

DTU



Embedded AI: Small Models Are Not Simple

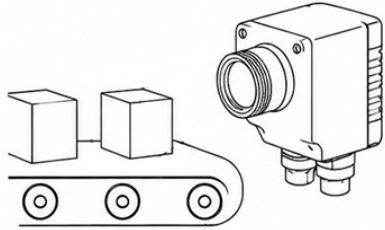
Luca Pezzarossa
Technical University of Denmark
DTU Compute

Embedded AI

Embedded AI refers to the execution of **machine-learning** models in **embedded computing systems**

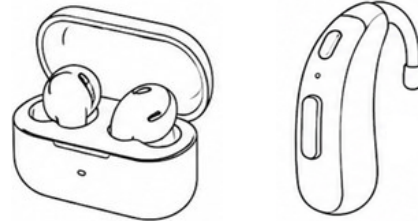
Smart industrial camera

Object detection and visual inspection



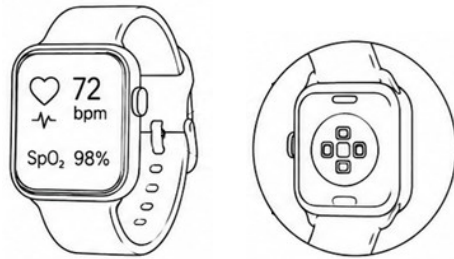
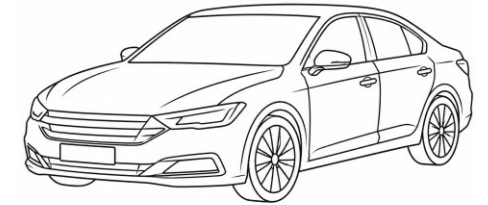
Hearing aid/earbuds

Speech enhancement

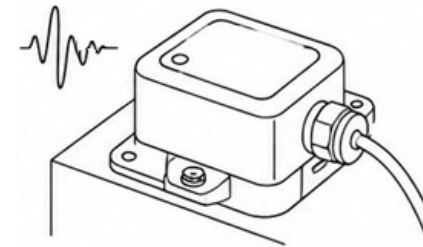


Automotive

Driver monitoring



Smartwatch or wearable sensor
Health monitoring



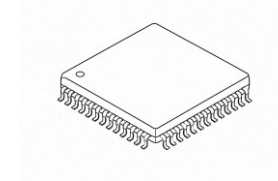
Vibration sensor node
Predictive maintenance

Cloud-based vs. Embedded AI



High compute
Large memory
Available energy
Easy deployment/updates

Needs connectivity
High latency
Privacy concerns
Hard for real-time systems

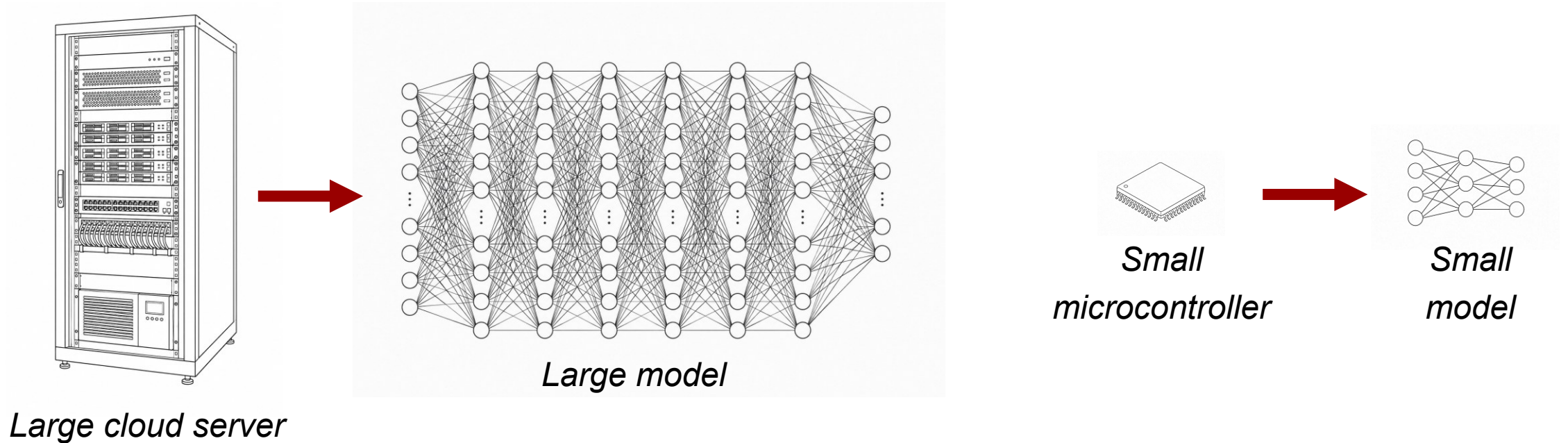


Limited compute
Limited memory
Limited energy
Hard deployment/updates

Works offline
Low latency
Better privacy
Suitable for real-time systems

Embedded AI

Embedded AI is often presented as a **model-size problem**

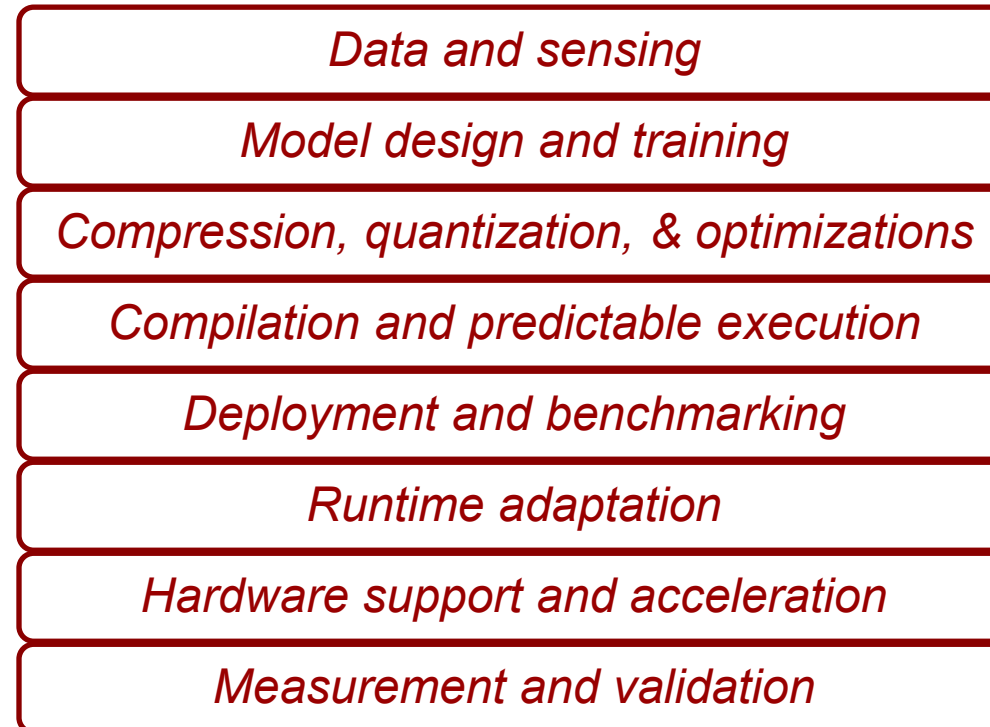


However: embedded AI is not only a small AI model on a small device

The full-stack perspective

The development of a deployable system needs a lot more

Embedded AI **spans the full stack**



Optimizing in one layer in isolation is often not enough

Our work at DTU Compute

Our embedded AI group works across the stack



Luca Pezzarossa



Xenofon Fafoutis



Massimo Pavan



Amin Hasanpour



Alessandro Cerioli



Charalampos Orfanidis



Ole Richter



Riccardo Miccini



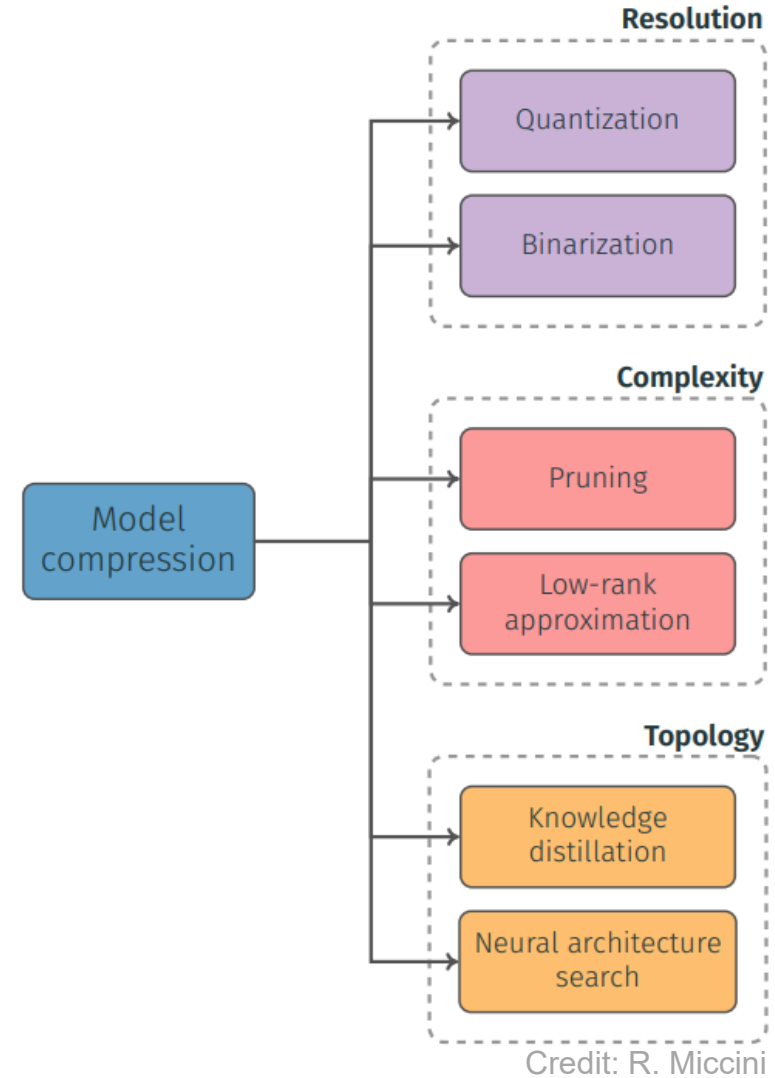
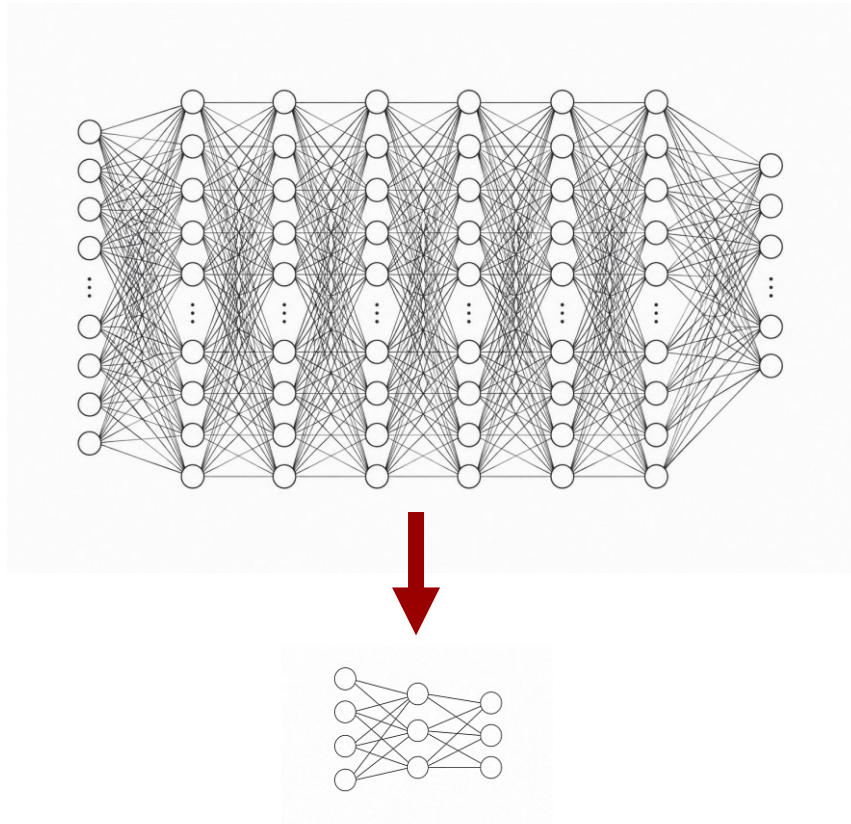
Vicenç P. Berenguer

The common theme is **deployable** and **adaptive** embedded AI

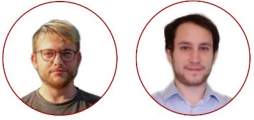
Model compression

Model compression

Compression **reduces** computational requirements while trying to **preserve** performance



Mixed precision quantization using NAS



R. Miccini et al.
[1]

Uniform quantization assumes one precision is good for everything

Some operations need more precision, others may tolerate fewer bits

Use **genetic algorithms** to find optimal bit widths for each operator

Quantization:

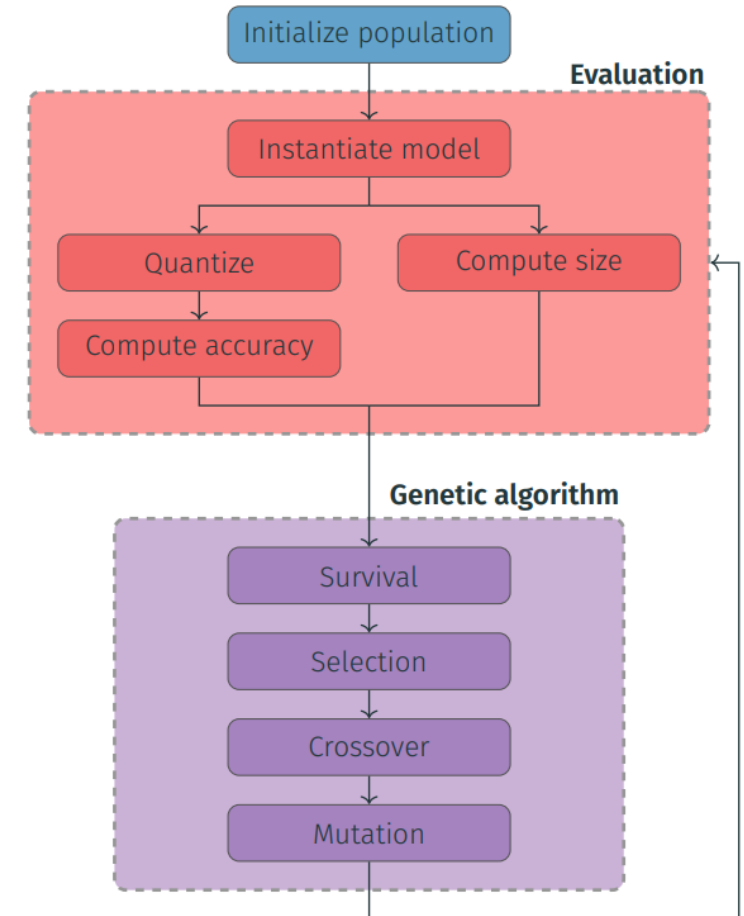
- uniform
- 2 to 8 bits
- weights and activations
- after and during training

Genome:

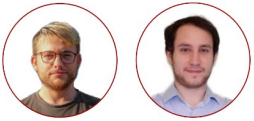
- bit widths of 17 GRU operators

Objectives:

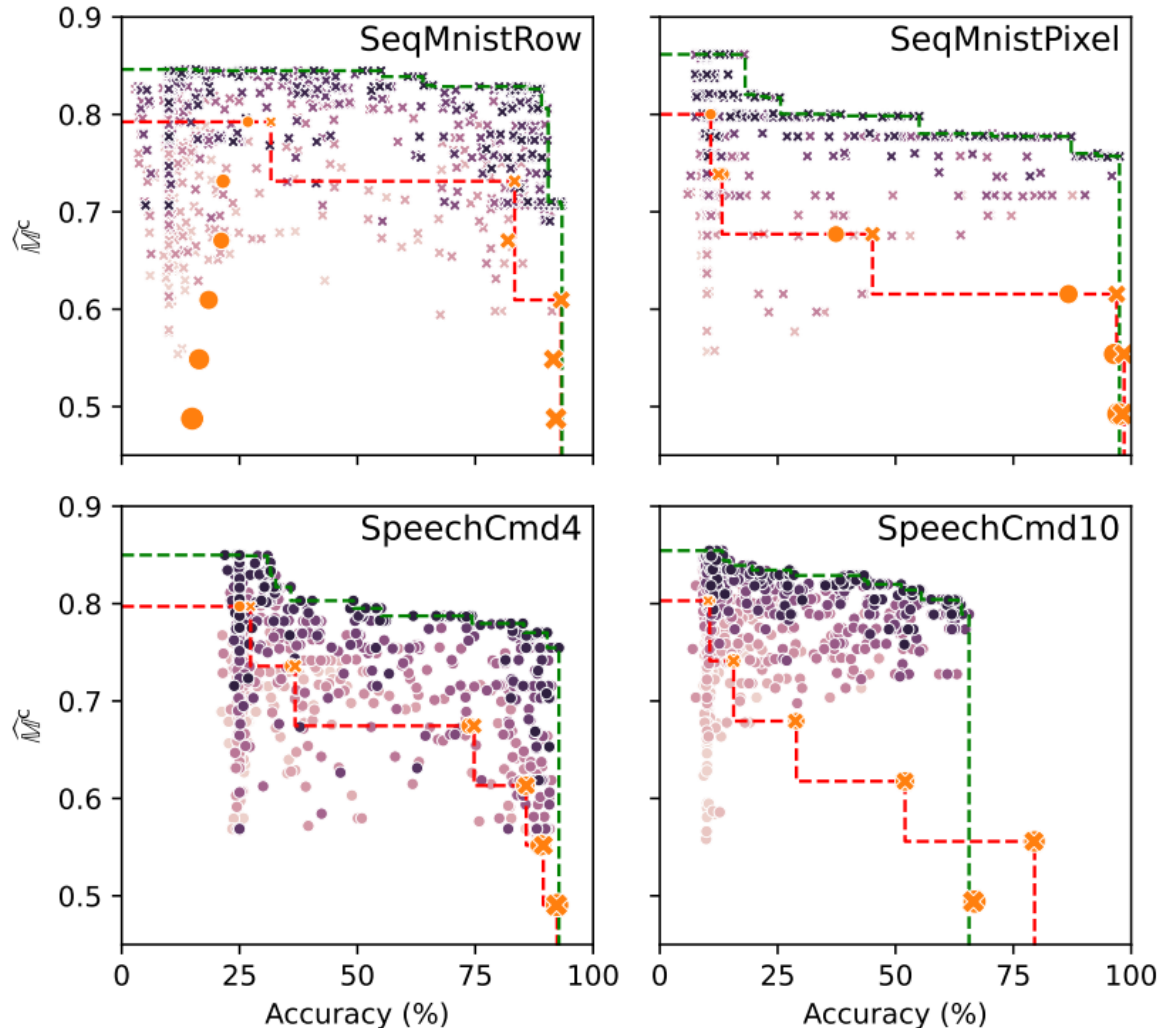
- ↓ model size
- ↑ accuracy



Mixed precision quantization using NAS



R. Miccini et al.
[1]



Dominates the Pareto front ($\nearrow$ better)

Maintains 8-bit performances while achieving 25% to 55% model size reduction

Improve consistently throughout the generations

$$\hat{M}^c = 1 - \frac{M_Q}{M_{FP16}}$$

Generation	Quantization Bits	Quantization Mode
0	3	ptq
4	4	qat
8	5	
12	6	
16	7	
Baseline	8	
		Pareto Fronts
		--- homogeneous
		- - - mixed-precision

Binary activations



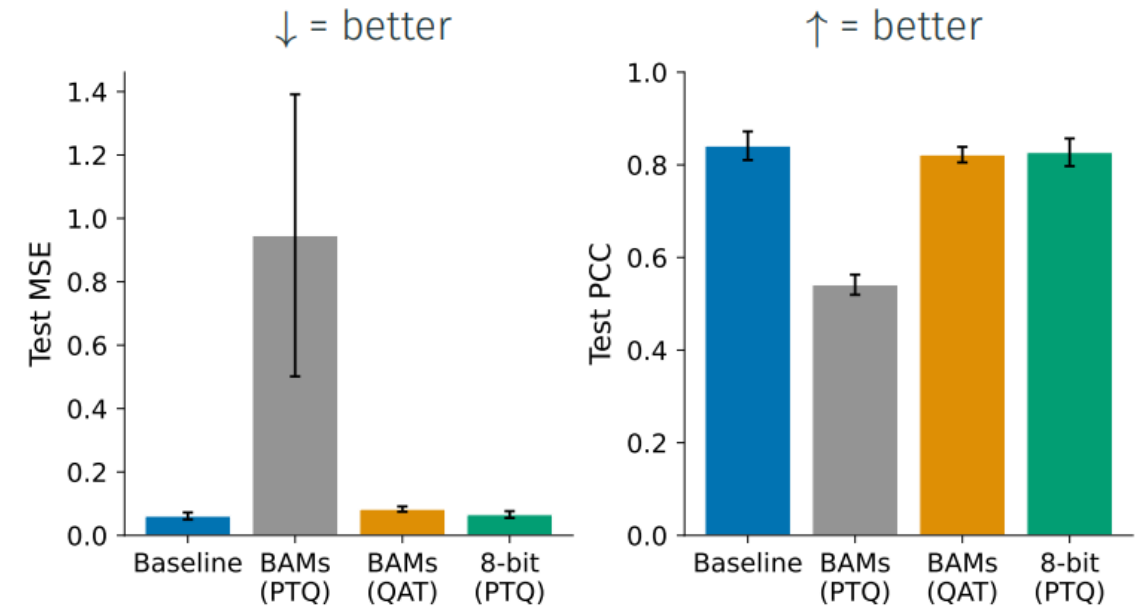
R. Miccini et al.
[2]

Replace floating-point activations with **binary activations**

Multiplications and dot products become sums

Use binarization-aware training to preserve accuracy

Results for DNSMOS convolutional model for speech quality prediction



25× reduction in memory footprint for inference

Hardware support

Hardware support is needed to leverage these benefits

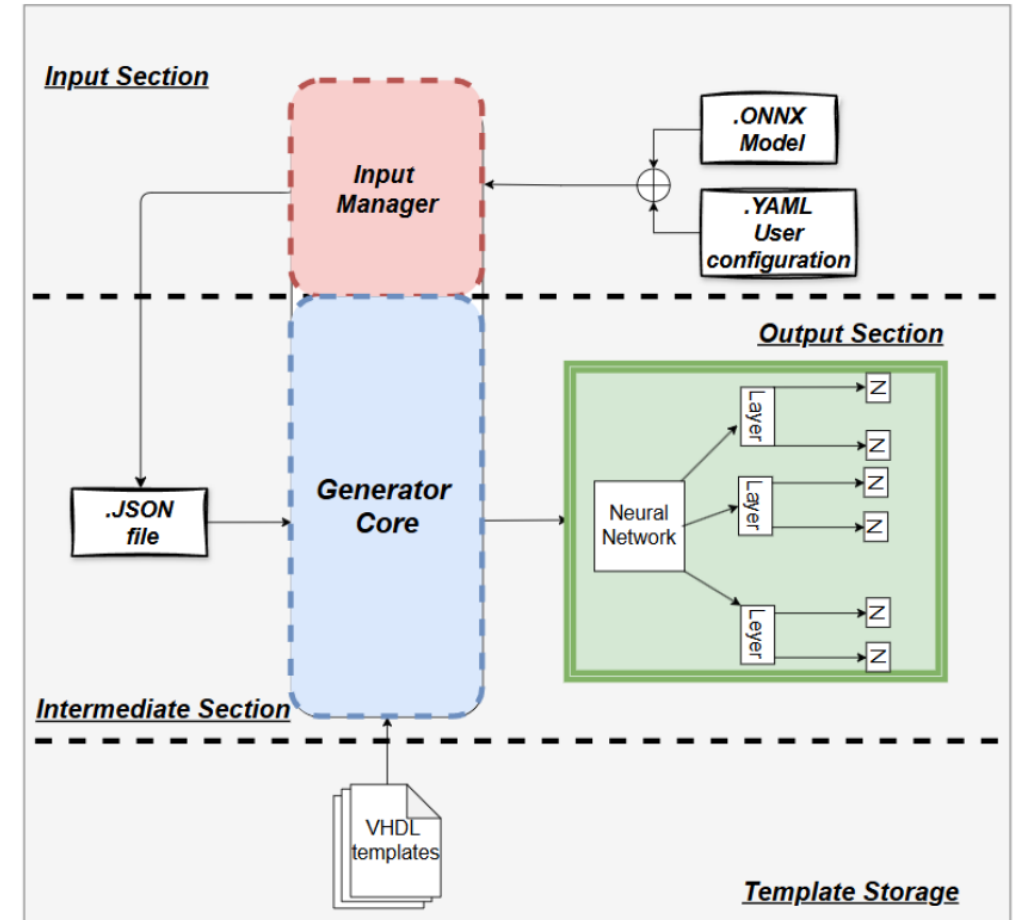
We have a **hardware generator** from ONNX

Parallelization can be configured (# of neurons)

FSM coordinates the execution

Available at:

<https://github.com/alsssss/NNGenerator/tree/main>



Credit: A. Mitola

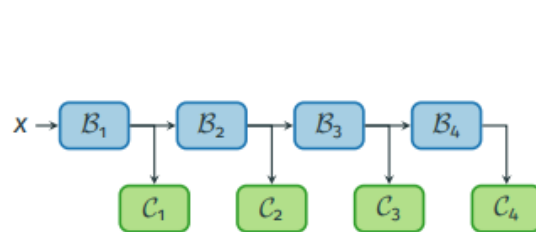
Dynamic model execution

From static to dynamic models

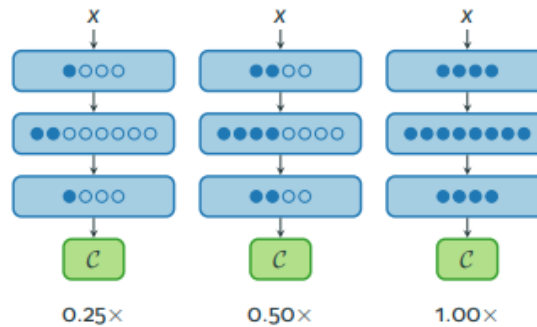
Embedded devices present changing conditions: computational budget, battery, easy/hard input, etc.

A **static model** always spends the same computation

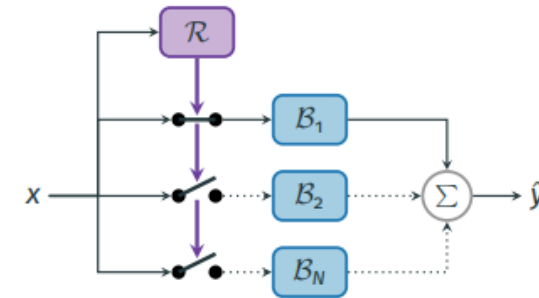
A **dynamic model** selects computation during inference



Early-exiting



Slimming



Mixture-of-experts

Adaptive slimming: One model, several widths



R. Miccini et al.
[4]

Dynamic slimming to DEMUCS

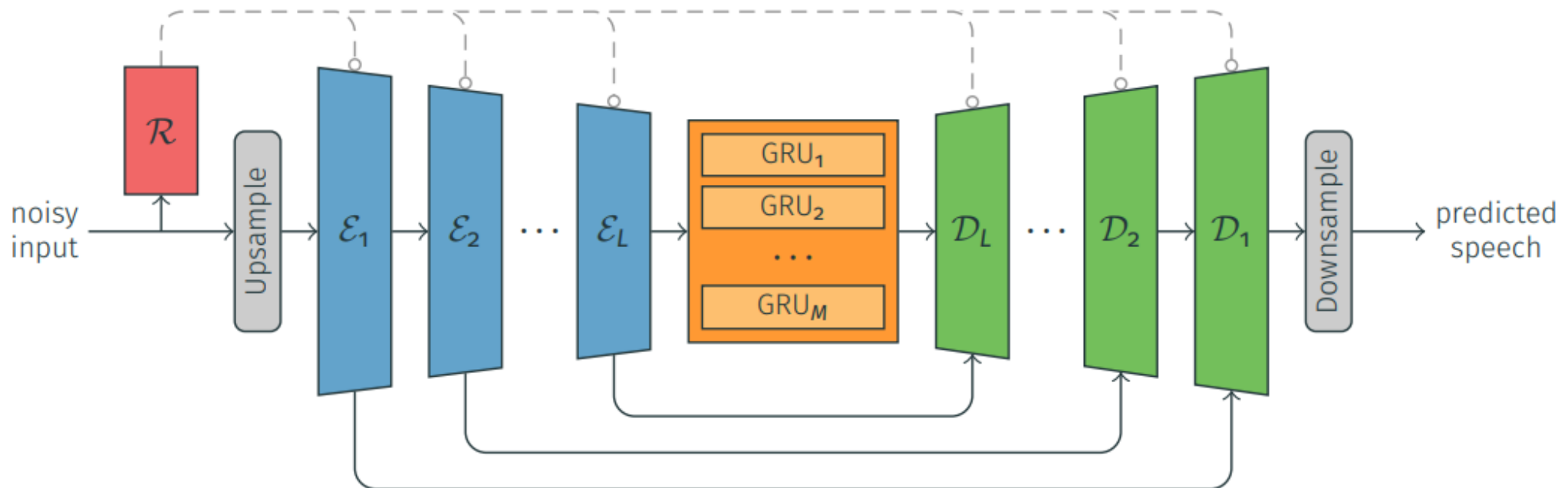
Define set of utilization factors (0.125, 0.25, 0.5, 1.0)

Slimmable encoder and decoder blocks

Lightweight router predicts utilization factor

Router decides the **utilization factor** for all blocks

(trained end-to-end)



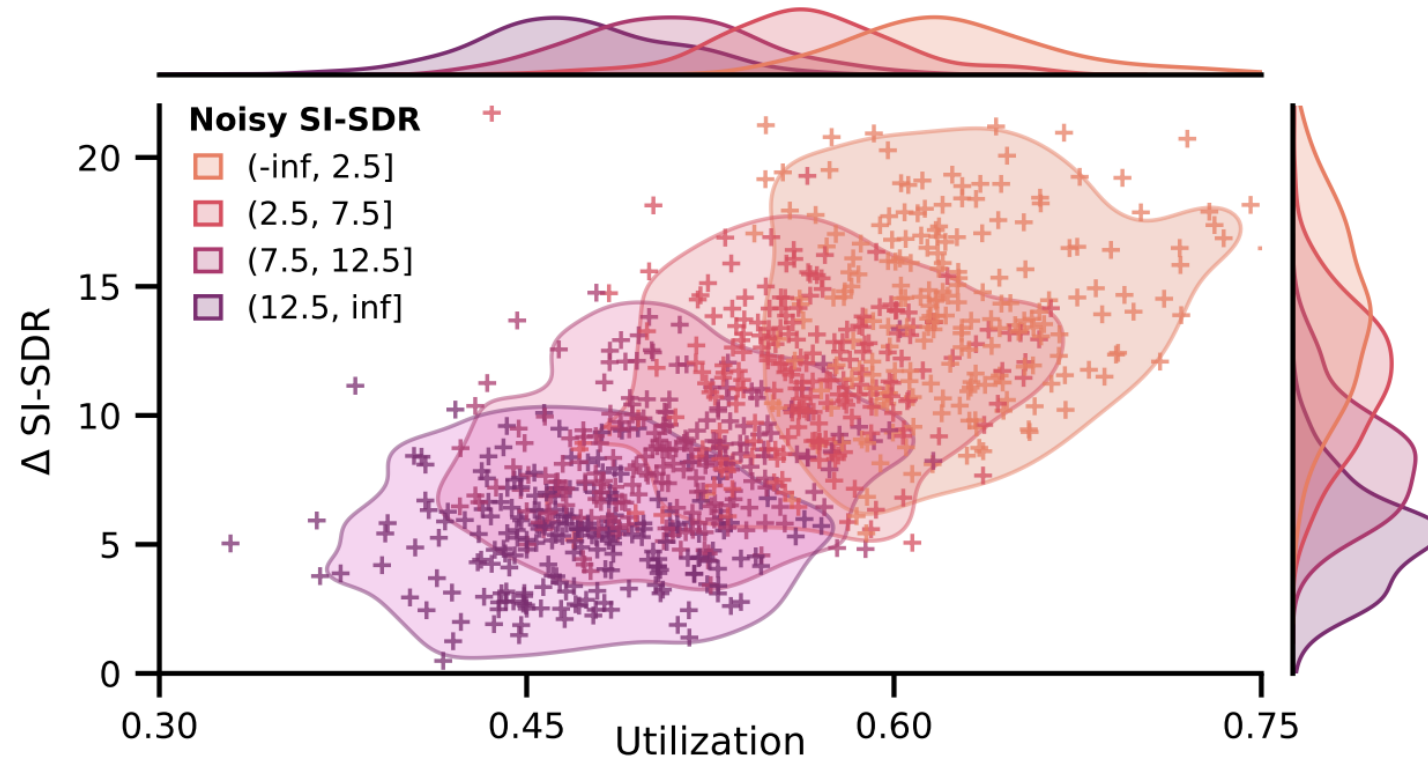
Adaptive slimming: One model, several widths



R. Miccini et al.
[4]

Noisier (harder) inputs are processed using larger utilization factors ($\rightarrow$) than cleaner inputs

Noisier (harder) inputs and obtain higher SI-SDR improvements ($\uparrow$) on average

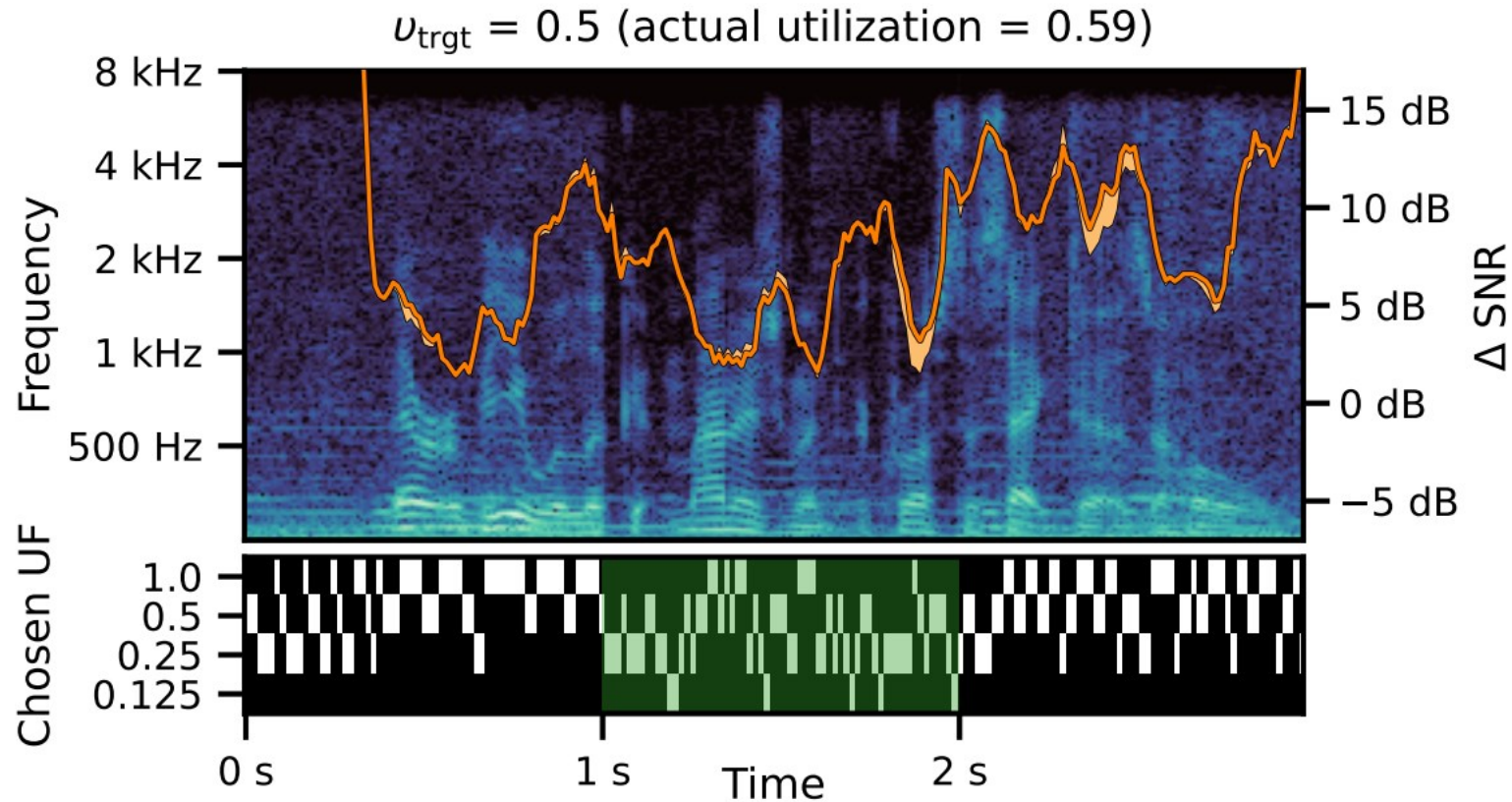


Adaptive slimming: One model, several widths



R. Miccini et al.
[1]

When the input is cleaner, the model uses smaller utilization factors



Deployment tools and benchmarking

The EdgeMark framework



Amin Hasanpour et al.
[5]

Many pipelines for embedded AI deployments are available

**Which deployment tool works best for this model,
on this hardware, under this memory and latency budget?**

EdgeMark is a framework that **automates** this pipeline and compares embedded-AI tools and configurations across real targets

The model gets converted, optimized, compiled, flashed, executed, and measured

Run controlled deployment experiments instead of manually testing one model, tool, and board at a time

Available at: <https://github.com/AminHasanpour/EdgeMark>

Models	FC	
	CNN	
	RNN	SimpleRNN
		LSTM
		GRU
	MLPerf Tiny	AE
		DS CNN
		ResNet
		MBNet
MBNetV2		

Tools	TFLM	
	Edge Impulse	
	Ekkono	
	Renesas eAI Translator	

Datasets	MNIST	
	CIFAR-10	
	ImageNet-V2	
	Boston Housing	
	Visual Wake Words	
	Shakespeare	
	Sinus	
	ToyADMOS	
	RandomSet	Classification
		Regression
		Sequence

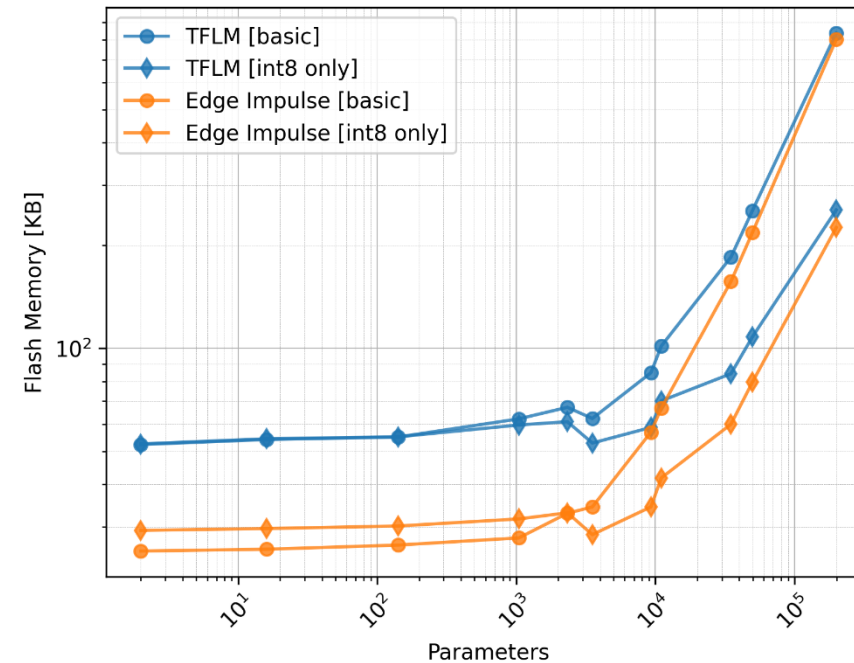
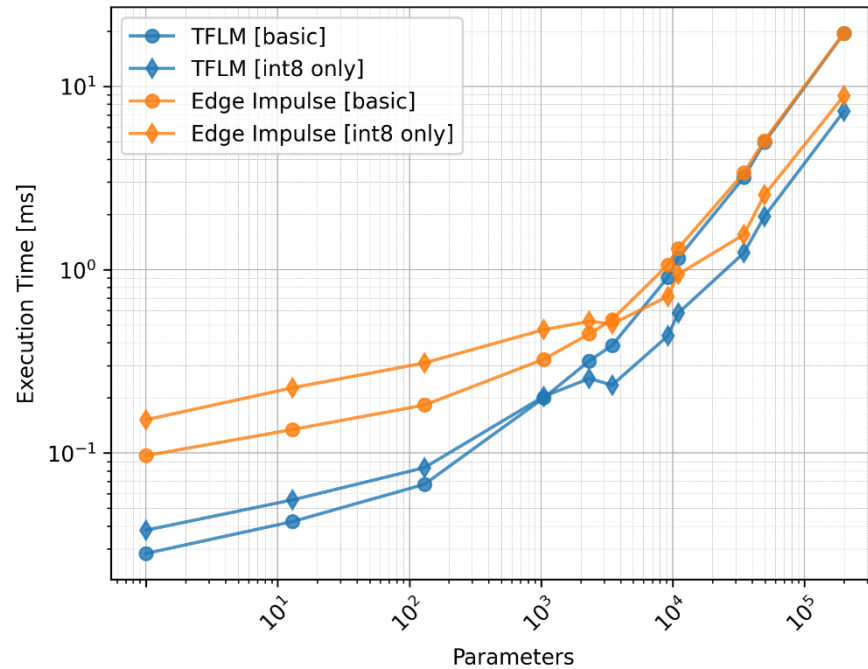
Hardware	STM	NUCLEO-L4R5ZI
	Renesas	RX65N

The EdgeMark framework



Amin Hasanpour et al.
[5]

Fully connected NN models perform on an STM microcontroller board when deployed with Edge Impulse and TensorFlow Lite for Microcontrollers (TFLM)



Compilation

NeuralCasting: From model to generated code



A. Cerioli et al.
[6,7]

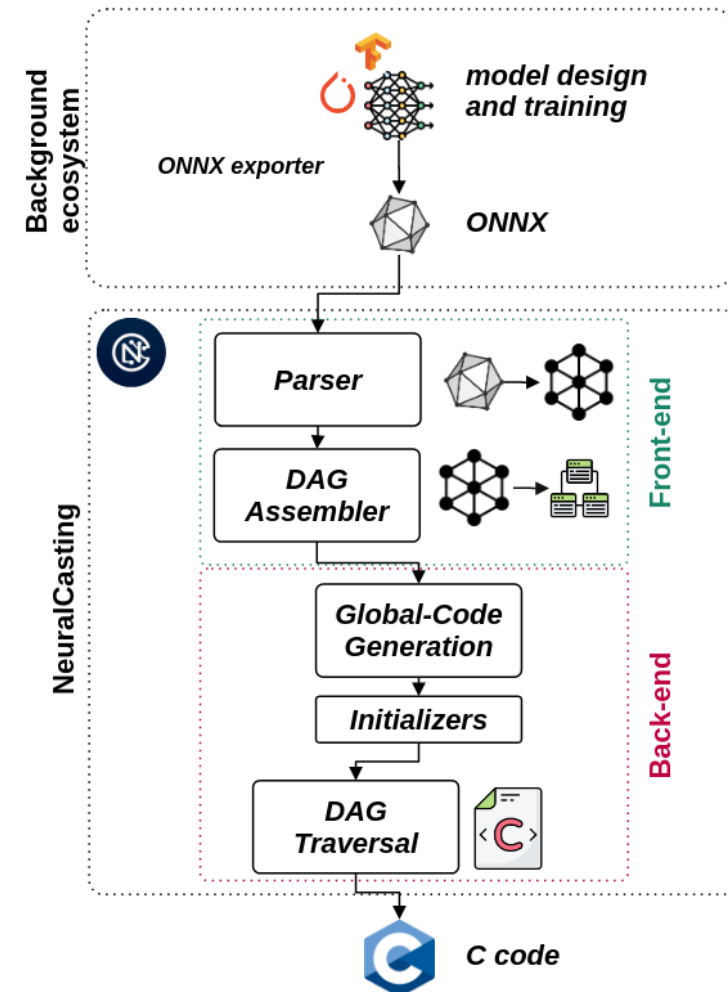
Inference engines are convenient

But they may introduce:

- runtime and memory overhead
- dependencies
- limited execution control
- and limited time predictability

NeuralCasting takes another path:

compile the network into native bare metal C code



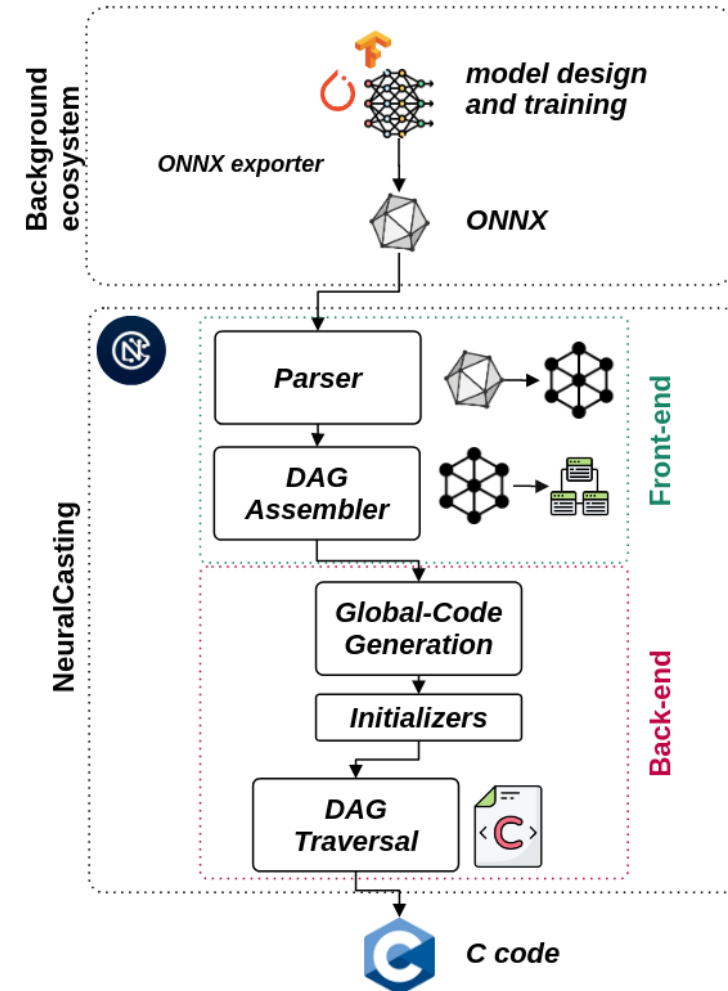
NeuralCasting: From model to generated code



A. Cerioli et al.
[6,7]

Full control over execution
Minimal memory footprint overhead
Suitable for bare metal embedded devices
High portability
Time-predictability

High maintainability effort
Low hardware specific optimizations
Low ecosystem support
Limited scalability



NeuralCasting: From model to generated code



A. Cerioli et al.
[6,7]

Some embedded-AI systems are **real-time systems**

To know the worst-case execution time, we need:

- deterministic memory behavior
- bounded loops
- static allocation
- predictable hardware/software interaction

NeuralCasting + Patmos + Platin

End-to-end NSNet2 WCET

Metric	Value
WCET (cycles)	64,397,009
Cache-Max-Cycles-Instr	6,405
Cache-Min-Hits-Instr	18,705
Cache-Max-Misses-Instr	39
Cache-Max-Cycles-Stack	0
Cache-Max-Misses-Stack	0
Cache-Max-Cycles-Data	36,549,597
Cache-Min-Hits-Data	0
Cache-Max-Misses-Data	1,732,400
Cache-Max-Stores-Data	8,057
Cache-Unknown-Address-Data	1,740,457
Cache-Max-Cycles	36,556,002

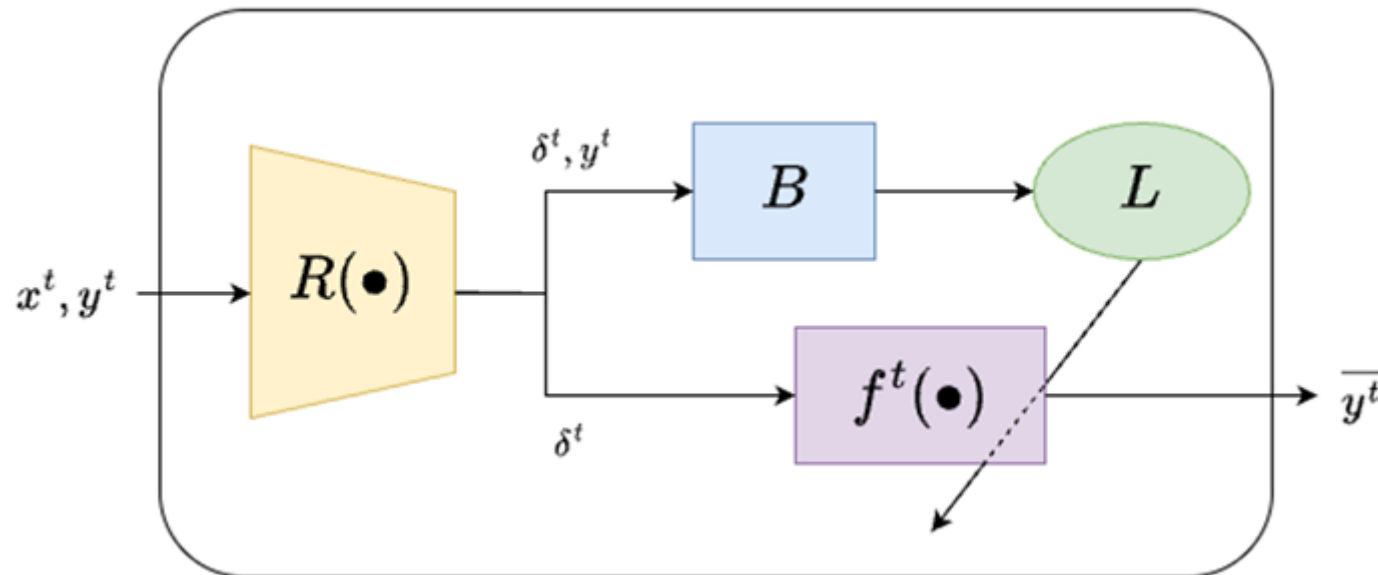
On-device learning

On-device learning

Training data and deployment data often **differ**

After deployment: users change, environments and sensors drift, ...

A deployed embedded-AI system may need to **learn locally**

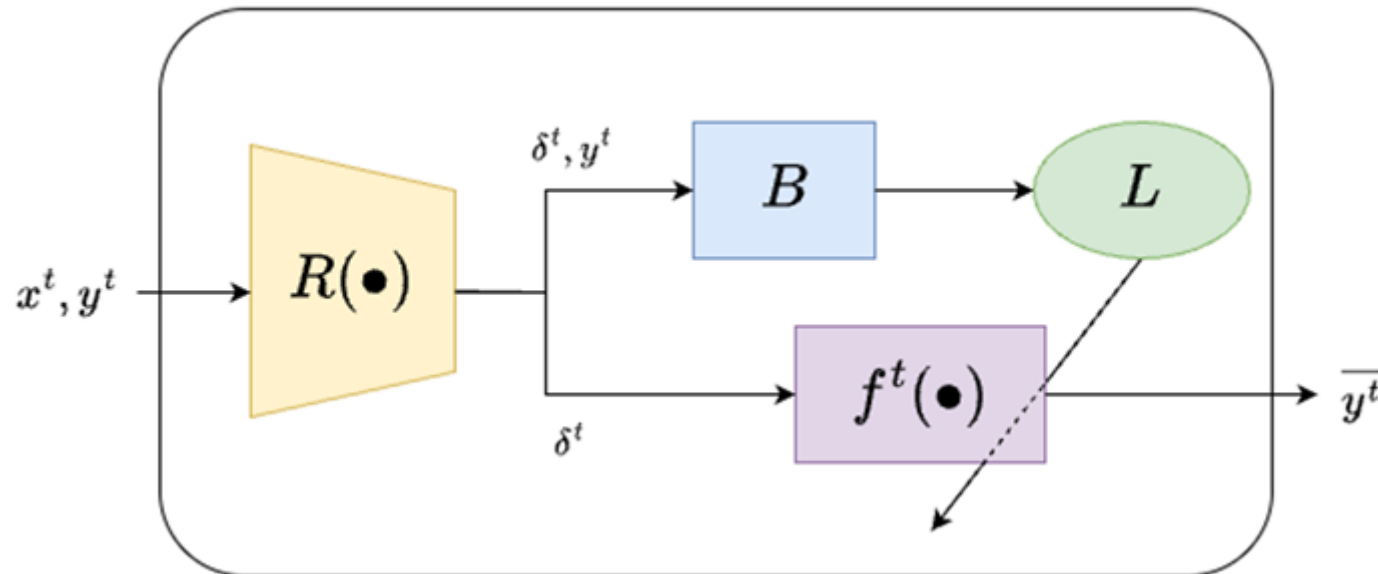


On-device learning

Literature review for ODL with microcontrollers

Gap between **benchmark papers** and **real deployments**

New library for ODL: ODL designed as a full pipeline, not just as a learning algorithm



Specialized hardware

Asynchronous NNs and event-driven AI hardware



O. Richter et al.
[9]

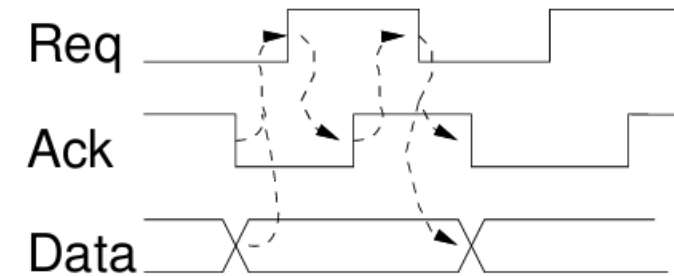
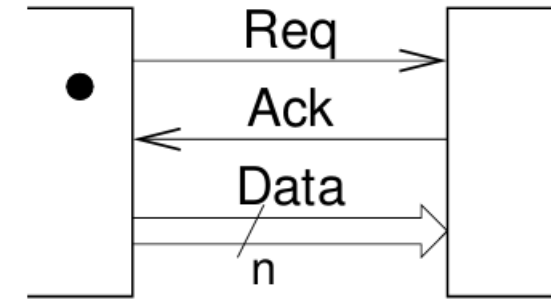
Many embedded-AI workloads are naturally **sparse**

- sensors do not always produce useful data
- events may arrive irregularly
- activity changes over time

Many computations **are unnecessary** when nothing changes

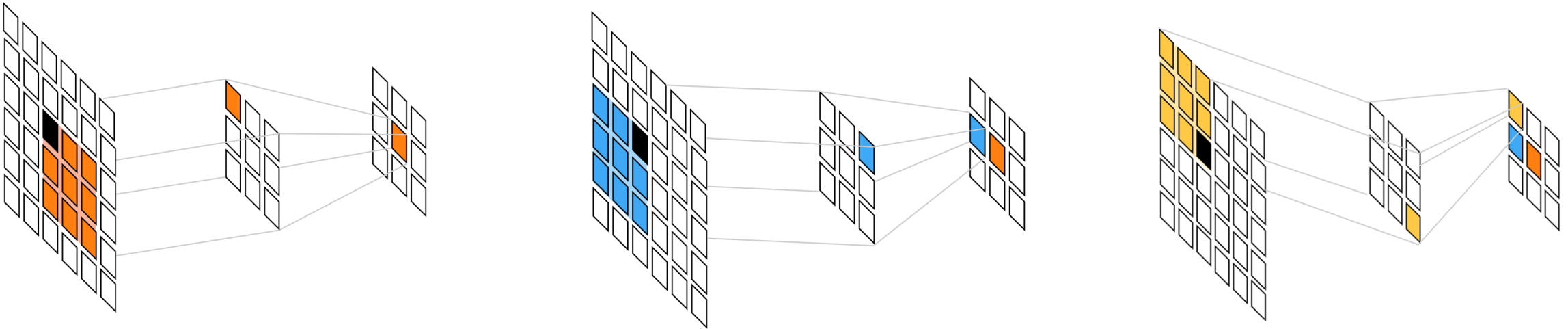
Asynchronous NN hardware exploits this sparsity

The circuit is active only when data needs to be processed



Credit: J. Sparsø

Speck1: smart vision SoC, 128×128 **event-based** vision sensor and **asynchronous** spiking CNN



One incoming visual event can be processed through **9 CNN** layers in **3.36 μ s**

Takeaways

Takeaways

Small models are not simple

Embedded AI exposes the full stack: models, compression, tools, runtime, hardware, and validation

Efficiency is not a property of the model alone

It depends on the deployment toolchain, the target device, and the execution context

Embedded AI is not only about making models smaller

It is also about making them deployable, adaptive, and predictable

Thank you!

Embedded AI group at DTU Compute



Luca Pezzarossa



Xenofon Fafoutis



Massimo Pavan



Charalampos Orfanidis



Ole Richter



Alessandro Cerioli



Riccardo Miccini



Amin Hasanpour



Vicenç P. Berenguer

References

1. R. Miccini, A. Cerioli, C. Laroche, T. Piechowiak, J. Sparsø, and L. Pezzarossa. Towards a Tailored Mixed-Precision sub-8-bit Quantization Scheme for Gated Recurrent Units using Genetic Algorithms. In TinyML Research Symposium. 2024. DOI: <https://doi.org/10.48550/arXiv.2402.12263>
2. M. Nilsson, R. Miccini, C. Laroche, T. Piechowiak, and F. Zenke. Resource-Efficient Speech Quality Prediction through Quantization Aware Training and Binary Activation Maps. In Interspeech. 2024. DOI: <https://doi.org/10.21437/Interspeech.2024-1979>
3. A. Mitola. Design and Implementation of Hardware Accelerators for Embedded AI. MSc thesis. DTU Department of Applied Mathematics and Computer Science. 2025. Link: <https://findit.dtu.dk/en/catalog/6897e3418b7fcb0102d2dc30>
4. R. Miccini, M. Kim, C. Laroche, L. Pezzarossa, and P. Smaragdis, Adaptive Slimming for Scalable and Efficient Speech Enhancement. In IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA). 2025. DOI: <https://doi.org/10.1109/WASPAA66052.2025.11230950>
5. M. A. Hasanpour, M. Kirkegaard, X. Fafoutis. EdgeMark: An Automation and Benchmarking System for Embedded Artificial Intelligence Tools. Journal of Systems Architecture. 2025. DOI: <https://doi.org/10.1016/j.sysarc.2025.103488>
6. A. Cerioli, R. Miccini, C. Laroche, T. Piechowiak, L. Pezzarossa, J. Sparsø, and M. Schoeberl. NeuralCasting: A Front-End Compilation Infrastructure for Neural Networks. In 11th International Conference on Internet of Things: Systems, Management and Security (IOTSMS). 2024. DOI: <https://doi.org/10.1109/IOTSMS62296.2024.10710209>
7. A. Cerioli, T. B. Strøm, C. Laroche, T. Piechowiak, L. Pezzarossa, M. Schoeberl. Time-predictable Deep Noise Suppression on an Edge Device. In International Symposium on Real-Time Distributed Computing (IEEE ISORC). 2025. DOI: <https://doi.org/10.1109/ISORC65339.2025.00062>
8. M. Pavan, L. Pezzarossa, F. Pittorino, M. Roveri, X. Fafoutis. What Changes After Deployment? A Survey on On-device Learning in TinyML. Submitted. DOI: <https://doi.org/10.48550/arXiv.2605.31226>
9. O. Richter, Y. Xing, M. De Marchi, C. Nielsen, M. Katsimpris, R. Cattaneo, Y. Ren, Y. Hu, Q. Liu, S. Sheik, T. Demirci, N. Qiao. Speck: A Smart Event-Based Vision Sensor with a Low Latency 327k Neuron Convolutional Neuronal Network Processing Pipeline. arXiv preprint. Link: <https://arxiv.org/abs/2304.06793>

Embedded AI: Small Models Are Not Simple

Luca Pezzarossa
Technical University of Denmark
DTU Compute

DTU

