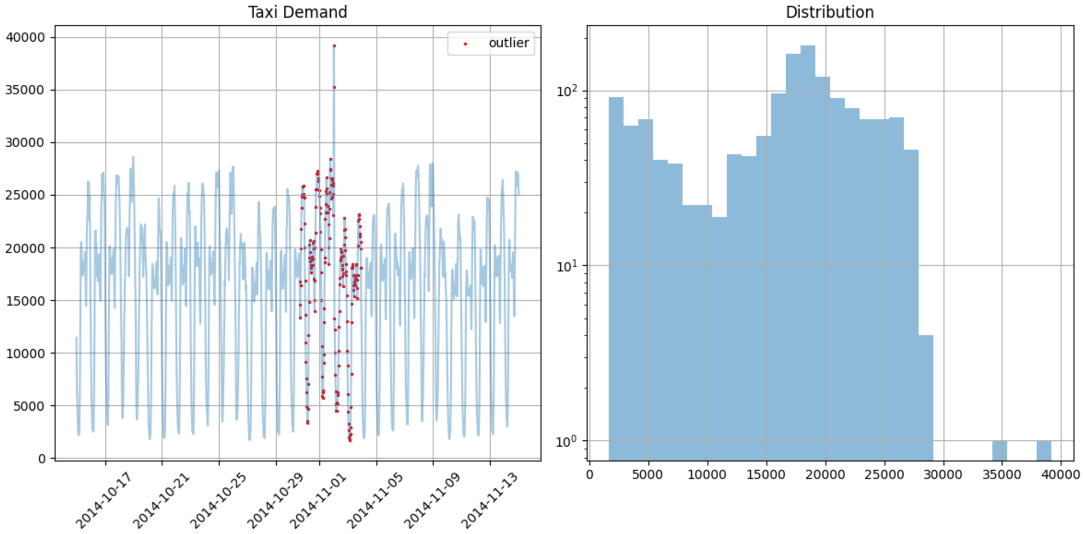

Figure 2.1: NYC Taxi demand - anomalies highlighted in red

2.2.2 KPI Anomaly Detection Dataset

The other labeled dataset that we use is the KPI Anomaly Detection Dataset (KPI AIOps) [15]. This dataset alongside the NAB dataset will be used to evaluate the predictive performance of the anomaly detection models.

The dataset consists of KPI (key performace index) time series data from many real scenarios of Internet companies with ground truth label. KPIs fall into two broad categories: service KPIs and machine KPIs. Service KPIs are performance metrics that reflect the size and quality of a Web service, such as page response time, page views, and number of connection errors. Machine KPIs are performance indicators that reflect the health of the machine (server, router, switch), such as CPU utilization, memory utilization, disk IO and network card throughput.

A sample time series of a sensor readings is presented in Figure 2.1. We can clearly see the outliers for some of the observations (colored in red).

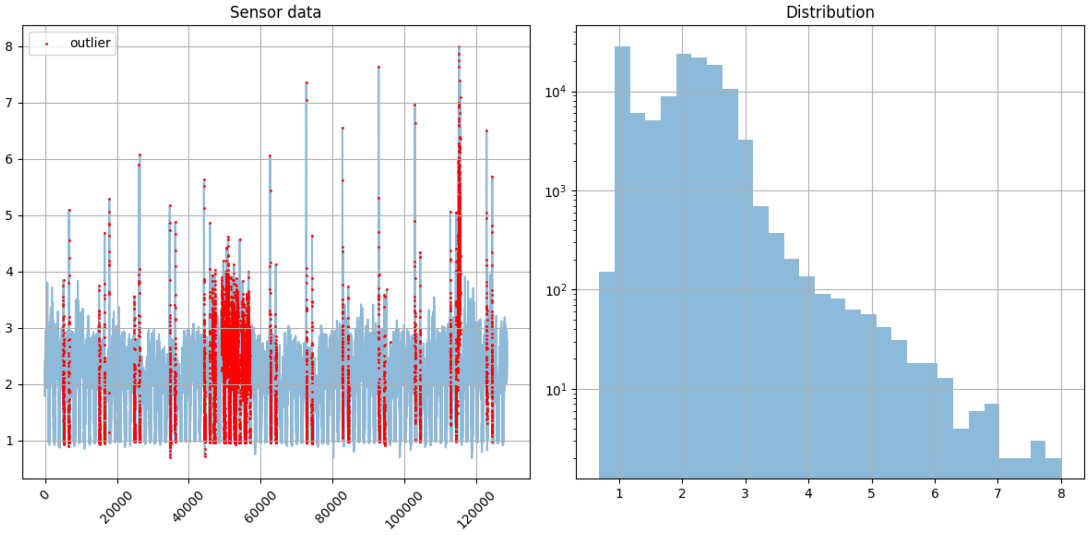

Figure 2.2: Sensor data from a machine in a data center. The red dots indicate the anomalies.

2.2.3 FI2010

In [16], authors described the first publicly available benchmark dataset of high-frequency limit order markets for mid-price prediction. The dataset contains 10-day limit order book data from June 2010 for five stocks that are listed on the Helsinki exchange. Each entry in the time series provides price details and aggregate order sizes for the top ten levels on both the bid and offer sides of the market, totaling forty data points. The time series consists of approximately four million messages, representing incoming buy/sell orders or cancellations. The dataset features order book snapshots taken after every 10 messages, resulting in approximately 400,000 records for the five stocks.

A number of versions of the dataset are available using different normalization schemes. We used the not normalized version of the dataset.

For the purpose of this work, we only extract only the mid price from the dataset which will be used for anomaly detection task.

Synthetic outliers

Since the dataset is not labeled, we have to inject synthetic anomalies into the dataset. We employ the approach similar to [17] with a slight modification. The algorithm can be summarized as follows:

1. Select n samples from the time series which will be contaminated (i.e., anomalous)

2. Replace the sample S_i with $\hat{S}_i = S_i(1+\delta)$ where δ is the injected outlier in the return space.

Authors model δ as a uniformly distributed random variable $\mathcal{U}[0, \rho]$. We instead use the normal distribution with matching mean and standard deviation of the returns time series.

An example of the injected outliers is presented in Figure 2.3

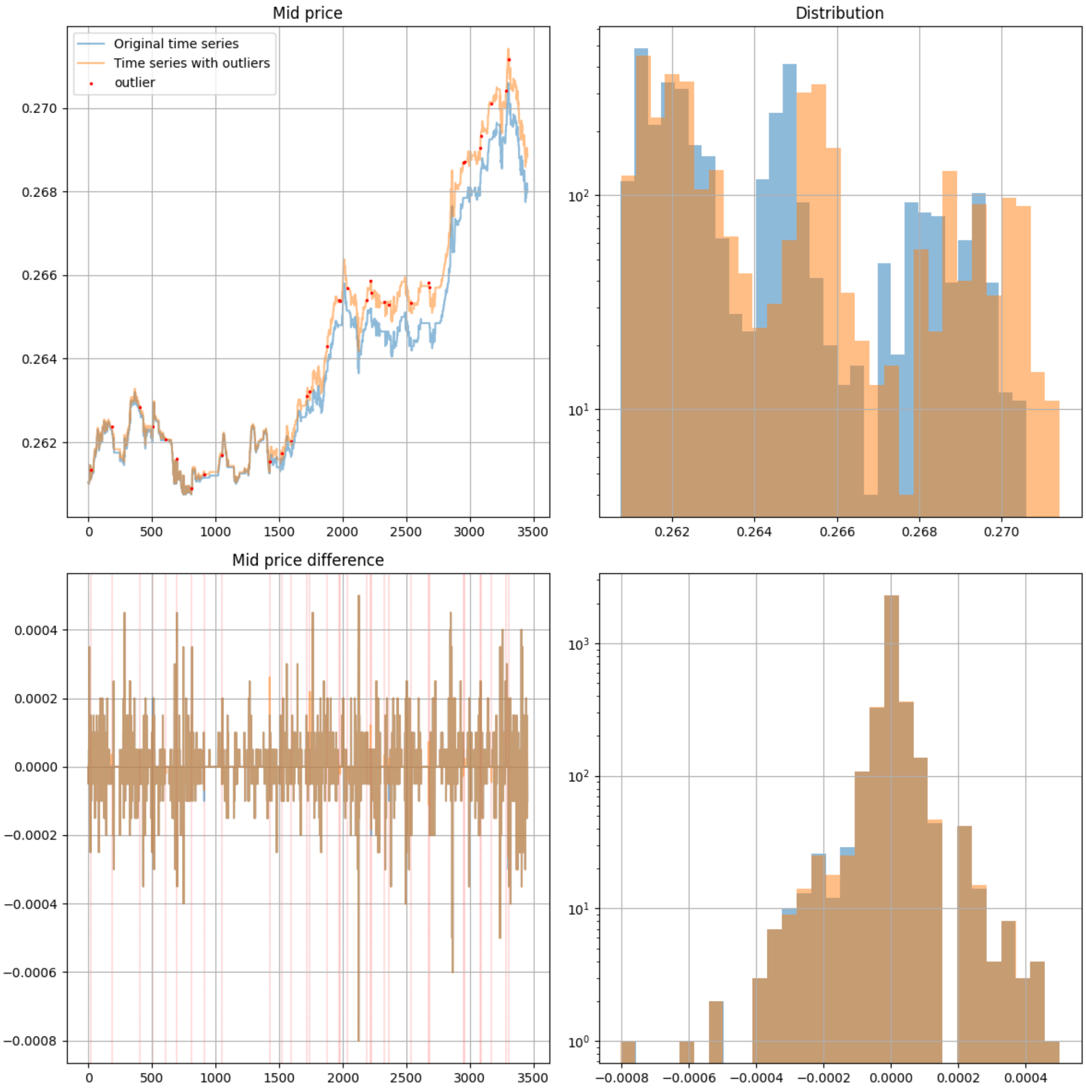

Figure 2.3: Example of the injected outliers in the FI2010 dataset.

2.3 Accuracy

TODO: describe the main metrics (accuracy, balanced accuracy, F1, Cohen's Kappa, Confusion matrix)

2.4 Performance/Speed

2.5 Resource utilization on FPGA

Conclusion

2.6 Future work

Bigger FPGA boards.

Evaluation of performance on more recent financial market data.

Appendix A

Code

A.1 Efficient matrix multiplication

This is Appendix A.1, which usually contained supporting material, or complicated proofs that might make the main text above less readable / fluid.

Bibliography

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[2] Thomas Neil Falkenberry CFA. High frequency data filtering, Sep 2008.

[3] Owen Vallis, Jordan Hochenbaum, and Twitter. Introducing practical and robust anomaly detection in a time series.

[4] Mohiuddin Ahmed, Nazim Choudhury, and Shahadat Uddin. Anomaly detection on big data in financial markets. In *2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 998–1001, 2017.

[5] Anthony Gillioz, Jacky Casas, Elena Mugellini, and Omar Abou Khaled. Overview of the transformer-based models for nlp tasks. In *2020 15th Conference on Computer Science and Information Systems (FedCSIS)*, pages 179–183, 2020.

[6] Qingsong Wen, Tian Zhou, Chaoli Zhang, Weiqi Chen, Ziqing Ma, Junchi Yan, and Liang Sun. Transformers in time series: A survey. 2022.

[7] Andreea-Ingrid Funie, Liucheng Guo, Xinyu Niu, Wayne Luk, and Mark Salmon. Custom framework for run-time trading strategies. In Stephan Wong, Antonio Carlos Beck, Koen Bertels, and Luigi Carro, editors, *Applied Reconfigurable Computing*, pages 154–167, Cham, 2017. Springer International Publishing.

[8] Julio-Omar Palacio-Niño and Fernando Berzal. Evaluation metrics for unsupervised learning algorithms, 2019.

[9] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2014.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[11] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[12] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention, 2020.

[13] Xilixn Inc. Vitis High-Level Synthesis User Guide, may 10 2023. [Online; accessed 2023-07-16].

[14] Subutai Ahmad, Alexander Lavin, Scott Purdy, and Zuha Agha. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing*, 262:134–147, 11 2017. [Online; accessed 2023-07-19].

[15] Zeyan Li, Nengwen Zhao, Shenglin Zhang, Yongqian Sun, Pengfei Chen, Xidao Wen, Minghua Ma, and Dan Pei. Constructing large-scale real-world benchmark datasets for aiops, 2022.

[16] Adamantios Ntakaris, Martin Magris, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis. Benchmark dataset for mid-price forecasting of limit order book data with machine learning methods. 2017.

[17] Stéphane Crépey, Noureddine Lehdili, Nisrine Madhar, and Maud Thomas. Anomaly Detection in Financial Time Series by Principal Component Analysis and Neural Networks. *Algorithms*, 15(10):385, oct 19 2022. [Online; accessed 2023-07-19].