The Journal of Supercomputing (2025) 81:270
https://doi.org/10.1007/s11227-024-06795-2

Optimizing Millimeter Wave MIMO Channel Estimation Through GPU-Based Edge Artificial Intelligence

Diego Lloria[1] · Sandra Roger[1] · Germán León[2] · José M. Badía[2] · Carmen Botella-Mascarell[1] · Jose A. Belloch[3]

Accepted: 28 November 2024

Abstract

In the context of upcoming sixth-generation (6G) wireless communication systems, the use of millimeter wave (mmWave) frequencies is a key technology for achieving high-throughput communications. Accurate parametric estimation of mmWave channels is critical for effective beamforming design and configuration, requiring sophisticated models to capture the directional characteristics of these channels. This work considers an innovative artificial intelligence (AI) approach for accurate estimation of angle-of-arrival (AoA) and angle-of-departure (AoD) parameters from frequency-domain channel observations. Our approach is based on the implementation of two convolutional neural networks (CNNs): a residual CNN (ResNet) and a U-Net CNN. Specifically, this work focuses on the efficient implementation of both schemes in an embedded system suitable for edge AI. We performed the experiments in a low-power NVIDIA Jetson Orin Nano platform and evaluated the effect of modifying the frequencies of its CPU and GPU on the performance of the inference process, both in terms of execution time and energy consumption. Experimental results showed that the U-Net model is more power consuming, but as it is faster, it consumes less energy per channel.

Keywords Artificial intelligence · Edge computing · Graphic processing units · MIMO communication systems · Channel estimation

Extended author information available on the last page of the article

Published online: 10 December 2024

1 Introduction

The proliferation of supercomputers with parallel computing capabilities has revolutionized several industries [1, 2]. Modern supercomputers typically consist of nodes containing multiple multicore central processing units (CPUs) and one or more graphic processing units (GPUs).[1] These architectures allow computationally intensive problems to be tackled efficiently by utilizing both types of processing elements, often through hybrid CPU-GPU implementations. This requires custom analysis to identify bottlenecks and allocate tasks to the most appropriate computing resource to maximize overall performance. One area that has benefited significantly from these advances is digital signal processing for wireless communications, particularly in the implementation of efficient multiple-input multiple-output (MIMO) algorithms [3, 4].

MIMO techniques have been widely employed since the third generation (3G) of mobile communications. In the current fifth generation (5G) and future sixth generation (6G), the use of MIMO has expanded to deployments referred to as massive MIMO, which includes tens or hundreds of antenna elements. A second major improvement is the use of higher frequency bands, as it is the case of millimeter wave (mmWave) spectrum, which is a key technology for achieving enhanced mobile broadband (eMBB) services [5, 6]. The deployment of mmWave requires the use of beamforming techniques with highly directional beams to increase the gain of the communication link between the transmitter (Tx) and receiver (Rx) to compensate for the larger channel propagation losses. The high directivity is typically achieved by using massive MIMO systems, which often require more complex signal processing techniques, highlighting the need for efficient implementations. This study focuses on this specific scenario, a massive MIMO system using a mmWave frequency band, where achieving an efficient and robust channel estimation is critical.

Accurate parametric modeling of mmWave channels allows the implementation of beamforming techniques. In this case, the directional characteristics of the channel are captured through the estimation of angle-of-arrival (AoA) and angle-of-departure (AoD) parameters. In this work, we focus on an artificial intelligence (AI) approach using two convolutional neural networks (CNNs) to accurately retrieve AoA and AoD values from frequency-domain channel observations in an analog beamforming scenario, which exploit the sparse nature of the mmWave channel. In particular, the estimation is addressed as a supervised image-to-image translation problem, where the aim is to transfer an image from one domain to another while preserving the content of the given image. Our approach builds upon the framework proposed in [7], where the 2D input matrix can be interpreted as the discrete Fourier transform (DFT) of a sum of 2D complex sinusoids defined by the existing channel paths. This matrix, which can be seen as an input image, is fed into AI-based methods for signal denoising and peak enhancement,

[1] http://top500.org.

producing as output a real image that encodes the underlying paths in the main peaks. The first AI architecture evaluated in this paper is a residual CNN, denoted as ResNet, based on the *2D-DeepFreq* architecture presented in [8], which is an adaptation of the residual CNN proposed in [9]. The second evaluated architecture is based on a well-known CNN architecture called U-Net, which has been widely used to solve problems of semantic segmentation [10], audio source separation [11], etc.

GPUs have become the preferred choice for deploying deep neural networks (DNNs) in both training and inference due to their high computational power and flexibility. In the context of mmWave MIMO systems, where data parallelism and efficient handling of intensive matrix operations are essential, GPUs offer significant advantages. The ability of GPUs to process multiple data points in parallel enables rapid and accurate channel estimation. In this contribution, the focus of the evaluation is on the efficient implementation of the CNN models in an embedded system. We evaluate the effect of modifying the frequencies of its CPU and GPU on the performance of the inference process, both in terms of execution time and energy consumption. These parameters are important metrics in the context of 6G systems, where substantial effort is located into placing sustainability in the center of the architecture design and future implementation [12, 13]. In fact, obtaining energy-efficient and low-complexity AI solutions is a challenge for future sustainable 6G systems.

A key technology trend toward achieving sustainable communication systems [14] is edge computing, and more specifically edge AI, which besides has emerged as a promising solution to the challenges of real-time data processing in wireless communication systems [15–17]. By processing data at the edge of the network, close to the data source, edge AI reduces latency and bandwidth consumption, enabling real-time decision making and analysis. In our contribution, the low-power NVIDIA Jetson Orin Nano [18] exemplifies this paradigm shift with its advanced GPU architecture and energy-efficient design. This low-power system on a chip (SoC) incorporates Tensor Cores optimized for DNN processing, allowing it to meet the real-time demands of edge AI applications with a high performance-per-watt ratio, essential for energy-efficient operation in 5G and 6G environments. While GPUs provide a robust solution for edge computing tasks, alternative hardware platforms such as FPGAs and specialized AI accelerators offer promising future directions. FPGAs, for instance, allow for customized configurations that can potentially optimize latency and energy efficiency in specific DNN applications. It could be interesting to assess their feasibility in scenarios where extreme customization or ultralow power consumption is required. However, given the GPU's current advantages in flexibility, widespread support, and optimization for DNNs, it remains the most suitable solution for our present study.

The main contributions of this work are the following:

– The paper showcases the deployment of two deep learning models for mmWave MIMO channel estimation, specifically, a ResNet and a U-Net, on a low-power embedded system (NVIDIA Jetson Orin Nano), showcasing the feasibility and benefits of edge AI for 6G networks.

- This study provides a detailed comparison of energy consumption between the ResNet and U-Net models, revealing that, despite a higher power usage, U-Net's faster execution time leads to a superior overall energy efficiency, a crucial consideration for sustainable edge AI deployments.
- The research highlights the trade-offs between power consumption and execution speed, offering valuable insights into optimizing AI model selection for resource-constrained environments in future wireless communication systems.
- By implementing the models on an actual edge computing platform, the paper demonstrates the practical challenges and solutions when implementing AI-based channel estimation.

Experimental results show that the U-Net model is consistently faster than the ResNet model for all CPU and GPU frequencies. Both models behave similarly regarding their execution time and power consumption as we modify the frequencies of both devices. The U-Net model is more power consuming, but as it is faster, it consumes less energy per channel. Specifically, the U-Net model is about 3.5 times faster than the ResNet independently of the frequencies and consumes about 1.5 times more power. As a result, in the best case, the ResNet model consumes 0.59 joules per channel, while the U-Net only consumes 0.30 joules per channel. Our results highlight the trade-offs between computational performance and power efficiency, providing valuable insights for optimizing AI applications on low-power embedded systems.

The remainder of the paper is structured as follows. Section 2 provides an overview of the system model and background. In Sect. 3, we describe the proposed deep learning models, including the architecture of the ResNet and U-Net, and their implementation on the NVIDIA Jetson Orin Nano platform. Next, in Sect. 4, we detail the experimental setup, evaluation metrics, and the results of our performance and energy efficiency analysis. Finally, Sect. 5 offers concluding remarks, summarizing the key findings of the study.

2 System Model

2.1 Channel Model

Consider a single-user mmWave MIMO geometric channel where both the Tx and Rx are equipped with uniform linear arrays consisting of n_t and n_r antenna elements, respectively. Let L represent the number of scatterers, with each scatterer contributing a single propagation path between Tx and Rx. The complex channel coefficient for each path is given by α_l, $l = 1, \ldots, L$, while ψ_l and ϕ_l denote the AoA and AoD, respectively. The full parametric model expresses the channel as [19]:

$$\mathbf{H}(\boldsymbol{\theta}) = \sqrt{n_t n_r} \sum_{l=1}^{L} \alpha_l \mathbf{a}_r(\psi_l)\mathbf{a}_t^H(\phi_l), \tag{1}$$

where $\boldsymbol{\theta} \triangleq [|\alpha_1|, \angle\alpha_1, \phi_1, \psi_1, \ldots, |\alpha_L|, \angle\alpha_L, \phi_L, \psi_L]^T$ is the parameter vector. Here, $|\alpha_l|$ and $\angle\alpha_l$ represent the magnitude and phase of each channel coefficient,

respectively. The α_l values are independent identically distributed (i.i.d.) random variables with the distribution $\alpha_l \sim \mathcal{CN}(0, \sigma_\alpha^2/L)$. The AoAs ($\psi_l$) and AoDs ($\phi_l$) are uniformly distributed within $[0, 2\pi]$.

The array responses for the Tx and Rx antennas, assuming half-wavelength antenna spacing, are given by:

$$\mathbf{a}_t(\phi_l) = \frac{1}{\sqrt{n_t}}[1,\ e^{-j\pi\cos\phi_l}, \cdots, e^{-j\pi(n_t-1)\cos\phi_l}]^T, \tag{2}$$

$$\mathbf{a}_r(\psi_l) = \frac{1}{\sqrt{n_r}}[1,\ e^{-j\pi\cos\psi_l}, \cdots, e^{-j\pi(n_r-1)\cos\psi_l}]^T. \tag{3}$$

Estimating the channel $\mathbf{H}(\boldsymbol{\theta})$ is equivalent to estimating the parameters in $\boldsymbol{\theta}$. Measurements have shown that mmWave channels exhibit significant sparsity [20]. Consequently, the L paths are typically well-separated, which simplifies the task of channel estimation.

2.2 Pilot-Based Training Phase

The initial phase is the open-loop pilot-based training phase. The beam search space is defined by a codebook containing P and Q code words/directions at the Tx and Rx sides, respectively. After transmitting the pilot symbol through the $Q \times P$ direction combinations, the observation matrix is formed.

The process works as follows: a pilot symbol s, known to both Tx and Rx, is transmitted and received through a subset of $P \leq N_{max}$ and $Q \leq N_{max}$ spatial directions, respectively. Here, N_{max} denotes the maximum number of angle quantization levels, limited by the realistic phase shifters' angle resolution. Since the beamforming scenario is analog, the Tx and Rx each have a single radio frequency chain, so beamforming and combining operations are performed in the analog domain [7].

Beamforming/combining vectors are computed to match the channel response [21], thus $\mathbf{f} = \mathbf{a}_t(\bar{\phi}_p)$ for $p = 0, 1, \ldots, P-1$, and $\mathbf{w} = \mathbf{a}_r(\bar{\psi}_q)$ for $q = 0, 1, \ldots, Q-1$. For each $\{q, p\}$ direction pair, the received signal is given by

$$\mathrm{y}_{q,p} = \sqrt{\rho}\,\mathbf{w}_q^H \mathbf{H}\mathbf{f}_p\,\mathrm{s} + \mathbf{w}_q^H \mathbf{n}, \tag{4}$$

where $\rho \in \mathbb{R}^+$ represents the transmit power. The noise term $\mathbf{n} \sim \mathcal{CN}(0, \boldsymbol{\Sigma}_\mathbf{n})$ is a complex additive white Gaussian noise vector of size $1 \times n_r$ with covariance $\boldsymbol{\Sigma}_\mathbf{n} = \sigma_n^2 \mathbf{I}_{n_r}$, where $\mathbf{I}_{n_r}$ is the $n_r \times n_r$ identity matrix. During training, the symbol s is set to 1 for simplicity, making the system signal-to-noise ratio (SNR) ρ/σ_n^2.

The observation matrix, formed by transmitting the pilot symbols through the $Q \times P$ directions, is given by

$$\mathbf{Y} = \begin{bmatrix} y_{0,0} & y_{0,1} & \cdots & y_{0,P-1} \\ y_{1,0} & y_{1,1} & \cdots & y_{1,P-1} \\ \vdots & \vdots & \ddots & \vdots \\ y_{Q-1,0} & y_{Q-1,1} & \cdots & y_{Q-1,P-1} \end{bmatrix} = \sqrt{\rho}\,\mathbf{G}(\boldsymbol{\theta}) + \mathbf{N}. \tag{5}$$

The noise matrix $\mathbf{N} \in \mathbb{C}^{Q\times P}$ contains i.i.d. elements $\sim \mathcal{CN}(0, \sigma_n^2)$, and $\mathbf{G} \in \mathbb{C}^{Q\times P}$ encodes the channel parameter vector $\boldsymbol{\theta}$.

To separate the impact of different path components, the elements in the observation matrix are rearranged as follows:

$$\mathbf{Y} = \sqrt{\rho} \sum_{l=1}^{L} \mathbf{G}^{(l)}(\boldsymbol{\theta}_l) + \mathbf{N}. \tag{6}$$

In this expression, the observation matrix is written as a sum of path contributions $\mathbf{G}^{(l)}(\boldsymbol{\theta}_l) \in \mathbb{C}^{Q\times P}$, each depending on a parameter vector

$$\boldsymbol{\theta}_l \;=\; [|\alpha_l|, \angle\alpha_l, \phi_l, \psi_l]^T. \tag{7}$$

The problem of estimating the AoA and AoD is approached by searching for spectral peaks in the 2D image formed by the values in the observation matrix $\mathbf{Y}$. Specifically, the estimation is treated as a supervised image-to-image translation problem, where images from one domain are transformed to exhibit the characteristics of images from another domain.

3 Deep-Learning-Based AoA/AoD Estimation

The procedure for estimating the AoA and AoD follows a three-step process. Initially, in a preprocessing phase, the complex-valued observation matrix $\mathbf{Y}$ is separated into its real and imaginary components, denoted as $\mathbf{Y}_{\Re} \in \mathbb{R}^{Q\times P}$ and $\mathbf{Y}_{\Im} \in \mathbb{R}^{Q\times P}$, respectively. To enlarge their dimensions to $Q' = \beta Q$ and $P' = \beta P$, nearest-neighbor interpolation [22] is applied, resulting in the new matrices $\bar{\mathbf{Y}}_{\Re} \in \mathbb{R}^{Q'\times P'}$ and $\bar{\mathbf{Y}}_{\Im} \in \mathbb{R}^{Q'\times P'}$:

$$\bar{\mathbf{Y}}_{\Re} = \mathcal{I}(\mathbf{Y}_{\Re}),$$
$$\bar{\mathbf{Y}}_{\Im} = \mathcal{I}(\mathbf{Y}_{\Im}),$$

where $\mathcal{I}(\cdot)$ represents the interpolation function and $\beta \in \mathbb{Z}_+$ is a design parameter. In this work, interpolation with $\beta \in \{2, 4\}$ has been applied to set $Q' = P'$ (input matrix of size 64×64). By interpolating all inputs to a uniform size, we ensured that a single network model could be used across different antenna setups, simplifying the overall implementation.

After this preprocessing, the domain consists of the matrices $\bar{\mathbf{Y}}_{\Re}$ and $\bar{\mathbf{Y}}_{\Im}$ from which we aim to estimate the AoAs and AoDs. Consequently, the network input is of size $Q' \times P' \times 2$. Next, these matrices are fed into the NN architectures, obtaining an output matrix (or image) $\mathbf{Z} \in \mathbb{R}^{M\times N}$. The parameters M and N are chosen as integer multiples of the input dimensions Q' and P'. In the last step, the output

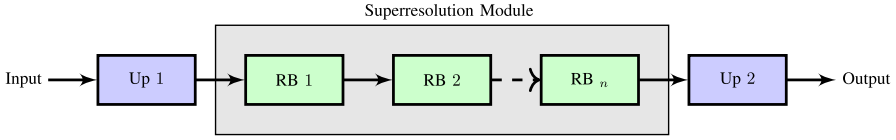

Fig. 1 ResNet architecture

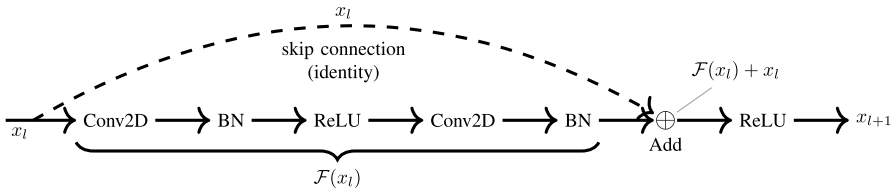

Fig. 2 Residual block (RB)

images are post-processed to find the most prominent peaks. In the following, we present the ResNet and U-Net models, as well as the algorithm used in the post-processing step for peak detection and the training configuration for the NNs.

3.1 ResNet Model

The first approach utilizes a residual CNN denoted as ResNet, which is an adaptation of the *2D-DeepFreq* architecture introduced in [9] (refer to Fig. 1). This modified ResNet integrates transposed convolution layers to achieve super-resolution. The goal is to address the AoA and AoD estimation problem by identifying the complex sinusoids present in the input complex-valued observation matrix $\mathbf{Y}$. Unlike the *2D-DeepFreq* model, our input $\mathbf{Y}$ already represents a frequency-domain sum of multiple sinusoids. Therefore, the matched filtering module found in *2D-DeepFreq* before the upsampling stage has been omitted. Let w_n and h_n represent the width and height of the output feature map at the n-th layer, respectively. The number of output channels at the n-th layer is denoted by c_n, the number of filters by f_n, and the length of the filter kernels by k_n (assuming they are square). All filters used in the convolutions have a consistent size of $k_n = k = 5$, as detailed in [9]. The input consists of the interpolated real and imaginary components of the observation matrix, namely $\bar{\mathbf{Y}}_{\Re}$ and $\bar{\mathbf{Y}}_{\Im}$. These components are processed through an upsampling stage (Up 1) using a transposed 2D convolutional layer with a $k \times k$ filter kernel (ConvT_k) and a stride of 2.

Then, the image enters the super-resolution module which iteratively stacks 64 residual blocks (RBs), as described in [9]. As shown in Fig. 2, RBs follow the Conv_k-BN-ReLU order, where Conv_k stands for a 2D-convolution with a filter with dimensions $k \times k$, ReLU is the rectified linear unit function and BN stands for batch normalization. For a generic input x to any RB, the output is given by:

$$\mathrm{RB}(x) := \mathrm{ReLU}(\mathcal{F}(x) + x),$$

where

$$\mathcal{F}(x) := \mathrm{BN}(\mathrm{Conv}_k(\mathrm{ReLU}(\mathrm{BN}(\mathrm{Conv}_k(x))))).$$

Finally, another transposed convolutional layer (ConvT_k) with a stride of 2 is applied for a second upsampling (Up 2). The output of the network is a matrix $\mathbf{Z} \in \mathbb{R}^{M \times N}$, where M and N are selected as integer multiples of the input dimensions Q' and P'. In this work we set a minimum upsampling factor of 2.

3.2 U-Net Model

The second proposed method is based on the well-known U-Net architecture (see Fig. 3), which was originally created for biomedical image segmentation tasks [10]. U-Net is a fully CNN designed for applications involving large amounts of data and high variability. Its architecture, characterized by a "U" shape, includes a contracting path to capture context and a symmetric expanding path to enable precise localization. In this study, we utilize the Wave-U-Net variant, which was introduced for audio source separation [11].

The core component of the U-Net model is the U-Net Block, depicted in Fig. 4. The U-Net Block consists of an encoder section (left) and a decoder section (right), which are composed of depth blocks (D-Blocks) and upsampling blocks (U-Blocks), respectively. The encoder is made up of five D-Blocks, defined as

$$\text{D-Block}(x) := \mathrm{MaxPool}(\mathrm{ConvBlock}(\mathrm{ConvBlock}(x))),$$

where x is the input tensor, and ConvBlock represents a standard convolutional block (with $k = 5$), defined as:

$$\mathrm{ConvBlock}(x) = \mathrm{ReLU}(\mathrm{BN}(\mathrm{Conv}_k(x))).$$

Between the encoder and decoder of the U-Net Block, there is a Dense Block. This Dense Block transforms an input tensor x with shape (B, H, W, C) into a 2D tensor with shape ($B, H \times W \times C$) using a flattening operator (Fl). The output is then passed through two dense layers and a final reshaping operator (Rshp) to fit the decoder input:

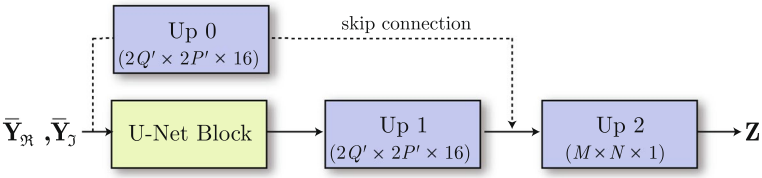

Fig. 3 U-Net overall scheme

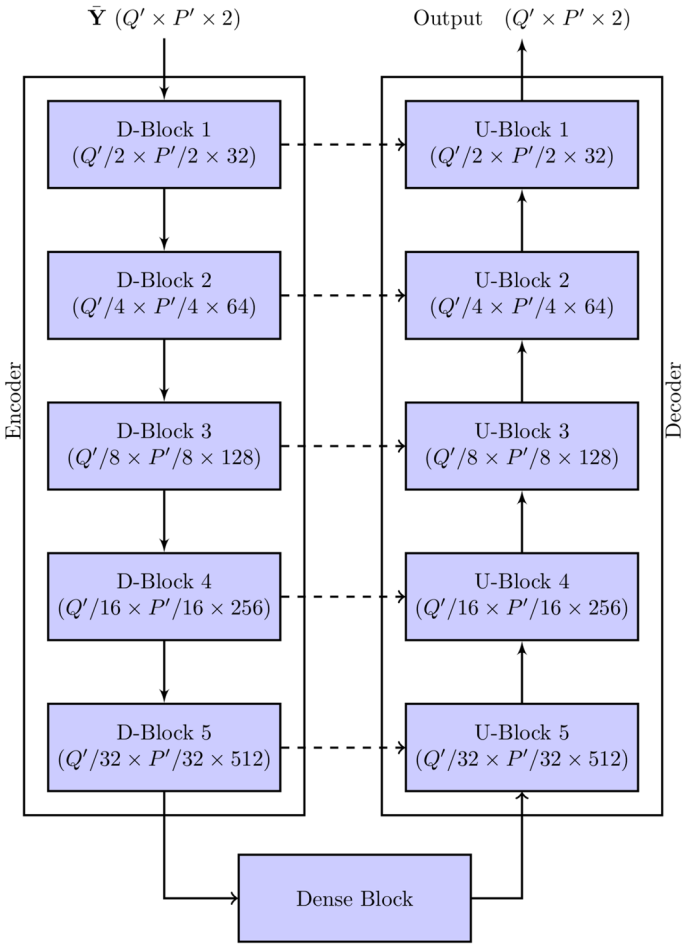

Fig. 4 U-Net block

Table 1 Model parameters (thousands) and size for both models

Model	Trainable	Non-trainable	Total	Size (MB)
ResNet	466,321	3072	469,393	1.79
U-Net	31,261,441	15,040	31,276,481	119.31

$$\text{DenseBlock}(x) = \text{ConvBlock}(\text{Rshp}(\text{Dense}_2(\text{Dense}_1(\text{Fl}(x))))).$$

Here, the number of neurons in Dense_2 is chosen to match the required size for the decoder input, while Dense_1 defines the bottleneck size, which is set to 1024. Similarly, the decoder contains five U-Blocks and a super-resolution layer. Each U-Block is composed of a ConvT_k layer, followed by batch normalization (BN) and ReLU activation, defined as:

$$\text{U-Block}_i(x) = \text{ConvBlock}(\text{ConvBlock}(\text{ConvT}_k(x \oplus y_i))),$$

where y_i represents the output of the corresponding D-Block in the encoder, and $\oplus$ indicates concatenation. Finally, the inputs and outputs of the U-Net block pass through the upsampling layers "Up 0" and "Up 1" before being concatenated and forwarded to the final "Up 2" layer, as illustrated in Fig. 3.

Table 1 shows the size of both models based on their trainable and non-trainable parameters. The U-Net model is much larger than the ResNet model, but it is still a small DNN that can be processed in a low-power SoC with 8GB of DDR memory.

3.3 Peak Detection

To detect the most prominent peaks in the image resulting from the output of either the ResNet or the U-Net, where the useful angular information is, we use the `SimpleBlobDetector` class, which implements the well-known blob detection method included in the OpenCV library.[2] Specifically, we first normalize the scale pixel values between 0 and 255, and then we apply a Gaussian blur filter to smooth the image and make the peak detection more reliable. To finish the preprocessing, we binarize the image to highlight the peaks. Then, we apply the blob detection method to detect the regions with higher average intensities and then sort them based on this parameter to get the most prominent ones [23].

3.4 Training Configuration

Similar to the method described in [9], during the training phase, the target for each channel in the dataset is a high-resolution matrix $\mathbf{X} \in \mathbb{R}^{M \times N}$, composed of the sum of L 2D Gaussian functions. The elements of the matrix $\mathbf{X}$, for $m = 0, \ldots, M-1$ and $n = 0, \ldots, N-1$, are calculated as follows:

$$\mathbf{X}_{m,n} = \frac{1}{2\pi\sigma_1\sigma_2} \sum_{l=1}^{L} e^{\left(\frac{(\omega_m - \tilde{\omega}_{\psi_l})^2}{2\sigma_1^2} + \frac{\left(\omega_n - \tilde{\omega}_{\phi_l}\right)^2}{2\sigma_2^2} \right)}, \tag{8}$$

where $\omega_m = m\frac{2\pi}{M}$, $\omega_n = n\frac{2\pi}{N}$, and $\tilde{\omega}_{\phi_l} = \mathcal{W}_{2\pi}(\omega_{\phi_l})$, $\tilde{\omega}_{\psi_l} = \mathcal{W}_{2\pi}(\omega_{\psi_l})$ denote the 2π -wrapped ground-truth frequencies. The standard deviations σ_1 and σ_2 determine the width of the Gaussian functions, with values set to $\sigma_1 = \sigma_2 = 0.2$. The models are trained by minimizing the mean square error (MSE) between the ground truth and the NN output, defined by the loss function:

$$\text{Loss} = \frac{1}{MN} \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} (\mathbf{Z}_{m,n} - \mathbf{X}_{m,n})^2. \tag{9}$$

[2] https://docs.opencv.org/.

4 Edge AI Implementation

4.1 Experimental Environment

For this investigation, we utilized Jetson Orin Nano modules, crafted using advanced 8nm technology, as our testing units [18]. Each module houses an Orin SoC that integrates a six-core ARM Cortex-A78AE CPU with 1.5 MB L2 and 4 MB L3 cache, alongside a GPU based on NVIDIA's Ampere architecture. This GPU includes four Streaming Multiprocessors, 512 CUDA cores, and 16 Tensor Cores. The system also includes 4GB of 64-bit LPDDR5 memory, providing a bandwidth of 34 GB/s, shared between the CPU and GPU. The modules operate in two power modes: a low-power 7-watt mode and a higher 10-watt mode for increased performance. Figure 5 depicts the main components of the Jetson Orin Nano modules, including its Orin SoC.

To measure power consumption accurately, we employed the `PMLIB` framework [24]. This system captures power data at 10 Hz from the device's integrated sensors. This power data is averaged over the duration of each test to evaluate the total power usage across three main components: the entire module, the combined load from the CPU, GPU, and other processing units like Computer Vision devices, and the memory subsystem along with other key components like the image signal processor.

Energy usage was calculated by multiplying their power consumption by the operational time during specific tasks, providing a measure in Joules (J). This approach allowed us to quantify the energy consumption necessary to evaluate the efficiency of the components associated to each sensor under different operational scenarios.

The board's power, thermal, and electrical management is supported by the NVIDIA board support package (BSP) [25]. This package includes CPU dynamic frequency scaling, which is managed by the `ondemand` governor based on Linux. This governor adjusts the CPU's frequency in response to real-time system demands, a strategy similarly employed for the GPU. This adaptive frequency scaling ensures that both the CPU and GPU modulate their operational frequencies based on their computational load, optimizing power efficiency. We can also set a fixed frequency both for the CPU and GPU that does not very during a given test. Our experiments

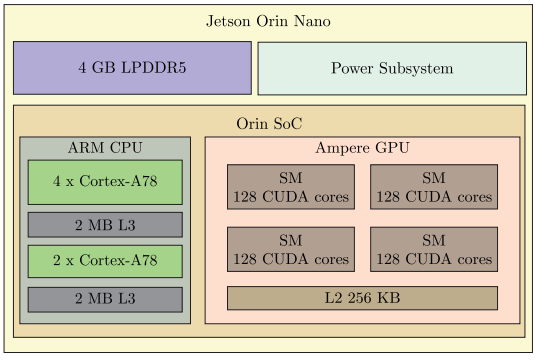

Fig. 5 Main components of the Jetson Nano Orin module

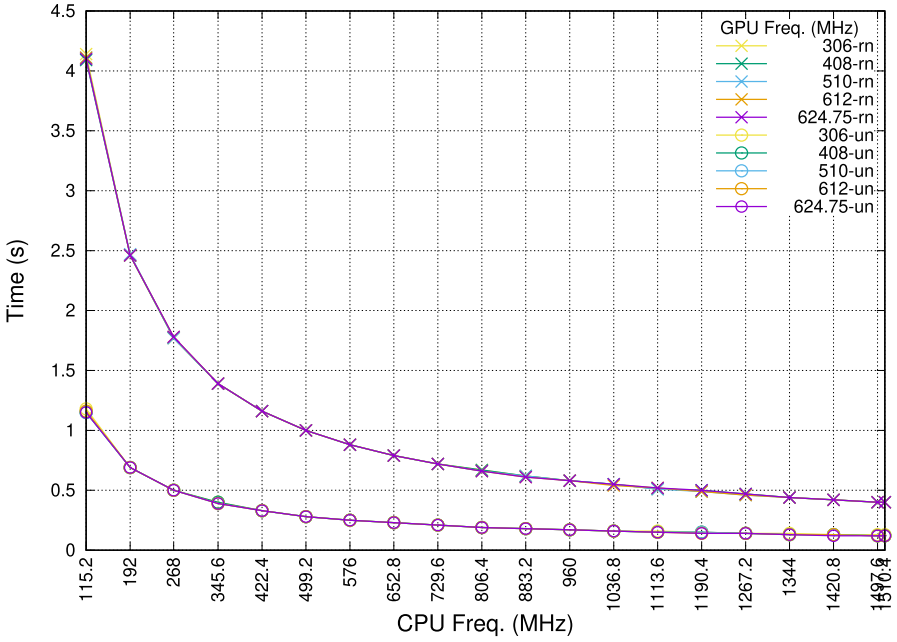

Fig. 6 Execution time using both models (`rn` and `un`) to perform the inference of one channel varying both the CPU and GPU frequencies. Note that the results for the different GPU frequencies overlap for each of the two models

explored various frequency settings for both the CPU and GPU to determine their effects on the board's overall power consumption.

4.2 Experimental Results

For the training and validation of the ResNet and U-Net architectures, datasets composed of 10,000 and 1000 different observation matrices were used, respectively. The datasets were synthetically generated following the model in Eq. (5) with $\rho = 1$, i.e., the SNR is $1/\sigma_n^2$. More specifically, each dataset element contains a random observation matrix generated for a random SNR in the range between -10 dB and 25 dB, with a number of channel paths randomly selected from $L = 1$ to 10. The codebook size is also selected randomly by setting $Q = P$ to either 16 or 32. Channel coefficients α_l, $l = 1, \ldots, L$, are drawn from a zero mean complex Gaussian distribution with variance $1/L$, while AoA and AoD angles are drawn from a uniform random distribution in the range $[0, \pi]$. Each dataset element also contains the corresponding ground-truth AoA and AoD for the L channel paths, to assist the model training.

The experimental evaluation of the resulting CNN model's implementations assumes equal values for the parameters P, Q, n_r and n_t for the sake of simplicity (note that in more realistic systems these parameters may differ), which implies that

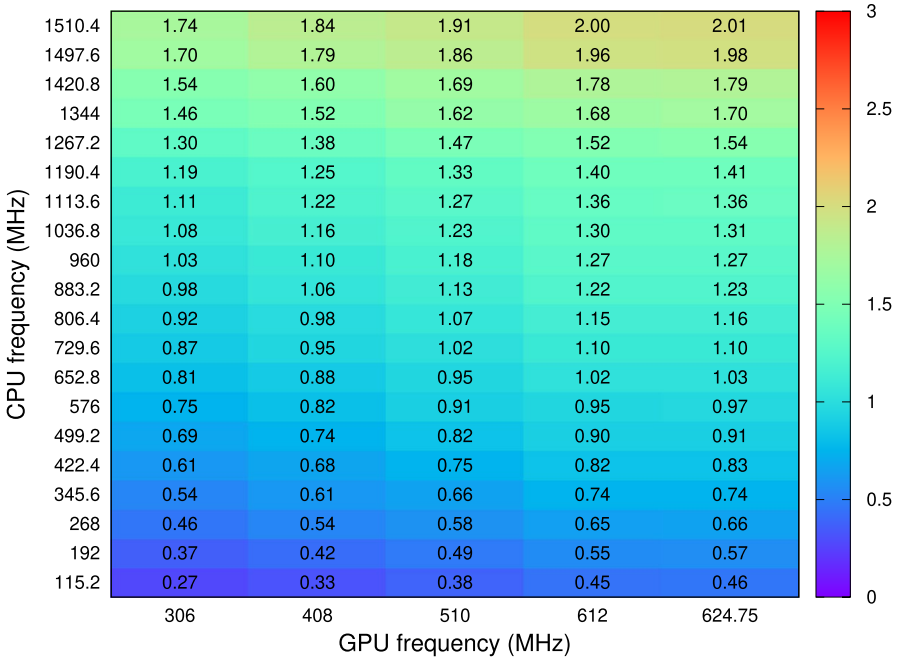

Fig. 7 Watts consumed by the whole platform to perform the inference using the ResNet model

the size of the observation matrix $\mathbf{Y}$ is directly $n_r \times n_t$. As a typical case for evaluation, we chose a number of paths equal to $L = 3$ and observation matrices of size $n_r \times n_t = 16 \times 16$.

All the experiments have been performed using a batch size equal to one. That means that for every channel we transfer the information from the CPU to the GPU, run the inference on the GPU and post-process the result on the CPU. We are measuring the time and power consumption of the whole process while iterating over several channels and calculating the average among the measurements from all the iterations.

Figure 6 presents the inference times (in seconds) for both models across various combinations of CPU and GPU frequencies. It is evident that the U-Net model (`un`) consistently outperforms the ResNet model (`rn`) in speed, irrespective of the frequencies. Both models exhibit similar behavior in response to changes in CPU and GPU frequencies: the execution time decreases rapidly at lower CPU frequencies and more gradually at higher frequencies, achieving the fastest execution time at the highest frequency. Interestingly, GPU frequency does not affect the execution time of either model, as a batch size of one does not fully utilize the GPU resources. Using the `jetson-stats` package, we monitored the platform's resource usage during the inference process.[3] At the highest GPU frequency, only about 50% of the

[3] https://github.com/rbonghi/jetson_stats.

GPU's capacity is utilized. However, as the GPU frequency decreases (while maintaining the CPU frequency), the communication latency to transfer each channel to the GPU is reduced, leading to up to 70% GPU utilization. Thus, the reduction in GPU frequency is offset by increased utilization, resulting in consistent inference times regardless of the GPU frequency. In order to confirm the effect of the GPU frequency, we have also carried out the inference process using a batch size of 32 channels. In this case, the GPU load is always very close to 100% for every GPU frequency. As a consequence, the GPU frequency affects the execution time, which decreases as we increase the frequency.

Reducing inference execution time is important, but in many applications using edge devices, minimizing power consumption can be paramount. Extending battery life is crucial in many environments, and one effective approach is to reduce the frequencies of the device components. Figures 7 and 8 illustrate the power consumption of the entire platform while executing both models at various combinations of CPU and GPU frequencies. We determined the power consumed by the application by subtracting the power consumed by the platform at idle with the minimum CPU and GPU frequencies. Both figures use the same color scale to facilitate comparison between the two models.

As previously observed, the U-Net model is consistently faster than the ResNet model. However, it also consumes more power across all CPU and GPU frequencies. The power consumption patterns of both models are similar, gradually increasing from the bottom-left corner to the top-right corner as both CPU and GPU

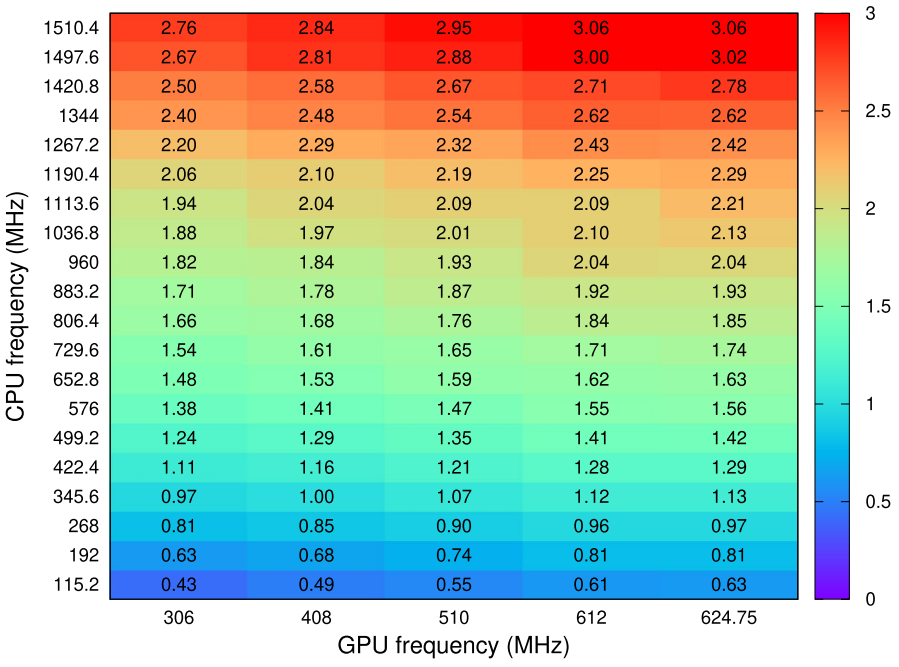

Fig. 8 Watts consumed by the whole platform to perform the inference using the U-Net model

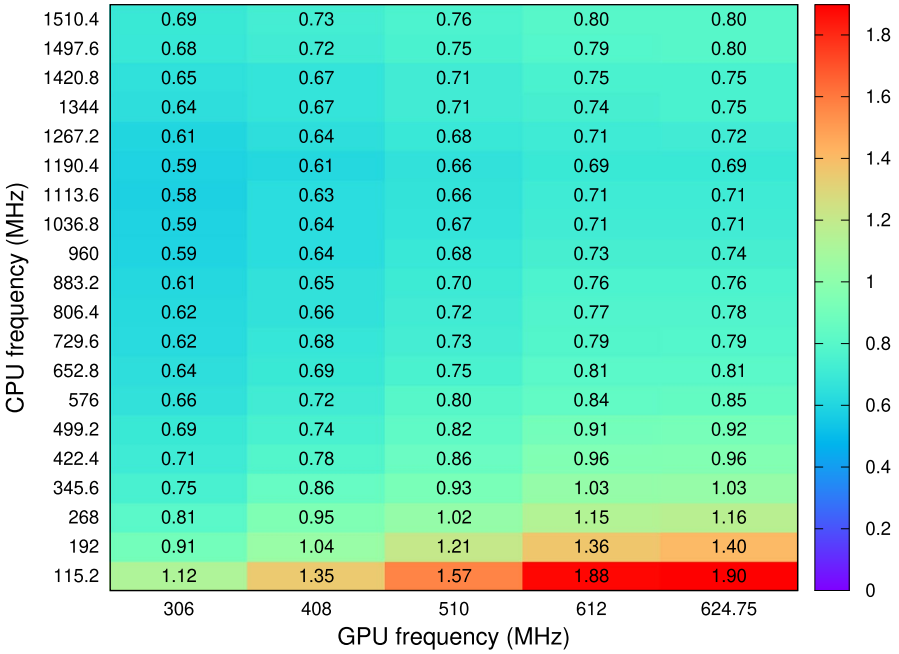

Fig. 9 Total energy (J) consumed by the whole platform to perform the inference of one channel for each combination of CPU and GPU frequencies using the ResNet model

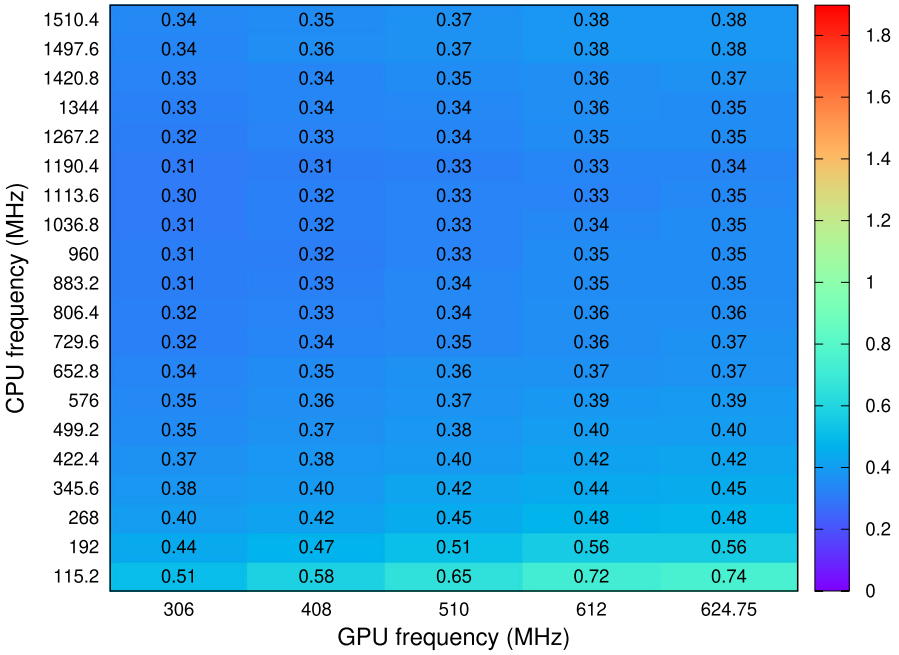

Fig. 10 Total energy (J) consumed by the whole platform to perform the inference of one channel for each combination of CPU and GPU frequencies using the U-Net model

frequencies rise. Therefore, to minimize power consumption, it is essential to use the lowest frequencies for both computing devices.

To assess the trade-off between execution time and power consumption, we calculated the energy required to perform the inference of one channel. This was done by multiplying the power consumption shown in the previous figures by the execution time shown in Fig. 6, resulting in the energy consumption per channel in joules. Figures 9 and 10 present this metric for both models, using the same color scale for easy comparison.

As demonstrated, the U-Net model consumes less energy per channel than the ResNet model across all combinations of CPU and GPU frequencies. This indicates that its faster execution time more than compensates for its higher power consumption, leading to lower energy consumption per channel. When analyzing the evolution of energy consumption with varying CPU and GPU frequencies, both models exhibit similar behavior. The highest energy consumption occurs at the highest CPU and GPU frequencies, while the lowest energy consumption is observed at the two lowest GPU frequencies and when the CPU runs between 883.2 MHz and 1267.2 MHz.

5 Conclusion

In this paper, we have presented a comprehensive study on the implementation of deep learning-based models, specifically a residual convolutional neural network (ResNet) and a U-Net architecture, for the efficient estimation of angle-of-arrival (AoA) and angle-of-departure (AoD) in mmWave MIMO systems. Our focus was on evaluating the performance and energy efficiency of these models when deployed on the NVIDIA Jetson Orin Nano platform, a low-power embedded system optimized for edge AI applications.

The experimental results demonstrate that while the U-Net model consumes more power than the ResNet model, it is approximately 3.5 times faster, leading to significantly lower energy consumption per channel estimation. This finding underscores the importance of considering both execution time and power consumption in the design and selection of AI models for edge computing in the future 6G systems. The trade-offs between computational performance and power efficiency highlighted in this study provide valuable insights for optimizing AI applications on low-power embedded systems.

Our research suggests that deploying deep learning models like U-Net on edge devices is a promising approach for enhancing the efficiency of mmWave MIMO channel estimation. This approach not only supports the high-throughput requirements of 6G communications but also aligns with the growing emphasis on sustainability in modern communication system design.

Author Contributions All the authors contributed equally to this work.

Funding Thanks to Grant PID2023-148671OB-I00 funded by MICIU/AEI/ 10.13039/501100011033 and by ERDF/EU. Thanks to Grants PID2020-113785RB-100 and PID2020-113656RB-C21, all of them

funded by the MICIU/AEI/10.13039/501100011033. Thanks to Grant TED2021-131003B-C21, funded by the "European Union NextGenerationEU/PRTR" and MICIU/AEI/10.13039/501100011033. Grant CIAICO/2022/179 from Generalitat Valenciana of Spain funded also partially this work.

Availability of data and materials No additional data or materials available.

Declarations

Ethical approval Not applicable.

Conflict of interest The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

1. Czarnul P, Proficz J, Drypczewski K (2020) Survey of methodologies, approaches, and challenges in parallel programming using high-performance computing systems. Sci Program 2020(1):4176794. https://doi.org/10.1155/2020/4176794
2. Belloch JA, Amor-Martin A, Garcia-Donoro D, Martínez-Zaldívar FJ, Garcia-Castillo LE (2019) On the use of many-core machines for the acceleration of a mesh truncation technique for FEM. J Supercomput 75:1686–1696. https://doi.org/10.1007/s11227-018-02739-9
3. Ramiro C, Roger S, Gonzalez A, Almenar V, Vidal AM (2013) Multicore implementation of a fixed-complexity tree-search detector for MIMO communications. J Supercomput 65:1010–1019. https://doi.org/10.1007/s11227-012-0839-x
4. Roger S, Ramiro C, Gonzalez A, Almenar V, Vidal AM (2012) Fully parallel GPU implementation of a fixed-complexity soft-output MIMO detector. IEEE Trans Veh Technol 61(8):3796–3800. https://doi.org/10.1109/TVT.2012.2210576
5. Roh W, Seol J-Y, Park J, Lee B, Lee J, Kim Y, Cho J, Cheun K, Aryanfar F (2014) Millimeter-wave beamforming as an enabling technology for 5G cellular communications: Theoretical feasibility and prototype results. IEEE Commun Mag 52(2):106–113. https://doi.org/10.1109/MCOM.2014.6736750
6. Giordani M, Polese M, Mezzavilla M, Rangan S, Zorzi M (2020) Toward 6G networks: use cases and technologies. IEEE Commun Mag 58(3):55–61. https://doi.org/10.1109/MCOM.001.1900411
7. Roger S, Cobos M, Botella-Mascarell C, Fodor G (2021) Fast channel estimation in the transformed spatial domain for analog millimeter wave systems. IEEE Trans Wireless Commun 20(9):5926–5941. https://doi.org/10.1109/TWC.2021.3071315
8. Lloria D, Roger S, Botella-Mascarell C, Cobos M, Svensson T (2024) A ResNet approach for AoA and AoD estimation in analog millimeter wave MIMO systems. In: IEEE International Symposium on Personal. Indoor and Mobile Radio Communications
9. Pan P, Zhang Y, Deng Z, Qi W (2021) Deep learning-based 2-D frequency estimation of multiple sinusoidals. IEEE Trans Neural Netw Learn Syst 33(10):5429–5440. https://doi.org/10.1109/TNNLS.2021.3070707

10. Ronneberger O, Fischer P, Brox T (2015) U-net: Convolutional networks for biomedical image segmentation. In: Medical Image Computing and Computer-Assisted Intervention-MICCAI, 2015 18th International Conference, Munich, Germany, October 5–9, 2015, Proceedings, Part III 18. Springer 2015:234–241. https://doi.org/10.1007/978-3-319-24574-4_28
11. Stoller D, Ewert S, Dixon S (2018) Wave-u-net: a multi-scale neural network for end-to-end audio source separation. arXiv preprint arXiv:1806.03185. https://doi.org/10.48550/arXiv.1806.03185
12. Pennanen H, Hänninen T, Tervo O, Tölli A, Latva-aho M (2024) 6G: the intelligent network of everything–A comprehensive vision, survey, and tutorial. arXiv preprint arXiv:2407.09398. https://doi.org/10.48550/arXiv.2407.09398
13. Bulakcı O, Li X, Gramaglia M, Gavras A, Uusitalo M, Rugeland P, Boldi M (2023) Towards sustainable and trustworthy 6G: challenges, enablers, and architectural design. Now Publishers. https://doi.org/10.1561/9781638282396
14. Roger S, Botella-Mascarell C, Martín-Sacristán D, García-Roger D, Monserrat JF, Svensson T (2024) Sustainable mobility in B5G/6G: V2X technology trends and use cases. IEEE Open J Vehicular Technol. https://doi.org/10.1109/OJVT.2024.3375451
15. Shi Y, Yang K, Jiang T, Zhang J, Letaief KB (2020) Communication-efficient edge AI: algorithms and systems. IEEE Commun Surv Tutor 22(4):2167–2191. https://doi.org/10.1109/COMST.2020.3007787
16. Letaief KB, Shi Y, Lu J, Lu J (2021) Edge artificial intelligence for 6G: vision, enabling technologies, and applications. IEEE J Sel Areas Commun 40(1):5–36. https://doi.org/10.1109/JSAC.2021.3126076
17. Zhu G, Lyu Z, Jiao X, Liu P, Chen M, Xu J, Cui S, Zhang P (2023) Pushing AI to wireless network edge: an overview on integrated sensing, communication, and computation towards 6G. Sci China Inf Sci 66(3):130301. https://doi.org/10.1007/s11432-022-3652-2
18. NVIDIA (2023) Nvidia jetson orin nano developer kit user guide. https://manuals.plus/nvidia/jetson-orin-nano-developer-kit-manual
19. Alkhateeb A, El Ayach O, Leus G, Heath RW (2014) Channel estimation and hybrid precoding for millimeter wave cellular systems. IEEE journal of selected topics in signal processing 8(5):831–846. https://doi.org/10.1109/JSTSP.2014.2334278
20. Akdeniz MR, Liu Y, Samimi MK, Sun S, Rangan S, Rappaport TS, Erkip E (2014) Millimeter wave channel modeling and cellular capacity evaluation. IEEE J Sel Areas Commun 32(6):1164–1179. https://doi.org/10.1109/JSAC.2014.2328154
21. Zhang C, Guo D, Fan P (2016) "Tracking angles of departure and arrival in a mobile millimeter wave channel," in 2016 IEEE international conference on communications (ICC). IEEE 1–6. https://doi.org/10.1109/ICC.2016.7510902
22. Park SC, Park MK, Kang MG (2003) Super-resolution image reconstruction: a technical overview. IEEE Signal Process Mag 20(3):21–36. https://doi.org/10.1109/MSP.2003.1203207
23. Szeliski R (2022) *Computer vision: algorithms and applications*. Springer Nature
24. Barrachina S, Barreda M, Catalán S, Dolz MF, Fabregat G, Mayo R, Quintana-Ortí E (2013) An integrated framework for power-performance analysis of parallel scientific workloads. Energy 114–119
25. NVIDIA, "NVIDIA jetson linux developer guide. release 32.7.3," (2022), https://docs.nvidia.com/jetson/archives/l4t-archived/l4t-3273/

Authors and Affiliations

Diego Lloria[1] · Sandra Roger[1] · Germán León[2] · José M. Badía[2] · Carmen Botella-Mascarell[1] · Jose A. Belloch[3]

✉ Sandra Roger
sandra.roger@uv.es

Diego Lloria
diego.lloria@uv.es

Germán León
leon@uji.es

José M. Badía
badia@uji.es

Carmen Botella-Mascarell
carmen.botella@uv.es

Jose A. Belloch
jbelloc@ing.uc3m.es

[1] Computer Science Department, Universitat de València, Burjassot, Spain

[2] HPCA, Universidad Jaume I de Castellón, Castellón de la Plana, Spain

[3] Depto. de Tecnología Electrónica, Universidad Carlos III de Madrid, Madrid, Spain