Machine Learning-based Deep Packet Inspection at Line Rate for RDMA on FPGAs

Maximilian J. Heer*
maximilian.heer@inf.ethz.ch
System Group, ETH Zurich
Zurich, Switzerland

Benjamin Ramhorst*
benjamin.ramhorst@inf.ethz.ch
System Group, ETH Zurich
Zurich, Switzerland

Gustavo Alonso
alonso@inf.ethz.ch
System Group, ETH Zurich
Zurich, Switzerland

Abstract

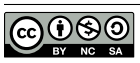
FPGAs are becoming increasingly important in the cloud and data centers, especially as network-attached accelerators or reconfigurable Network Interface Cards (NICs). In the cloud, Remote Direct Memory Access (RDMA) over Converged Ethernet (RoCEv2) has emerged as the de facto standard protocol for data transport due to its low latency and high throughput. However, RDMA has several access control weaknesses limiting its applicability in the cloud. In this paper, we explore using machine learning-based deep packet inspection (DPI) as an enhancement to an open-source FPGA RDMA stack. The ultra low-latency ML model is integrated on the RDMA datapath and allows for detection of specific content in RDMA payloads (e.g., executables) at a line rate of 100Gbps while using less than 1% of the available resources. Compared with existing work, our solution operates on the full message payload, at the transport level, and on a complete RDMA stack without sacrificing compatibility with RoCEv2 and its native performance characteristics, proving its potential as an end-to-end solution.

CCS Concepts: • **Hardware** → **Hardware accelerators**; • **Networks** → **Network management**; **Cloud computing**.

Keywords: FPGA, Remote Direct Memory Access (RDMA), Deep Packet Inspection, Machine Learning

ACM Reference Format:
Maximilian J. Heer, Benjamin Ramhorst, and Gustavo Alonso. 2025. Machine Learning-based Deep Packet Inspection at Line Rate for RDMA on FPGAs. In *The 5th Workshop on Machine Learning and Systems (EuroMLSys '25), March 30-April 3, 2025, Rotterdam, Netherlands.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3721146.3721935

*Both authors contributed equally to this research.



EuroMLSys '25, Rotterdam, Netherlands

ACM ISBN 979-8-4007-1538-9/2025/03
https://doi.org/10.1145/3721146.3721935

1 Introduction

Modern data centers rely heavily on Remote Direct Memory Access (RDMA) over Converged Ethernet (RoCEv2) for storage services and data transport with RDMA accounting for up to 70% of all traffic in a typical cloud network [3]. This is also reflected in emerging implementations of RDMA on FPGAs [38, 56, 35]. The key advantages of RDMA over classic networking protocols (TCP/IP and UDP) are low latency and high throughput which are achieved through kernel-bypass, zero-copy, and polling [30]. However, these techniques bypass control mechanisms of the operating system (OS) and therefore pose serious security concerns in the public cloud. The RoCEv2-inherent access control mechanisms are incapable of preventing side-channel attacks to established communications, and due to the inherent nature of the protocol, a hijacked connection exposes a strong attack vector through direct memory access [32]. One proposed direction for future research to harden RDMA-based cloud systems against malicious attacks is the enhancement of RDMA NICs with hardware-based security mechanisms, therefore enforcing strict access control rules while maintaining the RDMA-specific performance advantages [39].

In this paper, we explore the use of FPGA-based SmartNICs in the context of RDMA access control through on-datapath deep packet inspection (DPI). Besides traditional application acceleration [22, 17, 7, 14, 18], FPGAs are often utilized as SmartNICs for on-data-path acceleration and offloading of network virtualization functions [13]. This makes them a suitable platform to explore offloading functionality to the NIC. In this paper, we evaluate the possibility of leveraging deep learning and symbolic regression techniques to implement Deep Packet Inspection (DPI) capable of enforcing strict RDMA access control based on allowed and prohibited payloads. We show that DPI inspection at line rate can be practically achieved in a high-throughput data center environment. The contributions of the paper are:

- Exploring the use of deep learning and symbolic regression techniques for packet inspection as a way to extend the checks to full message payloads, instead of just headers. Most previous works proposed rule-based approaches [53] or ML-based DPI on aggregated header information [21].
- Utilization of compression and approximation techniques enabling 100G line-rate packet inspection. Our

models perform inference in less than 50ns and use less than 1% of the available resources, while achieving an accuracy close to 90% and also being able to generalize to previously unseen packet types.
- We extend an open-source, fully RoCEv2-compatible, 100Gbps FPGA stack with DPI functionality without compromising neither performance nor functionality and evaluate our contributions on an end-to-end system with Alveo FPGAs and Mellanox NICs connected via a 100G switched network.

2 Background and Related Work

2.1 RDMA

RDMA can directly access the memory of a remote host through either two-sided synchronous (`RDMA SEND` and `RDMA RECEIVE`) or one-sided asynchronous operations (`RDMA WRITE` and `RDMA READ`). In the latter case, the ACKing of RDMA operations and subsequent memory accesses are entirely handled by the remote-side NIC, underlining the key feature of RDMA: host-bypassing with zero-copy, since incoming traffic is not handled by the receiver-side OS [38]. RDMA was originally intended for high performance computing (HPC) [24], which has vastly different trust and security assumptions from the public cloud [29]. Therefore, porting of RDMA to the public cloud introduced various concerns, mostly centered around three main vulnerabilities:

- Host-bypassing includes bypassing of all host-enabled security and access control mechanisms residing in the OS [30].
- The RDMA-standard lacks encryption and cryptographic authentication, exposing both header- and payload-included information to side-channel attacks [39].
- The security functions originally included in RDMA such as memory protection domains and sequence number checks have been proven to be insufficient, making it relatively easy to hijack RDMA connections [32, 42] and exploit system weaknesses in memory management [44, 47].

Different solutions and partial improvements to these problems have been proposed, including changes to the control flow of RDMA [30] and offloading of encryption to SmartNICs and switches [43, 52].

2.2 Network Access Control and Deep Packet Inspection

A central piece of security in the public cloud are network access control rules [4] that seek to prevent malware injection [36] and DDOS attacks [51]. Access control is especially important for RDMA, due to its bypassing of OS control mechanisms, which are normally used for enforcing network rules. For example, well-known *shellcode* injection attacks [31, 41] hide malicious executables in network payloads and utilize memory vulnerabilities, which have been extensively described for RDMA [45], to gather control over the entire system. Deep Packet Inspection (DPI) is one possible solution for network access control. Opposed to traditional packet inspection, DPI evaluates the payload of incoming packets, instead of the headers or aggregated information, imposing additional complexity due to the potential irregularity of payload content compared to structured headers. Due to the strict performance requirements, FPGAs have been considered for DPI due to their suitability for stream computation [28]. Multiple FPGA-approaches have been discussed, including the aforementioned nondeterministic finite automata [5], Bloom filters for efficient search in incoming traffic [34], content-addressable memory (CAM) for expression matching [55, 57], and bitmaps for hashing [2].

The use of ML for DPI has been focused mostly on packet classification [54] in software frameworks, and only rarely on reconfigurable hardware. Presented systems for ML-based detection of malware in payloads [6, 1] stress the general importance of the topic, but apply computationally heavy convolutional neural networks (CNNs) on the host CPU for TCP traffic. However, the very nature of OS bypassing in RDMA limits the applicability of these models in a real system. The complexity of deploying intrusion detection in a full networked system is further underlined by previous work on ML-based classification on FPGAs [49] that operates on derived and aggregated traffic features and has no integration with an actual networking stack. To the best of our knowledge, ML on FPGAs for low-latency and high-throughput DPI, fully integrated with an RDMA stack, has not been studied thus far. This combination of urgency and performance-criticality is exactly the reason why offloaded DPI as a form of access control is an interesting research topic for RDMA out of all the networking protocols.

3 Architectural Implementation

3.1 Design overview

The general task of DPI can be summarized in four key steps:

- Input extraction, separating the packet payload and its unique identifier from the accompanying header. A single packet is received sequentially in multiple chunks of equal width, dictated by the network bus width. The maximum packet size is dictated by the maximum transmission unit (MTU). Any messages larger than the MTU are transmitted over several packets.
- Sequential inference of the ML-based DPI on the payload chunks which is executed in parallel to the network datapath, as demonstrated in Fig. 1.
- Aggregation of the individual decisions and forwarding to the networking stack, issuing an ACK or NAK.

The networking stack implements standard header processing functions, cyclic redundancy checks (CRC) etc. on the received packet. Towards the end of this packet processing

pipeline, the acceptability of the packet has to be communicated to the sender via an ACK or NAK. With the addition of DPI, an ACK is only issued if the packet successfully passed through the packet processing pipeline and has not been flagged by the DPI module. Since the DPI decision and packet processing pipeline only need to be merged at the end of the processing pipeline, the inference latency of the ML-based DPI module can be completely hidden by the packet processing pipeline. This latency budget allows us to achieve line-rate DPI without interfering with the processing flow of the baseline networking stack.

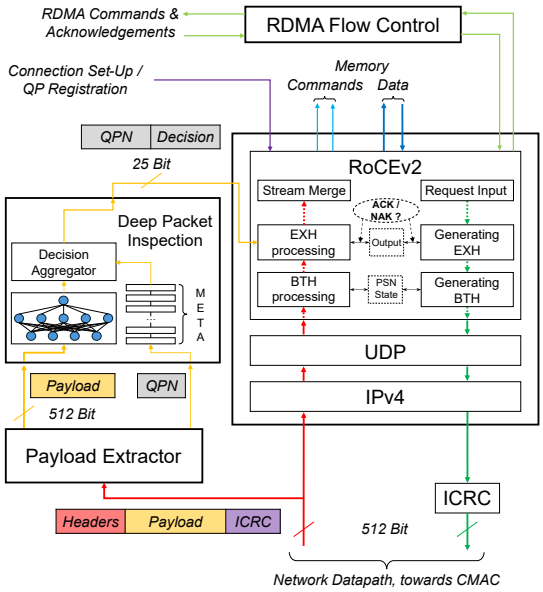


Figure 1. Architecture overview of the RDMA stack with DPI on a parallel datapath.

3.2 Integration of DPI with RDMA

Our baseline RDMA stack[1], which was extended with our DPI (Fig. 1), is included in the open-source FPGA shell, Coyote[2] [20]. The whole shell is built around 512-bit wide AXI4-stream busses, capable of maintaining the 100 Gbps bandwidth. Therefore, each packet is delivered in 512 bits chunks, which also corresponds to the input dimensionality of our ML models. Our DPI module is added as a side-channel to the network processing pipeline (Fig. 1): incoming packets on the datapath are consumed by a Payload Extractor, isolating the packet payload from its headers and the checksum, and forwarding it to the DPI module. Alongside the payload, the extracted Queue Pair Number (QPN) of the incoming packets is forwarded to the Decision Aggregator as a unique identifier of each RDMA connection, thus matching the DPI decision to the respective packet and connection. There, the DPI decisions are aggregated over the full length of the current packet: as soon as one chunk of the packet gets flagged as potentially malicious, the whole packet is also considered malicious. However, our Decision Aggregator is modular and easily extendable to only reject a packet after a certain threshold of its chunks has been marked as malicious, minimizing false positives. Finally, the QPN and the aggregated decision are forwarded to the Extended Header-processing step, where the final decision is issued: (i) an ACK, writing the transmitted payload into host memory or (ii) a NAK, discarding the potentially malicious packet.

To maintain 100 Gbps throughput, the side-channeled DPI module must not block the processing of packets in the network pipeline. The latency for the aggregated DPI decision can be determined from the end-to-end latency of the ML model l_{ML}, the initiation interval of this model ii_{ML}, the general overhead latency, l_{OH}, incurred by the Payload Extractor and Decision Aggregator, the network bus width BW and the MTU. On the receiving FPGA, each clock cycle BW of bytes are available. Then, the maximum number of chunks in a single packet, and therefore, the maximum amount of single ML decisions that need to be aggregated is equal to $\frac{MTU}{BW}$. Therefore, the latency of the DPI module is given by:

$$l_{DPI} = l_{ML} + l_{OH} + ii_{ML} \cdot (\frac{MTU}{BW} - 1) \quad (1)$$

The packet processing pipeline in Coyote takes 44 clock cycles (cc), with $MTU = 4096$ and $BW = 64$. As indicated by Equation 1, the initial interval is the most important factor in achieving the target latency and not causing system backpressure.

3.3 Quantized ML-based DPI on FPGAs

To tackle the problem of DPI for remote code injection via RDMA, we consider quantized neural networks, due to their high performance and low area on reconfigurable hardware, as well and symbolic regression, a light-weight ML technique used for fitting analytical equations to a dataset. Symbolic regression can be seen as a generalization of linear regression, using weights, as well as binary and unary operators, such as $x_1 \cdot x_2$, $\texttt{tanh}(\cdot)$, $\texttt{exp}(\cdot)$ etc. We train neural networks and symbolic regression to distinguish between 512-bit binary vectors obtained from the binary representation of executables and other files, such as PDF, CSV, DOCX, JPEG etc. A subset of C and C++ binaries from two sources, *The Programming Homework Dataset For Plagiarism Detection* [25] and *C++ Templates from GitHub* [46], is compiled. For the non-executable files, a subset of files with the following extensions is used:

- CSV, DOCX, PPTX, RTF, SQL, TXT and XLSX files from *Govdocs1* [15]
- PDF files from *CC-PDF* [10]
- JPEG images from ImageNet (*ILSVRC-2012*) [9]

[1]GitHub repository: https://github.com/fpgasystems/fpga-network-stack
[2]GitHub repository: https://github.com/fpgasystems/Coyote

Notably, instead of performing packet inspection on aggregated header information, we perform DPI using models trained on very small binary chunks obtained from the raw binary representation of publicly available files. When choosing the model architecture, the following hardware constraints are taken into consideration:

- Low initiation interval: As explained in Equation 1, the initiation interval is the determining factor of DPI latency. We aim for a initiation interval of 1, to avoid any potential back-pressure and the need to buffer out-standing packets.
- High operating frequency: To achieve the target network throughput of 100Gbps, the entire network stack, including DPI, needs to be clocked at 250MHz.
- Reconfigurability: As network threats and data center hardware are constantly evolving, the hardware implementation of the ML model should be easy to update and replace.

Given these requirements, we select the following constraints for our DPI model. First, to achieve reconfigurability, the models are implemented with `hls4ml` [11, 12], an open-source framework for low-latency neural network inference on FPGAs, supporting both neural networks and symbolic regression. Secondly, ML models operate on a series of matrix-vector multiplications. To achieve the target initiation interval, all the multiplication stages must be able to accept a new input every clock cycle, corresponding to all multiplications being executed in parallel. Finally, we consider small and quantized neural models that can fit into on-chip memory and use logic elements for multiplication. This approach keeps the area and congestion low to achieve the target frequency.

For the neural network we select an architecture with three hidden layers with 32, 64 and 64 units, and quantized ReLU activation. The last layer has one output with the sigmoid activation. The models are trained in `QKeras` [8] by minimizing binary cross-entropy using the Adam [19] optimizer. For symbolic regression, we use SymbolNet [48] to train the model, but contrary to the original work, which minimized mean squared error (MSE), we include an additional output layer with sigmoid activation, which is better suited for binary classification tasks. In both cases, the sigmoid activation produces a probability, which can be used to classify a packet, given a threshold, t. To further minimize resources and latency, the monotonically-increasing, resource-heavy sigmoid can be removed during inference by changing the threshold, t, and performing the comparison with an equivalent threshold, $\hat{t} = \texttt{ln}(\frac{t}{1-t})$. Using the modified, but equivalent, threshold, $\hat{t}$, enables us to avoid computing exponentials and reciprocals that depend on run-time parameters; instead our DPI module needs to only perform a comparison between a fixed threshold and the run-time input. The modified comparison can be executed in one clock cycle using LUT comparators, compared to several clock cycles and DSP blocks required to implement the sigmoid calculation. Finally, symbolic regression relies on a series of unary operators, such as $\texttt{sin}(\cdot)$, $\texttt{exp}(\cdot)$, $\texttt{tanh}(\cdot)$ etc. To minimize the latency, resources and critical path incurred by these operators, look-up tables stored in BRAM can be pre-populated for an expected range of values, producing an approximately correct output in one clock cycle. Assuming a sufficiently large look-up table, the drop in accuracy is negligible. Empirically, we determine that for our use cases the tables should contain between 1,024 and 4,096 elements.

4 Results

4.1 DPI accuracy

For evaluation, we consider a ternary neural network and symbolic regression (SR). Each architecture is trained 10 times, and, since the pool of available non-executable files (PDF, CSV, JPEG etc.) is significantly larger than the pool of compiled executables, a random subset of the data is sampled. This process enables us to not only capture the stochastic nature of training, but also any variance incurred from data variation, which is likely to occur in a data center setting. In a first experiment, we test the ability to distinguish executables from other files (e.g. PDFs, JPEGs, etc.). In total, 200,000 executable and 200,000 non-executable (mixture of PDF, CSV, JPEG etc.) 512-dimensional, binary vectors, obtained from the binary representation of at least 100 files in both categories, are sampled. Any duplicate points are removed before training. Due to the high input dimensionality, such occurrences are rare; however, they do occur for executables: inspecting the compiled code shows a large number of input vectors with all elements equal to zero. We remove these and only keep one of these zero-vectors to avoid skewing the training to one data point. The accuracy results, calculated on a held-out set of 80,000 (20%) points, including average accuracy, false positive (FPR) and false negative (FNR) rate are reported in Table 1.

Table 1. Accuracy of DPI models distinguishing executables from non-executable files over 10 trials.

Model	Accuracy [%]	FPR [%]	FNR [%]
SR	95.31 ± 0.33	5.04 ± 0.53	4.33 ± 0.35
Ternary	97.83 ± 0.16	1.74 ± 0.37	2.59 ± 0.38

In the second experiment, we insert binary chunks found in executables into non-executable files. The chunks can vary in length (64, 128, 192, 256, 320, 384 or 448 bits long) and are sampled randomly for training iteration. Similarly, training for each architecture was conducted 10 times, each time selecting 400,000 executable (incl. 200,000 from non-executable vectors with embedded chunks) and 400,000 non-executable (mixture of PDF, CSV, JPEG etc.) 512-dimensional,

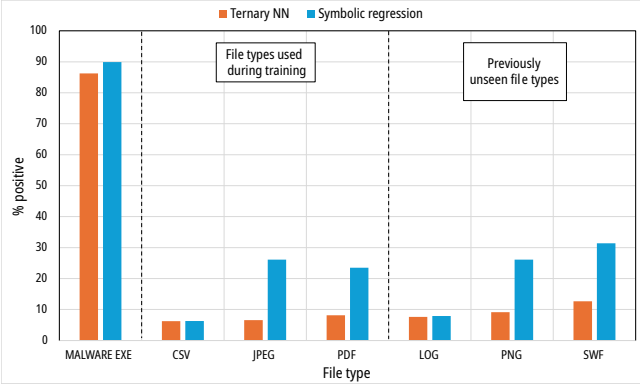


Figure 2. Ratio of 512-bit chunks in whole files, with duplicates removed, predicted as executable, per file type. A high value is desired for executables (detection rate), a low value for all other data types (false positive rate).

binary vectors. The accuracy, FPR and FNR, obtained from a held-out set of 160,000 (20%) points, are reported in Table 2. By embedding chunks of lower dimensionality into non-executable payloads, we show that our approach can generalize to (i) networking systems with a different configuration, namely AXI4 Stream widths and (ii) identify subtle changes in non-executable files. The latter is particularly important for documents that can be corrupted with embedded malcode [23] or JavaScript embeddings [40]. Due to the added complexity and data augmentation, the overall accuracy drops slightly. However, it is important to note that the values reported in Table 2 result from embedding extremely small (as low as 64b) chunks of executables into documents, corresponding to very few CPU instructions. Therefore, realistic code injection attempts are expected to fill up the full 512b, corresponding to results shown Table 1.

Table 2. Accuracy of DPI models identifying executables and non-executable payloads with embedded executables.

Model	Accuracy [%]	FPR [%]	FNR [%]
SR	83.47 ± 0.57	13.11 ± 0.72	19.95 ± 0.58
Ternary	89.36 ± 0.39	6.25 ± 1.37	15.04 ± 0.86

This time, SR performed worse than the ternary model. A possible reason for that is input pruning in SymbolNet [48]. To achieve low latency and avoid a cascade of complex mathematical operations in hardware, SymbolNet prunes a large portion of the inputs. However, in our case, the inputs are binary, hence, already sparse and additional pruning can adversely affect the performance. For hardware and networking evaluation, we use the models trained with code embedding. While SR achieves lower accuracy than the the ternary neural network and has a higher FPR/FNR, we consider accelerating it on the FPGA as it can offer significant area savings and latency reductions, while still achieving satisfactory accuracy for some applications.

4.2 DPI Generalizability and malware detection

To study the generalization of the proposed models, we also evaluate the models on whole files, with varying extensions and ranging in size from 154B to 22MB, and, report the fraction of 512-bit chunks evaluated as executable in a given file. By doing so, it is possible to identify any weaknesses on specific file types, which is not possible from the overall accuracy reported in Tables 1 and 2 on a mixture of file types. Importantly, the models are evaluated on previously unseen files, previously unseen file types, as well as malware executables.

- For a realistic evaluation of its capabilities, we tested the trained ML models with 47 malware executables from the VX-Underground collection [50], comprising rootkits, Mirai-type viruses and backdoors as typical for *shellcode* attacks, as well as virus-modified Linux bash commands.
- For CSV, JPEG and PDF (extensions encountered during training), files from the same source [15, 10] as training files are used, but unseen during training.
- Finally, we consider file types not even encountered during training: LOG and SWF from *Govdocs1* [15], and PNG from the *SVHN* dataset [26].

To avoid skewing the results for any possible duplicate inputs, we again remove any duplicate 512-bit vectors within a single file. The results are illustrated in Fig. 2, using the best ternary and SR models. Both the ternary and SR model are able to correctly identify more than 85% of the 512-bit chunks in any given malware executable. On the other hand, for non-executable files, the ternary model achieves consistent performance, even on previously unseen data types, while SR shows large variance and FPR, depending on the file type. Generally, the FPR increases with the complexity of the file (e.g. SWF, JPEG). Finally, while we acknowledge that both the ternary and SR model exhibit a somewhat high FPR at around 10%-20%, however, we argue that compared to the detection rate of actual malware executable, there exists a more than sufficient gap for efficient threshold and thus correct rejection of malware. As future work, we plan to extend the Decision Aggregator to be able to set such a threshold determining the number of flagged packets in the payload to reject the entire message.

4.3 Hardware evaluation

The best-performing ternary and SR models are implemented with `hls4ml`. The resource consumption and latency are reported in Tables 3. The resources are reported post-Place

Table 3. Post-PnR resource consumption and timing of the DPI models on Alveo U55C with 4ns clock period.

	Ternary	SR
LUT [%]	30062 [2.3%]	472 [0.04%]
FF [%]	11363 [0.4%]	271 [0.01%]
DSP [%]	0 [0%]	2 [0.02%]
BRAM [%]	0 [0%]	4 [0.05%]
Latency	11 cc [44 ns]	6 cc [24 ns]
Initiation interval	1 cc [4 ns]	1 cc [4 ns]

and Route (PnR) and the latency is reported from Vivado co-simulation, and, confirmed in the following section, with networked FPGAs and ChipScopes. Both models achieve ultra-low latency, which can be hidden by the packet-processing pipeline, and minimal resource consumption. While SR can achieve an extremely low resource footprint, it comes at the expense of accuracy and generalizability, which makes it a potentially interesting approach for lower-dimensional tasks such as header or metadata analysis. As expected, the SR model uses more BRAM for storing the look-up tables; however, after operator pruning [48], the only unary operation remaining was `exp(·)` and the BRAM utilization remained low. Since `hls4ml` performs additional post-training quantization, a slight accuracy drop is observed when running the model on hardware - the ternary model achieved an accuracy of 89.66%, while the SR achieved an accuracy of 84.47%.

4.4 Network performance

Finally, we perform end-to-end networking experiments with the ternary neural network for executable detection, confirming line rate and 100Gbps throughput. The evaluation was performed with the DPI-extended RDMA stack deployed as part of the Coyote shell [20] on two Alveo U55C accelerator cards, connected via a switched 100G-Ethernet network. The Alveo U55C FPGAs play the role of a traditional NIC, implementing the packet processing pipeline, the newly added DPI and finally, writing the packet to host CPU memory, if deemed acceptable. The tests follow the idea of the standard, open-source `perftest` library [27], with 1000 ping-pong exchanges of single messages for latency evaluation and large batches of 1000 messages for throughput testing, both realized via one-sided `RDMA WRITE` operations. Illustrated in Fig. 3, both throughput and latency across a wide range of message sizes are within a 5% margin of difference for the RDMA stack with and without DPI, with those differences being caused by inevitable network noise and traffic in a non-isolated public cluster. This confirms that DPI does not bottleneck the network.

In the second test, a commodity, ASIC-based NIC (Mellanox ConnectX-5) was used to send various test payloads to the DPI-enhanced FPGA-NIC. By transmitting both safe

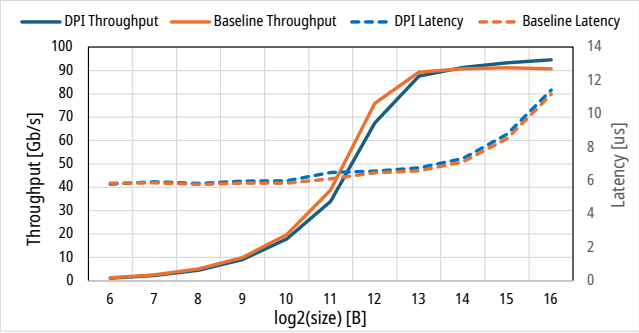


Figure 3. Comparison of RDMA with DPI (ternary ML model) and baseline (no DPI) for `RDMA WRITE` operations.

and potentially malicious executables and observing the resulting ACKs and NAKs issued by the FPGA-based NIC, we could confirm the desired functionality of the design and the full compatibility with the RoCEv2 protocol standard. It could therefore be deployed in a heterogeneous networking fleet with commodity NICs.

5 Conclusions

In this paper, we proposed leveraging the reconfigurability of FPGAs to extend an open-source RDMA stack with DPI enhancements. By integrating ultra-low latency neural networks and symbolic regression directly on the datapath, DPI achieves line-rate performance without impacting latency or throughput, while consuming less than 1% of resources. The models are able to generalize to previously unseen data types as well as systems with different configurations, including varying bus widths. When evaluated on malware executables, the models achieve a high classification rate. We integrate our DPI module with a fully open-source, RoCEv2-compatible RDMA stack due to the protocol's inherent lack of access control. However, our solution is not tied to RDMA; instead it adopts standardized AXI4 Stream interfaces and is therefore highly portable to various different FPGA-based networking stacks such as TCP/IP [37, 33] or UDP [16]. In conclusion, our approach showcases the advantages of FPGAs in the cloud and data centers as in-network SmartNICs, which benefit from the unmatched on-datapath acceleration for additional packet processing which is achieved with streaming computation and the high degree of parallelism available on reconfigurable fabric.

Acknowledgments

We would like to thank AMD for the donation of the Heterogeneous Accelerated Compute Cluster (HACC) which was used for the development and testing of this project. The work was funded in part through an unrestricted grant from AMD.

References

[1] Abdulwahab Ali Almazroi et al. 2024. Deep learning hybridization for improved malware detection in smart internet of things. *Scientific Reports*, 14, 1, (Apr. 2024), 7838. doi:10.1038/s41598-024-57864-8.

[2] N. S. Artan et al. 2007. Tribica: trie bitmap content analyzer for high-speed network intrusion detection. In *IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications*, 125–133. doi:10.1109/INFCOM.2007.23.

[3] Wei Bai et al. 2023. Empowering azure storage with RDMA. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, (Apr. 2023), 49–67. ISBN: 978-1-939133-33-5. https://www.usenix.org/conference/nsdi23/presentation/bai.

[4] Fangbo Cai et al. 2019. Survey of access control models and technologies for cloud computing. *Cluster Computing*, 22, 3, (May 2019), 6111–6122. doi:10.1007/s10586-018-1850-7.

[5] Milan Ceška et al. 2019. Deep Packet Inspection in FPGAs via Approximate Nondeterministic Automata. In *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 109–117. doi:10.1109/FCCM.2019.00025.

[6] Ronald Cheng et al. 2017. D2pi: identifying malware through deep packet inspection deep learning. https://www.cs.umd.edu/sites/default/files/scholarly_papers/Cheng%2C%20Ronald_1801.pdf.

[7] Eric Chung et al. 2018. Serving DNNs in Real Time at Datacenter Scale with Project Brainwave. *IEEE Micro*, 38, (Mar. 2018), 8–20. https://www.microsoft.com/en-us/research/publication/serving-dnns-real-time-datacenter-scale-project-brainwave/.

[8] Claudionor N Coelho et al. 2021. Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors. *Nature Machine Intelligence*, 3, 8, 675–686.

[9] Jia Deng et al. 2009. Imagenet: a large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 248–255. doi:10.1109/CVPR.2009.5206848.

[10] 2023. Digital Corpora: corpora/files/CC-MAIN-2021-31-PDF-UNTRUNCATED/. (2023). https://corp.digitalcorpora.org/corpora/files/CC-MAIN-2021-31-PDF-UNTRUNCATED/.

[11] J. Duarte et al. 2018. Fast inference of deep neural networks in FPGAs for particle physics. *Journal of Instrumentation*, 13, 07, (July 2018), P07027. doi:10.1088/1748-0221/13/07/P07027.

[12] [SW] FastML Team, fastmachinelearning/hls4ml version v0.7.1, 2023. doi:10.5281/zenodo.1201549, URL: https://github.com/fastmachinelearning/hls4ml.

[13] Daniel Firestone et al. 2018. Azure Accelerated Networking: SmartNICs in the Public Cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. (Apr. 2018). https://www.microsoft.com/en-us/research/publication/azure-accelerated-networking-smartnics-public-cloud/.

[14] Jeremy Fowers et al. 2018. A Configurable Cloud-Scale DNN Processor for Real-Time AI. In *Proceedings of the 45th International Symposium on Computer Architecture, 2018*. ACM, (June 2018). https://www.microsoft.com/en-us/research/publication/a-configurable-cloud-scale-dnn-processor-for-real-time-ai/.

[15] Simson Garfinkel et al. 2009. Bringing science to digital forensics with standardized forensic corpora. *Digital Investigation*, 6, (Sept. 2009). doi:10.1016/j.diin.2009.06.016.

[16] Fernando Luis Herrmann et al. 2009. A Gigabit UDP/IP network stack in FPGA. In *2009 16th IEEE International Conference on Electronics, Circuits and Systems - (ICECS 2009)*, 836–839. doi:10.1109/ICECS.2009.5410757.

[17] Amazon Web Services Inc. 2024. Amazon EC2 F1 Instances. Accessed on March 16th, 2024. (2024). https://aws.amazon.com/ec2/instance-types/f1/?nc1=h_ls.

[18] Wenqi Jiang et al. 2021. Fleetrec: large-scale recommendation inference on hybrid GPU-FPGA clusters. In *The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD*.

[19] Diederik P. Kingma et al. 2014. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

[20] Dario Korolija et al. 2020. Do OS abstractions make sense on FPGAs? In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, (Nov. 2020), 991–1010. ISBN: 978-1-939133-19-9. https://www.usenix.org/conference/osdi20/presentation/roscoe.

[21] Laurens Le Jeune et al. 2021. Sok - network intrusion detection on fpga. In *Security, Privacy, and Applied Cryptography Engineering: 11th International Conference, SPACE 2021, Kolkata, India, December 10–13, 2021, Proceedings*. Springer-Verlag, Kolkata, India, 242–261. ISBN: 978-3-030-95084-2. doi:10.1007/978-3-030-95085-9_13.

[22] M. Leeser et al. 2021. FPGAs in the Cloud. *Computing in Science & Engineering*, 23, 06, (Nov. 2021), 72–76. doi:10.1109/MCSE.2021.3127288.

[23] Wei-Jen Li et al. 2007. A study of malcode-bearing documents. In *Detection of Intrusions and Malware, and Vulnerability Assessment*. Bernhard M. Hämmerli et al., (Eds.) Springer Berlin Heidelberg, Berlin, Heidelberg, 231–250. ISBN: 978-3-540-73614-1.

[24] Jiuxing Liu et al. 2004. Evaluating the Impact of RDMA on Storage I/O over InfiniBand. In *SAN-03 Work- shop (in conjunction with HPCA), 2004*.

[25] Vedran Ljubovic. 2020. Programming homework dataset for plagiarism detection. (2020). doi:10.21227/71fw-ss32.

[26] Yuval Netzer et al. 2011. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*. http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf.

[27] 2024. Open Fabrics Enterprise Distribution (OFED) Performance Tests. Accessed on March 23rd, 2024. (2024). https://github.com/linux-rdma/perftest.

[28] Péter Orosz et al. 2019. FPGA-Assisted DPI Systems: 100 Gbit/s and Beyond. *IEEE Communications Surveys & Tutorials*, 21, 2, 2015–2040. doi:10.1109/COMST.2018.2876196.

[29] Sean Peisert. 2017. Security in high-performance computing environments. *Commun. ACM*, 60, 9, (Aug. 2017), 72–80. doi:10.1145/3096742.

[30] Maksym Planeta et al. 2023. CoRD: Converged RDMA Dataplane for High-Performance Clouds. (2023). arXiv: 2309.00898 [cs.OS].

[31] Michalis Polychronakis et al. 2010. Comprehensive shellcode detection using runtime heuristics. In *Proceedings of the 26th Annual Computer Security Applications Conference* (ACSAC '10). Association for Computing Machinery, Austin, Texas, USA, 287–296. ISBN: 9781450301336. doi:10.1145/1920261.1920305.

[32] Benjamin Rothenberger et al. 2021. Redmark: bypassing rdma security mechanisms. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, (Aug. 2021), 4277–4292. ISBN: 978-1-939133-24-3. https://www.usenix.org/conference/usenixsecurity21/presentation/rothenberger.

[33] Mario Ruiz et al. 2019. Limago: An FPGA-Based Open-Source 100 GbE TCP/IP Stack. In *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, 286–292. doi:10.1109/FPL.2019.00053.

[34] Arish Sateesan et al. 2020. Novel bloom filter algorithms and architectures for ultra-high-speed network security applications. In *2020 23rd Euromicro Conference on Digital System Design (DSD)*, 262–269. doi:10.1109/DSD51259.2020.00050.

[35] Niklas Schelten et al. 2022. A high-throughput, resource-efficient implementation of the rocev2 remote dma protocol and its application. *ACM Trans. Reconfigurable Technol. Syst.*, 16, 1, (Dec. 2022). doi:10.1145/3543176.

[36] Zhiyong Shan et al. 2012. Enforcing mandatory access control in commodity os to disable malware. *IEEE Transactions on Dependable and Secure Computing*, 9, 4, 541–555. doi:10.1109/TDSC.2012.36.

[37] David Sidler et al. 2015. Scalable 10Gbps TCP/IP Stack Architecture for Reconfigurable Hardware. In *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*, 36–43. doi:10.1109/FCCM.2015.12.

[38] David Sidler et al. 2020. StRoM: smart remote memory. In *Proceedings of the Fifteenth European Conference on Computer Systems* (EuroSys '20). Association for Computing Machinery, Heraklion, Greece. ISBN: 9781450368827. doi:10.1145/3342195.3387519.

[39] Anna Kornfeld Simpson et al. 2020. Securing RDMA for High-Performance datacenter storage systems. In *12th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 20)*. USENIX Association, (July 2020). https://www.usenix.org/conference/hotcloud20/presentation/kornfeld-simpson.

[40] Charles Smutz et al. 2012. Malicious pdf detection using metadata and structural features. In *Proceedings of the 28th Annual Computer Security Applications Conference* (ACSAC '12). Association for Computing Machinery, Orlando, Florida, USA, 239–248. ISBN: 9781450313124. doi:10.1145/2420950.2420987.

[41] Kevin Z. Snow et al. 2011. SHELLOS: enabling fast detection and forensic analysis of code injection attacks. In *20th USENIX Security Symposium (USENIX Security 11)*. USENIX Association, San Francisco, CA, (Aug. 2011). https://www.usenix.org/conference/usenix-security-11/shellos-enabling-fast-detection-and-forensic-analysis-code-injection.

[42] Konstantin Taranov et al. 2022. NeVerMore: Exploiting RDMA Mistakes in NVMe-oF Storage Applications. (2022). arXiv: 2202.08080 [cs.CR].

[43] Konstantin Taranov et al. 2020. sRDMA – efficient NIC-based authentication and encryption for remote direct memory access. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, (July 2020), 691–704. ISBN: 978-1-939133-14-4. https://www.usenix.org/conference/atc20/presentation/taranov.

[44] Andrei Tatar et al. 2018. Throwhammer: Rowhammer Attacks over the Network and Defenses. In *USENIX ATC*. Pwnie Award Nomination for Most Innovative Research. (July 2018). Paper=https://download.vusec.net/papers/throwhammer_atc18.pdf%20Web=https://www.vusec.net/projects/throwhammer%20Code=https://github.com/vusec/alis%20Press=https://goo.gl/GrZ87e.

[45] Andrei Tatar et al. 2018. Throwhammer: rowhammer attacks over the network and defenses. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. USENIX Association, Boston, MA, (July 2018), 213–226. ISBN: ISBN 978-1-939133-01-4. https://www.usenix.org/conference/atc18/presentation/tatar.

[46] 2024. TheAlgorithms/C-plus-Plus: Collection of various algorithms in mathematics, Machine Learning, computer science and physics implemented in C++ for educational purposes. Accessed on October 7th, 2024. (2024). https://github.com/TheAlgorithms/C-Plus-Plus.

[47] Shin-Yeh Tsai et al. 2019. Pythia: remote oracles for the masses. In *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, Santa Clara, CA, (Aug. 2019), 693–710. ISBN: 978-1-939133-06-9. https://www.usenix.org/conference/usenixsecurity19/presentation/tsai.

[48] Ho Fung Tsoi et al. 2024. SymbolNet: Neural Symbolic Regression with Adaptive Dynamic Pruning. *arXiv preprint arXiv:2401.09949*.

[49] Jure Vreča et al. 2021. Detecting network intrusion using binarized neural networks. In *2021 IEEE 7th World Forum on Internet of Things (WF-IoT)*, 622–627. doi:10.1109/WF-IoT51360.2021.9595961.

[50] 2024. Vx-underground malwaresourcecode. Accessed on October 4th, 2024. (2024). https://github.com/vxunderground/MalwareSourceCode.

[51] Yang Wang et al. 2019. Sgs: safe-guard scheme for protecting control plane against ddos attacks in software-defined networking. *IEEE Access*, 7, 34699–34710. doi:10.1109/ACCESS.2019.2895092.

[52] Jiarong Xing et al. 2022. Bedrock: programmable network support for secure RDMA systems. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Boston, MA, (Aug. 2022), 2585–2600. ISBN: 978-1-939133-31-1. https://www.usenix.org/conference/usenixsecurity22/presentation/xing.

[53] Chengcheng Xu et al. 2016. A survey on regular expression matching for deep packet inspection: applications, algorithms, and hardware platforms. *IEEE Communications Surveys & Tutorials*, 18, 4, 2991–3029. doi:10.1109/COMST.2016.2566669.

[54] Bowen Yang et al. 2019. Research on network traffic identification based on machine learning and deep packet inspection. In *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, 1887–1891. doi:10.1109/ITNEC.2019.8729153.

[55] S. Yusuf et al. 2005. Bitwise optimised CAM for network intrusion detection systems. In *International Conference on Field Programmable Logic and Applications, 2005.* 444–449. doi:10.1109/FPL.2005.1515762.

[56] Guanwen Zhong et al. 2023. A primer on reconic: rdma-enabled compute offloading on smartnic. (2023). arXiv: 2312.06207 [cs.DC].

[57] Chengjin Zhou et al. 2024. Netdpi: efficient deep packet inspection via filtering-plus-verification in programmable 5g data plane for multi-access edge computing. *IEEE Transactions on Mobile Computing*, 1–16. doi:10.1109/TMC.2024.3450691.

Received 11 February 2024