Embedded Audio Event Recognition: Architecture Trade-offs for Real-Time Inference on Resource-Constrained Platforms*

Dylan Molinié[1 [0000−0002−6499−3959]] and Andréa Macario Barros[1 [0000−0002−8871−9153]]

Université Paris-Saclay, CEA-List, Palaiseau, France

Abstract. Audio Event Recognition is a key perceptual capability for physically embodied systems operating in real-world environments, enabling robots and autonomous devices to detect and respond to security-critical sounds such as gunshots, glass breaking, or human screams. However, deploying Deep Learning-based AER on resource-constrained embedded hardware remains a significant challenge, as state-of-the-art models are rarely designed with embedded constraints in mind. In this work, we present a structured, layer-by-layer compression methodology for adapting AER architectures to embedded deployment without sacrificing recognition accuracy. Starting from established reference architectures, a standard ConvNet, SincNet, and DENet, we systematically ablate specific layers (learned filterbanks, denoising-enhancement branches, recurrent cells), network depth, and network width, evaluating at each stage the trade-off between sensitivity, specificity, inference time, computational load, and memory footprint. Experiments are conducted on the MIVIA Audio Events Dataset across multiple signal-to-noise ratios, and models are deployed on an ARM-based embedded platform using the Aidge embedded AI framework. Our results show that a compact ConvNet with two convolutional and two fully-connected layers achieves competitive accuracy while enabling real-time inference within a 100 ms latency, using under 17 MB of RAM. These findings contribute to the broader goal of enabling robust, real-time perception in physically embodied intelligent systems.

Keywords: Audio Event Recognition · Convolutional Neural Networks · Embedded AI.

1 Introduction

Audio Event Recognition (AER) consists of analyzing audio streams to detect critical sounds such as gunshots, glass breaking, or human screams from their distinctive acoustic signatures [1]. Audio signals can be processed in the time, frequency, or time-frequency domain. While the time-frequency domain is the most informative, it requires computationally expensive transformations such as

* An open-source implementation of the proposed architecture analysis pipeline, including the models and evaluation scripts, is available at https://gitlab.in2p3.fr/andrea.barros/embeddedaudioeventrecognition.git.

the Short-Time Fourier Transform; many prior works exploit such representations through features like MFCCs, spectrograms, or gammatonegrams [2], [3], at a computational cost that limits their applicability on resource-constrained hardware. The time domain, by contrast, requires no prior transformation and is directly compatible with standard convolutional operators.

Deploying Deep Learning models on embedded systems is challenging: restricted memory, limited compute, and energy constraints make state-of-the-art architectures difficult to run in real time. Despite this, most AER works optimize exclusively for recognition accuracy without regard for embedded deployment constraints [4], [5], [6]. Two time-domain architectures stand out as candidates for embedded AER: SincNet [5], which introduces a trainable sinc-function filterbank as its first layer, and DENet [6], which extends SincNet with a denoising-enhancement attention branch and a recurrent cell.

In this paper, we present a structured, three-stage analysis of the influence of architectural complexity on classification accuracy and embedded resource consumption, ablating specific layer types, network depth, and network width across SincNet and DENet reference architectures. Experiments are conducted on the MIVIA Audio Events Dataset [1] across multiple SNRs, and models are deployed on an ARM-based platform using the Aidge embedded AI framework [7].

2 Problem Formulation

Embedded systems impose hardware constraints on the models that run on them: restricted energy capacity, reduced available memory, limited computation capability, and latency in data transfers. These constraints jointly define the embeddability limitations to be respected by any candidate architecture for real-time embedded operation. They can be resumed as follows:

Latency. Audio is processed in chunks of a fixed duration, corresponding to given number of samples at a sampling rate. For real-time processing to be possible, the full inference pipeline, including data normalization and model forward pass must be complete within this chunck duration window. This constitutes a latency constraint: any model whose inference time exceeds the chunk duration cannot process audio continuously without introducing delay.

Memory footprint. The RAM available for the model and its activations is shared with the operating system and the acquisition pipeline. A practical upper bound is defined by the target board (a few tens of megabytes in this work, as shown in Section 4) and therefore imposed on the model's memory footprint, including both the weight tensors and the intermediate activation buffers.

Computing load. The number of Multiply-Accumulate (Mult-Add) operations per inference directly drives CPU load and, consequently, energy consumption. Minimizing this quantity is essential not only to satisfy the latency constraint, but also to leave sufficient headroom for the operating system and peripheral management tasks.

2.1 The Compression Objective

Given the constraints above, the objective of this work is to identify the minimal architecture that simultaneously satisfies the accuracy and hardware requirements for embedded AER. Formally, let $\mathcal{A}$ denote the set of candidate architectures parameterized by their layer types, depth, and width. The goal is to find $a^* \in \mathcal{A}$ such that:

$$a^* = \arg\min_{a \in \mathcal{A}} \mathcal{C}(a) \quad \text{subject to} \quad \text{sens}(a) \geq \tau_s, \;\; t_{\text{inf}}(a) \leq t_{\text{chunk}}, \;\; \text{RAM}(a) \leq \text{RAM}_{\text{max}} \tag{1}$$

where $\mathcal{C}(a)$ is a resource cost (such as number of parameters or Mult-Add operations), $\text{sens}(a)$ is the weighted mean sensitivity across classes, τ_s is the minimum acceptable sensitivity threshold, $t_{\text{inf}}(a)$ is the inference time on the target hardware, t_{chunk} is the chunk duration, and RAM_{max} is the available memory budget. Rather than solving this combinatorial problem through exhaustive search or neural architecture search, we adopt a structured, top-down compression pipeline that ablates complexity along three orthogonal axes: layer type, network depth, and network width. This approach yields interpretable findings at each stage and provides practical design guidelines for embedding-aware AER systems.

3 Architectures & Compression Benchmarks

This section presents the compression methodology and the architectures it operates on. The reference architectures that define the starting point and the design space of the proposed pipeline are introduced. Following, the methodology is structured as a top-down pipeline of three successive ablation stages, each targeting a distinct axis of architectural complexity.

3.1 Reference Architectures

This work focuses on relatively standard layers, in particular convolutional and fully-connected layers, operating in the time domain only. Transformer-based architectures such as AST [8] and PaSST [9] have been increasingly employed in audio classification tasks. However, these architectures operate on spectral representations and rely on self-attention mechanisms whose computational complexity and memory requirements are significantly higher than those of the convolutional architectures considered here. Their evaluation on the target embedded platform is outside the scope of this study, which is intentionally restricted to time-domain convolutional architectures whose operators are compatible with the embedded deployment pipeline and real-time constraints on resource-constrained hardware. Three reference architectures are therefore considered, chosen to cover the principal design choices found in the state of the art for time-domain AER.

ConvNet. The baseline architecture consists of a stack of one-dimensional convolutional layers for extracting time-local features, followed by a series of fully-connected layers for time-global classification, and a final SoftMax output. Each convolutional block includes a Max Pooling layer and a Normalization step; the

pooling reduces the size of the intermediate tensors, which directly decreases the computational burden of the subsequent layers. This architecture uses no learned filterbank and no recurrent component, making it the most compact and interpretable reference point.

SincNet. The SincNet model relies on the custom SincLayer as its first input layer: it is a filterbank using sinc functions as filters whose cut-off frequencies are trainable parameters. The Fourier transform of a sinc function is a gate function centered around zero; by subtracting two sinc functions with different cut-off frequencies in the time domain, one obtains the subtraction of two gate functions in the frequency domain. By applying the correct normalization factors, this yields a band-pass filter, that can extract full frequency bands from the input waveform through standard convolution [5]. SincLayer requires only two parameters per filter, the lower and upper cut-off frequencies, against $F \times L$ parameters for a standard convolutional filter of length L, making it highly parameter-efficient.

DENet. The DENet model is an improvement on SincNet, in which a Denoising-Enhancement operation, referred to as DELayer, is added as an attention branch: a smaller model that runs in parallel to the SincLayer and whose feedback serves as additional input to it. This attention mechanism essentially consists of additional convolutions and fully-connected layers with normalization, aiming to mitigate the noise of the signal and enhance the segments of interest. In addition, a Bidirectional Gated Recurrent Unit (BGRU) is stacked after the convolutional feature extractor to capture the temporal evolution of the frequencies of interest across successive frames [6]. The complete DENet architecture used as the starting point of this study is summarized in Figure 1, where B denotes the number of filters, L the convolution kernel length, and *Neurons* the number of units in fully-connected layers.

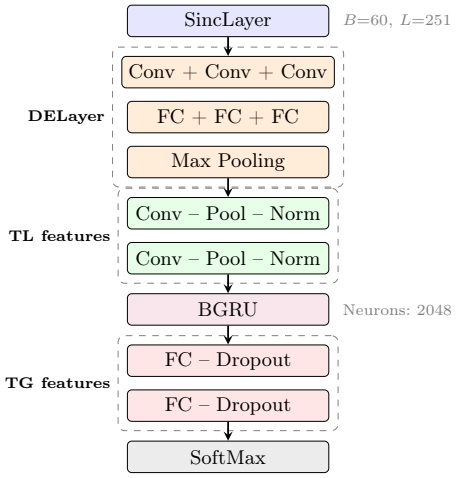

Fig. 1. DENet architecture [6]. TL = Time-local, TG = Time-global.

3.2 Ablation Stages

Stage 1 - Specific Layer Ablation. The first stage evaluates whether the added complexity of SincLayer, DELayer, and the recurrent cell is justified for embedded AER. Evaluating their contribution allows to correctly scale the AER model to meet the requirements and expectations of an embedded system. Nine configurations are derived from the ConvNet baseline by independently adding SincLayer, DELayer, and a recurrent cell (GRU or LSTM). For each configuration, inference time, RAM consumption, and number of parameters are measured, alongside sensitivity and specificity evaluated on both the training and testing sets of the MIVIA Audio Events Dataset at SNRs of 10, 20, and 30 dB. Stage 1 evaluates whether the components that distinguish the state-of-the-art DENet from a plain ConvNet, *i.e.*, the SincLayer, the DELayer, and the recurrent cell, are compatible with the constraints of the embedded pipeline. Its output is not a compressed model but a justified choice of architecture family.

Stage 2 - Depth Ablation. Adding convolutional or fully-connected layers is expected to increase accuracy, but this also increases computation and the number of parameters, and is thus expected to increase inference time and memory footprint. The purpose of this second compression stage is to identify the minimum number of layers that preserves accuracy while reducing resource consumption. Starting from the ConvNet baseline without recurrent cell, the number of convolutional (CV) and fully-connected (FC) layers is independently varied between 1 and 3. The number of filters per convolutional layer is fixed at 60 with a kernel length of 5 samples; the first FC layer uses 2048 neurons, the second 1024, and the third 512. This yields nine configurations, covering the full Cartesian product of depth choices.

Stage 3 - Width Ablation. The third and final stage investigates how far the number of filters in the convolutional layers and the number of neurons in the fully-connected layers can be reduced without incurring a significant loss. This stage operates on a fixed architecture and varies the number of convolutional filters across $\{20, 40, 60\}$ and the number of neurons in the first FC layer across $\{512, 1024, 2048\}$, with the second FC layer always set to half this value. This yields nine configurations covering the width design space.

4 Experimental Setup

This section describes the experimental conditions under which the compression methodology introduced in Section 3 is evaluated. All benchmarks share a common dataset, evaluation protocol, training procedure, and target hardware, ensuring that results across the three compression stages are directly comparable.

4.1 Dataset

This study uses the Audio Events Dataset elaborated by the MIVIA lab from the University of Salerno [1], a dataset oriented toward security applications that comprises three categories of sounds: Glass Breaking (GB), Gunshot (GS), and

Human Screams (S); the Background (B) is additionally considered as a fourth class to account for the continuous ambient noise present in real surveillance environments.

The dataset comprises 95 audio files of approximately 3 minutes each, split into two fixed sets: 66 files for training and 29 for testing. The training set contains 4200 occurrences of each of the three event classes, and the testing set contains 1800 occurrences of each. Any file has a background environment to which short-time events are locally superimposed, spaced with no fixed pattern, with one event occurring every 4 to 8 seconds depending on the file. The typical duration of each event class reflects its acoustic nature: Glass Breaking lasts 1 to 3 seconds, Gunshots approximately 500 ms, and Human Screams 1 to 2 seconds. Every file is provided at six Signal-to-Noise Ratios (SNRs): 5, 10, 15, 20, 25, and 30 dB, simulating varying distances between the source and the microphone. In this study, three SNRs (10, 20, and 30 dB) are used across all benchmarks, covering a representative range from moderately noisy to clean conditions.

Preprocessing. Audio data are pre-processed into non-overlapping chunks of 100 ms, corresponding to 3200 samples at the 32 kHz sampling rate of the dataset. This duration results from three concurrent constraints: it is short enough for the quasi-stationarity assumption to hold for all event classes while providing a 10 Hz frequency resolution sufficient for spectral discrimination; it ensures that even the shortest event class (Gunshot, $\sim$500 ms) spans at least five consecutive chunks, providing multiple independent classification opportunities per event; and it defines the real-time latency constraint directly, as the full inference pipeline must complete within 100 ms for continuous processing without delay. The non-overlapping scheme minimizes buffer memory requirements, as each chunk is written and read once, and ensures a deterministic, fixed-size input at every inference step, simplifying both the C++ deployment pipeline and real-time scheduling on the embedded board. Chunks are represented as 32-bit floating-point values and normalized to $[-1, +1]$ before being passed to the model.

4.2 Evaluation Metrics

Model accuracy is assessed through sensitivity and specificity, derived from the standard True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN) indicators:

$$\text{sens} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \quad \text{spec} = \frac{\text{TN}}{\text{TN} + \text{FP}} \tag{2}$$

In a security context, sensitivity is the more critical metric, as a missed event carries a higher operational cost than a false alarm. Both are reported as weighted means across the four classes at SNRs of 10, 20, and 30 dB, averaged across all files of the training and testing sets. Resource consumption is characterized through inference time, RAM usage, number of Multiply-Accumulate (Mult-Add) operations, and number of synaptic weights; the former driving CPU load, the latter providing a hardware-independent estimate of model size.

4.3 Training Protocol

All models are trained on the training set files using the Adam optimizer [10] with the Mean Squared Error (MSE) as the loss function. During training, the order of the 100 ms chunks is randomized across the full training set before each epoch. This shuffling strategy is standard practice in batch-based gradient descent: it breaks temporal correlations between consecutive samples, stabilizes gradient estimates, and reduces the risk of overfitting to the sequential structure of individual training files. However this protocol is structurally incompatible with recurrent architectures, which rely on ordered temporal sequences to propagate meaningful hidden states. The chunk-level shuffling adopted here reflects a deliberate embedded-first design choice: it is consistent with a pipeline where the system processes independent, fixed-size audio blocks rather than continuous, ordered streams, and it avoids the memory and latency overhead that sequence-level training would impose on the deployment pipeline. The implications of this choice on recurrent cell performance are analyzed in Section 5.

4.4 Target Hardware and Deployment Framework

A standard laptop (Intel Core i7, 64GB of RAM memory) serves to the models' development, training and preliminary evaluations; and the primary target is an i.MX8 Plus ARM-based embedded board equipped with a quad-core CPU running at 1.8 GHz, 2 GB of SDRAM, and 8 GB of flash memory, with a nominal power consumption of 1685 mW. No hardware accelerator is assumed; all inference runs on the CPU only. Inference is performed in the model generated using the Aidge embedded AI framework [7], which takes a model trained under a high-level framework (*e.g.*, Pytorch), and exports it as a standalone C++ project whose kernels are compiled natively for the target architecture. This export pipeline preserves the trained weights and allows full inference to run without any Python runtime or third-party dependency on the embedded board, which is an advantage requirement for deployment in security applications.

5 Results and Discussion

5.1 Stage 1 - Specific Layer Ablation

The results in terms of ressources consumption are evaluated firstly on the laptop using Pytorch framework and reported in Figure 2, and the accuracy values for the different implemented models are presented in Table 1.

Resource consumption. As illustrated in Figure 2, adding a recurrent cell significantly reduces the inference time; this may seem counter-intuitive since the number of Mult-Add operations increases. This is due to the data reshaping operated by the recurrent cell: since it outputs less data than it takes as input, this reduces the length of the intermediate tensors, thus reducing the required number of neurons in the subsequent fully-connected layers, and consequently the overall model size. This effect also applies to DELayer, whose normalization stage incorporates pooling, reducing both the tensor size and the inference time. However, recurrent cells require a large number of parameters and are

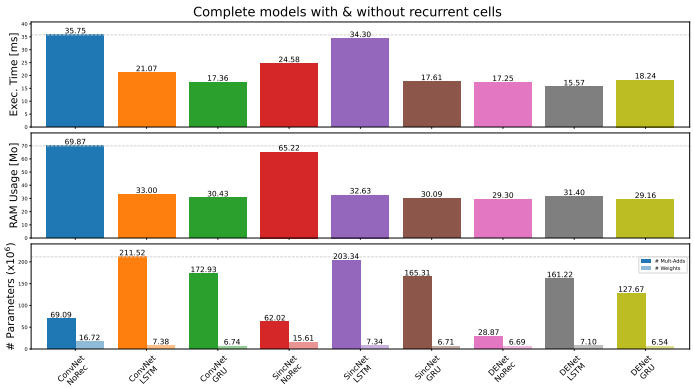

Fig. 2. Resource consumption of the nine Stage 1 configurations evaluated on the development platform with PyTorch. From top to bottom: inference time, RAM usage, and number of Mult-Add operations and synaptic weights.

Table 1. Mean sensitivity and specificity of the Stage 1 configurations, averaged across the four classes, at SNRs of 10, 20, and 30 dB on both the training and testing sets.

Model	Rec. Cell	Sensitivity						Specificity					
		Train Set			Test Set			Train Set			Test Set		
	SNR:	10	20	30	10	20	30	10	20	30	10	20	30
ConvNet	NoRec	0.82	0.91	**0.95**	0.79	0.89	**0.94**	0.94	**0.98**	**1.00**	0.93	0.96	0.99
ConvNet	LSTM	0.79	0.90	0.94	0.78	0.89	0.93	0.93	**0.98**	**1.00**	0.92	**0.98**	0.99
ConvNet	GRU	0.75	0.89	0.94	0.73	0.88	0.93	0.94	0.96	**1.00**	0.93	0.95	0.99
SincNet	NoRec	**0.84**	**0.92**	**0.95**	**0.82**	**0.90**	**0.94**	0.95	**0.98**	**1.00**	**0.94**	**0.98**	**1.00**
SincNet	LSTM	0.81	0.89	0.90	0.79	0.89	0.90	**0.96**	**0.98**	0.99	**0.94**	**0.98**	0.99
SincNet	GRU	0.78	0.88	0.93	0.77	0.88	0.93	0.95	**0.98**	0.99	**0.94**	0.97	0.99
DENet	NoRec	0.81	0.86	0.90	0.80	0.85	0.90	0.95	**0.98**	0.99	**0.94**	0.97	0.99
DENet	LSTM	0.70	0.83	0.87	0.64	0.81	0.87	0.94	**0.98**	0.99	0.92	0.96	0.99
DENet	GRU	0.64	0.81	0.85	0.61	0.80	0.86	0.92	0.97	0.99	0.92	0.96	0.99

highly computation-demanding at the individual operation level. From a pure resource-usage standpoint, the most compact models in terms of both parameters and Mult-Add operations are therefore those without a recurrent cell.

Accuracy. Most models achieve a equivalent specificity on both the training and testing sets, meaning that when they do not detect an event, it is very likely that no such event was present. However, sensitivity is more mixed, indicating that models are prone to miss events. The best-performing models for this metric are ConvNet and SincNet, both without a recurrent cell, which achieve the highest weighted mean sensitivity μ on both the training and testing sets, as reported in Table 1.

Conclusion of Stage 1. The degradation in sensitivity observed when adding GRU or LSTM cells is counter-intuitive, since recurrent cells are precisely designed to improve classification of temporally structured signals. The explanation lies in the interaction between the recurrent architecture and the chunk-level shuffling adopted during training: recurrent cells exploit the temporal relationship between consecutive inputs, information that is entirely lost when individual chunks are shuffled. As established in Section 4, this shuffling is a deliberate embedded-first design choice, but it creates a train-test mismatch as the BGRU is never exposed to meaningful temporal sequences during training, yet receives a continuous ordered stream at inference time, that manifests as over-smoothed class probabilities and increased False Negatives. In the original DENet formulation [6], by contrast, the BGRU is trained on ordered sequences of 10 consecutive 50 ms frames, preserving the temporal structure the cell requires. The degradation observed here is therefore a consequence of the training protocol imposed by the embedded pipeline constraints, not an inherent weakness of recurrent architectures. Under these constraints, the best-performing candidate of Stage 1 is SincNet-NoRec. For Stage 1, SincNet-NoRec achieves an embedded inference time of 192.75 ms, nearly twice the 100 ms latency constraint, confirming that Stage 1 alone is insufficient to meet the embedded deployment requirements. Since SincNet is architecturally equivalent to ConvNet augmented with a SincLayer, as established in Section 3, the following stages analyse the influence of depth and width on the shared ConvNet backbone to bring inference time within the target constraint.

5.2 Stage 2 — Depth Ablation

Resource consumption for all nine depth configurations is evaluated on the development platform using the PyTorch framework and reported in Figure 3, and the accuracy values for the different configurations are presented in Table 2.

Resource consumption. As illustrated in Figure 3, increasing the number of convolutional layers yields a substantial reduction in both inference time and RAM usage, while the number of fully-connected layers has comparatively little effect on either metric. This asymmetry is explained by the pooling-driven tensor reduction mechanism: each additional convolutional block equipped with Max Pooling compresses the temporal dimension of the intermediate feature map, reducing the input size of all subsequent layers. As a result, the 3CV configurations achieve the lowest resource consumption across the full design space, with inference times between 24.60 and 30.36 ms and RAM usage between 59.36 and 70.57 MB, regardless of the number of fully-connected layers.

Accuracy. As reported in Table 2, the configuration using 1 convolutional and 1 fully-connected layer is the least accurate across all SNRs and both sets, while simultaneously being the most resource-intensive. Among the remaining configurations, 2CV3FC achieves the highest mean sensitivity (0.887) and specificity (0.977) averaged across all SNR levels and both sets. However, several other configurations, including 3CV3FC (0.882, 0.967) and 2CV2FC (0.867, 0.970),

achieve comparable accuracy values. Given this accuracy equivalence, depth alone cannot unambiguously determine the optimal configuration, and the choice must account for the role that the remaining architectural axes, in particular network width, play in resource consumption.

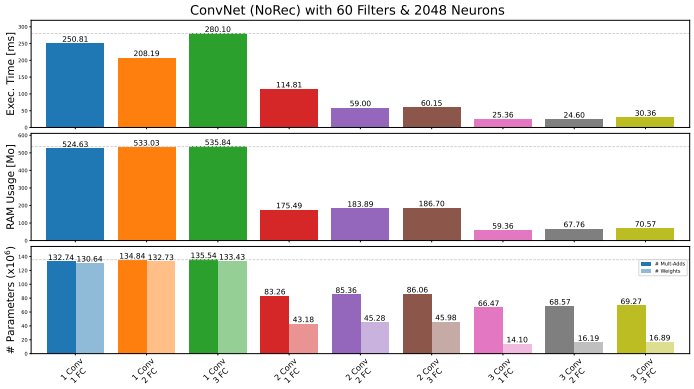

Fig. 3. Resource consumption of the nine Stage 2 configurations evaluated on the development platform with PyTorch. From top to bottom: inference time, RAM usage, and number of Mult-Add operations and synaptic weights

Table 2. Mean sensitivity and specificity of the Stage 2 configurations, averaged across the four classes, at SNRs of 10, 20, and 30 dB on both the training and testing sets.

#CV	#FC	Sensitivity						Specificity					
		Train Set			Test Set			Train Set			Test Set		
	SNR:	**10**	**20**	**30**	**10**	**20**	**30**	**10**	**20**	**30**	**10**	**20**	**30**
1	1	0.74	0.71	0.66	0.71	0.72	0.68	0.95	0.97	0.98	0.94	0.97	0.98
1	2	**0.84**	0.91	0.93	0.74	0.85	0.89	**0.96**	**0.99**	**1.00**	**0.95**	**0.98**	**0.99**
1	3	0.82	0.90	0.89	0.75	0.86	0.84	0.95	**0.99**	0.99	0.94	**0.98**	0.98
2	1	0.81	0.90	0.92	0.79	0.90	0.92	0.94	**0.99**	**1.00**	0.93	**0.98**	**0.99**
2	2	0.81	0.89	0.91	0.80	0.88	0.91	0.94	**0.99**	0.99	0.94	**0.98**	0.98
2	3	0.82	**0.93**	**0.95**	0.80	0.90	0.92	0.95	**0.99**	**1.00**	**0.95**	**0.98**	**0.99**
3	1	0.78	0.85	0.92	0.75	0.85	0.91	0.94	0.98	0.99	0.93	**0.98**	**0.99**
3	2	**0.84**	0.87	0.94	**0.81**	0.84	**0.93**	**0.96**	0.96	**1.00**	0.94	0.94	**0.99**
3	3	0.81	0.92	0.94	0.78	**0.91**	**0.93**	0.94	0.98	**1.00**	0.92	0.97	**0.99**

Conclusion of Stage 2. The depth analysis identifies the number of convolutional layers as the dominant factor governing resource consumption, through the pooling-driven tensor reduction mechanism, while the number of fully-connected

layers has comparatively little influence on either inference time or memory footprint. Among the configurations achieving acceptable accuracy, 2CV2FC is retained as the basis for Stage 3 for two complementary reasons. First, it represents the minimum convolutional depth at which accuracy remains acceptable across all SNR levels, ensuring that width is the dominant remaining factor driving resource consumption; in this regime, the influence of width on both accuracy and resource consumption can be clearly characterized without being masked by the pooling-driven reduction already achieved by deeper configurations. Second, its embedded inference time of 180.03 ms and RAM usage of 182.75 MB, measured using Aidge framework generated C++ code on the target board, confirm that depth reduction alone is insufficient to meet the 100 ms latency constraint, establishing width compression as a necessary and well-motivated next step.

5.3 Stage 3 — Width Ablation

Resource consumption for all nine width configurations is evaluated on the development platform using the PyTorch framework and reported in Figure 4, and the accuracy values for the different configurations are presented in Table 3.

Resource consumption. As illustrated in Figure 4, the number of neurons in the fully-connected layers is the dominant factor governing both inference time and RAM usage across the width design space. Moving from 512 to 2048 neurons increases inference time by a factor of approximately 3, from 11.12 to 32.80 ms for 20 filters, and from 17.31 to 100.25 ms for 60 filters, and RAM usage by a factor of approximately 4, from 15.86 to 66.93 MB and from 46.56 to 183.89 MB respectively. The number of convolutional filters, by contrast, has a comparatively modest effect: moving from 20 to 60 filters at fixed neuron count multiplies inference time by at most 1.6 and RAM by at most 3. This asymmetry arises from the structure of the architecture: after two pooling stages, the temporal dimension of the feature maps is substantially compressed, so the convolutional operations process short sequences regardless of the number of filters; in contrast, the fully-connected layers operate on the full flattened representation, whose size scales directly with the number of neurons. The configurations with 512 neurons therefore achieve the lowest resource consumption across all filter counts, with inference times of 11.12, 14.38, and 17.31 ms and RAM usage of 15.86, 31.19, and 46.56 MB for 20, 40, and 60 filters respectively.

Accuracy. As reported in Table 3, all configurations achieve comparable specificity across all SNR levels and both sets, with values consistently above 0.93; the number of filters and neurons has negligible influence on this metric. Regarding sensitivity, the 512-neuron configurations achieve the highest mean values across all filter counts: 0.880 for 20 filters, 0.885 for 40 filters, and 0.882 for 60 filters, averaged across all SNR levels and both sets. Increasing the neuron count to 1024 or 2048 does not improve sensitivity and in most cases marginally degrades it, with the 2048-neuron configurations consistently achieving the lowest mean sensitivity values of 0.875, 0.860, and 0.865 for 20, 40, and 60 filters respectively. The influence of the filter count on sensitivity is also limited: differences between filter counts at fixed neuron count remain minimal. This result reinforces

the finding already observed in Stage 1: under this training protocol, increasing model capacity does not improve classification accuracy while increasing resource consumption.

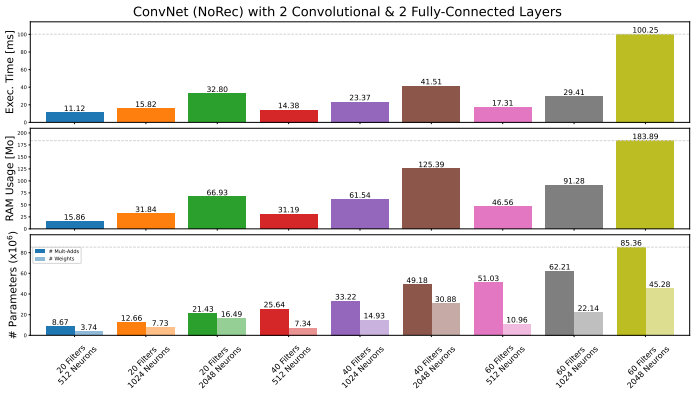

Fig. 4. Resource consumption of the nine Stage 3 configurations evaluated on the development platform with PyTorch. From top to bottom: inference time, RAM usage, and number of Mult-Add operations and synaptic weights

Table 3. Mean sensitivity and specificity of the Stage 3 configurations, averaged across the four classes, at SNRs of 10, 20, and 30 dB on both the training and testing sets.

Filters	Neurons	Sensitivity						Specificity					
		Train Set			Test Set			Train Set			Test Set		
	SNR:	10	20	30	10	20	30	10	20	30	10	20	30
20	512	0.80	**0.92**	0.94	0.79	**0.90**	**0.93**	**0.95**	**0.99**	**1.00**	0.94	**0.98**	**0.99**
20	1024	0.80	0.91	0.94	0.79	0.89	**0.93**	**0.95**	0.98	**1.00**	0.94	0.97	**0.99**
20	2048	0.79	0.91	0.93	**0.80**	**0.90**	0.92	**0.95**	0.99	0.99	0.95	0.98	**0.99**
40	512	**0.82**	**0.92**	**0.95**	**0.80**	0.89	**0.93**	**0.95**	0.98	**1.00**	0.94	0.97	**0.99**
40	1024	0.80	0.91	0.94	0.77	0.89	**0.93**	**0.95**	0.98	**1.00**	0.94	0.97	**0.99**
40	2048	0.78	0.89	0.92	0.77	0.89	0.91	0.94	0.98	0.99	0.93	**0.98**	**0.99**
60	512	**0.82**	**0.92**	0.94	**0.80**	0.89	0.92	**0.95**	0.98	**1.00**	0.94	0.96	**0.99**
60	1024	**0.82**	**0.92**	0.94	**0.80**	0.89	0.92	**0.95**	0.98	**1.00**	0.94	0.97	**0.99**
60	2048	0.79	0.89	0.92	0.78	0.89	0.92	0.94	0.98	1.00	0.94	**0.98**	**0.99**

Conclusion of Stage 3. The width analysis identifies the number of fully-connected neurons as the dominant factor governing both resource consumption and, to a lesser extent, accuracy, while the number of convolutional filters has comparatively little influence on either metric. The configuration using 20 filters and

512-256 neurons achieves the best trade-off across all criteria simultaneously: it is the most resource-efficient configuration, with an inference time of 11.12 ms and RAM usage of 15.86 MB on the development platform, while achieving a mean sensitivity of 0.880 and specificity of 0.975 across all SNR levels and both sets. This configuration was compiled and deployed on the target board via Aidge, yielding an embedded inference time of 19.81 ms and a RAM footprint of 15.53 MB, well within the 100 ms latency limitation and confirming that the width ablation successfully brings the architecture within the embedded deployment constraints established in Section 2.

5.4 Synthesis

Table 4 summarizes the selected candidate from each stage, evaluated on the embedded target platform via Aidge. The three-stage analysis reveals a consistent finding: across all axes of architectural variation, *i.e.*, layer type, depth, and width, the number of fully-connected neurons is the dominant factor governing embedded resource consumption, while convolutional depth controls the tensor size that propagates through the network via the pooling-driven reduction mechanism.

Table 4. Comparison of the best candidate from each compression stage on the embedded target. Sensitivity and specificity are weighted means across the four classes at SNR 20 dB on the testing set. Inference time and RAM are measured with Aidge on the ARM platform, single-threaded.

Model	**Sens.**	**Spec.**	**Inf. Time (ms)**	**RAM (MB)**
SincNet-NoRec (S1)	0.90	0.98	192.75	64.02
ConvNet-2CV-2FC (S2)	0.88	0.98	180.03	182.75
ConvNet-20F-512N (S3)	0.90	0.98	19.81	15.53

Three observations emerge from the synthesis. First, the classification accuracy is preserved across all three stages: the final Stage 3 configuration recovers the sensitivity of the Stage 1 best candidate while achieving a tenfold reduction in embedded inference time, from 192.75 ms to 19.81 ms, and a fourfold reduction in RAM usage, from 64.02 MB to 15.53 MB. Second, the progression from Stage 1 to Stage 3 reveals that neither depth reduction alone nor the Stage 1 architecture alone is sufficient to meet the 100 ms latency limitation; it is the combination of minimum sufficient depth and reduced fully-connected width that brings the model within the embedded deployment constraints. Third, the Mult-Add and the weight count decrease between Stage 1 and Stage 3, as shown in Figures 2-4, confirming that the final model is not only faster and lighter in memory but also computationally more efficient per inference, leaving greater headroom for the operating system and peripheral tasks on the target board.

Real-Time Validation The model identified at Stage 3 was integrated into a real-time demonstrator running entirely on the embedded board, illustrated in Figure 5. A Shure MV7 microphone USB-connected to the board continuously

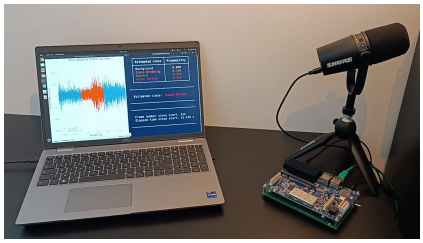

Fig. 5. Illustration of the real-time AER demonstration running on a ressource-constrained board.

captures audio, which is written as 100 ms chunks into a ring buffer, normalized to $[-1, +1]$, and passed to the model; the connected laptop is used for visualization only. The complete pipeline (data transfer, ring buffer write and read, normalization, and model inference) executes in 53.28 ms on average with a maximum of 63 ms, well within the 100 ms chunk duration, leaving sufficient headroom to absorb transient CPU overload. CPU load stabilizes at 54% and total RAM usage, including the operating system and acquisition pipeline, reaches 145.92 MB.

Limitations of this work. Some limitations of the present study warrant explicit acknowledgment. First, the evaluation is conducted on three event classes only; extending the methodology to a larger and more diverse set of classes would be beneficial to assess the generalization of the compression findings. Second, the study targets a single hardware platform; validating the methodology on other embedded targets, microcontrollers, FPGAs, or platforms with NPU acceleration, would strengthen the generalizability claim. Finally, the models evaluated in this work operate in 32-bit floating-point precision throughout. Post-Training Quantization (PTQ) to 8-bit integer precision could yield further reductions in memory footprint and inference time without retraining, at the cost of a potential accuracy trade-off that remains to be characterized for this specific task.

6 Conclusion

This paper addressed the deployment of Deep Learning-based Audio Event Recognition on resource-constrained embedded hardware, a challenge largely overlooked in the literature in favor of accuracy-only optimization. Rather than proposing a new architecture, we presented a structured, three-stage analysis of the influence of architectural complexity on classification accuracy and embedded resource consumption, exploring orthogonal axes of variation: specific layer types, network depth, and network width. Applied to the MIVIA Audio Events Dataset and deployed on an ARM-based embedded platform using the Aidge framework, the analysis identifies a compact ConvNet as the configuration offering a favorable trade-off across all evaluated criteria. This result is validated by a real-time demonstrator running entirely on the embedded board at an average pipeline latency of 53.28 ms with 54% CPU load.

Two contributions of broader significance emerge from this work. The first is methodological: the three-stage analysis framework is architecture-agnostic and transferable to other embedded sensing tasks involving one-dimensional time-domain signals. The second is the analysis suggesting a practical design principle for embedded AER: invest the parameter budget in convolutional depth with pooling, minimize fully-connected width, and avoid recurrent components unless the training protocol preserves temporal ordering. Future work includes extending the evaluation to all six SNR levels, applying Post-Training Quantization to 8-bit integer precision, and validating the framework on additional event classes and hardware targets.

Acknowledgements This work is part of the DeepGreen Project (ANR-23-DEGR-0001). The authors thank the project team for their support that contributed to this research.

References

1. Foggia, P., Petkov, N., Saggese, A., Strisciuglio, N., Vento, M.: Reliable detection of audio events in highly noisy environments. Pattern Recognit. Lett. **65**, 22–28 (2015). https://doi.org/10.1016/j.patrec.2015.06.026
2. Greco, A., Petkov, N., Saggese, A., Vento, M.: AReN: a deep learning approach for sound event recognition using a brain inspired representation. IEEE Trans. Inf. Forensics Secur. **15**, 3610–3624 (2020). https://doi.org/10.1109/TIFS.2020.2994740
3. Podda, A.S., Balia, R., Pompianu, L., Carta, S., Fenu, G., Saia, R.: CARgram: CNN-based accident recognition from road sounds through intensity-projected spectrogram analysis. Digit. Signal Process. **147**, 104431 (2024). https://doi.org/10.1016/j.dsp.2024.104431
4. Prashanth, A., Jayalakshmi, S.L., Vedhapriyavadhana, R.: A review of deep learning techniques in audio event recognition (AER) applications. Multimed. Tools Appl. **83**, 8129–8143 (2024). https://doi.org/10.1007/s11042-023-15891-z
5. Ravanelli, M., Bengio, Y.: Speaker recognition from raw waveform with SincNet. In: Proceedings of the IEEE Spoken Language Technology Workshop (SLT), pp. 1021–1028. IEEE, Athens (2018). https://doi.org/10.1109/SLT.2018.8639585
6. Greco, A., Roberto, A., Saggese, A., Vento, M.: DENet: a deep architecture for audio surveillance applications. Neural Comput. Appl. **33**(17), 11273–11284 (2021). https://doi.org/10.1007/s00521-021-05873-3
7. Chersi, F., Bichler, O., Moineau, C., Naud, M., Soutier, L., Templier, V., Allenet, T., Kucher, I., Lorrain, V.: AIDGE: a framework for deep neural network development, training and deployment on the edge. In: River Publishers Series in Communications, pp. 41–54 (2025)
8. Gong, Yuan, Yu-An Chung, and James Glass.: Ast: Audio spectrogram transformer. arXiv preprint arXiv:2104.01778 (2021).
9. Koutini, Khaled, et al.: Efficient training of audio transformers with patchout. arXiv preprint arXiv:2110.05069 (2021).
10. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. In: Proceedings of the International Conference on Learning Representations (ICLR) (2015). https://doi.org/10.48550/arXiv.1412.6980