Performance Evaluation of Convolutional Neural Networks with oneAPI and OpenMP for Network Intrusion Detection

Rafael Rodrigues Oliveira, Thiago Augusto Demeto Leão, Henrique Cota de Freitas

Computer Architecture and Parallel Processing Team (CArT)
Department of Computer Science
Pontifícia Universidade Católica de Minas Gerais (PUC Minas)
30.535-901, Belo Horizonte, MG, Brazil

{rafael.oliveira.1404477, tadleao}@sga.pucminas.br, cota@pucminas.br

Abstract. *The accelerated evolution in the Machine Learning (ML) field has made the use of hardware solutions indispensable for training and inferring models. This article presents the performance evaluation of three implementations of a Convolutional Neural Network: a sequential version and two parallel versions, one using Intel oneAPI and the other with OpenMP, all running on the CPU. The comparative analysis is based on a network intrusion dataset (NSL-KDD), focusing on speedup and scalability. It shows that the implementation using Intel® oneAPI achieves the best performance compared to the sequential and OpenMP versions.*

1. Introduction

Recent advances in Machine Learning (ML) have led to an increasingly larger number of layers and parameters for deep neural network models, aiming for better inferences. This led to longer training times for each model, sometimes numbering weeks. With the advent of new systems, such as autonomous cars (Silva et al., 2024; Silva and de Freitas, 2025), which have interconnected devices with different architectures, it is also necessary to integrate them to quickly respond to information received by the system, such as sensor signals.

To reduce the time spent training models and perform this integration, high-performance computing (HPC) is increasingly present in the development of these models, using accelerator devices such as Graphics Processing Units (GPUs) and Field-Programmable Gate Arrays (FPGAs) (Kuon et al., 2008) to achieve both goals. In this context, Intel® oneAPI emerges, an environment developed by Intel® aiming to ease heterogeneous programming for different accelerators and devices using a single codebase. For this same purpose, OpenMP (Chandra et al., 2001) has been used and achieved good performance results. However, to the best of our knowledge, there are no previous works addressing a comparison between them in this specific context.

Thus, this work aims to conduct tests in the oneAPI environment and compare them with the sequential and OpenMP-based parallel versions. To this end, three versions

Rafael and Thiago are undergraduate students in Computer Science at PUC Minas.

The authors would like to thank the National Council for Scientific and Technological Development of Brazil (CNPq - Codes 311697/2022-4 and 402837/2024-0), the Foundation for Research Support of Minas Gerais State (FAPEMIG - Code APQ-05058-23), and PUC Minas FIP 2025/32469.

of a Convolutional Neural Network (CNN) were developed: sequential, OpenMP- and oneAPI-based parallel versions, to evaluate speedup and scalability. The training and testing dataset used was the NSL-KDD (Dhanabal and Shantharajah, 2015), a dataset for network attacks, with approximately one hundred thousand instances and two classes.

This article is structured as follows: Section 2 briefly presents a background about CNNs; Section 3 discusses related works; Section 4 presents the methodology, including materials and development method; Section 5 describes the evaluation of results obtained; and finally, Section 6 shows the conclusions of the work, besides proposed future works.

2. Convolutional Neural Networks

Convolutional Neural Network (CNN) (Li et al., 2022) is a machine learning model initially created to classify visual data. They are a class of deep neural networks, inspired by the organization of the human visual cortex, and are particularly effective for tasks involving images. A CNN architecture is composed of layers such as convolutional and dense layers, often interleaved with activation functions like ReLU. The model goes through two main processes: training (to learn weights) and prediction (inference using the trained model).

2.1. Convolutional Layer

The first learning stage involves passing a 1D kernel through the data, multiplying the input by the weights of the kernel by region of the input, like a sliding window, to extract features from the input and place them in a new matrix with smaller dimensions than the original. In this work, convolution was used in only one dimension, as the inputs were feature vectors and not matrices, as in the case of image classification.

2.2. Dense Layer

The dense layer functions similarly to a Multi Layer Perceptron (Popescu et al., 2009), comprising layers of neurons that receive the output of the convolutional layers for classification and adjust their weights through backpropagation. This algorithm applies the chain rule to calculate the gradient of the loss function and, thus, adjust the weights in the direction that minimizes the error.

2.3. Training Process

The training process of a CNN is a repetition composed of two steps: forward propagation and backward propagation. This repetition is known as the number of epochs and is completed after passing through the entire dataset. To optimize this process, especially when using a GPU, the data is not processed all at once. Instead, the dataset is divided into batches. This approach is essential to minimize the data transfer overhead between the system memory (host) and the GPU memory (device), in addition to taking full advantage of the SIMD (Single Instruction Multiple Data) nature of these devices.

In the forward stage, the input data, grouped into batches, passes through the network layer by layer. It begins at the first convolutional layer. The output is passed through an activation function and then sent to the next layer, which can be another convolutional layer or dense layers. Dense layers perform a weighted sum of their inputs, apply an activation function to introduce nonlinearity, and finally generate the model's prediction.

The backward stage begins when the previous one ends. Firstly, the prediction error is calculated by comparing the model's output with the expected value. This error is then propagated backward. Based on this information, the weights are adjusted so that the error is smaller in the next epoch.

2.4. Prediction Process

After training is complete, the model has a set of optimized parameters that minimize the loss function for the training data. These parameters represent the knowledge gained during training and are now used to make predictions on unknown instances, a process known as inference.

3. Related Work

This section describes other research works related to parallel CNNs and intrusion detection studies.

Maciel et al. (2024) proposes and evaluates an energy-efficient hardware architecture for intrusion detection systems using a Convolutional Neural Network (CNN). The authors implemented it on a heterogeneous platform that combines a CPU and FPGA, aiming to improve performance and energy efficiency in security applications.

Wang et al. (2023) proposes the application of deep learning models for intrusion detection in networks, exploring architectures such as CNNs, RNNs, LSTMs and hybrid combinations. The implementation was developed in Python with TensorFlow library, using the CSE-CIC-IDS2018 dataset, with GPU-accelerated training and inference processes.

Peng et al. (2019) proposes the use of a Convolutional Neural Network (CNN) for the detection of intrusion in vehicle networks, achieving excellent accuracy results and low latency times in attacks. The model was trained using a GPU. The programming language chosen for development was Python, along with the TensorFlow library and its high-level API, Keras.

Ioannou and Fahmy (2019) demonstrates a high-performance network intrusion detection system by implementing a CNN on an FPGA SoC. The model training was done in software using Python with TensorFlow (on CPU/GPU), while real-time inference was accelerated using an FPGA.

The main difference between this study and the existing literature, as summarized in Table 1, lies in the programming tools and the focus on optimization. While most related work utilizes specialized hardware, such as FPGAs and GPUs, to accelerate intrusion detection, this research focuses on optimizing performance in multi-core CPUs. For this, two distinct parallelization approaches were analyzed and compared: an implementation based on the OpenMP standard and another using the oneAPI ecosystem. This comparative analysis seeks to determine the most efficient approach for this class of hardware.

Table 1. Overview of related work

	Language	OpenMP	oneAPI	OpenCL	CPU	GPU	FPGA
Maciel et al. (2024)	C++	X		X	X		X
Wang et al. (2023)	Python				X	X	
Peng et al. (2019)	Python				X	X	
Ioannou and Fahmy (2019)	Python				X	X	X
This work	C++	X	X		X		

4. Methodology

This section presents the materials and method used to design and evaluate our proposal, besides design decision related to CNN versions.

4.1. Materials

The environment used in the development and test is present in Table 2:

Table 2. Environment

Technical Specifications	
OS	Windows 10
CPU	AMD Ryzen 7 5700X 3.4GHz 8-Cores 16-Threads
GPU	NVIDIA GeForce RTX 3060 12GB
RAM	32 GB DDR4

The database used was NSL-KDD, with, after preprocessing (Álvarez Almeida and Carlos Martinez Santos, 2019), 33 columns, 102,350 training instances, and 10,369 test instances, where each record represents a server-client connection labeled as attack or normal. This database is an improvement of a previous database, KDD99 (Tavallaee et al., 2009), having a record balance between classes and no duplicate records.

4.2. Developed code

A sequential CNN algorithm was developed using the C++ language as a baseline. The next step was to produce two parallel versions[1] using the oneAPI SYCL Khronos Group environment and OpenMP, parallelizing the batch execution steps, layer forwarding, and weights adjustment per layer in backward, respecting the existing dependencies between these steps.

It is worth noting that because the graphics card used in the experiments is an NVIDIA®, which is not natively supported by oneAPI, it was necessary to use a plugin(Codeplay Software Ltd. / Intel) to enable execution on these devices. However, this plugin does not offer the same level of integration and optimization available for hardware that is natively supported by the oneAPI environment.

4.3. Trained Models

Table 3 describes the layers of each trained network. The goal is to identify the factors that influence the time reduction between code versions. The networks were trained three times in fifty epochs to collect the average time for each, with 400 instances per batch. The standard deviation is below 3.59%, considering all results presented in Section 5.

[1]Code available at: https://github.com/cart-pucminas/parallel-ML

Table 3. Network Architectures

Model	Convolutional Layers (1D)	Dense Layers (Neurons per layer)
1	1×5	$20 \to 20 \to 20 \to 20 \to 20 \to 2$
2	$1 \times 5 \to 1 \times 5$	$45 \to 20 \to 2$
3	1×5	$45 \to 20 \to 2$
4	1×5	$45 \to 45 \to 20 \to 2$
5	1×5	$100 \to 100 \to 100 \to 20 \to 2$

5. Results

In this section, the best results in speedup and scalability will be presented, comparing oneAPI with OpenMP and sequential codes. Tests were conducted using the same random seed to ensure replicability and the accuracy of each model for the dataset test batches, as shown in Table 4.

Model	Accuracy (%)
1	88.31
2	89.31
3	89.26
4	91.73
5	92.4

Table 4. Accuracy per Model

5.1. Speedup

Figures 1, 2, 3, 4 and 5 show each model speedup. The models with the highest speedup for OpenMP and oneAPI compared to the sequential approach were models 1, 2, and 5.

Model 2 indicates that convolutions are an impactful factor, as they are essentially matrix multiplications, which is a highly parallelizable operation, and are expected to show better results for larger inputs.

Models 1 and 5 indicates that the number of layers and neurons per layer are two other factors that influence speedup, since the neural network's forward and backward operations depend on the previous and subsequent layers, respectively, and are only parallelizable within themselves, justifying parallelization only in cases of a large number of neurons per layer.

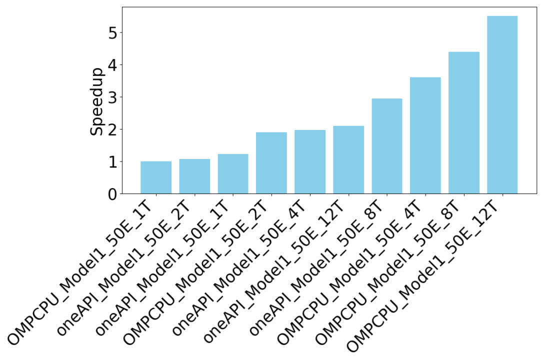

Figure 1. Model 1 Speedup

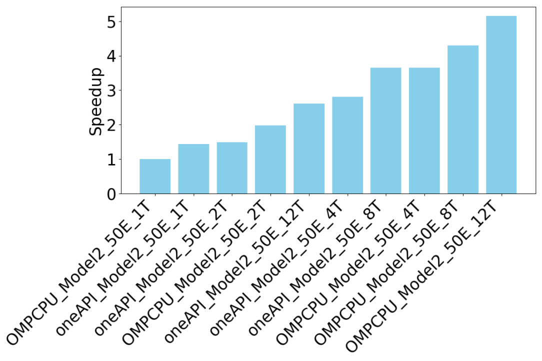

Figure 2. Model 2 Speedup

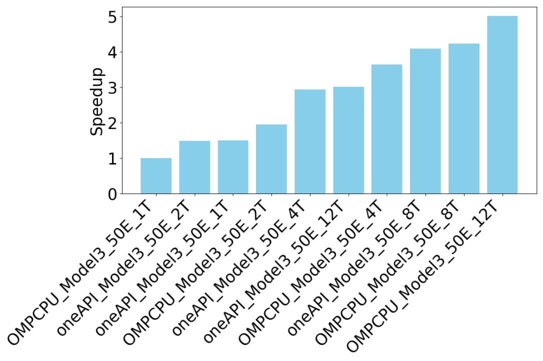

Figure 3. Model 3 Speedup

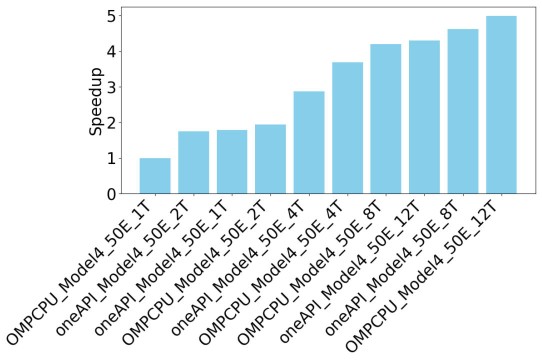

Figure 4. Model 4 Speedup

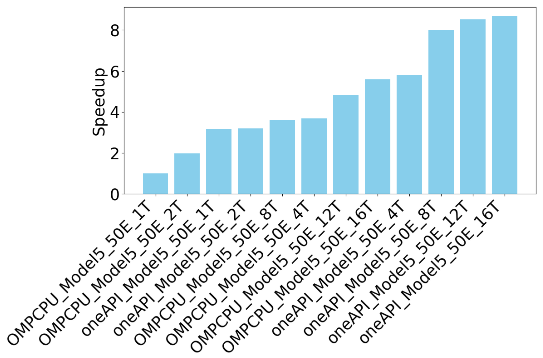

Figure 5. Model 5 Speedup

5.2. Scalability test

Scalability tests were performed varying the number of threads between configurations of 1 (sequential), 2, 4, 8, 12, and 16. Figure 6 illustrates the graph obtained for Model 5 as an example of the code's behavior at different thread counts. It shows that the oneAPI version scaled well up to eight threads, while being faster than the OpenMP one at any count and 42% faster than OpenMP's 16-thread version, despite the latter showing better scalability results for twelve and sixteen threads.

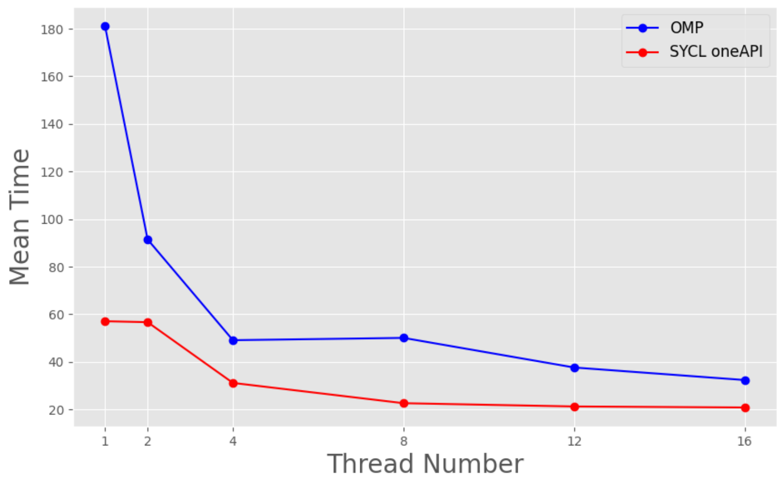

Figure 6. Model 5 Training Times

6. Conclusion

This work presented a comparative performance analysis of three distinct implementations of a Convolutional Neural Network (CNN) for network intrusion detection, using the NSL-KDD database. A sequential version in C++, a parallel version with OpenMP, and another with Intel's SYCL implementation were developed, evaluated, and compared. The objective was to calculate the performance gain and strong scalability in heterogeneous hardware environments, specifically CPU. The research focused on how different network architectures and the choice of parallelization technology influence training and inference times, which are indispensable factors in this field.

The overall results show an advantage for OpenMP parallelism in the smaller models. However, as more layers with more neurons were added, SYCL's oneAPI began to show far better results in the CPU context. The expected results for the GPU environment were not achieved due to memory management issues between devices, which prevented it from outperforming the oneAPI best CPU tests in this experiment. In the best-case scenario, it stood at the same timescale as the oneAPI CPU for Model 5 at 8 threads.

As future work, extending this study to GPUs and FPGAs is suggested, leveraging the oneAPI ecosystem's support for these kinds of architectures. This would enable a more comprehensive analysis of code portability and the efficiency of this technology across a new hardware category.

References

R. Chandra, L. Dagum, D. Kohr, R. Menon, D. Maydan, and J. McDonald. *Parallel programming in OpenMP*. Morgan kaufmann, 2001.

Codeplay Software Ltd. / Intel. oneAPI for NVIDIA® GPUs. `https://developer.codeplay.com/products/oneapi/nvidia/home`. Accessed in August 20, 2025.

L. Dhanabal and S. Shantharajah. A study on nsl-kdd dataset for intrusion detection system based on classification algorithms. *International journal of advanced research in computer and communication engineering*, 4(6):446–452, 2015.

L. Ioannou and S. A. Fahmy. Network intrusion detection using neural networks on fpga socs. In *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, pages 232–238, 2019. doi: 10.1109/FPL.2019.00043.

Khronos Group. Sycl overview. `https://www.khronos.org/sycl/`. Accessed in August 21, 2025.

I. Kuon, R. Tessier, J. Rose, et al. Fpga architecture: Survey and challenges. *Foundations and Trends® in Electronic Design Automation*, 2(2):135–253, 2008.

Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou. A survey of convolutional neural networks: Analysis, applications, and prospects. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

L. A. Maciel, M. A. Souza, and H. C. Freitas. Energy-efficient cpu+fpga-based cnn architecture for intrusion detection systems. *IEEE Consumer Electronics Magazine*, 13 (4):65–72, 2024. doi: 10.1109/MCE.2023.3283730.

R. Peng, W. Li, T. Yang, and K. Huafeng. An internet of vehicles intrusion detection system based on a convolutional neural network. In *2019 IEEE Intl Conf on Parallel Distributed Processing with Applications, Big Data Cloud Computing, Sustainable Computing Communications, Social Computing Networking (ISPA/BDCloud/SocialCom/SustainCom)*, pages 1595–1599, 2019. doi: 10.1109/ISPA-BDCloud-SustainCom-SocialCom48970.2019.00234.

M.-C. Popescu, V. E. Balas, L. Perescu-Popescu, and N. Mastorakis. Multilayer perceptron and neural networks. *WSEAS Transactions on Circuits and Systems*, 8(7): 579–588, 2009.

E. Silva, F. Soares, W. Júnio de Souza, and H. Freitas. A systematic mapping of autonomous vehicle prototypes: Trends and opportunities. *IEEE Transactions on Intelligent Vehicles*, 9(11):6777–6802, 2024. doi: 10.1109/TIV.2024.3387394.

E. M. R. Silva and H. C. de Freitas. Task scheduling for autonomous vehicles with heterogeneous processing via integration of the carla simulator and starpu runtime. In *2025 IEEE International Conference on Consumer Electronics (ICCE)*, pages 1–6, 2025. doi: 10.1109/ICCE63647.2025.10929988.

M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani. A detailed analysis of the kdd cup 99 data set. In *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, pages 1–6, 2009. doi: 10.1109/CISDA.2009.5356528.

Y.-C. Wang, Y.-C. Houng, H.-X. Chen, and S.-M. Tseng. Network anomaly intrusion detection based on deep learning approach. *Sensors*, 23(4), 2023. ISSN 1424-8220. doi: 10.3390/s23042171. URL `https://www.mdpi.com/1424-8220/23/4/2171`.

L. A. Álvarez Almeida and J. Carlos Martinez Santos. Evaluating features selection on nsl-kdd data-set to train a support vector machine-based intrusion detection system. In *2019 IEEE Colombian Conference on Applications in Computational Intelligence (ColCACI)*, pages 1–5, 2019. doi: 10.1109/ColCACI.2019.8781803.