Distributed Deep Learning Inference on DPU-Based FPGA Systems for On-Board Earth Observation

Distributed Deep Learning Inference on DPU-Based FPGA Systems for On-Board Earth Observation

Invited Paper

Federico Buccellato
Politecnico di Torino
Turin, Italy
federico.buccellato@polito.it

Luca Mannini
Politecnico di Torino
Turin, Italy
s291065@studenti.polito.it

Corrado De Sio
Politecnico di Torino
Turin, Italy
corrado.desio@polito.it

Abstract

The growing adoption of Artificial Intelligence is driving demand for accurate, robust models, often characterized by deep, large-scale neural networks. While these models achieve high performance, their computational cost leads to significant inference latency, particularly on embedded, resource-constrained platforms such as FPGA-based architectures. This challenge is especially relevant in the space domain, where on-board deep learning is required for data analysis and decision-making under strict constraints on latency and computational resources. To address this challenge, this work proposes a distributed inference architecture based on FPGA accelerators, combining Deep Processing Units with a cluster-oriented execution model over TCP. The approach leverages a high-level development flow based on Vitis AI, enabling deployment without low-level hardware design expertise. Inference is executed across multiple FPGA boards, overcoming the limitations of a single device while maintaining portability and ease of integration. The proposed solution is validated on a cloud-detection task using satellite imagery, representative of realistic on-board scenarios. Experimental results show that the proposed approach achieves a speedup of up to 1.93× over single-board execution, demonstrating its effectiveness for large neural networks and suitability for on-board processing in space applications.

CCS Concepts

• **Computer systems organization → n-tier architectures**.

Keywords

FPGA acceleration, Distributed inference, Earth Observation, DPU

ACM Reference Format:
Federico Buccellato, Luca Mannini, and Corrado De Sio. 2026. Distributed Deep Learning Inference on DPU-Based FPGA Systems for On-Board Earth Observation: Invited Paper. In *Proceedings of the 23rd ACM International Conference on Computing Frontiers (CF Companion '26), May 19–21, 2026, Catania, Italy*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3801488.3808246

CF Companion '26, Catania, Italy

ACM ISBN 979-8-4007-2569-2/26/05
https://doi.org/10.1145/3801488.3808246

1 Introduction

In recent years, Artificial Intelligence has become a key enabling technology across several application domains, including computer vision [5], autonomous systems [2], and edge computing [16]. The rapid evolution of deep neural networks has led to increasingly accurate solutions for complex tasks, but also to a substantial growth in model complexity and computational demand. This trend is particularly critical in the space domain, where on-board processing is required for applications such as satellite image analysis, cloud detection, anomaly detection, and object recognition. In such scenarios, embedded platforms must operate under stringent constraints in terms of computational resources, communication bandwidth, and inference latency.

To address these challenges, hardware acceleration has become an important solution for improving inference performance. In this context, FPGA-based System-on-Chip platforms are particularly effective, as they combine programmable logic with general-purpose processors in a single architecture. This enables the implementation of specialized accelerators for deep learning workloads. At the same time, high-level toolchains such as Vitis AI, together with preconfigured components like Deep Processing Units (DPUs), have made FPGA-based acceleration more accessible [3, 4], reducing the need for low-level hardware design expertise.

Despite these advances, the execution of large, complex deep learning models on a single FPGA remains limited by available computational and memory resources. This limitation becomes particularly relevant in scenarios where increasing model size is necessary to achieve higher accuracy, while maintaining strict latency constraints.

In this work, we address this limitation by proposing a distributed inference approach using multiple FPGA nodes, where neural network models are decomposed and executed across multiple devices. By distributing the computation across multiple accelerators, the proposed method enables the execution of larger models while maintaining latency within acceptable bounds. The main contribution of this work lies in enabling distributed FPGA-based inference within a fully high-level development flow, leveraging DPUs and the Vitis AI toolchain. Unlike approaches that rely on custom hardware design or specialized interconnects, the proposed solution can be implemented within a standard software-oriented workflow, making distributed inference accessible to a broader range of developers.

By combining distributed execution with a standardized and high-level toolchain, the proposed methodology improves both scalability and deployability. In particular, it allows larger neural

networks to be executed across multiple FPGA devices while preserving a development workflow based on widely adopted deep learning frameworks.

Experimental results, validated in a realistic Earth Observation scenario using cloud detection from satellite imagery, demonstrate that the proposed approach improves inference performance for large models, achieving speedups of up to 1.93× over single-device execution. These results highlight how distributed FPGA-based inference, when combined with accessible development frameworks, can effectively support on-board processing in resource-constrained environments.

2 Related Works

In recent years, research on accelerating deep neural network inference using FPGAs has attracted growing attention, especially for embedded and edge applications with strict latency and hardware resource constraints. Several studies have demonstrated that FPGA architectures, thanks to their high degree of parallelism and hardware specialization, represent a valid alternative to CPUs and GPUs for the efficient execution of deep learning models [8].

A significant body of work has focused on accelerating individual neural network models on FPGAs, often involving AMD Deep Processing Units and the Vitis AI toolchain to implement convolutional neural networks efficiently on AMD systems. These approaches show that optimization of DPU parameters, together with careful architectural configuration, can significantly reduce latency while preserving high accuracy, even in resource-constrained scenarios such as embedded systems [15], CubeSats [6], and real-time vision applications [12].

However, the deployment of increasingly complex deep learning models introduces challenges beyond simply mapping a model onto a single FPGA. The growth in network depth and computational complexity often requires larger memory resources and strategies to improve execution time through workload parallelization. Distributing a neural network across multiple compute units can both overcome single-device hardware limitations and reduce inference latency by exploiting concurrent execution.

Current research on distributed neural network execution mainly focuses on GPU-based platforms, which dominate HPC environments due to mature software ecosystems and standardized programming models [9]. Nonetheless, some FPGA-based works have explored model-level parallelism, showing that distributed execution of CNNs [7] and RNNs [14] can improve throughput for large-scale models.

Despite these results, FPGA-based distributed approaches often rely on custom hardware architectures, dedicated interconnects, or tightly coupled, platform-specific designs. While these solutions can achieve high performance, they may limit portability and scalability, require hardware design expertise and complicate integration into standard embedded environments.

Within this context, this work proposes a distributed inference architecture that combines FPGA-based acceleration with a cluster-oriented execution model built entirely on commercial off-the-shelf platforms, pre-configured DPUs, and standard TCP communication. The system adopts a master–worker organization in which the neural network is partitioned into multiple sub-models that are executed across different FPGA nodes.

Unlike prior approaches, the proposed solution enables model-level parallelism within a fully high-level development flow based on Vitis AI, without requiring custom RTL design or specialized hardware interconnects. Communication between nodes follows a request–response pattern over standard TCP sockets, allowing intermediate feature maps to be exchanged without relying on custom communication protocols. This design choice prioritizes accessibility and deployability while still enabling performance improvements for large models. As a result, the approach allows scalable distributed inference to be integrated into real-world systems, making it particularly suitable for on-board processing in space applications.

3 Methodology

This section describes the methodology adopted to enable efficient deep learning inference using a distributed execution model based on FPGA accelerators. The approach leverages Deep Processing Units, which are hardware accelerators specifically designed to efficiently support the core computational kernels of neural networks, such as convolutions, matrix multiplications, and activation functions [4].

Figure 1 provides an overview of the overall workflow, highlighting the main stages from model preparation to distributed execution across multiple FPGA nodes.

3.1 Partitioning Workflow

The deployment of neural networks on the FPGA-based cluster considered in this work requires a transformation pipeline that adapts models to the constraints imposed by the target DPU architecture. Models trained on general-purpose platforms cannot be directly executed on reconfigurable accelerators and must undergo a sequence of processing steps, including quantization, partitioning, and compilation.

The Vitis AI toolchain provides the core support for this process through two main components: (i) a quantizer, responsible for model compression via quantization and calibration, and (ii) a compiler, which generates hardware-specific executables optimized for the target DPU.

The workflow starts from a neural network trained using standard deep learning frameworks (e.g., TensorFlow or PyTorch) in 32-bit floating-point precision. This choice ensures maximum representational capacity during training. The trained model is then processed by the Vitis AI quantizer, which converts weights, activations, and biases into an 8-bit integer representation. This step reduces both memory footprint and computational complexity, enabling efficient execution on FPGA-based accelerators. A calibration phase is subsequently performed using a representative dataset in order to mitigate accuracy degradation introduced by quantization.

After quantization, the model undergoes a custom partitioning stage, which represents the key step in enabling distributed execution across multiple FPGA nodes. This step must be performed prior to compilation. Indeed, the compilation process produces a hardware-specific execution graph in which operators are bound

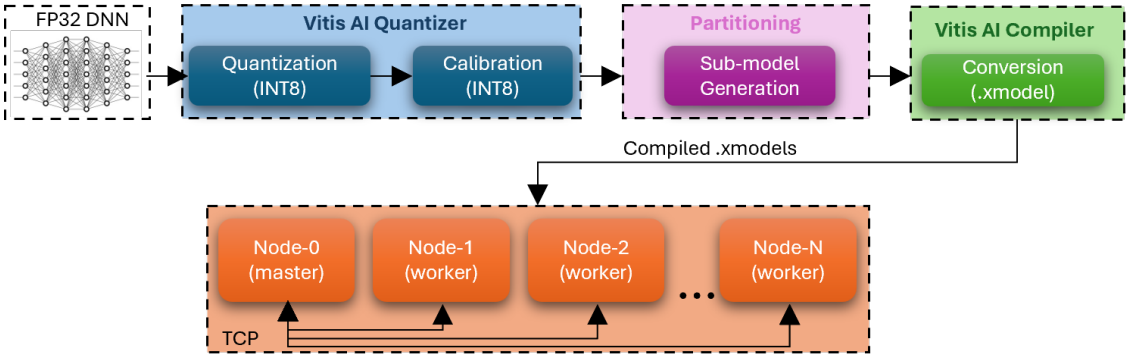

Figure 1: Overview of the workflow for model preparation, partitioning, and distributed inference on FPGA-based DPU accelerators.

to accelerator resources and scheduled accordingly, thereby preventing any further structural modifications to the model.

Formally, given a neural network represented as a computational graph G, the partitioning process consists in decomposing G into a set of N subgraphs $\{G_1, G_2, \ldots, G_N\}$, each corresponding to a sub-model assigned to an FPGA node. Each subgraph must satisfy the following requirements:

- **Functional completeness**: each partition must define a valid sub-model with well-defined input and output tensors.
- **Dependency closure**: all operations within a partition must depend only on tensors locally available or explicitly received from preceding partitions.
- **Graph consistency**: the composition of all partitions must preserve the functional equivalence with the original model.

The identification of partitioning points corresponds to selecting intermediate tensors that define valid cut points in the computational graph. These points must ensure both correctness and efficiency of the distributed execution.

The selection process is constrained by two competing objectives. On the one hand, the computational workload should be evenly distributed across partitions to maximize parallel efficiency. Load imbalance leads to idle time on faster nodes and limits the achievable speedup. On the other hand, the volume of data exchanged between partitions should be minimized, as inter-node communication contributes directly to inference latency.

To address these constraints, partitioning points are preferably selected at locations where tensor dimensionality is reduced, such as after pooling layers, bottleneck blocks, or other dimensionality-reduction operations. Furthermore, partitions should align with semantically coherent structures of the network, such as layer groups or independent branches, ensuring compatibility with the compilation flow and preserving structural clarity.

Once the partitioning points are identified, the actual graph decomposition is performed by operating on the intermediate representation of the model, namely the Xilinx Intermediate Representation (XIR). At this level, the computational graph can be directly accessed and modified prior to compilation. The XIR graph is explicitly split at the selected cut points, resulting in a set of independent subgraphs, each corresponding to a sub-model with well-defined inputs and outputs.

Each resulting subgraph is then exported as an independent model and compiled using the Vitis AI compiler, producing optimized `.xmodel` files targeting the DPU. These compiled models are subsequently deployed across the FPGA nodes to enable distributed inference.

3.2 Logical Execution Flow

The partitioned model results in a structured execution pipeline composed of $N + 2$ models, one *head*, N parallel sub-models, and one *tail*. This logical organization, illustrated in Figure 2, defines the dataflow of the distributed inference process.

During inference, the computation is divided into three main stages.

In the initial stage (*head*), the input data are processed to produce intermediate tensors representing feature maps at the partitioning boundaries. These tensors define the inputs for the parallel execution phase.

The second stage consists of parallel execution across the N sub-models. Depending on the network topology, intermediate tensors may be replicated or distributed across multiple branches. Each branch processes a portion of the computation independently, enabling parallelism.

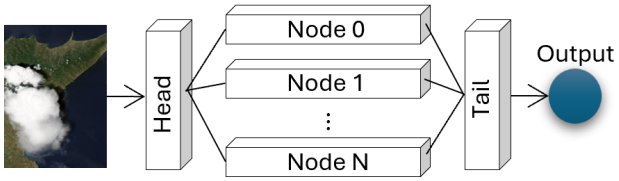

Figure 2: Distributed execution flow of the partitioned neural network.

In the final stage (*tail*), the outputs produced by the parallel branches are combined. This aggregation may involve concatenation, element-wise operations, or other transformations required

to reconstruct the original computational graph. The final layers are then executed to produce the inference result.

This head–distributed–tail decomposition separates sequential and parallel regions of the computation and defines explicit synchronization boundaries between stages.

3.3 Distributed Execution on FPGA Nodes

The logical execution model is mapped onto a distributed system composed of multiple FPGA nodes, as illustrated in Figure 1. Each node hosts a DPU and executes one or more sub-models generated during the partitioning phase.

The execution follows a *master–worker* paradigm, in which one node acts as a coordinator while the remaining nodes operate as compute units. The master node is responsible for orchestrating the execution, managing data transfers, and executing part of the neural network locally. In particular, it maintains the global execution state and enforces the correct ordering of operations according to the dependencies defined by the partitioned model.

The system is implemented on FPGA platforms interconnected via TCP communication. This design choice avoids the need for custom interconnects or tightly coupled hardware solutions, enabling portability across different deployment environments.

From a communication perspective, the system follows a request–response pattern. The master node generates intermediate tensors and transmits them to worker nodes, which process the data and return partial results. Each interaction involves serialization, transmission, deserialization, and synchronization. These steps directly affect overall inference latency and must be accounted for when evaluating system performance.

Each worker node executes a specific sub-model independently. Since sub-models are fully compiled and self-contained, workers do not require inter-node communication during computation. This property isolates computation from communication and simplifies the execution model, allowing each worker to operate autonomously once the input data has been received.

After processing, workers send their outputs back to the master node. The master aggregates the results and forwards them to the subsequent stage of the execution flow. This aggregation step introduces a synchronization point, as the master must collect all required outputs before proceeding.

The master node maintains a global view of the system, including model decomposition, node availability, and the mapping between sub-models and hardware resources. This enables consistent scheduling decisions and correct handling of data dependencies. As a result, the execution model remains deterministic, with well-defined data exchanges and synchronization boundaries across all nodes.

4 Experimental Analysis

This section presents the experimental evaluation of the proposed distributed inference architecture. It first describes the experimental setup used to conduct the evaluation and then reports the inference results, highlighting the impact of distributed execution and model-level parallelism on inference latency.

4.1 Experimental Setup

The experimental campaign was conducted on a cluster of three Kria KV260 FPGA boards, each configured as an independent compute node. All boards run Ubuntu 22.04 LTS on ARM64 and are equipped with FPGA-based accelerators for deep learning inference. To maximize performance, the most performant DPU B4096 overlay provided by AMD for the Kria platform was used. This overlay offers a high degree of internal parallelism and is designed to efficiently support large convolutional neural networks, enabling low-latency inference and effective FPGA resource utilization.

The FPGA nodes are interconnected through a standard Gigabit Ethernet network, with communication between nodes implemented over TCP sockets

Neural network model preparation was performed on a dedicated development server running Ubuntu 22.04 LTS. The server handled model partitioning, quantization, calibration, and compilation using the official Vitis AI toolchain.

The experimental evaluation focuses on a representative Earth Observation task, specifically cloud detection in satellite visual data. This application is particularly relevant in space scenarios, as it enables the identification of non-informative regions and supports more efficient use of communication bandwidth by reducing the amount of data transmitted to ground stations.

To this end, convolutional neural networks were used as representative workloads, given their widespread adoption in image-based Earth Observation tasks [1, 10, 13]. The input data are derived from the Cloud-38 dataset [11], which provides satellite imagery for cloud detection tasks. The images include multiple spectral channels, namely the visible bands (RGB) and Near-Infrared (NIR), which offer complementary information for distinguishing clouds from background surfaces. Accordingly, all models operate on inputs with shape $[1, 384, 384, 4]$.

To evaluate the scalability of the distributed inference approach, a set of convolutional neural network variants with increasing complexity was considered. All models share the same base architecture but differ in terms of channel width and total number of parameters. This allows a systematic analysis of how model size and computational load impact distributed execution.

The main architectural characteristics of the evaluated CNN variants are summarized in Table 1.

Table 1: Architectural characteristics of the evaluated CNN variants.

Variant	Input Size	Parameters (M)	Depth	Max Channels
CNN-S	[1, 384, 384, 4]	~1.3	12	128
CNN-M	[1, 384, 384, 4]	~26	18	512
CNN-L	[1, 384, 384, 4]	~133	20	1024
CNN-XL	[1, 384, 384, 4]	~364.4	22	1536

4.2 Experimental Results

This section presents the experimental results obtained by evaluating distributed inference on convolutional neural networks of increasing complexity. The analysis focuses on the impact of model size on inference latency and on the effectiveness of multi-node FPGA execution.

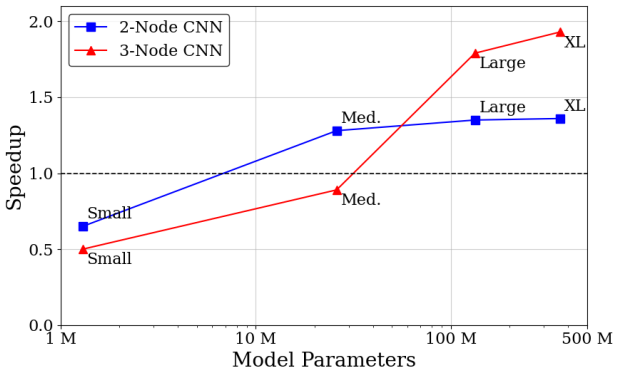

Figure 3: Inference latency of CNN models across single-node and distributed FPGA configurations.

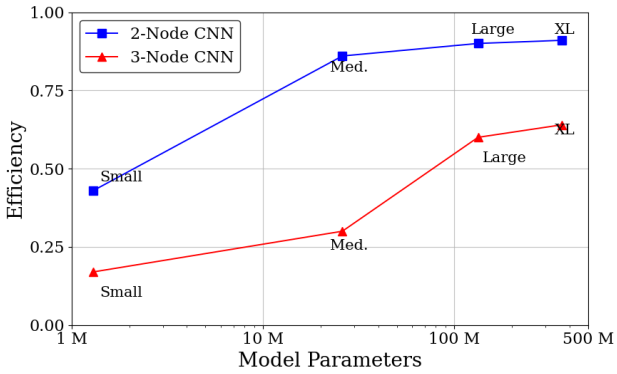

Figure 4: Parallel efficiency for 2-node and 3-node distributed execution.

Figure 3 reports the inference latency and corresponding speedup for the evaluated CNN models under single-node, 2-node, and 3-node configurations.

For small models, distributed execution introduces a clear performance degradation. The limited computational workload does not compensate for the overhead associated with data transfer, synchronization, and orchestration. As a result, single-node execution remains the most efficient configuration.

As model complexity increases, the benefits of distribution begin to emerge. In the medium configuration, a 2-node setup already shows a reduction in inference latency, indicating that model-level parallelism starts to amortize communication overhead. However, extending the execution to 3 nodes does not provide further improvements, as coordination costs still dominate.

For larger models, distributed execution becomes increasingly effective. In the large configuration, parallel execution is able to mask communication overhead, resulting in a substantial latency reduction. This trend becomes even more pronounced for the XL model, where inference latency decreases from 624.07 ms on a single node to 322.89 ms on 3 nodes, corresponding to a speedup of approximately 1.93×.

Overall, these results highlight a transition point in which distributed execution becomes advantageous only when the computational workload is sufficiently large to amortize communication and orchestration overhead.

To further analyze scalability, we consider the parallel efficiency, defined as

$$\eta = \frac{S}{N}$$

where S is the achieved speedup and N is the number of FPGA nodes.

Figure 4 shows the efficiency trends for both 2-node and 3-node configurations. While the 3-node configuration provides higher absolute speedup for large models, the 2-node configuration consistently achieves higher efficiency. This indicates that the additional FPGA node does not translate into proportional performance gains due to increased communication and synchronization overhead.

In particular, the 2-node configuration reaches efficiency values above 0.85 for medium and large models, whereas the 3-node configuration remains significantly lower. This behavior highlights a key trade-off in which increasing the number of nodes improves peak performance but reduces efficiency, since a larger fraction of execution time is spent in coordination rather than computation.

The observed communication overhead must be distinguished from raw network transfer time. For the considered workload, a typical intermediate tensor exchanged between nodes is approximately 600 KB and can be transferred in less than 1 ms over a Gigabit Ethernet connection, indicating that network bandwidth is not the limiting factor.

Instead, the dominant contribution arises from the distributed execution workflow, including tensor serialization and deserialization, TCP round-trip interactions between master and worker nodes, synchronization across parallel branches, and idle waiting along the critical path. This effect is quantified in Table 2, which reports a component-wise latency decomposition for the 3-node configuration.

Table 2: Component timing breakdown for 3-node distributed inference.

Component	CNN-S(ms)	CNN-M (ms)	CNN-L (ms)	CNN-XL (ms)
Head	0.42	0.70	2.84	29.62
Nodes	5.74	48.90	117.76	291.43
Tail	0.88	1.16	1.24	90.92
Total	7.22	50.75	121.84	322.89

The timing breakdown in Table 2 further clarifies this behavior. The nodes phase dominates the total latency across all models. For smaller networks, a significant portion of this phase is attributable to communication and orchestration overhead, which limits the benefits of distributed execution.

As model complexity increases, computation on the DPU becomes the dominant component of execution time, reducing the relative impact of communication overhead. This allows distributed execution to achieve higher efficiency, as the overhead is progressively amortized.

The *head* and *tail* phases contribute less to the total latency, although their cost increases for larger models due to higher-dimensional intermediate data and aggregation operations. In particular, the *tail* stage becomes more significant for CNN-XL, reflecting the cost of combining outputs from multiple branches.

Overall, these results highlight a fundamental trade-off between computation and communication. Increasing the number of nodes improves peak performance but reduces efficiency, while fewer nodes achieve better resource utilization. Consequently, distributed FPGA-based inference becomes advantageous only beyond a certain model complexity threshold, where computation dominates execution time and amortizes communication overhead.

5 Conclusion

This work presents a distributed inference architecture based on FPGA accelerators, designed to support efficient execution of deep learning models in resource-constrained environments. The proposed approach exploits model-level parallelism to distribute the computational workload across multiple FPGA nodes, enabling both reduced inference latency and the execution of neural networks that cannot be supported by a single platform. At the same time, by leveraging pre-configured DPUs and a high-level development flow based on Vitis AI, the approach reduces the complexity of FPGA-based distributed deployment, making it accessible without requiring low-level hardware design expertise.

The experimental validation, conducted on convolutional neural networks for cloud detection, shows that the benefits of distributed execution increase with model complexity. In particular, for large models, a speedup of up to 1.93× was observed over single-node execution, demonstrating the approach's effectiveness in improving inference performance.

From an application perspective, the results suggest that distributed FPGA architectures represent a promising solution for enabling onboard processing capabilities in Earth Observation scenarios, helping to reduce latency and dependence on downlink to ground-based infrastructures.

References

[1] 2021. Cloud detection using convolutional neural networks on remote sensing images. *Solar Energy* 230 (2021), 1020–1032. doi:10.1016/j.solener.2021.10.065

[2] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. E. Mohamed, and H. Arshad. 2018. State-of-the-Art in Artificial Neural Network Applications: A Survey. In *Proceedings of Heliyon*, Vol. 4. Article e00938.

[3] Advanced Micro Devices, Inc. 2023. Vitis AI User Guide (UG1414), Version 3.5. In *Proceedings of AMD Documentation*. https://docs.amd.com/r/en-US/ug1414-vitis-ai

[4] Advanced Micro Devices, Inc. 2024. Xilinx Deep Processing Unit (DPU). Available at https://docs.amd.com/r/en-US/Vitis-AI-User-Guide.

[5] F. Buccellato, E. Vacca, S. Azimi, C. De Sio, and L. Sterpone. 2025. From Detection to Intervention: An End-to-End System for Recognizing the "Signal for Help" Gesture in Real Time. In *Proceedings of Intelligent Systems with Applications*. Article 200536. doi:10.1016/j.iswa.2025.200536

[6] Angela Cratere, Mohamed Salim Farissi, Andrea Carbone, Marcello Asciolla, Maria Rizzi, Francesco Dell'Olio, Augusto Nascetti, and Dario Spiller. 2025. Efficient FPGA-Accelerated Convolutional Neural Networks for Cloud Detection on CubeSats. *IEEE Journal on Miniaturization for Air and Space Systems* 6, 3 (2025), 187–197. doi:10.1109/JMASS.2025.3533018

[7] Tong Geng, Tianqi Wang, Ahmed Sanaullah, Chen Yang, Rushi Patel, and Martin Herbordt. 2018. A Framework for Acceleration of CNN Training on Deeply-Pipelined FPGA Clusters with Work and Weight Load Balancing. In *Proceedings of the 28th International Conference on Field Programmable Logic and Applications (FPL)*. 394–3944. doi:10.1109/FPL.2018.00074

[8] Kaiyuan Guo, Shulin Zeng, Jincheng Yu, Yu Wang, and Huazhong Yang. 2019. [DL] A Survey of FPGA-based Neural Network Inference Accelerators. *ACM Transactions on Reconfigurable Technology and Systems* 12, 1, Article 2 (March 2019), 26 pages. doi:10.1145/3289185

[9] Ziyue Luo, Jia Liu, Myungjin Lee, and Ness B. Shroff. 2025. Prediction-Assisted Online Distributed Deep Learning Workload Scheduling in GPU Clusters. arXiv:2501.05563 [cs.DC] https://arxiv.org/abs/2501.05563

[10] Nan Ma, Lin Sun, Chenghu Zhou, and Yawen He. 2021. Cloud Detection Algorithm for Multi-Satellite Remote Sensing Imagery Based on a Spectral Library and 1D Convolutional Neural Network. *Remote Sensing* 13, 16 (2021). doi:10.3390/rs13163319

[11] S. Mohajerani and P. Saeedi. 2019. Cloud-Net: An End-To-End Cloud Detection Algorithm for Landsat 8 Imagery. In *IGARSS 2019 - 2019 IEEE International Geoscience and Remote Sensing Symposium*. 1029–1032. doi:10.1109/IGARSS.2019.8898776

[12] Camilo A. Ruiz-Beltrán, Adrián Romero-Garcés, Martín González-García, Rebeca Marfil, and Antonio Bandera. 2023. FPGA-Based CNN for Eye Detection in an Iris Recognition at a Distance System. *Electronics* 12, 22, Article 4713 (2023). doi:10.3390/electronics12224713

[13] Michal Segal-Rozenhaimer, Alan Li, Kamalika Das, and Ved Chirayath. 2020. Cloud detection algorithm for multi-modal satellite imagery using convolutional neural-networks (CNN). *Remote Sensing of Environment* 237 (2020), 111446. doi:10.1016/j.rse.2019.111446

[14] Yuxi Sun and Hideharu Amano. 2020. FiC-RNN: A Multi-FPGA Acceleration Framework for Deep Recurrent Neural Networks. *IEICE Transactions on Information and Systems* E103-D, 12 (2020), 2457–2462.

[15] Mehdi Trabelsi Ajili and Yuko Hara-Azumi. 2022. Multimodal Neural Network Acceleration on a Hybrid CPU–FPGA Architecture: A Case Study. *IEEE Access* 10 (2022), 9603–9617. doi:10.1109/ACCESS.2022.3144977

[16] Zhi Zhou, Xu Chen, En Li, Liekang Zeng, and Ke Luo. 2023. Edge AI: A Survey. *Internet of Things and Cyber-Physical Systems* 3 (2023), 71–92. doi:10.1016/j.iotcps.2023.02.004