Onboard Machine Learning for Satellite Edge Computing: The SPAICE Project Use Case

Luis M. Garcés-Socarrás[†], Raudel Cuiman[†], Flor Ortiz[†], Juan A. Vásquez-Peralvo[†], Jorge L. González-Rios[†], Mouhamad Chehaitly[†], Arkadii Kazanskii[†], Sahar Malmir[†], Amirhossein Nik[†], Jan Thoemel[†], Sumit Kumar[♭], Marcele Kuhfuss[†], Swetha Varadajulu[†], Eva Lagunas[†], Juan C. M. Duncan[†], Jorge Querol[†], Symeon Chatzinotas[†]

[†]*Interdisciplinary Centre for Security, Reliability and Trust (SnT), University of Luxembourg, Luxembourg*

[♭]*Luxembourg Institute of Science and Technology (LIST)*

***Abstract*—This work addresses the challenge of implementing an artificial intelligence-driven flexible payload onboard for next-generation satellites. Within the SPAICE project, we present the design and hardware deployment of hardware-optimized machine learning models for flexible payload and adaptive beamforming. The models are restructured to reduce memory and parameter overhead, then quantized and compiled for the Versal ACAP AI platform. Optimization strategies, including Cross-Layer Equalization and Fast Fine-Tuning, mitigate quantization losses while maintaining near-floating-point accuracy. Experimental results demonstrate significantly faster inference than workstation implementations, confirming the feasibility of deploying advanced machine learning models onboard satellites for real-time, reconfigurable payload operation with high computational efficiency.**

***Index Terms*—Onboard satellite processing, Flexible payload, Adaptive beamforming, Machine learning, Convolutional neural networks, Versal ACAP**

I. Introduction

Recent advances in satellite communications have highlighted the growing demand for greater flexibility and autonomy in payload operations. Conventional satellite systems depend heavily on ground-based processing, which introduces latency, limits adaptability, and increases operational costs. As mission requirements become more dynamic, onboard processing has emerged as a promising approach to enable real-time decision-making under strict resource constraints.

Rapid advancement of machine learning (ML) and artificial intelligence (AI) has led to their integration into a wide range of applications, including space-based systems, autonomous navigation, and communication networks. In the case of Low Earth Orbit (LEO) satellites, the limited visibility windows to ground stations make fast and autonomous decision-making essential. Moreover, functional splits in next-generation satellite communication architectures require a flexible distribution of signal processing tasks between the space and ground segments. In this context, efficient onboard ML integration enables real-time adaptation to traffic and channel dynamics, maximizing resource utilization while ensuring service reliability.

This work has been supported by the European Space Agency (ESA), which funded it under Contract No. 4000134522/21/NL/FGL, ”Satellite Signal Processing Techniques using a Commercial Off-The-Shelf AI Chipset (SPAICE).” Please note that the views of the authors of this paper do not necessarily reflect the views of ESA

However, deploying ML models in resource-constrained environments, such as onboard satellites and edge computing platforms, faces several challenges. These include computational limitations, memory constraints, power efficiency, and the need for real-time inference under dynamic conditions. Unlike traditional ML deployment on high-performance computing systems, where abundant resources enable complex model execution, embedded hardware platforms require substantial model optimization, quantization, and hardware-specific adaptations to achieve an optimal balance between performance and efficiency.

Machine learning for onboard satellite studies have focused initially on limited use cases and a narrow range of methods, restricting the generality of the results. Furthermore, the short evaluation periods have hindered comprehensive assessment and comparison against traditional non-ML solutions, leaving the practical benefits and limitations of onboard ML largely unexplored [1], [2]. Additionally, achieving acceptable performance requires extensive and well-labeled training datasets, which are often difficult to store or access in space environments, along with significant computing capabilities and power for running the training process on the satellite [3], [4].

A practical solution to mitigate these problems when integrating ML techniques into the onboard payload is to execute the training and validation of the payload model offline using a set of input/output values, as depicted in Figure 1. This pre-trained model is then incorporated into the satellite payload processor to perform the inference process. In this approach, any changes to the trained model must be implemented on the ground and subsequently updated onboard. The feeder uplink signals are processed onboard according to the demand requirements for the satellite's coverage area. Based on the model's outputs, the system generates control instructions for the low physical layer, thereby improving transmission efficiency, managing network congestion, and optimizing bandwidth and power allocation [5].

The integration of AI and ML into satellite systems has been explored, with a focus on onboard autonomy and efficient resource management. Davidson *et al.* [6] demonstrate this within the Flexible and Intelligent Payload Chain (FIPC) subsystem, combining reconfigurable architectures with machine learning techniques to support autonomous payload control,

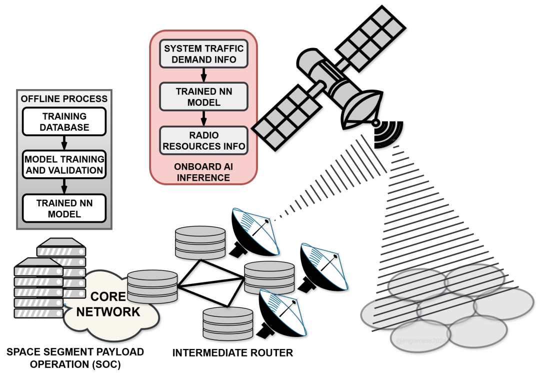

Fig. 1: Onboard AI Payload application [5].

while highlighting the challenges of balancing computational efficiency, flexibility, and reliability in space-qualified hardware and the importance of modular design. Similarly, Shenwai *et al.* [7] review the transformative impact of AI across the aerospace sector, underlining its growing role in enabling autonomy in dynamic environments, including satellite communications, Earth observation, and spectrum management. Complementing these perspectives, Kashyap and Gupta [8] survey resource allocation techniques in multibeam satellites, contrasting conventional optimization methods with AI/ML-based approaches and emphasizing the potential of data-driven techniques to enhance efficiency and adaptability, while noting the practical challenges imposed by computational, real-time, and hardware constraints of satellite payloads.

In this context, this paper presents the implementation, optimization, and evaluation of two ML models designed for flexible resource allocation on a Versal ACAP AI device as part of the Satellite Signal Processing Techniques using a Commercial Off-The-Shelf AI Chipset (SPAICE) project. SPAICE's objective is to leverage onboard intelligence to improve decision-making processes in satellite-based communication networks, particularly adaptive radio resource management (RRM) [9]. The paper explains the model's architecture, adaptation, and quantization for onboard processing constraints, while maintaining the expected accuracy. It also examines the model's onboard execution and performance metrics, comparing pre- and post-quantization model performance in terms of latency and prediction accuracy to assess their suitability for onboard deployment.

II. Machine Learning for flexible payload in onboard satellite applications

The use of regenerative satellite payloads motivates the adoption of flexible payload architectures and advanced beam management algorithms, which can be further accelerated through artificial intelligence and machine learning running directly on board. Regenerative payloads are capable of processing signals in the digital domain, enabling features such as inter-satellite links to relay connectivity to multiple gateways. This capability improves the link budget on the user link, improves the spectral efficiency on the feeder link, and simplifies the implementation of user and gateway handovers [5].

A flexible payload is a mission payload designed to be reconfigurable in orbit, allowing operators to adapt its functionality to evolving requirements after launch. Unlike traditional fixed payloads, which are hardwired for specific frequencies, coverage areas, and signal processing parameters, a flexible payload leverages technologies such as digital signal processors, beamforming networks, and software-defined radios. These technologies enable adjustments of frequency plans, beam shapes, bandwidth allocation, and routing on demand. The combination of regenerative and flexible payload capabilities allows satellites to dynamically respond to changing resource demands, emerging applications, and unexpected events. This flexibility ultimately extends mission life and maximizes return on investment [3], [10], [11].

Those principles are applied in the SPAICE project, which develops an Artificial Intelligence Satellite Telecommunications Testbed (AISTT), shown in Figure 2. The project implements a partial regenerative payload on a LEO satellite for AI-accelerated flexible payload and beam management algorithms, focusing on 5G New Radio. The testbed proposes the onboard integration of Next Generation NodeB (gNB) functionalities, enabling the evaluation of different functional split options between the space and ground segments. This allows the testbed to assess trade-offs between latency, feeder link load, and onboard processing complexity. The selected application features a software-controlled satellite payload connected to a multibeam Direct Radiating Array (DRA) antenna with hybrid beamforming, enabling dynamic adjustments to bandwidth, power, and beamwidth [5], [12]–[15].

The AISTT architecture integrates a Scenario Generator, a Base Station Emulator, an onboard payload, a Channel Emulator (ChEM), and user equipment (UE). The scenario generator is a MATLAB script within the payload control center that produces the inputs for the AI/ML algorithms, including traffic demand (R), beam pointing angles (Az and El), minimum Side Lobe Level (SLL_{min}), and beamforming coefficient phasors ($e^{j\theta_W}$).

The Base Station Emulator is a Next Generation NodeB distributed unit (gNB-DU) running OpenAir-Interface (OAI) that generates the downlink signal and receives the uplink. For implementation purposes, the partially regenerative functions running on the satellite payload are implemented on the downlink Low-PHY layer, which is the only link emulated on the physical layer due to hardware constraints.

The onboard payload is the principal component of the AISTT. It is composed of the AI/ML inference system and the Software-Defined Radio (SDR) as a Radio Frequency (RF) front-end. The inference system runs on a Versal ACAP AI Edge device, which executes the machine learning models, obtains the beamforming coefficients, and feeds them back to the payload control center. The SDR, powered by an RFSoC FPGA board, performs payload processing by receiving/transmitting the downlink/uplink signals to the gNB-DU computers via an Ethernet interface. It also receives the

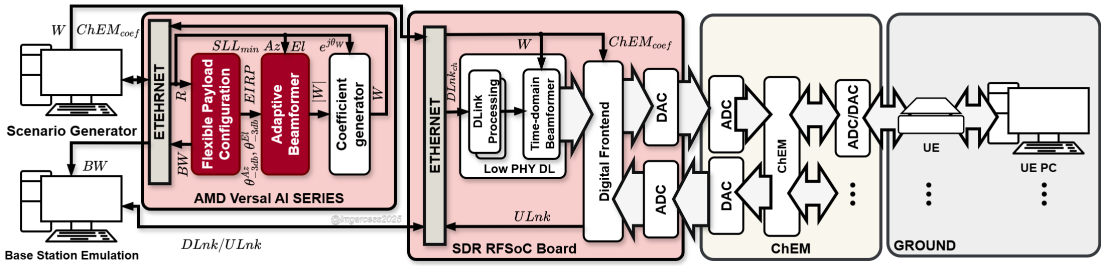

Fig. 2: AISTT functional diagram.

channel matrix coefficients ($ChEM_{\text{coef}}$) required to decode the downlink signals after mixing them in the time-domain beamformer. This operation is not computed onboard since, in a real scenario, channel emulation is not required.

The Channel Emulator receives the RF signal from the AI/ML payload, applies the channel effects, and sends it to the UE side. The UE aggregates all the traffic demand corresponding to all the users served by the beam. At the same time, the OAI UE generates different user information that is retransmitted to the gNB-DU (uplink) to modify the downlink requirements.

A. Models' selection

The inference system executes two machine learning models running in sequence. The first, the Flexible Payload model, is implemented in TensorFlow 2 (TF2) as a Keras sequential convolutional neural network (CNN) model, as shown in Figure 3a. The model is trained on a dataset, where each sample consists of a 401×501 matrix representing the traffic demand (R) as a function of the satellite position. The model classifies the output into 50 classes using a cost function designed to maximize the Effective Isotropic Radiated Power (EIRP), beamwidth, and bandwidth per beam.

When inspecting the model using the Vitis AI tools, several layers of the designed architecture—highlighted in red and yellow in Figure 3a—were reported as unsupported for the Deep Learning Processing Unit (DPU) IP core of AMD devices, responsible for the ML acceleration, requiring the execution of the operations of those layers in the CPU. The *Dense* layer after the *Flatten* operation requires a large matrix multiplication not supported by the hardware accelerator. The *Dropout* layer is removed during inference, and the *softmax* activation has to be implemented in SW.

As one of the project's objectives is to maximize performance and reduce inference latency, the architecture has been modified to remove the large fully connected layers—highlighted in blue in Figure 3b. The original *Flatten* and *Dense* layers, which together represented more than 47 million parameters, were replaced by a fully convolutional design that progressively reduces spatial dimensions through additional *MaxPooling2D* layers. The final classification is performed by a *Conv2D* layer with one channel per class, followed by a *GlobalAveragePooling2D* operation. After training, the model achieved an accuracy of 1.00 with a negligible loss, as reported in Table I. This result may vary depending on the weight values assigned by the tool each time the model is trained.

On the other hand, the Adaptive Beamforming model is a simpler Keras sequential CNN, illustrated in Figure 4a. Its input consists of a six-element vector dataset with a cost function to achieve the desired EIRP, beamwidth, and minimum SLL for the satellite position. The output corresponds to one of 15 predefined matrices for beam coefficient configurations. Due to the simplicity of the model and the number of parameters, the only layer that has been modified is the *softmax* activation of the last *Dense* layer, as shown in Figure 4b. In this case, the model achieved an accuracy of 0.99887 with a loss of 0.00546, as shown in Table II.

B. Models' quantization

The models must be quantized for the hardware implementation to reduce the precision of the weights from floating-point to an eight-bit representation. Although this process slightly impacts the model's accuracy, it optimizes the architecture for size, speed, and energy efficiency, making it suitable for deployment on resource-constrained devices. Furthermore, it ensures compatibility with hardware accelerators specifically optimized for low-precision formats.

During the quantization process, two versions have been evaluated: one without optimization parameters and the other with *Cross-Layer Equalization* (CLE) and *Fast Fine-Tuning* (FFT) enabled. CLE balances the weight distributions across consecutive layers, reducing scale mismatches and mitigating quantization errors without altering the network's functionality. On the other hand, FFT leverages a small calibration dataset to adapt the quantized model parameters, helping to recover the accuracy loss typically introduced during the quantization process [16], [17].

The performance results of the float models after quantization for the Flexible Payload model are presented in Tables I. The model's quantization reduces the accuracy of the float model by 0.0015 in the non-optimized and optimized versions. When the model is re-trained, this gap is reduced while exhibiting lower loss values. For the Adaptive Beamforming model (Table II), quantization preserves the accuracy relative to the float model, with the re-trained and optimized version achieving the lowest overall loss.

Although the overall accuracy of the model exceeds 0.99, an analysis of the accuracy per class is necessary to ensure correct performance across all classes. Figure 5 presents the bar graphs of the accuracy per class of the quantized Flexible Payload model versions, where each class represents

```
Model: model
Layer                Output Shape                Param #
InputLayer           (None, 401, 501, 1)               0
Conv2D<relu>         (None, 399, 499, 32)            320
MaxPooling2D         (None, 199, 249, 32)              0
Conv2D<relu>         (None, 197, 247, 64)          18496
MaxPooling2D         (None, 98, 123, 64)               0
Conv2D<relu>         (None, 96, 121, 128)          73856
MaxPooling2D         (None, 48, 60, 128)               0
Flatten              (None, 368640)                    0
Dense<relu>          (None, 128)                47186048
Dropout              (None, 128)                       0
Dense<softmax>       (None, 50)                     6450


Total params:                                 47,285,170
Trainable params:                             47,285,170
Non-trainable params:                                  0
```

```
Model: model
Layer                Output Shape                Param #
InputLayer           (None, 401, 501, 1)               0
Conv2D<relu>         (None, 399, 499, 32)            320
MaxPooling2D         (None, 199, 249, 32)              0
Conv2D<relu>         (None, 197, 247, 64)          18496
MaxPooling2D         (None, 98, 123, 64)               0
Conv2D<relu>         (None, 96, 121, 128)          73856
MaxPooling2D         (None, 48, 60, 128)               0
Conv2D<relu>         (None, 48, 60, 128)           16512
MaxPooling2D         (None, 24, 30, 128)               0
MaxPooling2D         (None, 12, 15, 128)               0
MaxPooling2D         (None, 6, 7, 128)                 0
Conv2D<linear>       (None, 6, 7, 50)               6450
GlobalAverage-       (None, 50)                        0
Pooling2D
Total params:                                    115,634
Trainable params:                                115,634
Non-trainable params:                                  0
```

Fig. 3: Flexible payload model structure: a) computer model, b) adapted model for Versal AI Edge implementation.

```
Model: model
Layer                Output Shape                Param #
InputLayer           (None, 6)                         0
Dense<relu>          (None, 64)                      448
Dense<relu>          (None, 64)                     4160
Dense<softmax>       (None, 15)                      975
Total params:                                      5,583
Trainable params:                                  5,583
Non-trainable params:                                  0
```

```
Model: model
Layer                Output Shape                Param #
InputLayer           (None, 6)                         0
Dense<relu>          (None, 64)                      448
Dense<relu>          (None, 64)                     4160
Dense<linear>        (None, 15)                      975
Total params:                                      5,583
Trainable params:                                  5,583
Non-trainable params:                                  0
```

Fig. 4: Adaptive beamforming model structure: a) computer model, b) adapted model for Versal AI Edge implementation.

TABLE I: Performance report of Flexible Payload model: Accuracy and loss during training & quantization.

Model	Type	Opt.	Accuracy	Loss
FPayl	Float	None	1.00000	0.00001
	Quantized	None	0.99853	0.00565
		CLE & FFT	0.99853	0.00646
	Retrained	None	0.99944	0.00139
		CLE & FFT	0.99898	0.00225

TABLE II: Performance report of Adaptive Beamforming model: Accuracy and loss during training & quantization.

Model	Type	Opt.	Accuracy	Loss
ABeam	Float	None	0.99887	0.00546
	Quantized	None	0.99515	0.03781
		CLE & FFT	0.99650	0.03462
	Retrained	None	0.99357	0.02180
		CLE & FFT	0.99639	0.00903

the required power, width, and bandwidths per beam in the satellite. The application of CLE and FFT does not improve the accuracy per class, as shown in Figures 5a and 5b, where three classes fail to reach maximum accuracy, with a minimum of 0.97. In contrast, re-training improves the accuracy of one class, but negatively affects others, with minimum accuracies of 0.34 without optimization (Figure 5c) and 0.62 when CLE and FFT are applied (Figure 5d).

Similar analyses have been carried out for the quantized Adaptive Beamformer models, as shown in Figure 6, where each class represents an activation matrix for the DRA antenna per beam. In the quantized versions (Figures 6a and 6b), only three classes do not reach the maximum accuracy, with a minimum of 0.93. After re-training, more than 66% of the classes are affected, with minimum accuracies of 0.76 in the non-optimized version (Figure 6c) and 0.77 in the optimized one (Figure 6d).

TABLE III: Performance report of Flexible Payload model: Accuracy, loss and inference time onboard.

Model	Type	Opt.	Accuracy	Loss	Time
FPayl	Quant.	None	0.98894	0.01106	0.00312
		CLE & FFT	0.99165	0.00835	0.00400
	Retrain.	None	0.99289	0.00711	0.00312
		CLE & FFT	0.99334	0.00666	0.00310

C. *Models'compilation*

Once the models are quantized, the next step is their conversion to the specific DPU IP block that executes them using the Vitis-AI compilation tools. This approach enables straightforward portability of the models to any supported AMD device.

To test the accuracy and inference speed of the compiled models for the iWave, each one has been executed separately with a test dataset of more than 8800 samples and compared with the reference values. On the Flexible Payload model, the reported minimum accuracy is greater than 0.98 (only 0.01 lower compared to the quantized model on the PC), with a maximum loss of less than 0.02 for the version without optimization, shown in Table III. The re-trained model with CLE and FFT optimization reports the best results.

Analyzing the per-class accuracy in Figure 7, it decreases in some cases, reaching a minimum of 0.06 in class 18 for the retrained model with optimizations. The most stable results are achieved by the quantized model with CLE and FFT, with only two classes below 0.90: class 18 (0.62) and class 47 (0.81), both of which had maximum accuracy before the on-board conversion. The average inference time is less than 4 ms, with slight variations depending on the type of model and the utilization of memory, processor, and DPU on the board.

For the Adaptive Beamforming model (Table IV), the accuracy is above 0.96 (0.03 lower than the quantized results),

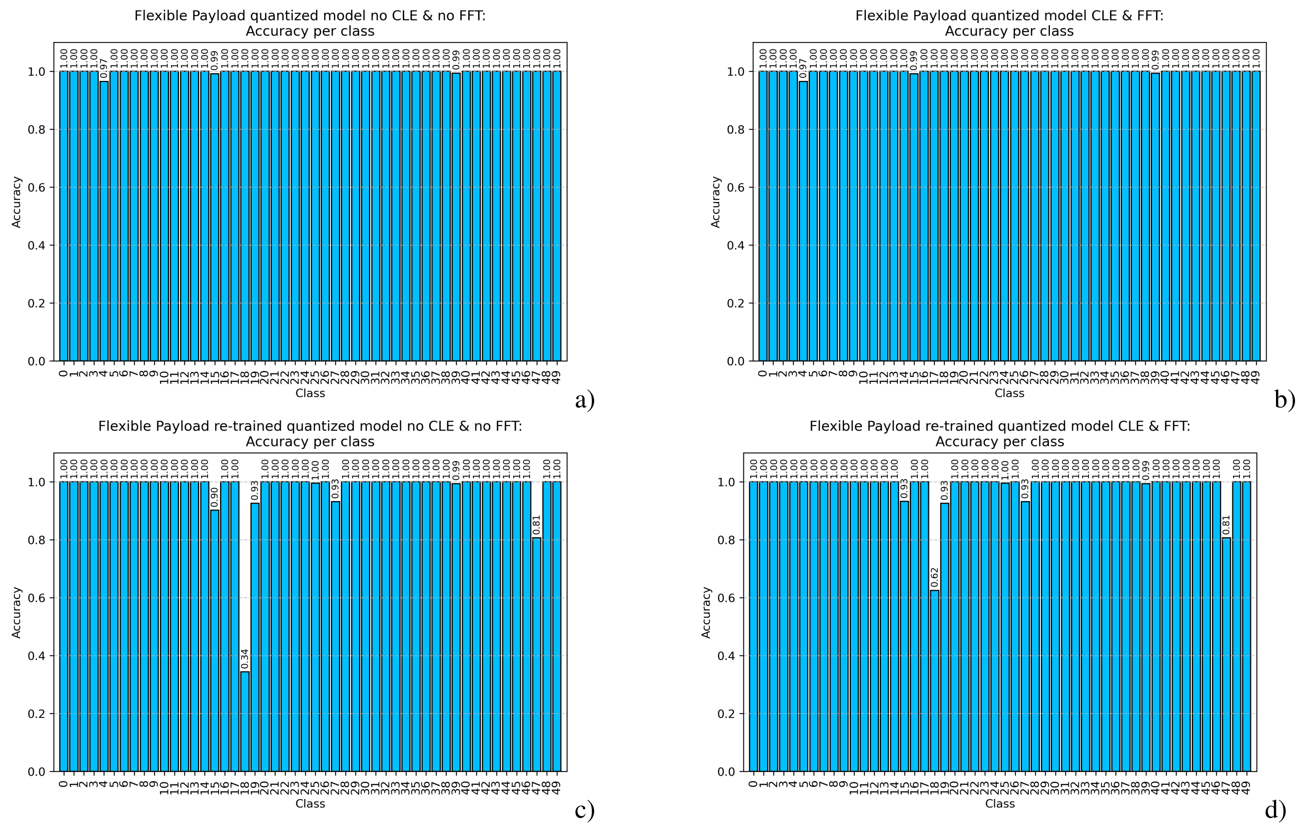

Fig. 5: Per-class accuracy of the flexible payload model under different configurations: a) without optimization, b) with CLE and FFT, c) re-trained without optimization, d) re-trained with CLE and FFT.

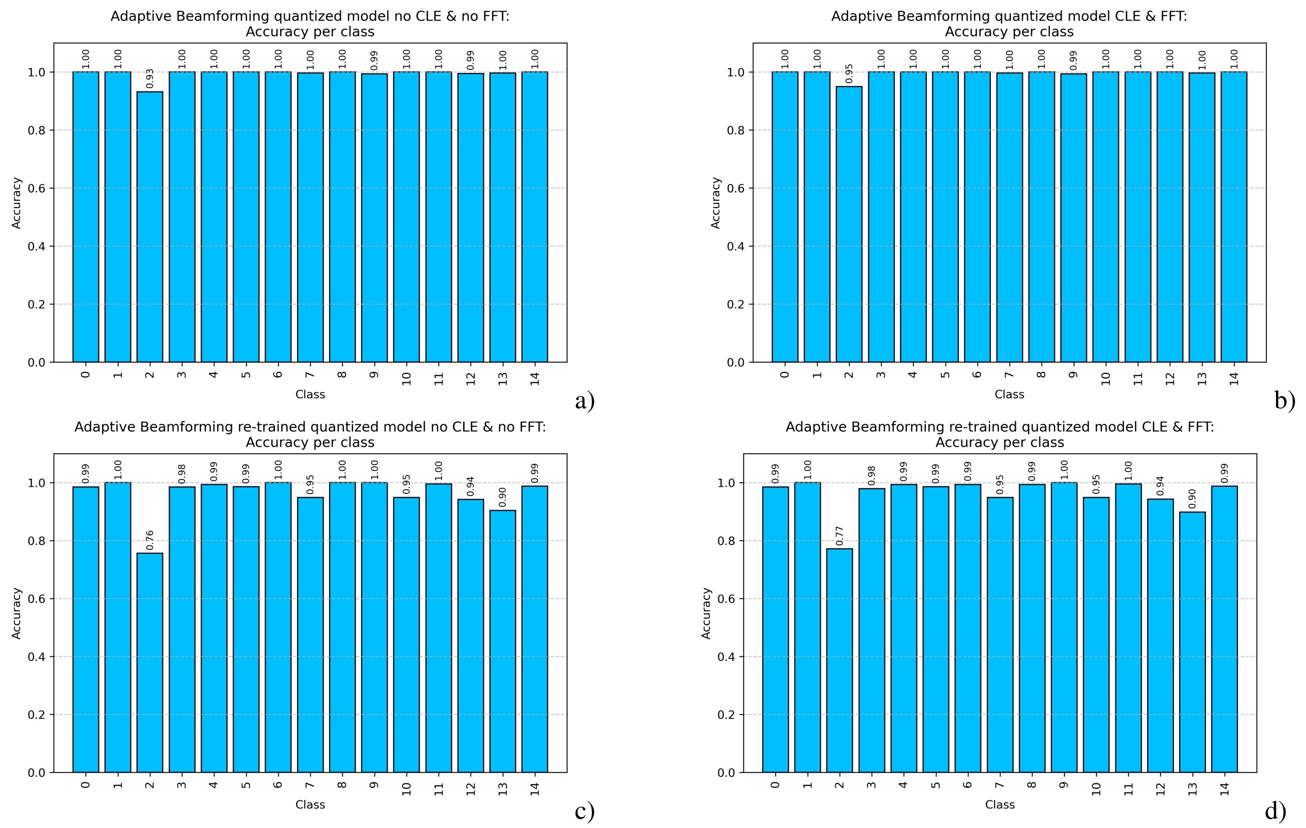

Fig. 6: Per-class accuracy of the adaptive beamforming model under different configurations: a) without optimization, b) with CLE and FFT, c) re-trained without optimization, d) re-trained with CLE and FFT.

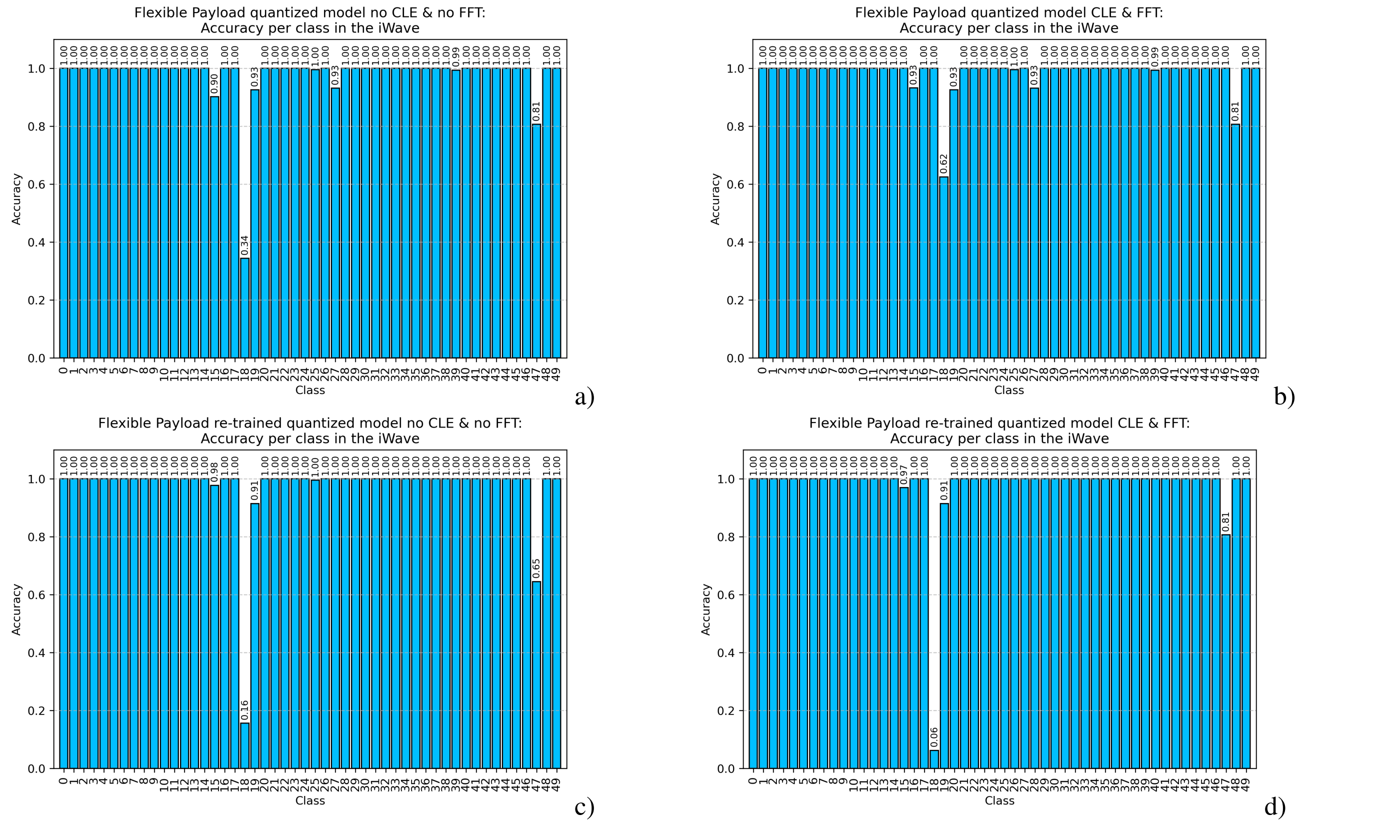

Fig. 7: Per-class accuracy of the flexible payload model onboard under different configurations: a) without optimization, b) with CLE and FFT, c) re-trained without optimization, d) re-trained with CLE and FFT.

TABLE IV: Performance report of Adaptive Beamforming model: Accuracy, loss and inference time onboard.

Model	Type	Opt.	Accuracy	Loss	Time
ABeam	Quant.	None	0.96537	0.03463	0.00018
		CLE & FFT	0.96548	0.03452	0.00018
	Retrain.	None	0.96165	0.03835	0.00018
		CLE & FFT	0.96086	0.03914	0.00018

while the loss is below 0.04. The per-class accuracy in Figure 8 shows similar variations across the four versions, with only one class below 0.90 (class 2 with a minimum of 0.76). This reduction compared with the PC quantized results is partially compensated in the re-trained version with optimization, which achieves the best performance. The inference time is similar for all versions.

III. System integration and execution

The execution of the models for the onboard payload is managed by standalone software that interacts with the Scenario Generator to provide the model inputs and receive the results via TCP sockets, as illustrated in Figure 9. The Scenario Generator transmits a new set of inputs to the machine learning model and waits for its response. When the processor in the Versal device detects the new input data set in the TCP socket, it is read and processed by executing the two machine learning algorithms in the DPU, which leverages the AI Engines (AIE) integrated into the Versal architecture to accelerate computation. After execution, the results are sent back to the Scenario Generator, which reads and processes them. This cycle is performed once every second, when new values are sent to the ML algorithms.

On the Versal side, after the *TCP read* operation, the traffic demand (R), corresponding to the coverage area, is normalized and provided as input to the *Flexible Payload* model. An *arg.max* function is then applied to the model output to map the arbitrary class scores (FP_{class}), thereby replacing the *softmax* activation in the original model's final layer. The resulting class is translated into seven $EIRP$s, beamwidths (θ^{Az}_{-3dB} and θ^{El}_{-3dB}), and bandwidths (BW), through a look-up table (*FPayl. Class LUT*).

In the second step, the per-beam $EIRP_i$ and beamwidths ($\theta^{Az}_{-3dB_i}$ and $\theta^{El}_{-3dB_i}$) are combined with the satellite position (Az, El) and the required minimum Side Lobe Level (SLL_{min}), obtained from the Scenario Generator. These inputs are normalized and provided to the *Adaptive Beamforming* model. The execution of this model produces arbitrary scores, which are converted into a class ($ABclass_i$) using the *arg.max* operation and decoded into the beamforming coefficient activation matrix for each beam ($|W_i|$) by a look-up table (*Abeam. Class LUT*). Finally, the coefficient activation matrix is combined with the beamforming phasors ($e_i^{j\theta_W}$) from the Scenario Generator (*Coefficient Generator*), yielding the complete beamforming coefficients for the actual beam (W_i).

This procedure is repeated for all beams, and the resulting beamforming coefficients (W), the per-beam bandwidths (BW), and the classes are returned to the Scenario Generator

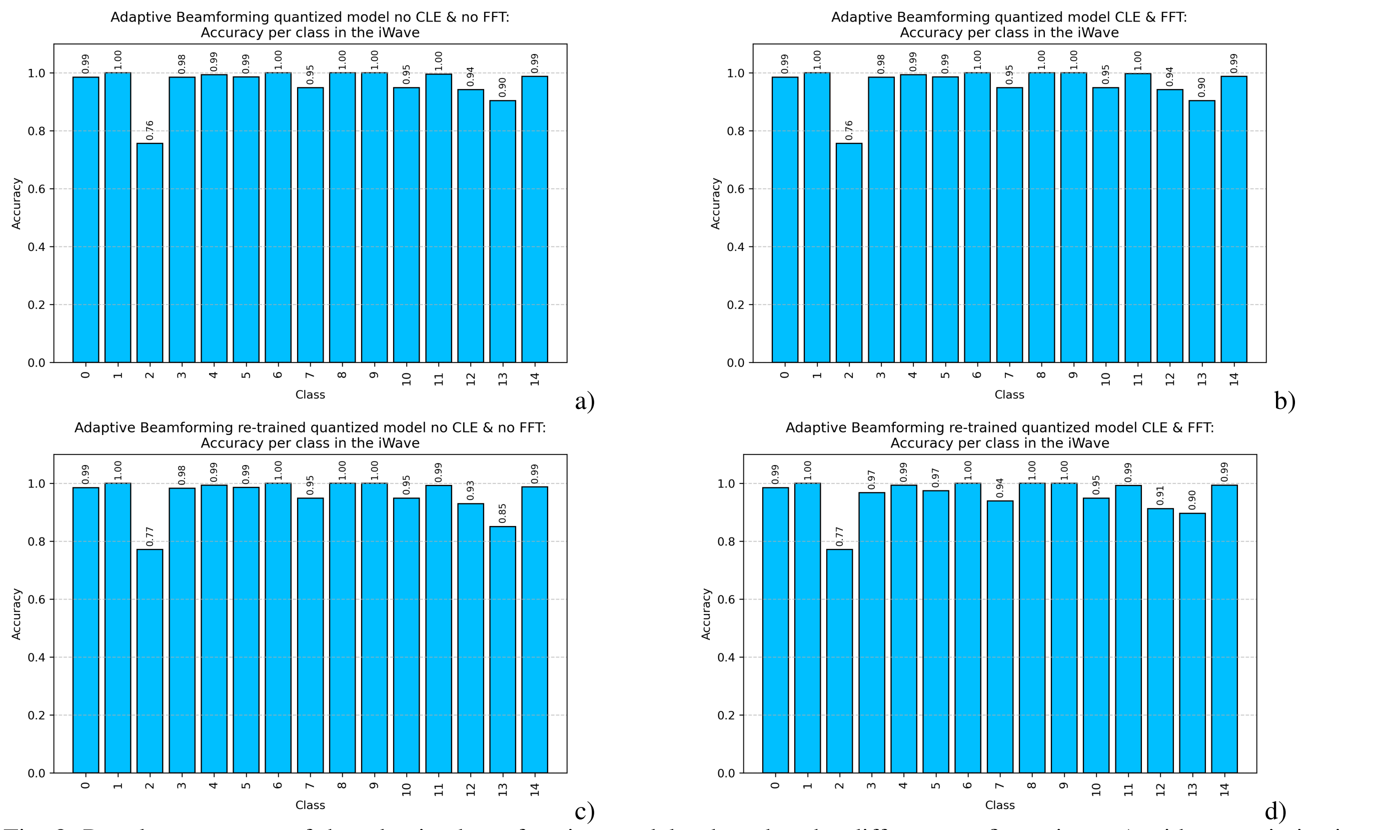

Fig. 8: Per-class accuracy of the adaptive beamforming model onboard under different configurations: a) without optimization, b) with CLE and FFT, c) re-trained without optimization, d) re-trained with CLE and FFT.

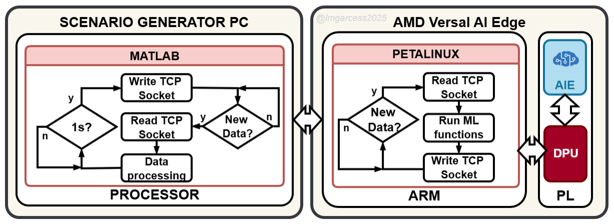

Fig. 9: Scenario Generator and onboard AI Payload integration.

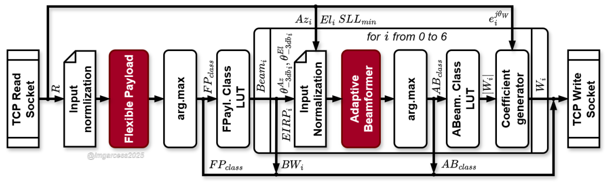

Fig. 10: Onboard AI Payload algorithm.

via a *TCP write* operation. The blocks highlighted in red in Figure 10 are executed in the DPU using the AIE, while the remaining blocks are executed in the board processor.

The average performance per stage for the entire system is shown in Table V after 10000 executions. Once the data is received, the flexible payload operation lasts 6.45 ms, where the highest delay corresponds to the inference process.

Comparing the execution of the original model on a workstation PC, powered by a dual Intel Xeon Gold 6230 CPU with a total of 80 processing cores up to 2.1 GHz and 512 GB

TABLE V: Performance report: Complete AI/ML algorithm execution

Stage	Operation	Time iWave	Time PC
Communication	Data reception	0.01470	0.00000
Flexible Payload	Data scaling	0.00280	0.00188
	Inference	0.00356	0.11688
	arg.max	0.00006	0.00000
	Class LUT search	0.00003	0.00063
Adaptive Beamform.	Data scaling	0.00007	0.00009
	Inference	0.00026	0.09991
	arg.max	0.00004	0.00005
	Class LUT search	0.00002	0.00000
	Coeff. generation	0.00005	0.00027
Communication	Data transfer	0.00010	0.00000
	Total	0.02434	0.82125

DDR4 RAM, the onboard implementation is more than 18 times faster. This improvement is due to the optimizations applied and the use of hardware accelerators in the onboard implementation, as well as the PC's processing load. On the other hand, the adaptive beamforming requires 441 μs, which is more than 227 times faster than the execution on the PC.

The inference time reported on both models is higher than in the standalone execution. This increment is a result of the additional overhead incurred when switching models on the DPU.

Considering the execution of adaptive beamforming seven times, the total onboard processing latency is 24.34 ms, representing an efficiency gain of more than 33 times compared to the PC implementation. This latency is compatible with coarse control loops, including satellite scenarios [18], [19]. From an absolute perspective, the system can adapt to changes in traffic demand in under one second, while ensuring a pointing error of less than 1°.

IV. Conclusions & Future Work

This work has presented the design, optimization, quantization, and hardware deployment of two machine learning models for onboard satellite payload processing. Several layers are inefficient for hardware acceleration due to their high parameter count and memory demands. Those layers need to be replaced to reduce the number of model parameters and improve the execution feasibility in the DPU after quantization.

Re-training the quantized models has improved the accuracy of specific underperforming classes, but it can also introduce degradations in other classes. These errors could lead to suboptimal resource allocation, such as incorrect beam assignment, reduced spectral efficiency, or transient underserved regions.

After model compilation and deployment in the DPU, the overall accuracy remains high, but reductions have been observed in some classes compared with the float and quantized versions executed on the PC.

The models have been successfully executed on the Versal ACAP AI platform, achieving an execution time that enables updating the beamforming coefficients and bandwidth 40 times per second. This demonstrates a significant acceleration compared to execution on a workstation PC.

For future work, additional data will be generated for the low-accuracy classes, and the models will be retrained to improve their performance. At the same time, the full AISTT is under development to enable simulation of different mission scenarios and hardware configurations, providing valuable insights and ensuring that future satellite missions can be further optimized for performance and efficiency.

References

[1] GMV, Centre Tecnològic de Telecomunicacions de Catalunya, Reply, Eutelsat, and European Space Agency, "Machine Learning and Artificial Intelligence for Satellite Communication (SATAI)," p. 1, 2020. [Online]. Available: https://connectivity.esa.int/projects/satai

[2] Joanneum Research, Inmarsat Navigation Ventures Ltd, Graz University of Technology, and European Space Agency, "Machine Learning and Artificial Intelligence for Satellite Communication (MLSAT)," p. 1, 2020. [Online]. Available: https://connectivity.esa.int/projects/mlsat

[3] F. Ortíz, V. Monzón Baeza, L. M. Garcés-Socarrás, J. A. Vásquez-Peralvo, J. L. González Rios *et al.*, "Onboard Processing in Satellite Communications Using AI Accelerators," *Aerospace*, vol. 10, no. 2, p. 101, 1 2023. [Online]. Available: https://www.mdpi.com/2226-4310/10/2/101

[4] G. Fontanesi, F. Ortiz, E. Lagunas, L. M. Garces-Socarras, V. M. Baeza *et al.*, "Artificial Intelligence for Satellite Communication: A Survey," *IEEE Communications Surveys and Tutorials*, 2025.

[5] L. M. Garcés-Socarrás, A. Nik, F. Ortiz, J. A. Vásquez-Peralvo, J. L. Gonzalez *et al.*, "Artificial Intelligence Satellite Telecommunication Testbed using Commercial Off-The-Shelf Chipsets," in *The First Joint European Space Agency / IAA Conference on AI in and for Space (SPAICE2024)*, D. Dold, A. Hadjiivanov, and D. Izzo, Eds., ESA European Centre for Space Applications and Telecommunications (ECSAT). Harwell, UK: European Space Agency (ESA), 9 2024, pp. 169–174. [Online]. Available: https://zenodo.org/records/13885551

[6] R. Davidson, A. H. Diaz, E. Simons, S. Hadfield, C. Bridges *et al.*, "The Development of an Onboard Processing Environment within the Flexible and Intelligent Payload Chain Sub-system for Small EO Satellites," *Proceedings of the 2023 European Data Handling and Data Processing Conference for Space, EDHPC 2023*, 2023.

[7] P. G. Shenwai, A. Choudhary, T. Pokuri, A. Basak, M. Manikandan *et al.*, "On the Role of Artificial Intelligence in Aerospace Engineering: Current State of the Art and Future Trajectories," *The Aeronautical Journal*, pp. 1–27, 2025. [Online]. Available: https://www.cambridge.org/core/journals/aeronautical-journal/article/on-the-role-of-artificial-intelligence-in-aerospace-engineering-current-state-of-the-art-and-future-trajectories/A69D4FABFC73CDE81FA0F4F79A4FBC5F

[8] S. Kashyap and N. Gupta, "Resource Allocation Techniques in Multibeam Satellites: Conventional Methods vs. AI/ML Approaches," *International Journal of Satellite Communications and Networking*, vol. 43, no. 2, pp. 97–121, 3 2025. [Online]. Available: https://onlinelibrary.wiley.com/doi/full/10.1002/sat.1548 https://onlinelibrary.wiley.com/doi/abs/10.1002/sat.1548 https://onlinelibrary.wiley.com/doi/10.1002/sat.1548

[9] Interdisciplinary Centre for Security Reliability and Trust (SnT) and European Space Agency (ESA), "Satellite Signal Processing Techniques using a Commercial Off-The-Shelf AI Chipset (SPAICE)," p. 1, 2022. [Online]. Available: https://connectivity.esa.int/projects/spaice

[10] E. Godino, L. Escolar, and A. P. Honold, "Flexible Payload Operations of Satellite Communication Systems," in *2018 SpaceOps Conference*. Marseille, France: American Institute of Aeronautics and Astronautics, may 2018, p. 9. [Online]. Available: www.gmv.com https://arc.aiaa.org/doi/10.2514/6.2018-2653

[11] F. Vidal, H. Legay, G. Goussetis, M. Garcia Vigueras, S. Tubau *et al.*, "A methodology to benchmark flexible payload architectures in a megaconstellation use case," *International Journal of Satellite Communications and Networking*, vol. 39, no. 1, pp. 29–46, 1 2021. [Online]. Available: https://onlinelibrary.wiley.com/doi/full/10.1002/sat.1344 https://onlinelibrary.wiley.com/doi/abs/10.1002/sat.1344 https://onlinelibrary.wiley.com/doi/10.1002/sat.1344

[12] J. A. Vásquez-Peralvo, J. Querol, F. Ortíz, J. L. González Rios, E. Lagunas *et al.*, "Flexible Beamforming for Direct Radiating Arrays in Satellite Communications," *IEEE Access*, vol. 11, no. August, pp. 79 684–79 696, 2023. [Online]. Available: https://ieeexplore.ieee.org/document/10197375/

[13] F. Ortiz, J. A. Vasquez-Peralvo, J. Querol, E. Lagunas, J. L. Gonzalez Rios *et al.*, "Supervised Learning Based Real-Time Adaptive Beamforming On-board Multibeam Satellites," in *18th European Conference on Antennas and Propagation (EuCAP)*. Glasgow, United Kingdom: IEEE, 3 2024, pp. 1–5. [Online]. Available: https://arxiv.org/abs/2311.01334

[14] J. A. Vasquez-Peralvo, J. Querol, F. Ortiz, J. L. Gonzalez-Rios, E. Lagunas *et al.*, "MultiBeam Beamforming for Direct Radiating Arrays in Satellite Communications Using Genetic Algorithm," *IEEE Open Journal of the Communications Society*, 2024.

[15] L. M. Garcés-Socarrás, A. Nik, F. Ortiz, J. A. Vásquez-Peralvo, J. Luis *et al.*, "Artificial Intelligence implementation of onboard flexible payload and adaptive beamforming using commercial off-the-shelf devices," in *5th ESA Workshop on Advanced Flexible Telecom Payloads*, ESA. Didcot, United Kingdom: ESA, may 2025, p. 8. [Online]. Available: https://arxiv.org/abs/2505.01853v1

[16] AMD Inc., "Vitis AI User Guide," AMD Inc., Tech. Rep., 9 2023. [Online]. Available: https://docs.amd.com/r/en-US/ug1414-vitis-ai/Vitis-AI-Overview

[17] Y. Fukuda, K. Yoshida, and T. Fujino, "Evaluation of Model Quantization Method on Vitis-AI for Mitigating Adversarial Examples," *IEEE Access*, vol. 11, pp. 87 200–87 209, 2023. [Online]. Available: https://ieeexplore.ieee.org/document/10216964

[18] J. L. Gonzalez-Rios, L. Martinez-Marrero, E. Lagunas, J. Krivochiza, L. M. Garcés-Socarrás *et al.*, "Doppler Shift in Precoded Cooperative Multi-Gateway Satellite Systems: Effects and Mitigation," in *IEEE Wireless Communications and Networking Conference (WCNC 2024)*. Dubai, United Arab Emirates: IEEE, 4 2024.

[19] L. Martinez Marrero, J. C. M. Duncan, J. L. Gonzalez, J. Krivochiza, S. Chatzinotas *et al.*, "Accurate Phase Synchronization for Precoding-Enabled GEO Multibeam Satellite Systems," *IEEE Open Journal of the Communications Society*, vol. 5, pp. 712–729, 2024.