

# TradeNIC: Trading-specific SmartNIC Design for Low Latency and High Throughput Algorithmic Trading

**Team Members:** Kerem SARI, İsmail Hakkı İPEK

**Supervisor:** Assist Prof. Dr. İsmail San

**University name:** Eskisehir Technical University

**GitHub Link:** <https://github.com/DHLSan/TradeNIC>

**Video Link:** <https://youtu.be/zLlkz4FTuO0>

## Abstract

Stock and currency exchange transactions are increasing with the emergence of cryptocurrencies. However, the latency in the exchanges have an important role due to its profit rates and high market demands. From an application-specific digital hardware design point of view, all the networking, real-time forecasting and investing computations can be smarter and faster with the FPGA based Smart Network Interface Cards (SmartNICs) that allows, multiple data processing on programmable logic by providing simultaneous multiple network-based communications. In our interdisciplinary study, we propose TradeNIC, low-latency and high throughput trading-specific SmartNIC design equipped with deep learning-based price forecasting method. Our TradeNIC design utilizes RNN-LSTM network as a powerful deep learning method for time-series based forecasting. The TradeNIC uses the lightweight IP network API with UDP protocol and accelerates the LSTM and Dense layers functionalities thanks to reconfigurable hardware design of SoC architecture. TradeNIC shows that with the hardware accelerators each prediction operation performs x5.81 times faster than software design. This shows that TradeNIC performs in low-latency, high throughput, and less power consumption. Considering the differences between Intel i7 7th generation processor and TradeNIC co-design, i7 shows only x3,25 times better computation time for each prediction. TradeNIC is much more advantageous in terms of source, power consumption and throughput. All the experiments showed that TradeNIC like designs are suitable for data centres and edge devices.

**Index Terms**— SmartNIC; LSTM; RNN; Deep Learning; Time series forecasting; UDP; SoC architecture

## 1. INTRODUCTION

In recent years, the number of users in the markets are increasing to invest in conventional stocks. Interestingly enough, with the cryptocurrency markets, the number of real-time investors in these new markets are increasing dramatically, due to their high earning rates. There is an opportunity to automate the trading and all these computations smarter and faster from an application-specific digital hardware design point of view. All the networking, investing, marketing computations using available market indicators and heavy neural network-based forecasting methods could be accelerated in a network interface card (NIC) without any communication to a CPU on the SmartNIC-based approach. We call this NIC card TradeNIC which is mainly responsible for accelerating the trading computation.

There are various indicators/mathematical methods to oversee/forecast the stock price changes, and market transactions and they especially need a real-time computation, and they are computationally-complex and require a speedup for a real use-case. Automating smart-trading in these markets promises a very high amount of profit but implementing them in software-based trading bots is not an efficient and viable approach, especially in high-frequency trading. Automated trading algorithms contain heavy decision-making processes for buying and selling assets, and artificial intelligence and machine learning techniques provide better decisions. However, their computations should be addressed in terms of latency and high throughput since every millisecond latency in stock exchange transactions may cause a significant amount of financial profit or loss depending on the correct decision with high throughput. Improving the latency by accelerating the trading computation on hardware, optimizing the compute-heavy trading algorithms, and offloading host CPU applications that traditionally run-in multi-core CPU/GPU based server applications will significantly help investors. Achieving high speed, low-latency trading in an optimized trading-specific SmartNIC will be more profitable.

In algorithmic trading, latency is the time delay between one exchange to another as seen in *figure 1*. In the example, it takes 0.5 seconds to reach the Trader's screen from the exchange, 0.4 seconds to process and analyse the trade and 0.5 seconds to reach a broker. It takes 1.4 seconds to actualize trading. In this period of time stock values can change many times for that reason, latency has a very important role in trading. To reduce the latency in trading exchanges our SmartNIC design proposes to accelerate the elapsed time between Trader's Screen and Analysis and Order Placed as mentioned in *figure 1*.

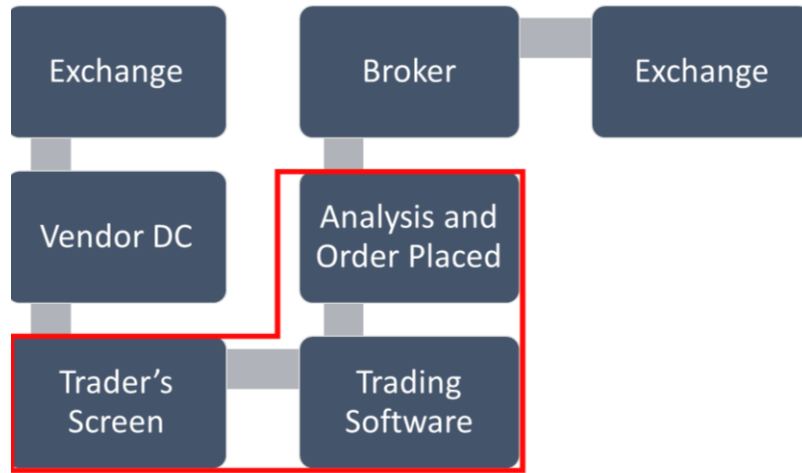


Figure 1 An example of a point-to-point exchange operation in trading.

Over the years, many neural networks techniques have been used in time-series stock market forecasting. Nowadays, deep learning models are very popular due to their speed and accuracy of prediction, but in longer sequences these kinds of models are facing dependency issues. This way it is convenient to not use longer sequential data processing in trading [1]. Because of that, in this paper, we present one of the most common time-series forecasting techniques: LSTM (Long Short-Term Memory) RNN (Recursive Neural Network). Using RNN without LSTM causes a cyclic structure that allows data to remain temporarily in the network. LSTM architecture solves the long-term time dependency problem, but large-scale

LSTMs can highly have compute-intensive executions [2]. To address this problem, we propose a SmartNIC design to accelerate this compute-heavy operation and offload from the host CPU in terms of reaching low latency, high throughput, and lower power consumptions.

The main contributions of this study as follows:

- Designing a high precision trading specific FPGA based SmartNIC for data centres or edge devices
- Generating high throughput forecasting with a Neural Network based deep learning model on a bare-metal system in a low latency way.
- Generating a trading specific co-design thanks to reconfigurable SoC architecture in an optimised way, using HLS tools.
- Generating network-based infrastructure for real-time forecasting for every type of financial markets (Cryptocurrency, Stock etc.) and timeframes.

Our responses of Xilinx Open Hardware as follows:

- **Technical Complexity:** Our SoC solution was performing trading based real-time, high throughput price prediction operations on an FPGA based SoC in a low latency way so the traders can make decisions in milliseconds by using mathematical models and complex algorithms. We achieved this operation by generating a SmartNIC concept by constructing the RNN-LSTM network deep learning method. We have developed a high precision and low latency model that performs real-time prediction on reconfigurable SoC architecture in the SmartNIC concept.
- **Re-usability and Implementation:** In this project, we implemented our proposed deep learning model to our TradeNIC design by testing different amounts of neurons and LSTM cells to choose the best precision model for the system in terms of throughput and computation. We have proven that our proposed TradeNIC design performs well in terms of latency, throughput, and power consumption. Also, TradeNIC can be configured with different types of trading models easily. (We built a structure to load the trained model to DDR3 through a network for re-usability.)
- **Marketability/Innovation:** In recent years, the global economy has been exposed to global trading systems. Interest in trading has become more popular in the world's economy. To manage all the networking, security and economic demands requires smart solutions in cloud environments. These types of traffic on the data centres have made SmartNICs more popular for distributing the application-specific workloads from CPUs. Cloud providers are offering custom SmartNIC designs for application-specific solutions to cloud environments (AWS Nitro or Azure SmartNIC). Many vendors are investing in the SmartNIC ecosystem, such as Xilinx (AMD), Mellanox (Nvidia), Intel and Marvell. Designing high throughput forecasting models such as LSTM-RNN to cloud integrations, our TradeNIC presents to both vendors and investors a SmartNIC design for high throughput and low latency for real-time trading.

## 2. THEORETICAL BACKGROUNDS

This section includes information about the basis of studies carried out within the scope of the project and that will help to have a better understanding of the processes given.

### 2.1 SmartNIC

SmartNIC, i.e., Smart Network Interface Card, is an embedded system card that provides multiple data processing on programmable logic by providing simultaneous multiple Ethernet-based communication. SmartNIC not only sends the data from the network to the processor and the data from the processor to the network but also accelerates the network-based operations on the incoming data in the network adapter in hardware. It performs hardware acceleration of complex algorithms and eliminates problems by enabling real-time data packets to be transmitted and processed in low latency mode. These SmartNIC cards can communicate with each other and also with different host machines.

### 2.2 UDP COMMUNICATION

UDP (User Datagram Protocol) is a transport layer protocol that does not establish any handshake connection between the server and the client, especially used for time-sensitive unreliable data transmissions. It does not apply a control mechanism to ensure the reliability and security of the data to be transmitted from the source to destination, and packet losses may occur during transmission. Also, UDP communication does not wait for the receiver to request transmission, so it occupies less bandwidth on the network and uses less memory. As a result, UDP provides faster data transmission because it does not establish connections and does not check for packet transmissions. In algorithmic trading, reaching the stock price data as fast as possible will ensure that the designed model is more optimized.

### 2.3 RNN-LSTM NETWORK MODEL

RNNs (recurrent neural networks) are powerful artificial neural networks that are mostly used in time series prediction, they can maintain the memory of the input. However, RNNs have a problem called vanishing gradient which causes the model of training to be extremely slow and difficult to handle in long term dependencies. LSTM is designed to solve RNN's longer dependency problems. An LSTM cell includes three special gates to provide RNN long-term dependencies in the sequence shown in *figure 2*. As the first step LSTM should have to decide which data will throw away to the cell state,  $C_t$ . This is done by using sigmoid  $\sigma$  on the "forget layer" as represented in (1).  $h_{t-1}$  and  $x_t$  are the inputs and the output is a number between 0 and 1 and decides how much of the value should be multiplied with cell state,  $C_t$ .

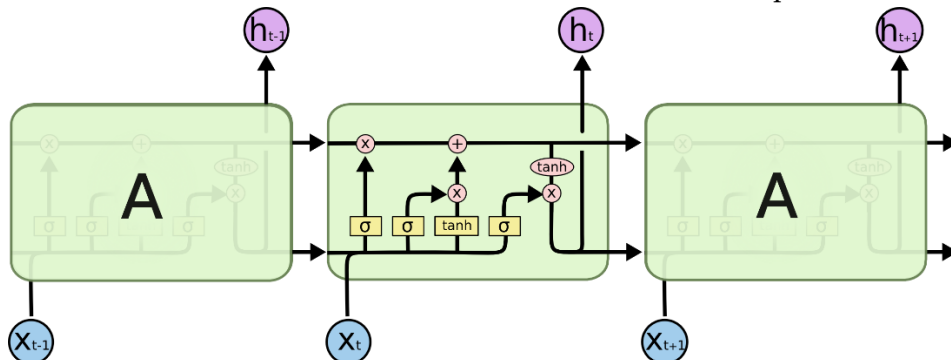


Figure 2 Repeating Module in a standard RNN that contains LSTM layers.

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \quad (1)$$

The other gate is the “input gate” (2)  $i_t$  decides which new data is going to add or not.  $\tilde{C}_t$  creates a tanh layer for new candidates (3).

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \quad (2)$$

$$\tilde{C}_t = \tanh(W_c[h_{t-1}, x_t] + b_c) \quad (3)$$

Then,  $f_t$  multiplies with the old state to forget and adds with the new candidate (4).

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \quad (4)$$

Finally, the “output gate”  $o_t$ , calculates the potential value for the first time (5) the output  $h_t$  regulates itself using present cell state (6).

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \quad (5)$$

$$h_t = o_t * \tanh(C_t) \quad (6)$$

Here,  $t$  is the time step and  $W_f, W_c, W_i, W_o$  are the weight matrices;  $b_f, b_c, b_i, b_o$  are the biases of the gates.

This calculation is the most common LSTM description for sequence learning, and it is used on the project. There are other variants of LSTM based on this calculation such as “*peephole connections*” designed by *Gers & Schmidhuber*, GRU (Gated Recurrent Unit) designed by *Cho* etc. [3]

## 2.4 Normalization Techniques

In recent digital areas, growing big data streams like financial data needs to be normalized effectively. It is important to reduce complexity and density in computations. To reduce input density on the trading-like operation using virtualized normalization techniques improves the computation time and throughput on the neural network. Using LSTM with the sliding window of fixed size mechanism and normalizing changing data in each window will allow to generate faster and more stable systems [4].

Neural Networks-based algorithms update their weights as they are trained, which provides inter-layer control. When updating these weights with BackPropagation, it creates an Internal Covariate Shift, which increases the time for training the model. The shift occurs due to the layers acting independently and separately from each other. Although each layer tries to correct its own error, the parameter changes in the layer causes changes in the next layers. Due to these shifts in the layers, the computational intensity increases and therefore the time required for the computation also increases.[5]

The shift and time elongation in layers can be reduced by using batch normalisation. In this algorithm, layers do not wait for the previous layer to be trained, that is, they are trained simultaneously. The standardisation and normalisation are performed on each layer with this method since other layers cannot benefit from the compression process between the values made at the input. Thus, while training the algorithm to be run on the Neural Network, it will be faster and more stable. [6]

### 3. TradeNIC: Trading-Specific SmartNIC Design for Low Latency and High Throughput Algorithmic Trading

In this study, the FPGA-based trading-specific SmartNIC system, TradeNIC, will improve the latency of the stock data forecasting on an LSTM based RNN network model. In this section, we explain which methods are being used, what protocol is being used and how the system is implemented.

Over the many years, traders are using conventional forecasting methods and trying to find high throughput trading models, there are many machine learning and artificial intelligent models in the market, but these are coming with performance and budget issues. Through this TradeNIC like platform in data centres and edge devices, traders can rent an instance from the nearest cloud services with low budget and high performance and perform its trading operations. In this TradeNIC like co-designs there are many functionalities in terms of hardware and software. In our project, one of the powerful deep learning network model for time-series forecasting RNN-based LSTM network method has been used to generate high throughput forecasting. Also, it has been created as a platform that supports LSTM functionality in an efficient way in terms of throughput and latency.

#### 3.1 UDP-BASED NETWORK COMMUNICATION

In this project, UDP based communication has been used to transmit stock market pricing from the host machine to the SoC architecture. Since stock values change rapidly over time, data transmission must be very fast and continuous, so that the strategy algorithm can respond quickly. In order to avoid packet loss, which is a common problem of UDP communication, we have implemented IP address forwarding over a fixed port. Only the special port we specified was used for the transmission of the created server and client definitions and stock data; Thus, data is prevented from being transmitted to random ports and lost. In addition, UDP communication increases the optimization of the designed model, as in hardware acceleration, consumed memory, power, and bandwidth are important. For the later use of case special network protocols can be designed to reduce latency and prevent data loss on the network.

In the project, An EchoBack system has been set up between the host and the TradeNIC via UDP packets over the ARM Cortex-A9 processor. The echo server ensures that the stock data packets are sent bi-directionally to the SoC architecture, and the predictions of the strategy algorithm are returned to the host without any loss. On this echo server port, the main thread listens continuously and receives a new socket identifier, as the echo service thread creates a separate thread for each connection request from the client whose input can be read from [7].

Open-source LwIP (Lightweight IP) libraries on Vitis SDK (Software Development Kit) are used to set up a server on the SoC side for UDP communication. The general working principle of the created UDP communication is summarized as follows; First, the UDP socket is created, and the socket connects to the server address, then the client waits for the packets to arrive, and the incoming packets are processed and sent back to the client, and finally the packets are returned to the waiting phase as shown in *figure 3*.

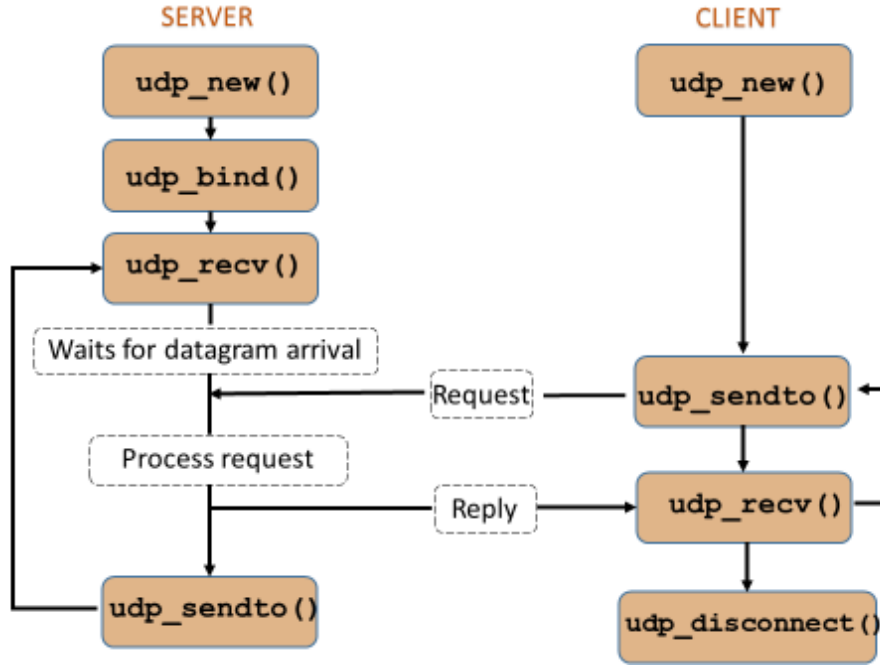


Figure 3 Server-Client UDP Communication with Functions of LwIP Library

For the creation of EchoBack Server in a thread, the following (figure 3) LwIP functions are used. `udp_new()`, creates a new UDP PCB (Protocol Control Block) which can be used for UDP communication. The PCB is not activated until it has either been bound to a local address or connected to a remote address. Since UDP is a connectionless protocol PCB is initialised to zero. `udp_bind()`, binds the UDP PCB to be bound with a local address and a designated port. `udp_recv()`, sets a receive call-back for a UDP PCB and this call-back is called when receiving a datagram for the PCB. `udp_sendto()`, sends data to a specified address and port using UDP.

### 3.2 RNN-LSTM NETWORK MODEL

This project contains RNN-LSTM deep learning concepts for time series forecasting both in host machine and TradeNIC design. Time-series forecasting uses the previous data and predicts the future data points. This is a common pattern used for trading applications. Financial market prices that change over time use a time series model to predict future changes more accurately. The RNN-based LSTM network model is more suitable [8] for estimating continuously increasing and decreasing pricing on a timeline. Since training the algorithm with historical data increases the accuracy of the applications, the LSTM model, which is suitable for feedback in the trading forecasting process, is more advantageous but using these forecasting models can be compute-intensive [9].

In the host machine to generate a trained model we used TensorFlow and Keras library in Python3 and generated trained models by using historical data of different types of financial market price exchanges. To reduce complexity and density in computations normalization techniques were used in RNN-LSTM layers. While generating model batch normalization of Keras library has been used in every layer and while predicting on TradeNIC, LSTM with the sliding window of fixed size mechanism has been used by normalizing changing data in each window. Also, the layer functionalities were tested in

different sizes of neurons (LSTMcells), window sizes and eliminated to achieve higher throughput and low latency on our TradeNIC design.

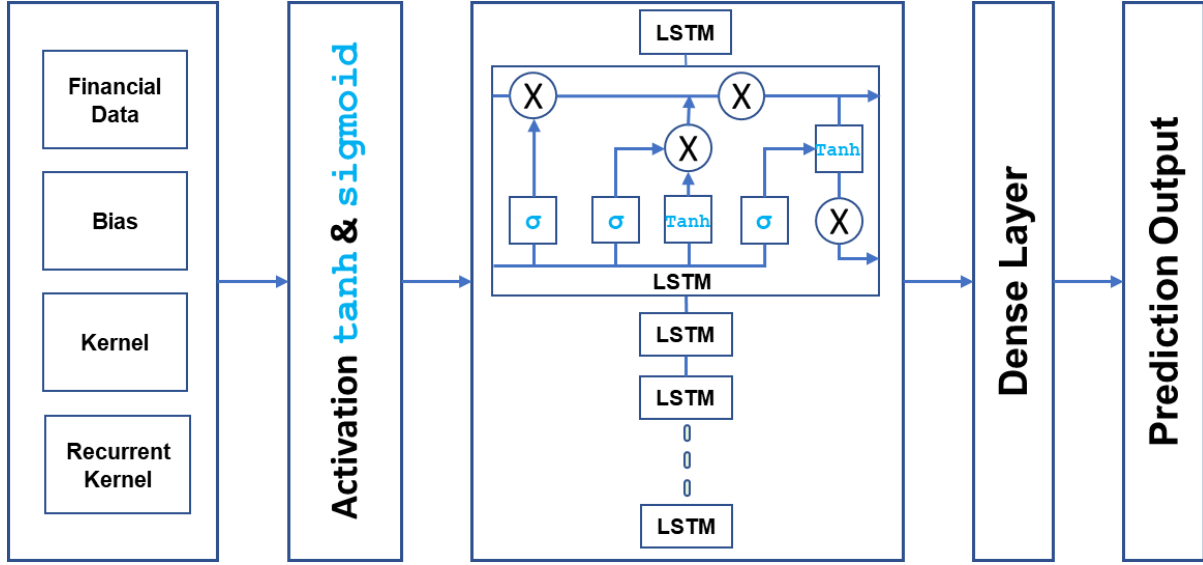


Figure 4 RNN-LSTM network architecture of TradeNIC

RNN-LSTM networks on SoC design represented in *figure 4*, the LSTM layer provides a recurrent segment with default tanh activation enabled and includes 100 LSTM units, DENSE layer (the core layer) has one output using sigmoid which represents a number between 0 and 1. The weights (kernel as Keras input call), hidden matrix (recurrent kernel as Keras input call) and biases of the trained model are used as inputs to the FPGA implementation of this LSTM network. The training, normalization of weights and hidden matrix of financial data is all completed on the Python software platform.

On the TradeNIC co-design the C implementation of the LSTM and DENSE layer functionalities which are programmed and generated via HLS tools. The designed layer accelerators implemented to UDP functionalities on specified ports by using LwIP API and generated a custom circular buffer structure with the sliding window technique for window size  $50 \times 2$  to perform real-time trading operations with dependencies on the configurable SoC architecture. In addition, Batch normalization is performed while loading weight, hidden matrix, and bias values to the DDR3 and computing the current state in the LSTM layer by using tanh and sigmoid activation functions.

### 3.3 Financial Data Pre-processing

It is necessary to examine the increases and decreases in small periods in order to predict the intraday pricing of cryptocurrency exchanges. In the project, the historical financial data are accessed via API at 1-minute intervals on the BITFINEX free online trading platform [10]. Our custom design uses Open and Close parameters of every exchange to perform prediction operations on close values. Also, TradeNIC allows to predict other data types such as open, high, volume etc. easily. Financial data format is a large number of floating points, to predict in higher precision all the input values must be normalized in the  $[-1:1]$  range by scaling and shifting in each layer so that these values do not create computational density for the system. For this reason, standardization and normalization of financial data is carried



out within our TradeNIC design, whenever new close and open parameters are received to TradeNIC it operates a window-based normalization operation for each financial data frame.

### 3.4 System Implementation

Our system implementation is simulated by using a host machine as a Traders' screen and TradeNIC as a SmartNIC. *Figure 5* shows all processes and states of the project.

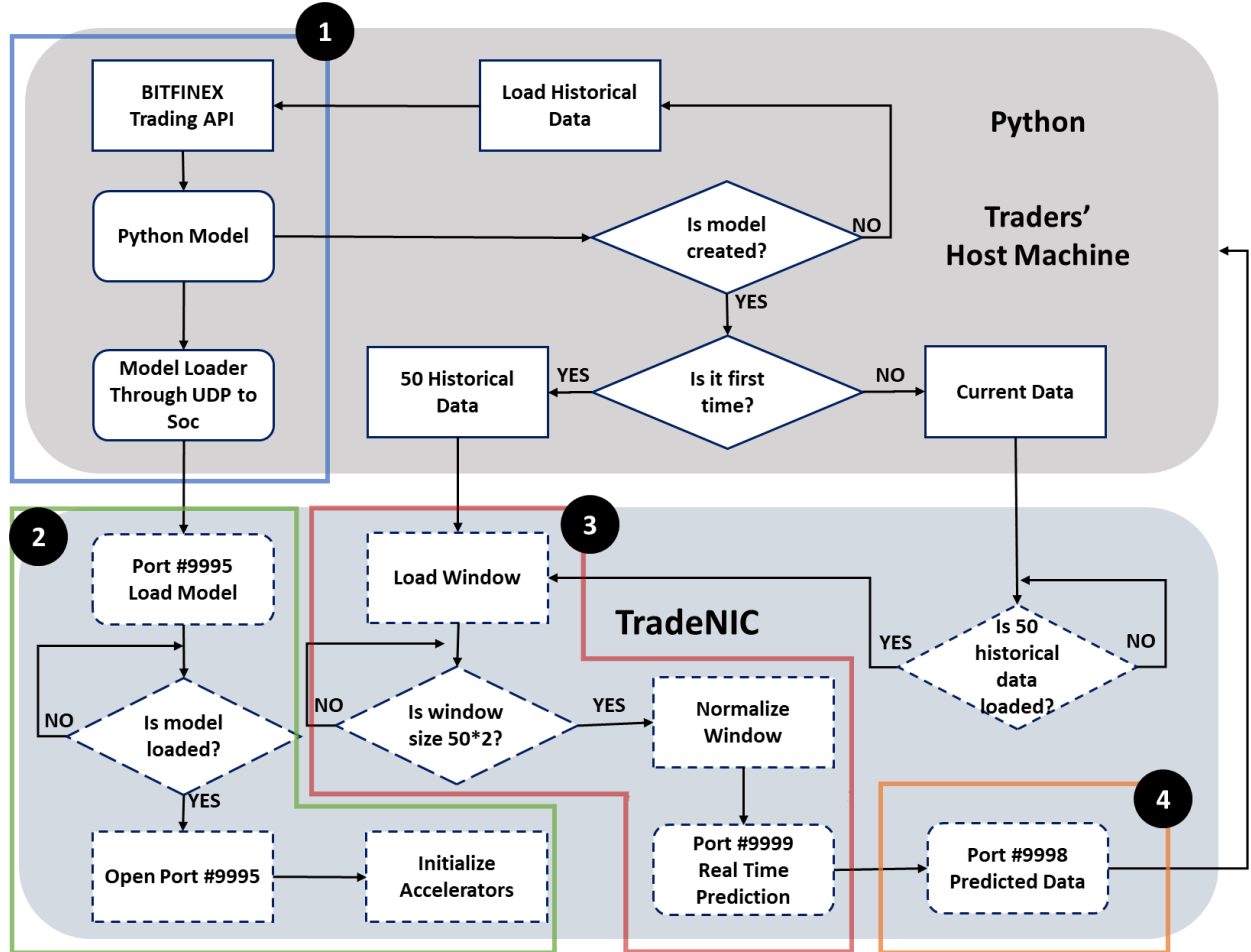


Figure 5 Flowchart of project model for performing a prediction operation on TradeNIC.

- 1) Represents the model generation process on Trader's host. The trader can generate every type of financial model with the specified timeframe and uploads to TradeNIC.
- 2) TradeNIC receives the traders' model weights, biases, hidden matrix values and initializes the LSTM and Dense network accelerators on UDP port 9995.
- 3) TradeNIC connects to Bitfinex API and receives specified real-time financial data with the specified timeframe on UDP port 9999 and fills and normalizes the window then performs prediction operation by using accelerators.
- 4) After prediction is performed, it sends the data to Traders' host machine on UDP port 9998.

In *figure 5*, optimized trading software application for stock and cryptocurrency price prediction is performed on TradeNIC. Both historical and real-time financial data is retrieved

from an open-source online trading API, Bitfinex. Purpose-oriented ports are specified for Ethernet communication between host and TradeNIC via UDP. In the model, the suggested Hardware Acceleration motivations are realized by running the LSTM and Dense functionalities on programmable logic. The algorithms are synthesized in the HLS program and IP Cores are generated in VHDL at RTL level. The algorithm is enhanced with pragmas, pipelines etc. to provide a more optimized hardware acceleration. Finally, a Hardware/Software co-design system is created with these accelerators.

## **4. Experiments**

In this interdisciplinary work, we aimed to generate an optimized SmartNIC design in terms of latency, throughput, and power consumption. All the throughput tests performed on Python 3.8.10 and evaluated the best model for the TradeNIC. We tested the LSTM and Dense functionalities on a VMware virtual machine Linux kernel Ubuntu 20.4, Zynq 7000 bare-metal applications and Zynq 7000 configurable SoC architecture with accelerators we have designed. All the tests on the FPGA are performed at a frequency of 100 MHz and on the Ubuntu 3800 MHz by using one core of i7 7th generation processor. Considering the experimental process, it includes the following titles:

- LSTM layer throughput optimization tests performed in Python 3.8.10 to generate a high precision model.
- Prediction latency optimizations on reconfigurable SoC architecture.
- RTL level generation of LSTM and Dense network layer with the HLS tools.
- SoC design by using Vivado program, PL (Programmable Logic) and PS (Processing System) connection of LSTM and DENSE IP Core are established with AXI Interface.
- Resource Utilisation information of TradeNIC.
- Latency and throughput comparisons between TradeNIC and i7 10th CPU.

### **4.1 Experimental Setup**

While implementing the TradeNIC model, basically certain steps were followed. (1) A trade model was trained with the help of Python Keras Library. (2) The financial data and the trained model were transferred over the network via UDP communication. (3) The IP Core was generated for LSTM and Dense layers within the framework of low-latency motivation by performing synthesis at RTL level with HLS configuration. (4) Hardware block design was created for TradeNIC on Vivado and communication between Vitis program and processing system and programmable logic was provided.

#### **4.1.1 Trading Model Generation**

The RNN-based LSTM network model was created with the Python Keras Library in order for TradeNIC to make predictions with higher throughputs. The last 4000 data with 1-minute timeframes received from Bitfinex via free-API were used about 3 days before the training was made.

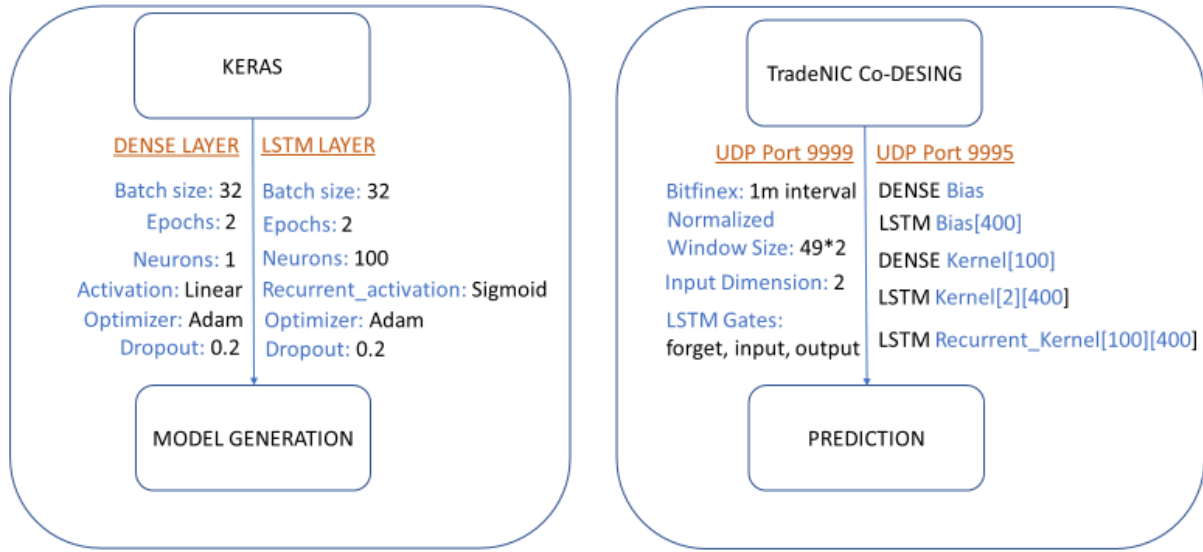


Figure 6 System configurations for both in Software platform and TradeNIC.

The parameters given in the Keras Library are used during the training of the model, as can be seen in *figure 6*. The recommended 32 batch sizes were used, and the maximum regularisation was tried to be achieved during the training. 100 neurons were used for the LSTM layer and the data were normalized again in each cell by batch normalization. Thus, a more effective model was created where both the computational intensity was avoided, and the shift of the input data was prevented. The number of epochs indicates how many cycles, i.e. iterations, of the train process will perform and in this model, 2 epochs were enough. The Activation function is used to normalize the output of hidden layers and the sigmoid function is used for LSTM. Since the dense layer does not have a multi-layer structure, non-linear transformation is not required, and linear activation is used. Adam, the adaptive momentum optimizer, was used in both layers. This optimizer, which is widely used in deep learning methods, converges to the optimum minimum quite quickly compared to normal gradient descent update by using gradients and squares while updating weights. The dropout value provides regularisation during the train and increases the generalisation ability of the network. In short, certain parts of neurons are randomly selected and disabled, thus forcing neurons to learn instead of relying on the outputs of other neurons. The value of 0.2 is the recommended ideal value, and above this, the training model begins to deteriorate.

It contains the weight and bias values in the model, and in the LSTM structure, the gates use these parameters to remember and forget the data. The parameter called weight constitutes the kernel matrix. The recurrent kernel parameter formed in the model is also the hidden state matrix. These parameters were used by the LSTM and Dense layers to make predictions.

#### 4.1.2 IP Core Generation

A Hardware Acceleration has been implemented to reduce the latency caused by the computational complexities of the LSTM algorithm. In order to run LSTM and Dense layers on programmable logic, IP Core was created by synthesising them at the RTL level. These algorithms in C language were converted to custom IP Cores in VHDL language with the HLS

program. The testbench was created by taking the weight and bias values of the trained model before the IP Core was synthesised. IP Core was generated after the layers were observed to work properly. Then, pipeline pragmas were used to optimise this synthesised block, and it was examined whether there was a change in the throughput of the algorithm.

```
#pragma HLS INTERFACE s_axilite port = input_data bundle = BUS_A
#pragma HLS INTERFACE s_axilite port = kernel bundle = BUS_A
#pragma HLS INTERFACE s_axilite port = recurrent_kernel bundle = BUS_A
#pragma HLS INTERFACE s_axilite port = bias bundle=BUS_A
#pragma HLS INTERFACE s_axilite port = lstm_out bundle = BUS_A
#pragma HLS INTERFACE s_axilite port = reset bundle = BUS_A
#pragma HLS INTERFACE s_axilite port = return bundle = BUS_A
```

Figure 7 Identifying the AXI Lite Interface and Bus A for LSTM layer IP Core which is generated

As shown in *figure 7*, the IP to be generated will enable communication with the AXI Lite Interface. This IP, which will work in slave mode, will produce output for the inputs as it is called. Inputs and outputs to be used are sent on the same channel. In this model, there is no need for burst mode as there is no need to provide multiple address locations for single input addresses. For this reason, AXI Lite Interface was used.

**#pragma HLS PIPELINE:** This pragma enables simultaneous execution of operations and computations, reducing the loop's initialization interval. In short, the loop does not have to wait for the previous loop to finish to start. [11]

#### 4.1.3 Soc Architecture Design

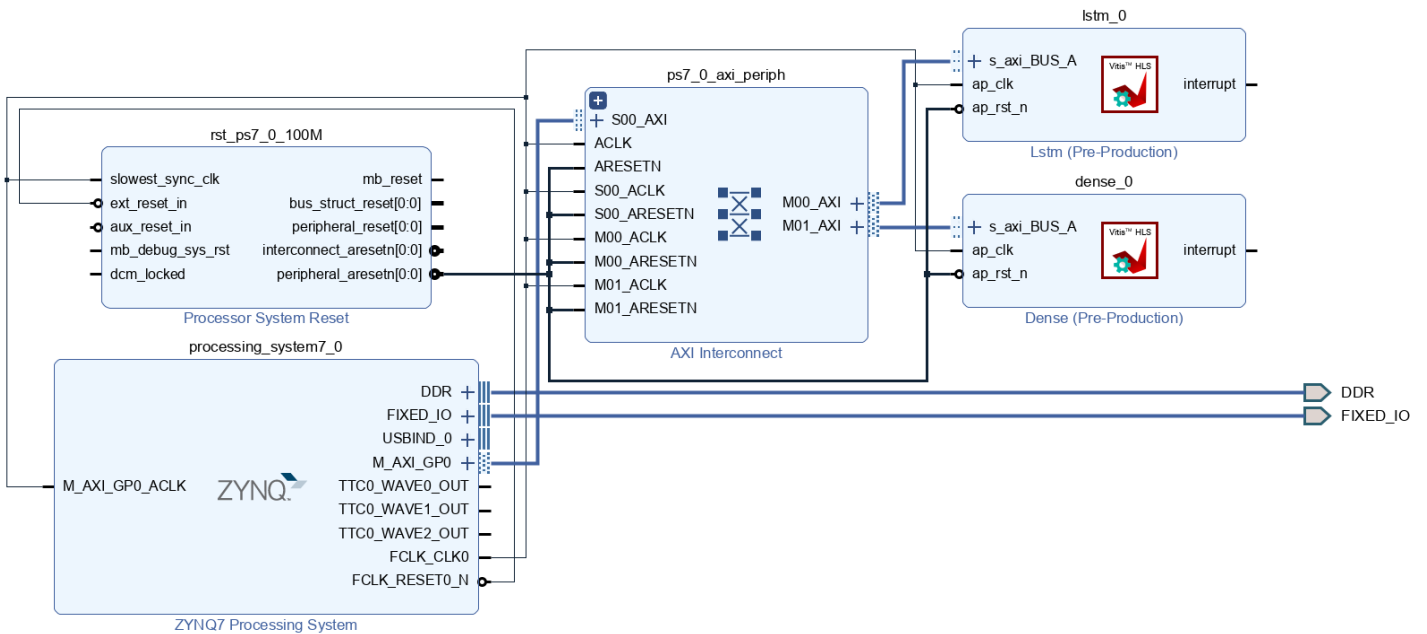


Figure 8 Vivado design of generated LSTM and Dense IP Core with the processing system

In order for LSTM and Dense IP Core Blocks produced with HLS to work on programmable logic, the hardware design is made as shown in *figure 8*. Here the connection between PL and PS is provided by AXI Interconnect. AXI Interconnect allows multiple IP

blocks with different addresses to be connected to the processor at the same time. When communicating with LSTM and Dense block, the interconnection block is in Master mode, but when communicating with PS, it behaves in Slave mode since the client is PS this time. The address range of the LSTM layer starts at 0x40000, and ends with 0x7FFFF, a 19-bit address length is used. The address range should therefore be  $2^{19}$  i.e. 512 KB. The 64 KB address range set by default was insufficient for LSTM IP Core, but it is sufficient for Dense IP Core.

In this SoC designed for TradeNIC, all software configurations are done on Vitis SDK; Operations such as the use of hardware design with UDP communications, LSTM and Dense blocks have been made. A UDP server was installed on the SoC using the LwIP library. With this server, TradeNIC can access the financial data and the trained model. The predicted values were sent to the host machine.

In the experiments performed on Vitis, the LSTM algorithm was first run on a bare metal system entirely on software. While measuring the running time of the algorithm on the ARM-based processor, the deviation rate was checked by comparing it with the actual values for the prediction throughput. Then, initialization and setup configurations of LSTM and Dense IPs for which hardware design was made were made. Pricing estimates were made using the trained model and real-time data received with UDP. Performance and throughput measurements of the system were made. The performance of TradeNIC has been measured using the `XTime_GetTime()` function in the "xtime\_1.h" library when running on both software and hardware.

## 4.2 System Optimization Experiments and Results

In order to make the TradeNIC design work more optimised, changes were evaluated on the parameters used, and their effects on the prediction throughput were examined as seen in Figure 9.

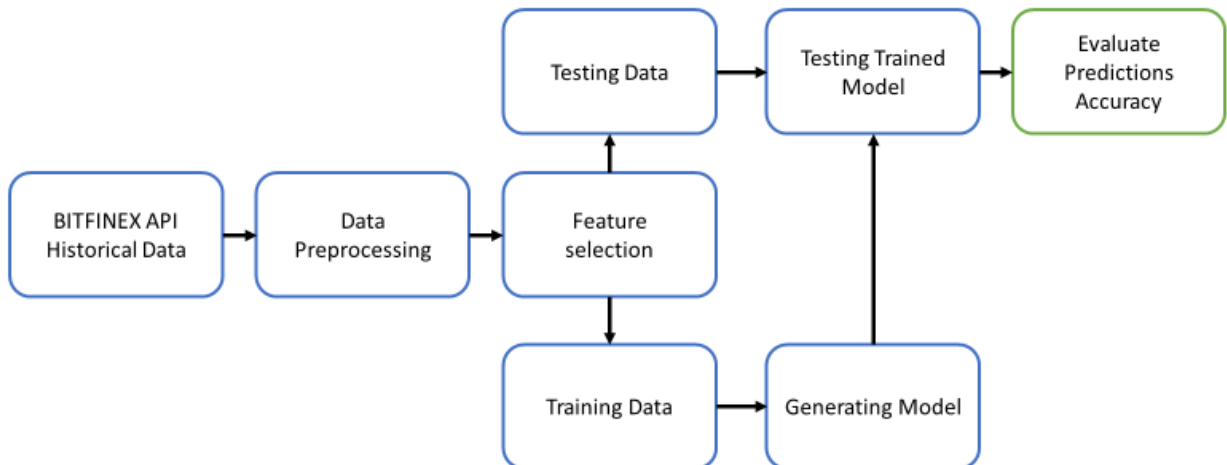


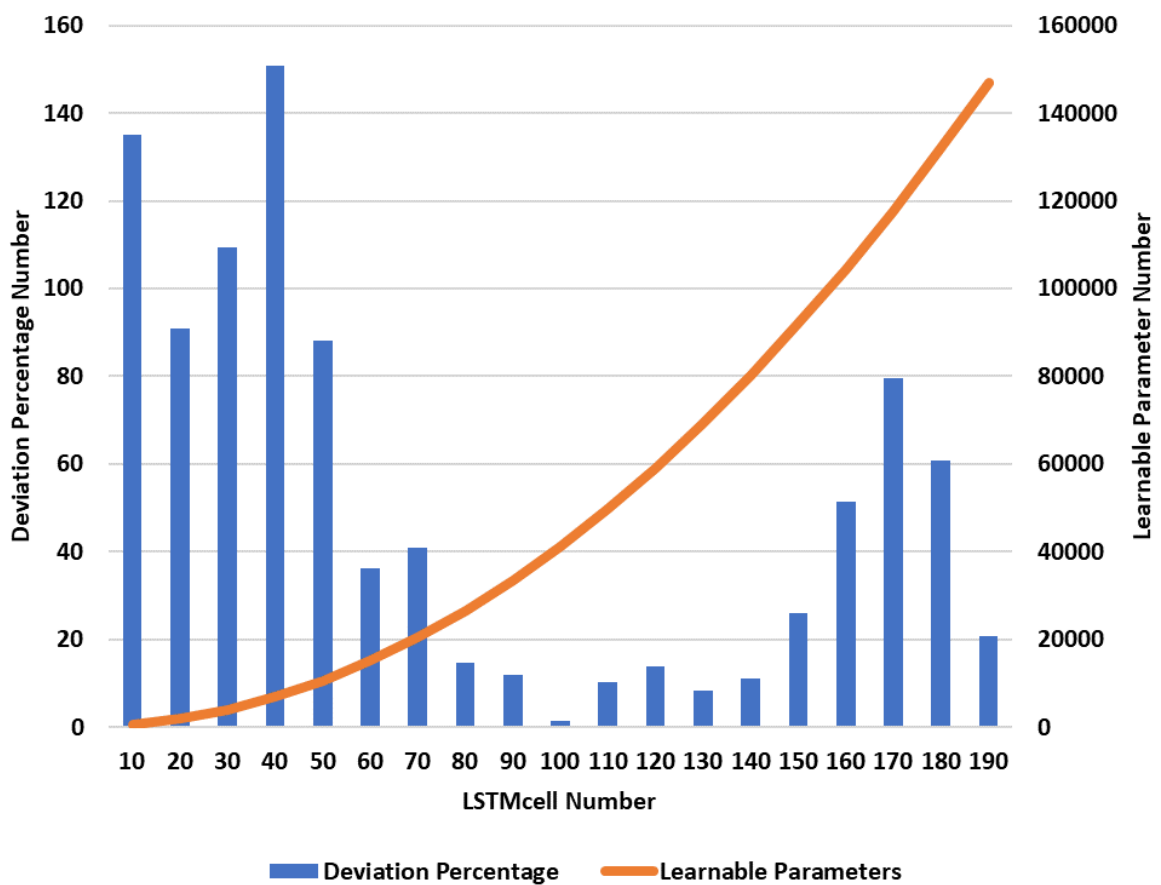
Figure 9 Generating and evaluating a financial model step on Python

### 4.2.1 LSTM Layer Throughput Optimizations

As the system works in the most optimised state, how the neuron and window size parameters in the LSTM layer affect the accuracy of the prediction has been examined. The prediction deviations of the models trained with parameter changes were examined and the results were graphed.

#### 4.2.1.1 LSTMcell Number Effect on Prediction Throughput

New models were trained by changing the number of neurons in the LSTM layer. It was aimed to examine the number of neurons with the least prediction deviation of the model depending on the number of LSTM cells and the resource usage of the model with this experiment. Thus, the efficiency of TradeNIC is increased by creating a model with the number of neurons from which the best values are obtained. A total of 19 models were trained on Python Keras by increasing the number of neurons in the range of 10-190 by ten. In this section the window size is always set to half the number of neurons, the reason for this will be explained by experiments in the next section. Predictions were made using the trained models and the historical data of the AMD coded stock. Then, predicted data and real data were compared by selecting five samples at different intervals. To find the prediction deviation rate during the comparison; The predicted data was subtracted from the real data and the value found was divided by the real data, i.e.  $((\text{Real Data} - \text{Predicted Data}) / (\text{Real Data}))$ . Deviation rate indicates relative error, which is an analysis parameter in statistical science, i.e. rate of deviation from the true value. After calculating the deviation values for each value, they were averaged and recorded as percentage values. In addition, depending on the number of neurons, the number of #params, i.e., learnable parameters, of the model were also included in the analysis. Params value indirectly shows how much hidden state matrix Keras library will generate in each layer during the train and how much resources it uses.

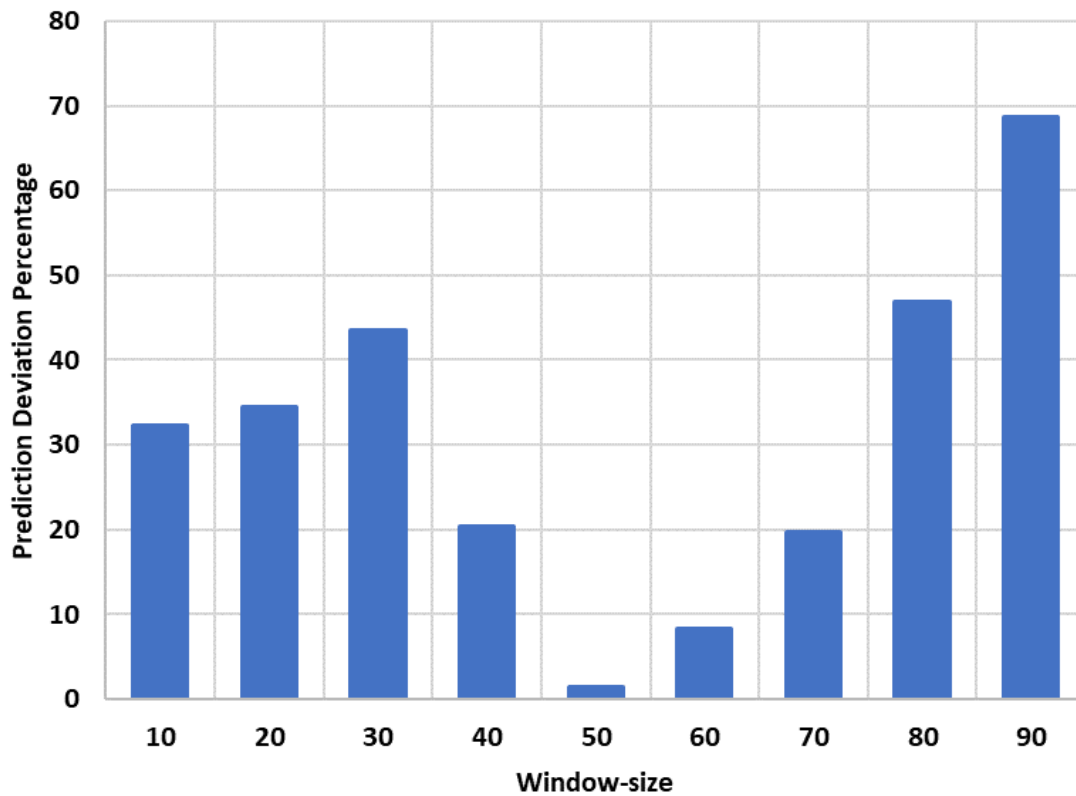


Graph 1 The graph of the learnable parameter numbers and the prediction deviation according to the increasing number of neurons (LSTMcell).

The results of the experiment of examining the effect of the LSTMcell increase made in *graph 1* on the prediction throughput are graphed. The values in the left column are the percentage equivalents of the deviation amount, and the ones in the right column are the learnable parameter values. When the number of neurons is small, the deviation rate can be more than 100%. In addition, even though it is not greater than 100%, the deviation rate is very high in large neuron numbers. The deviation rate was lowest when the model was trained with 100 neurons, which is approximately 1.5%. Although the model makes predictions with less than 10% deviation in the use of such 100-130 neuron values, it has created a higher learnable parameter here. As a result, using 100 neurons for maximum throughput makes for TradeNIC to work more optimised and efficiently with higher throughputs.

#### 4.2.1.2 Window-size Effect on Prediction Throughput

In this section, as in the experiment in which LSTMcell changes were examined, it was observed how much of a deviation the changes in the window-size parameter would cause in the prediction values. This time, the number of neurons was kept constant at 100. Since there was no change in the number of neurons, the model always used the same learnable parameters. Therefore, only the window-size and prediction deviation relationship were examined while making comparisons. In this experiment, in which a total of 9 models were created, window-sizes were used at the rate of 10-90% of the number of neurons by increasing them by ten. Thus, it was aimed to find the most ideal window-size value compared to the number of neurons. As in the previous experiment, five samples were selected at different intervals and the throughput deviation in each model was calculated. Then, these mean deviation values as percentages were graphed.



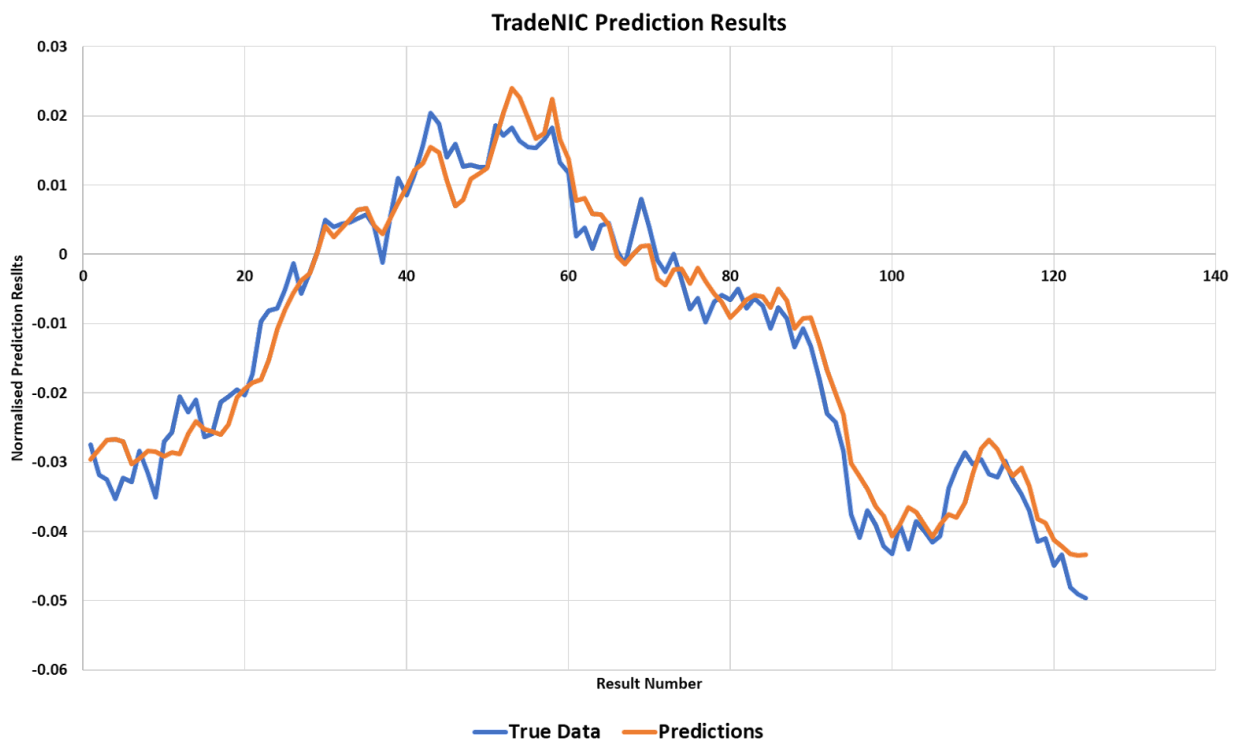
Graph 2 The graph of the prediction deviation according to the increasing number of window-size.



When the experimental results are examined on the *graph 2*, the prediction deviation was at least half the number of neurons when window-size was used, and there was a deviation of approximately 1.5% in this value. Usually, a deviation below 50% is obtained, while the window-size, when the number of neurons rises to 90% or more, the prediction deviation is above this value. As a result of this experiment, it was proved that the optimum value of window-size is half the number of neurons to keep the prediction throughput at maximum.

#### 4.2.2 Prediction Throughput Test Perform on TradeNIC

The prediction throughput of the system was tested using historical data after the RNN-based LSTM algorithm was implemented into the SoC architecture. Thus, it was aimed to evaluate the predictions on hardware design and to improve the throughput deviations that may occur before using TradeNIC as a real-time system. The weight and bias parameters obtained from the model trained on Python Keras were used. The IP Cores of the LSTM and Dense layers were both operated at 100 MHz on the PL. The prediction throughput test was performed with the remaining 700 data while 85% of the total 4700 historical BTC-USD data received was used for the train. The historical cryptocurrency data and the trained model were transferred to SoC via UDP-LwIP. The prediction results and the corresponding real data were compared on the graph.



*Graph 3 Comparison of true BTC-USD data and predicted data by using RNN-LSTM Deep Learning method with TradeNIC design.*

As a result of the experiment, the predicted and real data of the cryptocurrency BTC-USD are given in *graph 3* as normalised. Shown in the orange line are the predictions made with the LSTM hardware accelerator, and in the blue line are the correct historical data. A



total of 124 pricing forecasts made at 1-minute intervals are compared on the line graph. As can be seen, the prediction values mostly follow the true data closely. Also, it caught the rising and falling trends of the BTC-USD values. The values are more stable and closer to the real values because of the throughput optimizations which have been designed.

The system was studied by using real-time data at the next stage after observing that TradeNIC made correct predictions. In the experiment conducted on June 27, predictions were made using the open and close values of real-time BTC-USD. The model was trained using 4000 data, i.e. about 3 days of pricing changes, before the prediction process was launched, and this model was sent to the SoC. Real-time data was sent first with 50 window-sizes and predictions were made through the trained model. It was denormalized and sent to the trader as output after 1 minute of data was predicted.

Date-Time	Real Close Value	Predicted Close Value	Deviation Rate (‰)
⋮	⋮	⋮	⋮
27/06/2022 16:09	21244.77	21259.20703 ↑	⋮
27/06/2022 16:10	21251	21263.86133 ↑	0.386195061
27/06/2022 16:11	21276	21254.66211 ↓	0.570533553
27/06/2022 16:12	21248	21243.73828 ↓	0.313540539
27/06/2022 16:13	21221.61	21246.63281 ↑	1.042723962
27/06/2022 16:14	21234.3	21249.36914 ↑	0.580796753
27/06/2022 16:15	21242	21238.82617 ↓	0.346913691
27/06/2022 16:16	21234	21240.15625 ↑	0.69822793
27/06/2022 16:17	21250.09	21244.55078 ↓	0.467468608
27/06/2022 16:18	21221.89	21233.17383 ↑	1.06780222
⋮	⋮	⋮	⋮

Table 1 The comparison of real-time and predicted prices of BTC-USD cryptocurrency

Some of the predicted data as a result of the experiment is shown in the table in **Table 1**. The real and predicted close parameters of BTC-USD are compared in the table. Predicted close values show the data which is prediction of the actual value after 1 minute, and this is also indicated by blue arrows. The deviation rate values are shown as permillage and are often less than 1‰. When the rise and fall intervals of the real values are examined in the table; The green upward arrows which are next to predicted values indicate that BTC-USD will rise after 1 minute, and red downward arrows indicate that it will fall after 1 minute.

#### 4.2.3 Prediction Latency Optimization Experiment Results on TradeNIC

RNN-based LSTM algorithm accelerated on hardware with High Level Synthesis tools. Also, the loops within the algorithm were run with low latency by using pipeline pragmas. The algorithm was first run on the processing system and then was run on programmable logic with HW/SW co-design at the Zynq-7000 SoC. During this experiment, the ARM Cortex A9 processor was used with a single core and the CPU frequency was 666.67 MHz. The operating frequency of the hardware called PL Fabric Clock was 100 MHz. The

execution times for one prediction were measured for the LSTM layer, the Dense layer, and the whole algorithm using both layers. Also, execution times are calculated by averaging 50 predictions.

Execution Time for One Prediction(us)	Zynq-7000 ARM Cortex A9	TradeNIC Design	Speed-Up Rate
LSTM Layer	123295.1	21187.5	x5.82
Dense Layer	9.561	6.457	x1.48
RNN-based LSTM	123304.7	21221.05	x5.81

Table 2 Acceleration rates of LSTM Layer, Dense Layer and RNN-LSTM network for one prediction

The test results are shown in *table 2*. The entire algorithm spent about 123.3 milliseconds on the processor of the SoC to make a prediction by using 50 window-size. After the hardware acceleration was applied, the prediction time accelerated by x5.81 times and decreased to 21.22 milliseconds. The execution time of the dense layer decreased from about 9.5 up to 6.5 us. The reason for the runtime and acceleration rate is less than the LSTM layer; it does not depend on window-size and does not have many loops with heavy computations. The LSTM layer takes up most execution time of the entire algorithm. The prediction time on the LSTM layer was accelerated by x5.82 times.

### 4.3 Resource Utilisation

*Table 3* shows the use of FPGA resources in the implementation of TradeNIC on hardware. The Zynq-7000 SoC card has 140 BRAMs for storing large amounts of data, 106400 memory element Flip-Flops(FF), 200 digital signal processor (DSP) and 53200 Look-up Table (LUT) which is a real table that produces an output based on inputs. Dense layer uses very little of the resources and this is because it does not contain many loops and complex computations. In the LSTM layer, loading of trained model parameters, application of normalization techniques, and computational intensity with a gating mechanism are made. Therefore, the use of resources is quite high. It is seen that the resource capacity is suitable for the improvement of the prediction process as considering the current resource consumption.

Resource	Available	Dense Layer	Utilization (%)	LSTM Layer	Utilization (%)
LUT	53200	628	1.18	19550	36.75
FF	106400	568	0.54	27165	25.53
BRAM	140	2	1.43	74.5	53.21
DSP	220	5	2.27	100	45.46

Table 3 Resource utilization table of LSTM and Dense layer IP Cores

#### 4.4 LSTM Performance Comparison i7 7th processor vs TradeNIC

In this experiment, application-specific reconfigurable digital hardware design and the CPU on the host machine were compared for the trading prediction process. Intel i7 7700HQ processor with 3.8 GHz core speed and Artix-7 based Programmable Logic running with 100 MHz frequency were evaluated under the title of execution time and prediction deviation rate. The CPU experiment was performed on the Virtual Machine with the processor performance at the highest level. It has been ensured that it affects the prediction performance at a minimum level by restricting background operations. TradeNIC is working on SoC architecture with HW/SW co-design. The prediction deviation and execution time criterions are calculated by taking the average of approximately 100 values.



Execution Time (msec)		Average Deviation(‰)	
Intel i7 7700HQ	6.525	Intel i7 7700HQ	~15
x3.25 		x21.19 	
TradeNIC	21.187	TradeNIC	0.708

Table 4 The comparison of prediction time between Intel i7 processor and TradeNIC

In *table 4*, prediction time and average deviation rates between Intel i7 processor and TradeNIC design are shown. The i7 processor averagely spends 6.525 milliseconds for a prediction. TradeNIC design is only x3.25 times slower with 21,187 milliseconds. On the other hand, when looking at the prediction throughputs, the processor has an average deviation rate of about 15‰. This value is x21.19 times higher than the hardware design. Although the i7 processor gives better prediction time, it is a very powerful processor and is very expensive in terms of cost. The clock cycle interval is lower because they operate at high frequencies. However, they make predictions in a shorter time with high processing power, the accuracy of prediction remains quite low compared to TradeNIC. If the TradeNIC design is made by using much more powerful FPGAs, the prediction time and throughput will be improved. In addition, it allows improving and increasing prediction techniques because it provides more resource usage and higher data transmission speed.

## 5. Conclusion and Future Work

In this interdisciplinary work, we presented TradeNIC, a trading specific SmartNIC co-design. TradeNIC uses RNN-LSTM network instead of using conventional indicators to reach higher throughputs on the system. Also, TradeNIC performs in a low-latency way and low power consumption thanks to reconfigurable hardware design of SoC architecture.

SmartNICs allows multiple data processing on programmable logic by providing simultaneous multiple network-based communications. The traffic on the data centres has made SmartNICs more popular for distributing the application-specific workloads from CPUs. Cloud providers are offering custom SmartNIC designs for application-specific solutions to cloud environments (AWS Nitro or Azure SmartNIC). Many vendors are

investing in the SmartNIC ecosystem, such as Xilinx (AMD), Mellanox (Nvidia) etc. Our approach presented in this project, by using the TradeNIC like designs in data centres or edge devices will help cloud integration and customers to perform compute intensive trading specific operations in terms of latency, throughput, and power consumption.

Using RNN-LSTM model with Hw/Sw co-design and software only application, each experiment was run on both platforms and the execution times were recorded. Comparing the result of experiments, Hw/Sw co-design provides better results in terms of throughput and power consumption. The hardware acceleration has achieved a success of x5.81 times when the algorithm is compared with bare-metal and programmable logic on the SoC. Since the new generation processors are more powerful than the Zynq 7000 SoC hence, Intel i7 processor was only x3.25 times better than TradeNIC in terms of latency. If we consider budget and development time, designing an reconfigurable application specific hardware accelerator on bare-metal systems is more advantageous. The throughput deviation is approximately x21.2 times less by the hardware design. The SoC design with low cost and high prediction throughput can show much lower latency thanks to more powerful FPGAs to be used. Also, these powerful processors have to run on a machine and therefore use a lot of power, but TradeNIC runs in low power mode on FPGA based SoC architecture. Therefore, better performances can be achieved with SmartNIC systems in low cost and latency concept instead of conventional high cost CPU-based systems. In addition, it was observed that the RNN-based LSTM algorithm reached the maximum throughput for TradeNIC with 100 neurons and 50 window-size parameters as a result of the experiments which are made to increase prediction throughput.

Future work that can be developed with the TradeNIC design:

- TradeNIC like designs can be implemented to data centres so the traders can rent from the cloud instances whenever they want and perform trading operations with the trading bots that run on TradeNIC.
- The TradeNIC concept can be applied to more powerful SoC architectures with multiple different types of forecasting algorithms. We intend to generate more than one forecasting algorithm. Thus, users can easily use which method they want to use. We already design a configurable model loader structure on DDR3 of TradeNIC that allows predicting different types of market with different timeframes.
- Instead of using a trading API from the host machine, it can be embed to TradeNIC design, this way it can directly connects to the trading server and reduces latency on the network transactions to perform faster operations.

## REFERENCES

- [1] Wang, D., Fan, J., Fu, H., & Zhang, B. (2018). Research on optimization of big data construction engineering quality management based on RNN-LSTM. *Complexity*, 2018.
- [2] Siامي-Namini, S., Tavakoli, N., & Namin, A. S. (2019, December). The performance of LSTM and BiLSTM in forecasting time series. In *2019 IEEE International Conference on Big Data (Big Data)* (pp. 3285-3292). IEEE.
- [3] Touzani, Y., & Douzi, K. (2021). An LSTM and GRU based trading strategy adapted to the Moroccan market. *Journal of big Data*, 8(1), 1-16.
- [4] García, S., et al., "Tutorial on practical tips of the most influential data preprocessing algorithms in data mining", *Knowledge-Based Systems*, 98, 1-29, 2016.
- [5] Ioffe, S., & Szegedy, C. (2015, June). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning* (pp. 448-456). PMLR.
- [6] Santurkar, S., Tsipras, D., Ilyas, A., & Madry, A. (2018). How does batch normalization help optimization? *Advances in neural information processing systems*, 31.
- [7] Velusamy, S. (2009). Lightweight ip (lwip) application examples. Application note: Embedded , XA 1026. : [www. \\_ 1026](http://www._1026). "
- [8] Sunny, M. A. I., Maswood, M. M. S., & Alharbi, A. G. (2020, October). Deep learning-based stock price prediction using LSTM and bi-directional LSTM model. In *2020 2nd Novel Intelligent and Leading Emerging Sciences Conference (NILES)* (pp. 87-92). IEEE.
- [9] Azari, E., & Vrudhula, S. (2019, December). An energy-efficient reconfigurable LSTM accelerator for natural language processing. In *2019 IEEE International Conference on Big Data (Big Data)* (pp. 4450-4459). IEEE.
- [10] "Bitfinex finance," accessed: 2021-06-15. [Online]. Available: <https://docs.bitfinex.com/docs>
- [11] Vitis High-Level Synthesis User Guide UG1399 (v2020.2) March 22, 2021 (pp. 448-458; pp. 462-464)