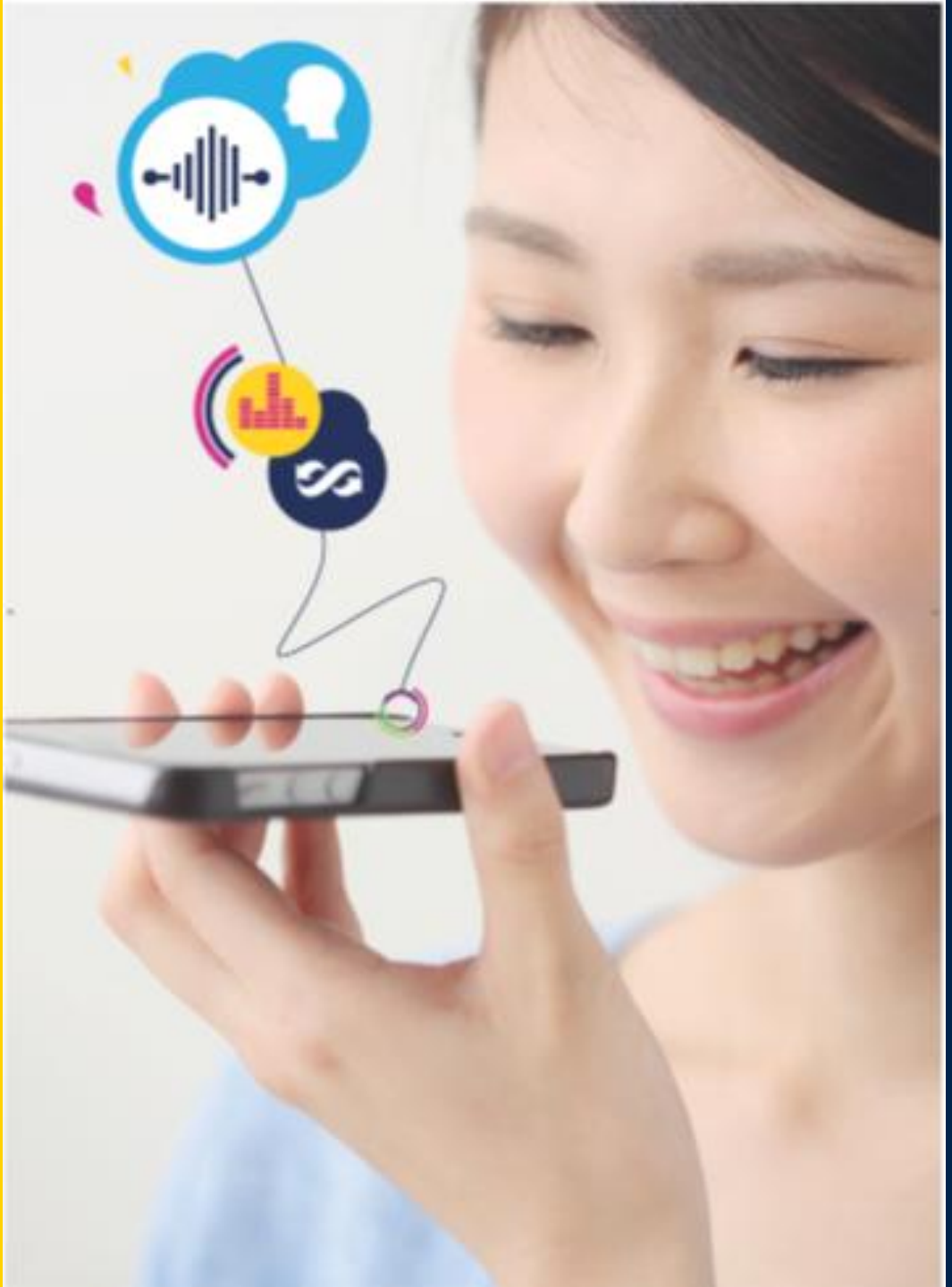


Vocal User Interfaces @ edge

STMicroelectronics

System Research & Applications

Ing. Ivana Guarneri

Agenda

1. Vocal User Interfaces
2. Keyword Spotting with demo@STM32
3. Vocal Commands Recognition with demo@STM32
4. Tiny Vocal Commands Recognition on MCU
5. Basic on Transformers
6. Transformers for Vocal Commands Recognition @STM32

Vocal User Interfaces for MCU

Voice Applications

Voice Interface

Speech as a computer interface has numerous benefits over traditional interfaces using mouse and keyboard:

- speech is natural for humans, requires no special training;
- improves multitasking by leaving the hands and eyes free;
- is often faster and more efficient to transmit than the information provided using conventional input methods.

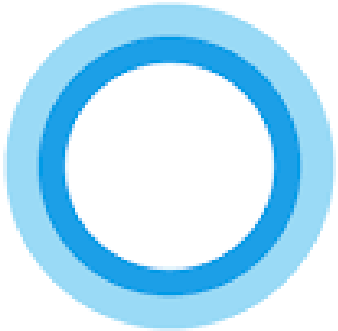
"Voice is the next great interface"
***Daren Gill**, director, Amazon Alexa*.

Intelligent Personal Assistent

- A voice assistant is a digital assistant that uses voice recognition, speech synthesis, and natural language processing (NLP) to provide a service through a particular application.

- Voice assistants allow us to do a variety of tasks hands-free, which is a major reason many people like using them, especially on their phones.

Siri



Alexa

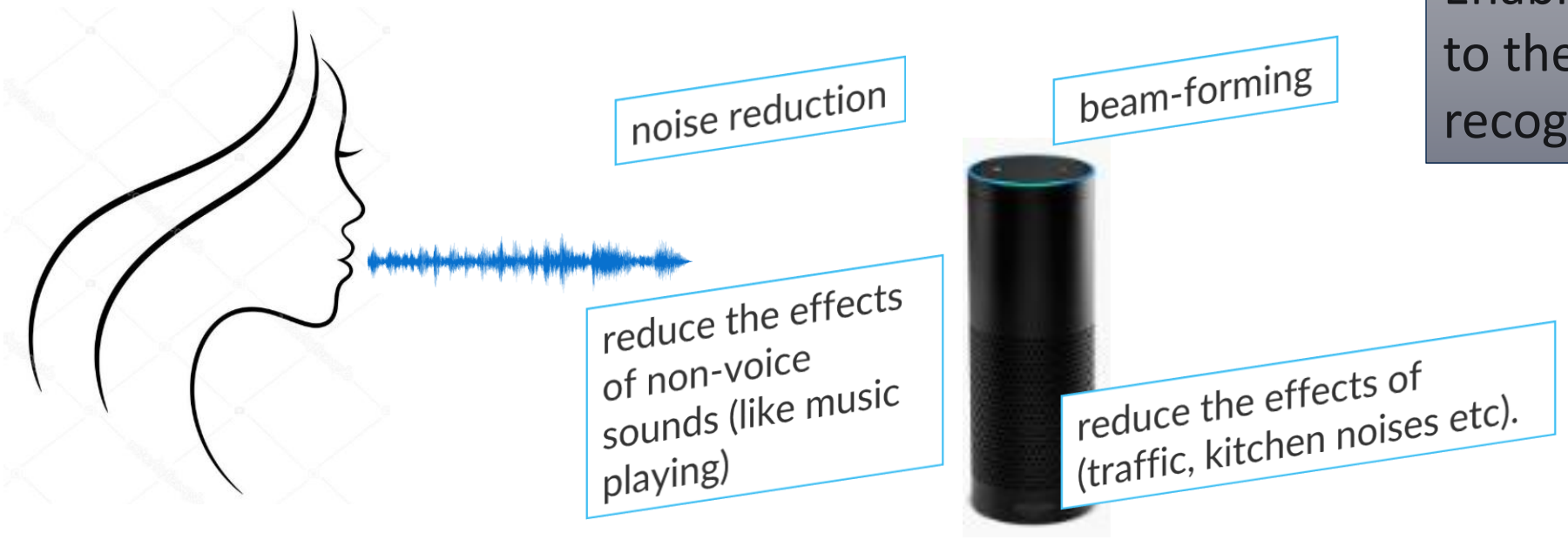
Google Assistant

Keyword Spotting as trigger for ASR

- Most of the major speech engines deploy a combination of ***embedded plus cloud-based recognition***. All of these cloud-based recognition systems use ***embedded "trigger" phrases or keywords*** to open the cloud connection to ready itself for the speech recognition.
- The ***keyword spotting (KWS) deals with the identification of keywords in utterances***, typically the *KWS* is used for enabling speech-based user interactions on smart devices.

Keyword Spotting as trigger for ASR

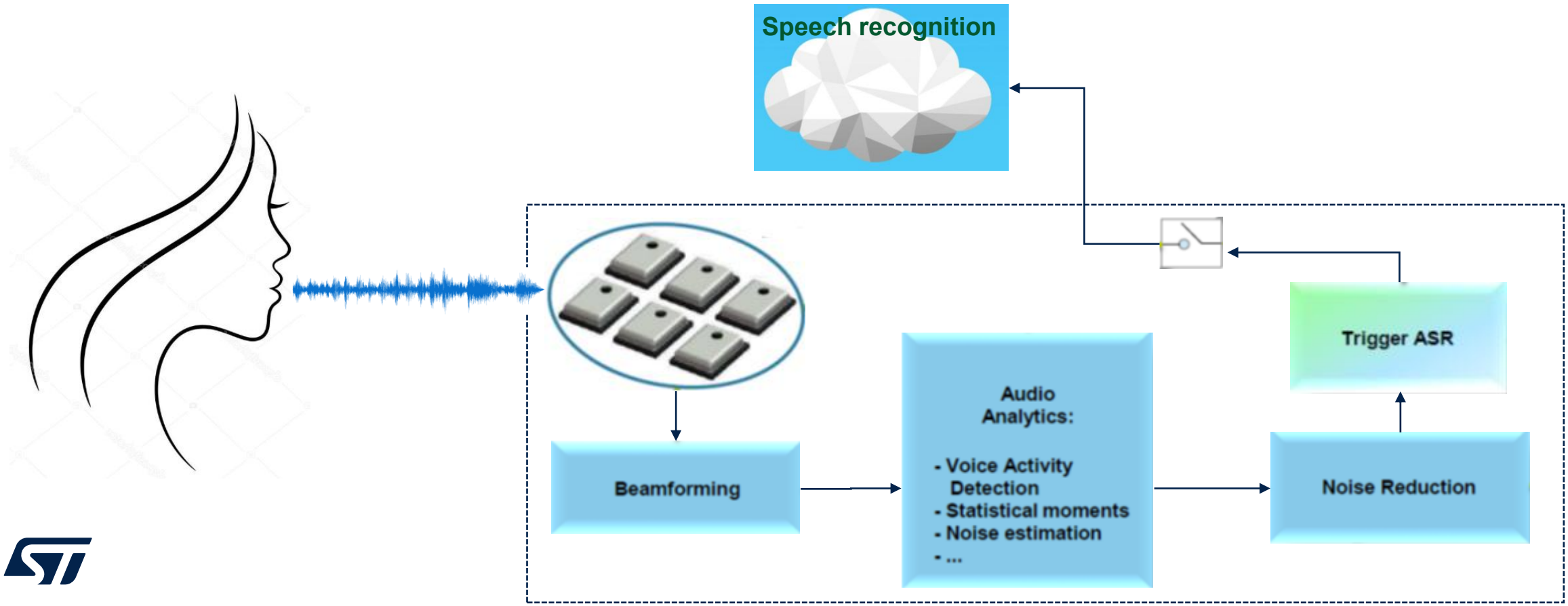
In speech processing ***the keyword spotting is the algorithm that is always listening for a wake word*** or phrase so that a phone, smart speaker, or something else can begin communicating with a server to do its job.



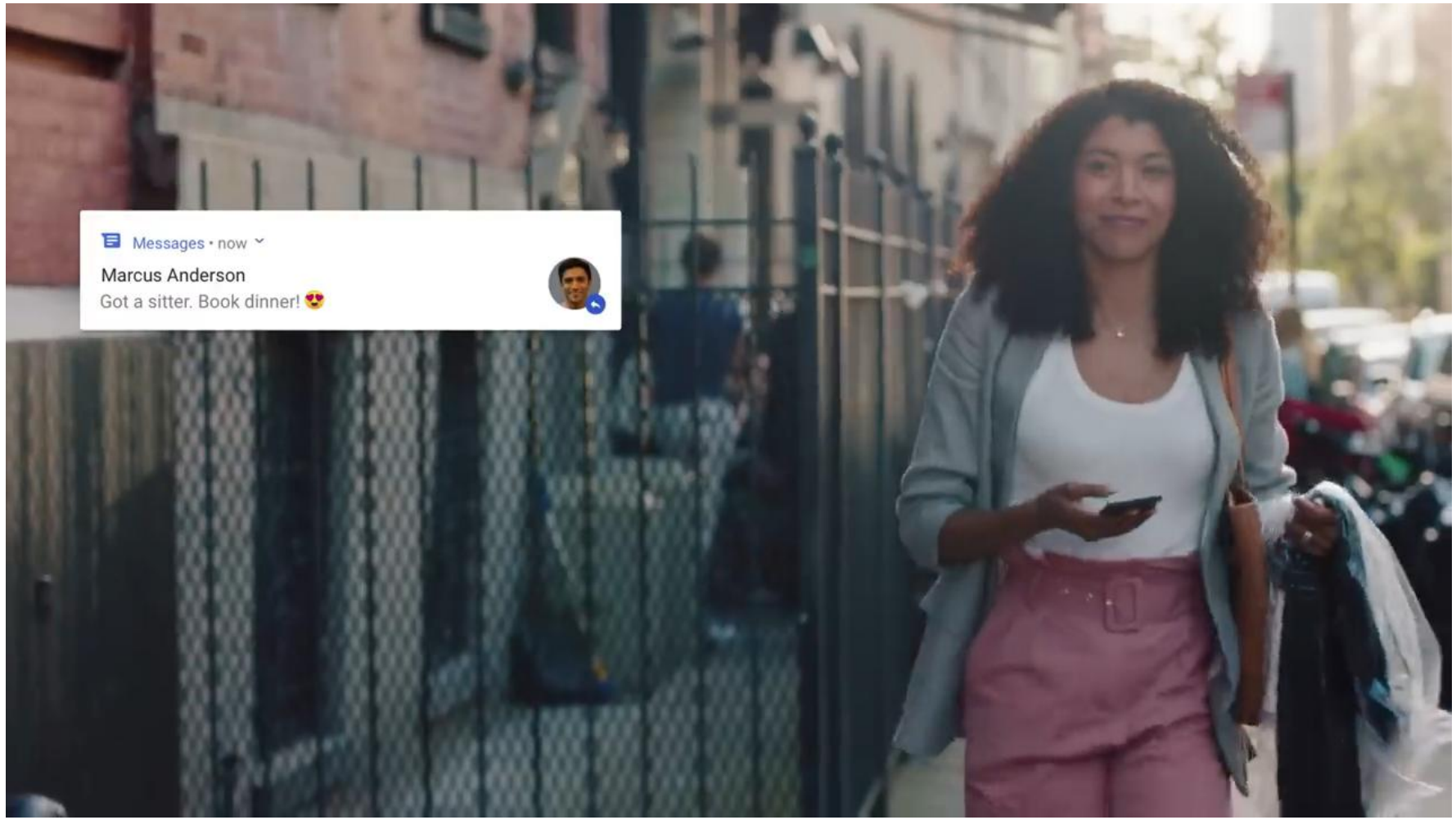
Enable the audio streaming to the cloud for speech recognition

Keyword Spotting as trigger for ASR

In speech processing ***the keyword spotting is the algorithm that is always listening for a wake word*** or phrase so that a phone, smart speaker, or something else can begin communicating with a server to do its job.



Google’s AI Assistant



https://www.youtube.com/watch?v=-qCanuYrR0g

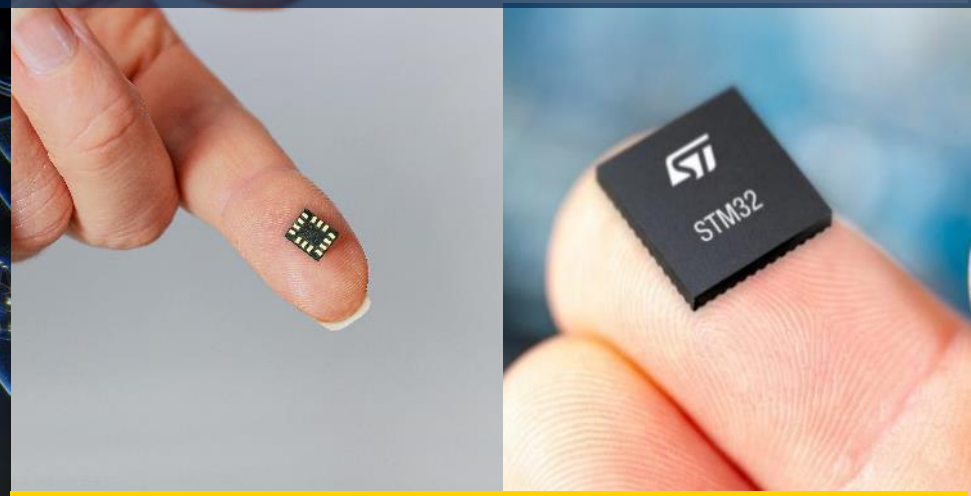
The explosion of AI-enabled devices is accelerating the inference shift from the cloud to the tiny edge

Inference **in the cloud** → Inference **on the device** → Inference **at the edge**

Cloud-centric

On device-centric

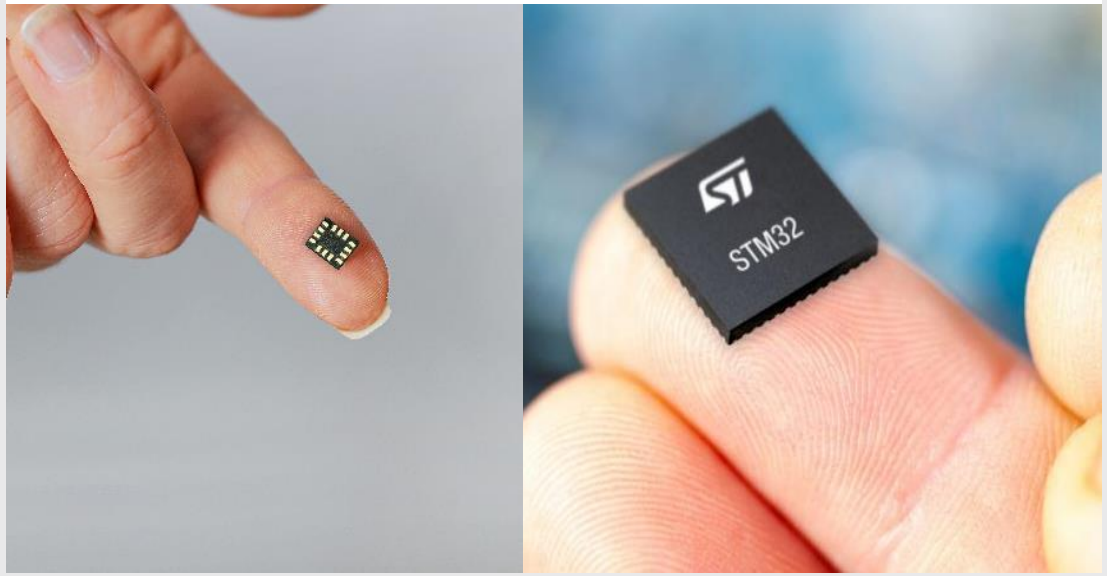
Tiny edge-centric



Enabled by a different class of hardware and software. Making AI more sustainable.

Inferring at the edge brings substantial benefits

01
10 **Reduced data transmission to generate meaningful information**



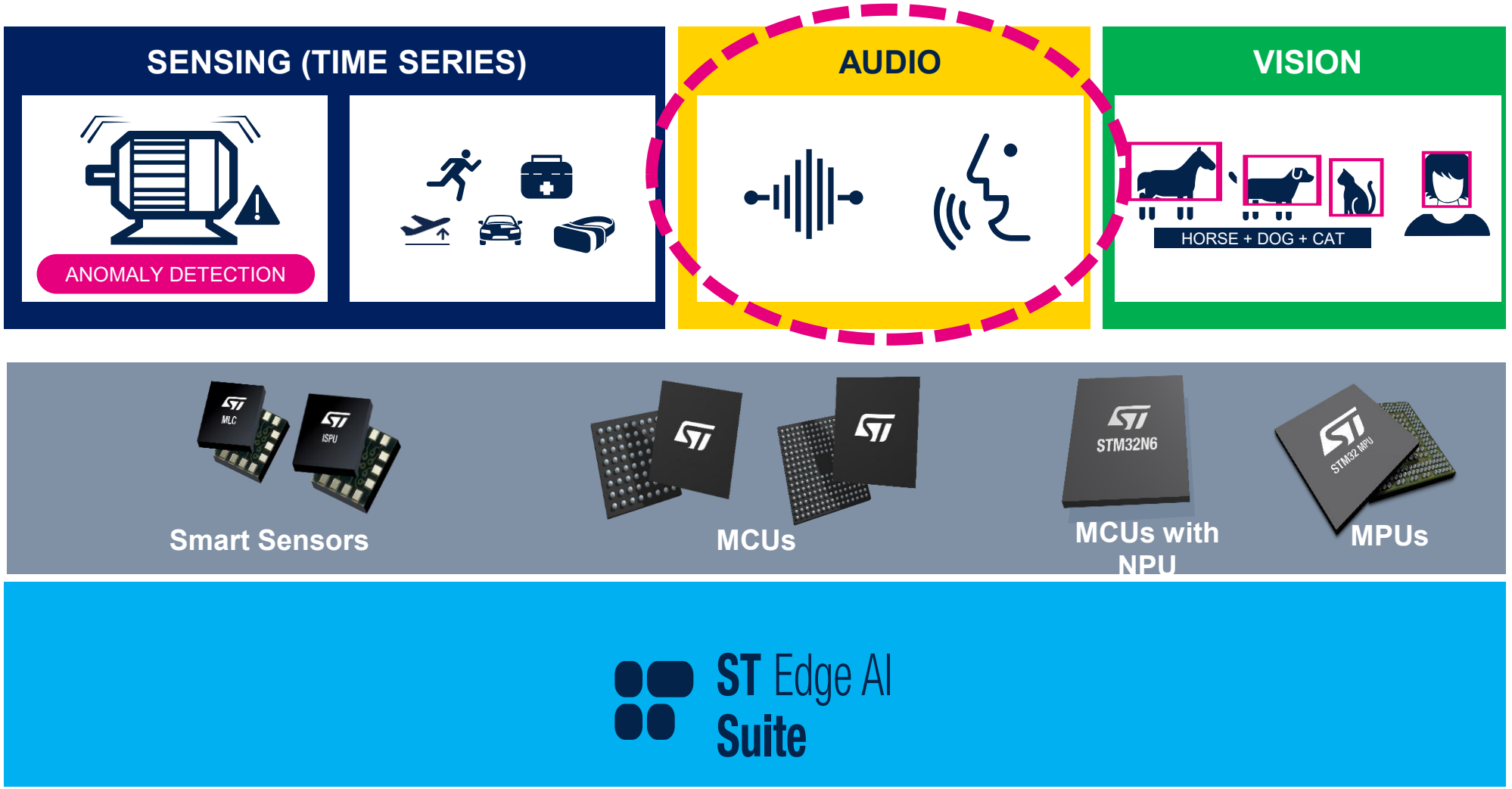
Ultralow latency
Real-time applications

Enhanced privacy and security
No data sharing in the cloud

Sustainable on energy
Low data, low power

Lower cost of inference to enable a new class of operations

Edge AI: ST solutions for multiple applications



Voice User Interfaces

Complexity of the Voice DL solutions

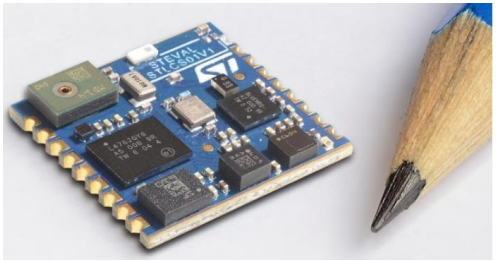
Automatic Speech Recognition

Recognition of a set of vocal commands

Cloud

Edge

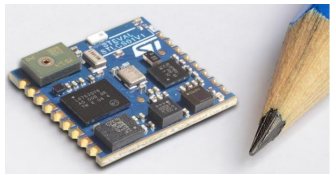
Deep Edge



Keyword Spotting with demo @STM32

Keyword Spotting on STM32

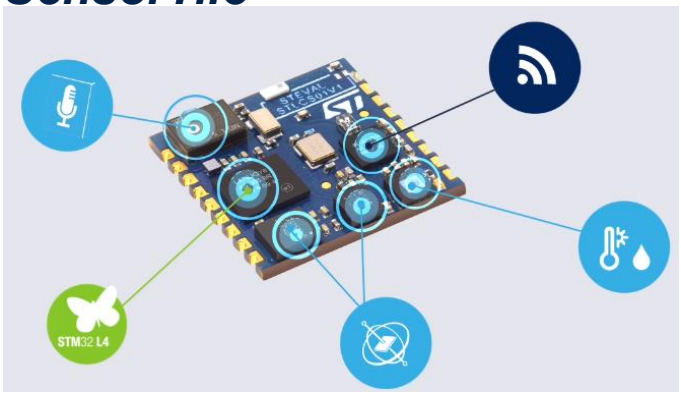
Keyword Spotting: to recognize a keyword against other words or environmental noises



→ Wake up the device

→ Continue to sleep

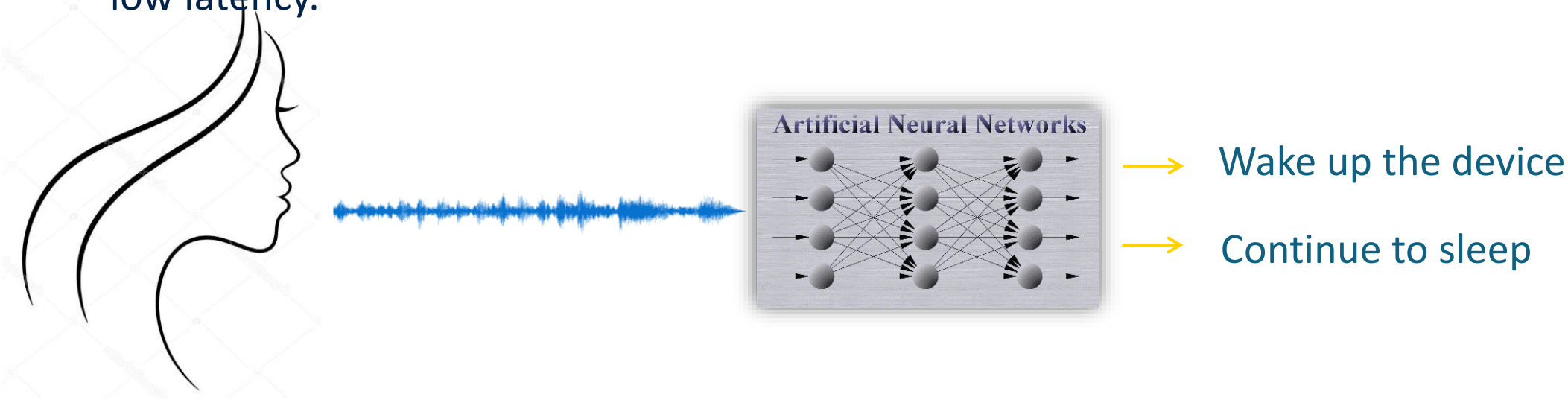
SensorTile



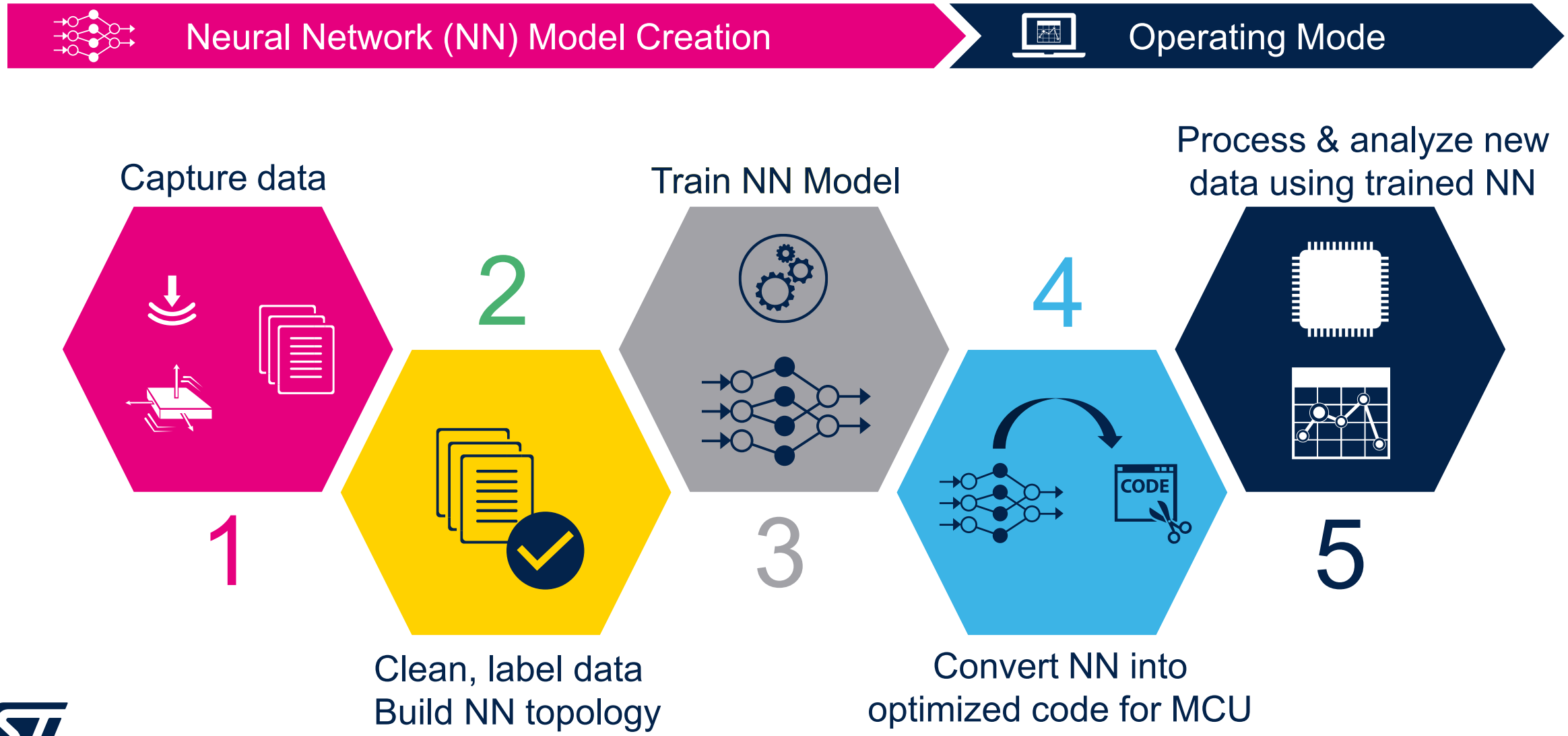
- STM32L476 @80MHz ;
- One digital microphone;
- 1MB FLASH;
- 128 KB RAM;

Keyword Spotting on the Edge

- ***Deep Edge computing is done at or near the source of the data;***
- ***To provide a real time user experience the KWS should ensure:***
  - low power consumption;
  - low memory requirements;
  - high accuracy;
  - low latency.



The key steps behind Neural Networks



Neural Network Model Creation

Database

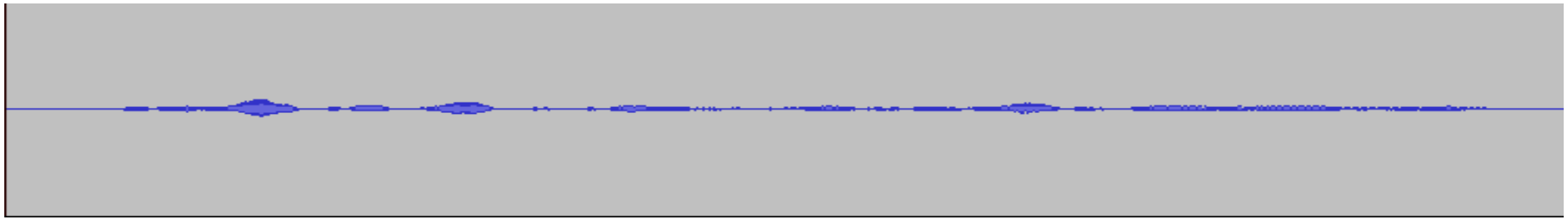
- TIMIT Acoustic-Phonetic Continuous Speech Corpus
- TIDIGITS
- 2000 HUB5 English Evaluation Transcripts
- WSJCAM0 Cambridge Read News
- Switchboard-1 Release 2
- NTIMIT
- CSR-II (WSJ1) Complete
- Switchboard Cellular Part 1 Transcribed Audio
- CSR-I (WSJ0) Complete
- Switchboard Cellular Part 1 Audio
- 2000 HUB5 English Evaluation Speech
- Fisher English Training Part 2, Transcripts
- Switchboard-2 Phase I
- Switchboard Cellular Part 1 Transcription
- Multi-Channel WSJ Audio
- Switchboard-2 Phase II
- Fisher English Training Speech Part 1 Transcripts

NN architecture

- Long Short term Memory
- Feed Forward
- Recurrent Neural Network
- Deep Convolutional Inverse Graphics Network
- Auto Encoder
- Kohonen Network
- Radial Basis Network
- Deconvolutional Network
- Variational AE
- Deep Feed Forward
- Hopfield Network
- Extreame Learning Machine
- Generative Adversarial Network
- Deep Belief Network
- Boltzmann Machine
- Restricted BM
- Denoising AESparse AEMarkov Chain
- Gated Recurrent Unit
- Echo State Network
- Deep Convolutional Network
- Liquid State Machine
- Neural Turing Machine
- Deep Residual Network

Speech Corpus example

- ***TIMIT*** *Acoustic-Phonetic Continuous Speech Corpus*



LDC93S1.txt → 0 46797 She had your dark suit in greasy wash water all year.

LDC93S1.phn → 0 3050 h#3050 4559 sh4559 5723 ix5723 6642 hv6642 8772 eh8772 9190 **dcl**9190 10337 **jh**10337 11517 **ih**11517 12500 **dcl**12500 12640 **d**12640 14714 **ah**14714 15870 **kcl**15870 16334 **k**16334 18088 **s**18088 20417 **ux**20417 21199 **q**21199 22560 **en**22560 22920 **gcl**22920 23271 **g**23271 24229 **r**24229 25566 **ix**25566 27156 **s**27156 28064 **ix**28064 29660 **w**29660 31719 **ao**31719 33360 **sh**33360 33754 **epi**33754 34715 **w**34715 36080 **ao**36080 36326 **dx**36326 37556 **axr**37556 39561 **ao**39561 40313 **l**40313 42059 **y**42059 43479 **ih**43479 44586 **axr**44586 46720 h#

LDC93S1.wrd → 3050 5723 **she**5723 10337 **had**9190 11517 **your**11517 16334 **dark**16334 21199 **suit**21199 22560 **in**22560 28064 **greasy**28064 33360 **wash**33754 37556 **water**37556 40313 **all**40313 44586 **year**

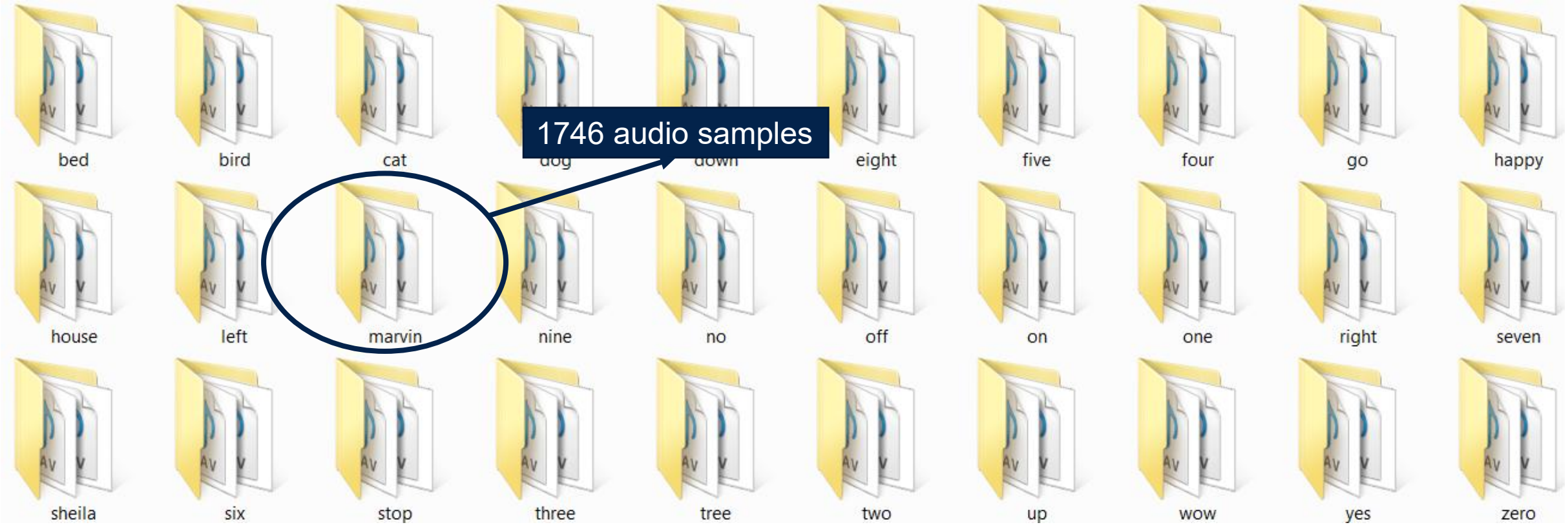
https://catalog.ldc.upenn.edu/LDC93S1W

Google Commands Dataset

- Google launched on August 2017 the Speech Commands Dataset. The dataset has 65,000 one-second-long utterances of 30 short words, by thousands of different people.

Google Commands Dataset

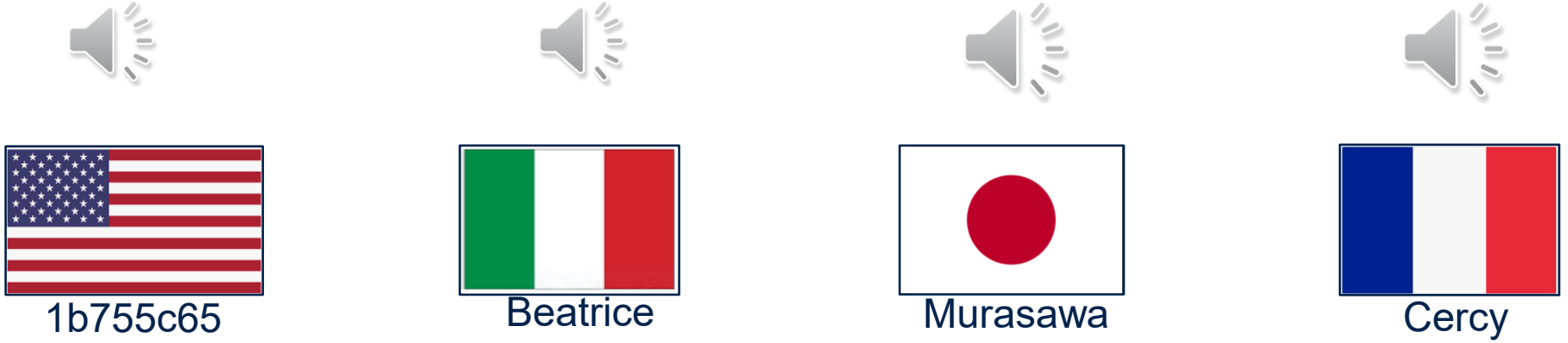
- Google launched on August 2017 the Speech Commands Dataset. The dataset has 65,000 one-second-long utterances of 30 short words, by thousands of different people.



The selected keyword for AST KWS system is Marvin

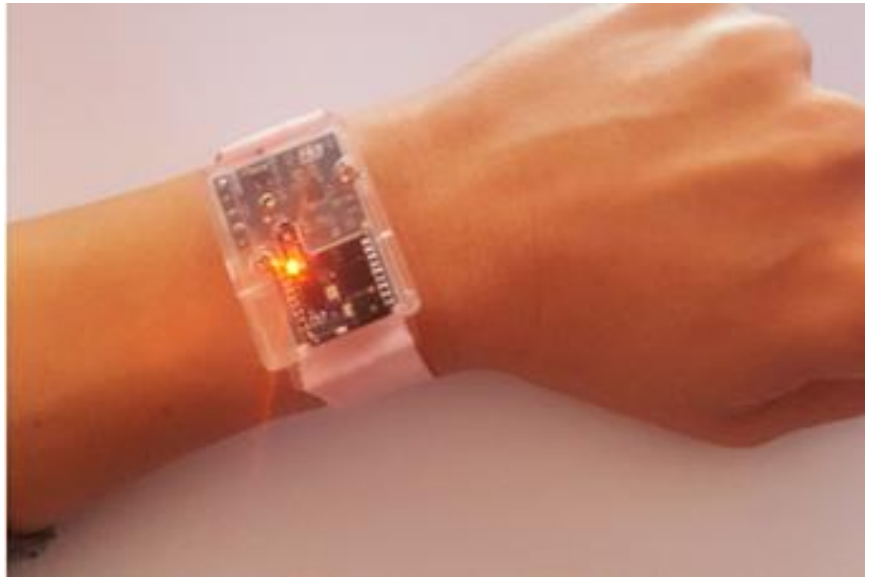
Heterogeneous Dataset

- **Speaker Language:**
  - Supposing to have trained a KWS algo by English words pronounced mainly by American people. What happen if the model is tested by French or Asiatic people?
  - Tones and pronunciations of other languages can affect the accuracy of the KWS algos, that's because specific language peculiarities are unknown to the model.



Dataset Collection

- A data collection for the keyword 'Marvin' and other vocal commands has been conducted using the ST ***SensorTile*** platform;
- Data are recorded with ***ST BLE Sensor*** application;
- The audio signal has been recorded at ***16 kHz***;
- Dataset is populated with a set of short commands in cooperation with other ST groups and Divisions.

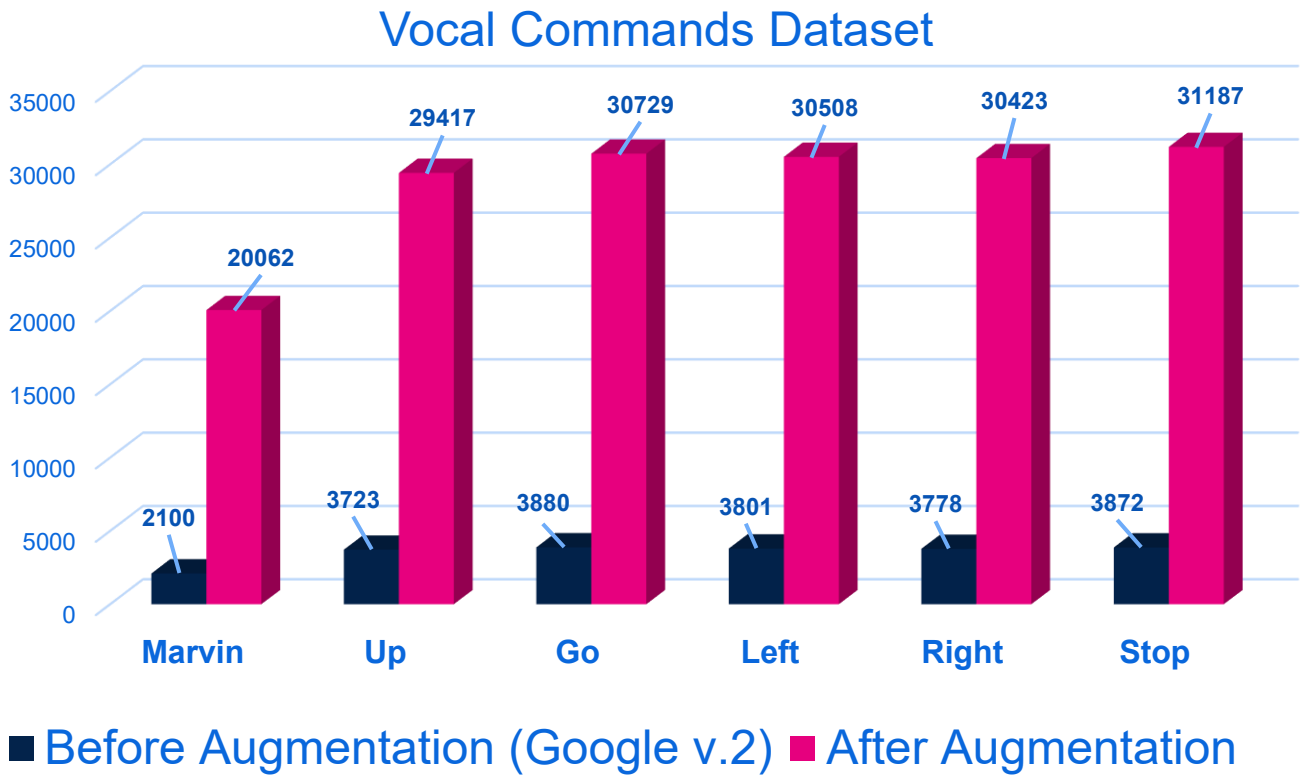


Vocal Commands Dataset

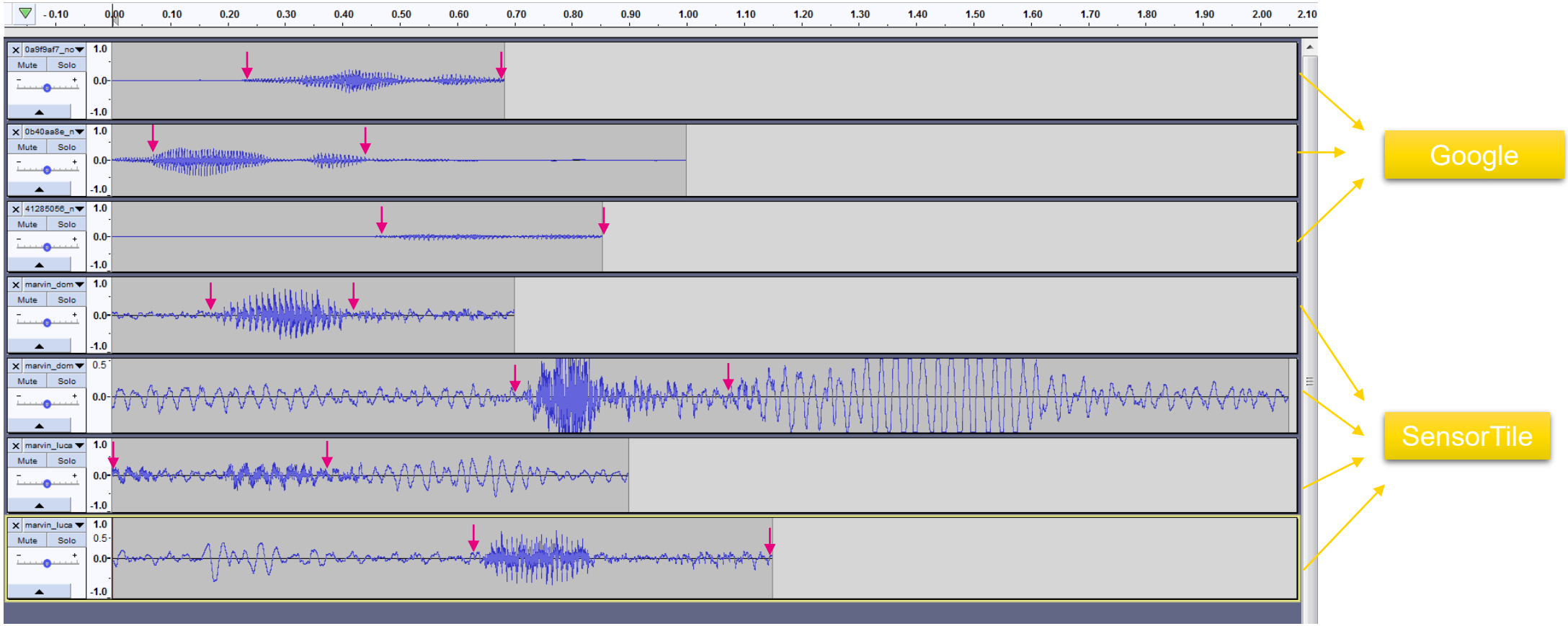
DATASET

1. **Started from Google dataset**
   - Around 3000 files for each command
2. **Data collection with new speakers**:
   - **French**: 11 speakers →1584 words
   - **Japanese**: 20 speakers → 2432 words
   - **Italian**: 31 speakers → 6405 words
3. **Data Augmentation**
   - Pitch variation
   - Time stretching
   - Noise mixing

21.154 words → 172.326 words

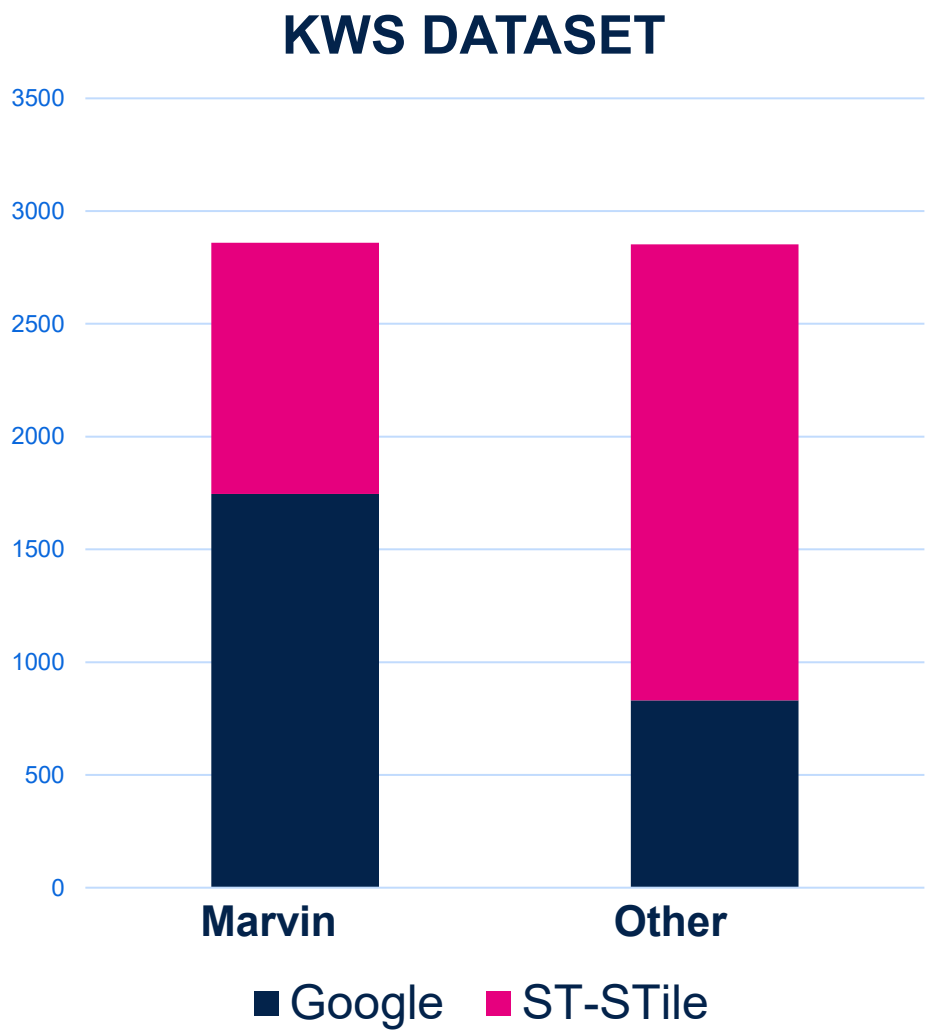


Isolated Words Dataset – weak labelling



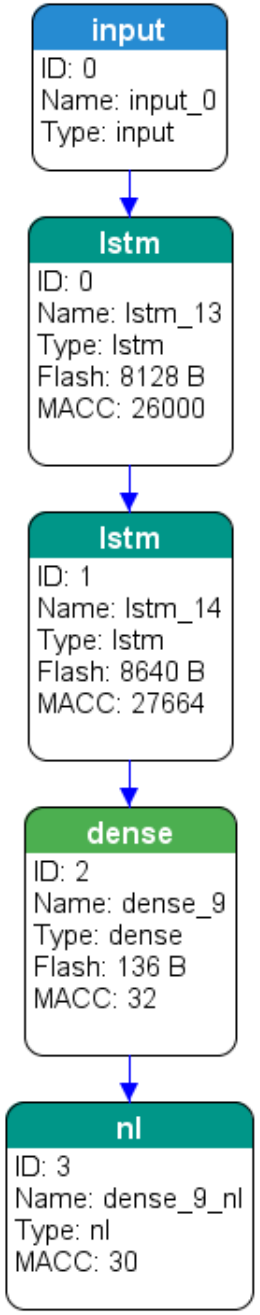
Keyword Spotting System on the edge (18-CT-0894)

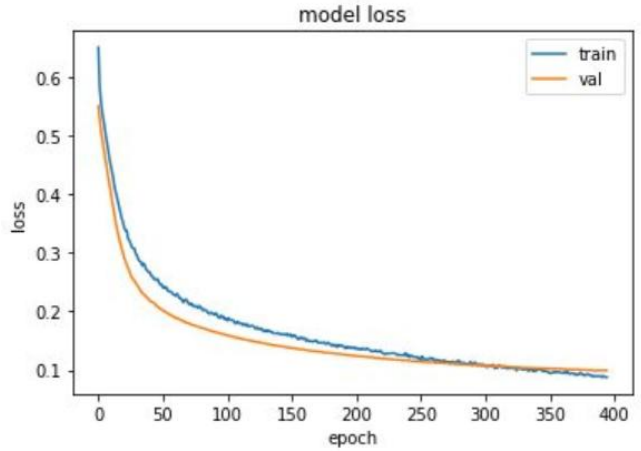
Keyword Spotting Dataset

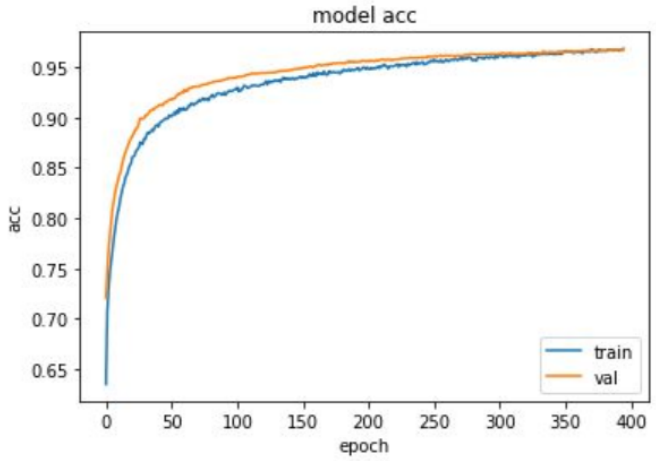


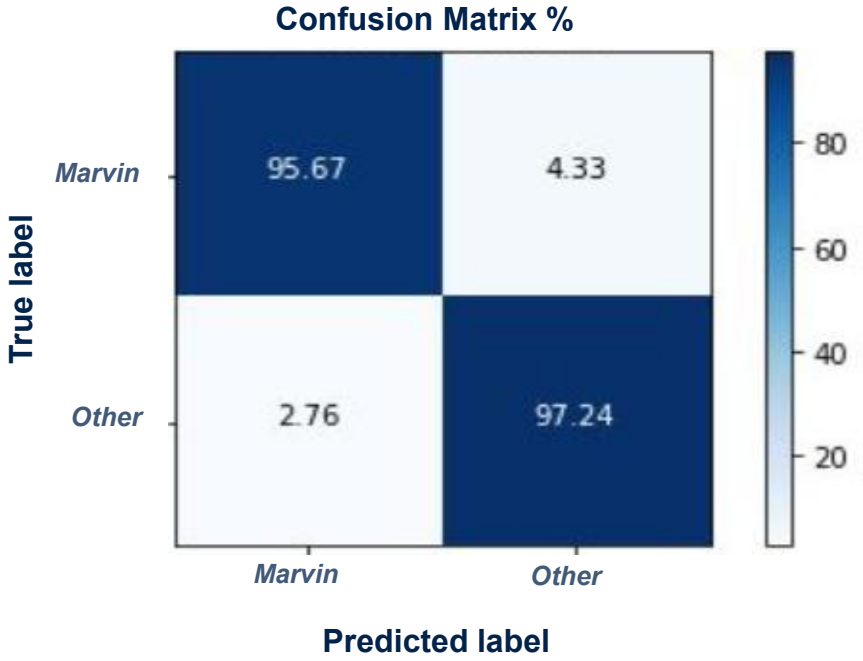
- The audio dataset contains 5712 audio files taken from:
  - Google public dataset
  - SensorTile microphone data
- The dataset is balanced with respect to the two classes: «Marvin» and «Other»
- The KWS dataset has been split in 70% - 30% for Training and Testing. The 20% of Training has been used as Validation set

KWS Neural Network & Performances



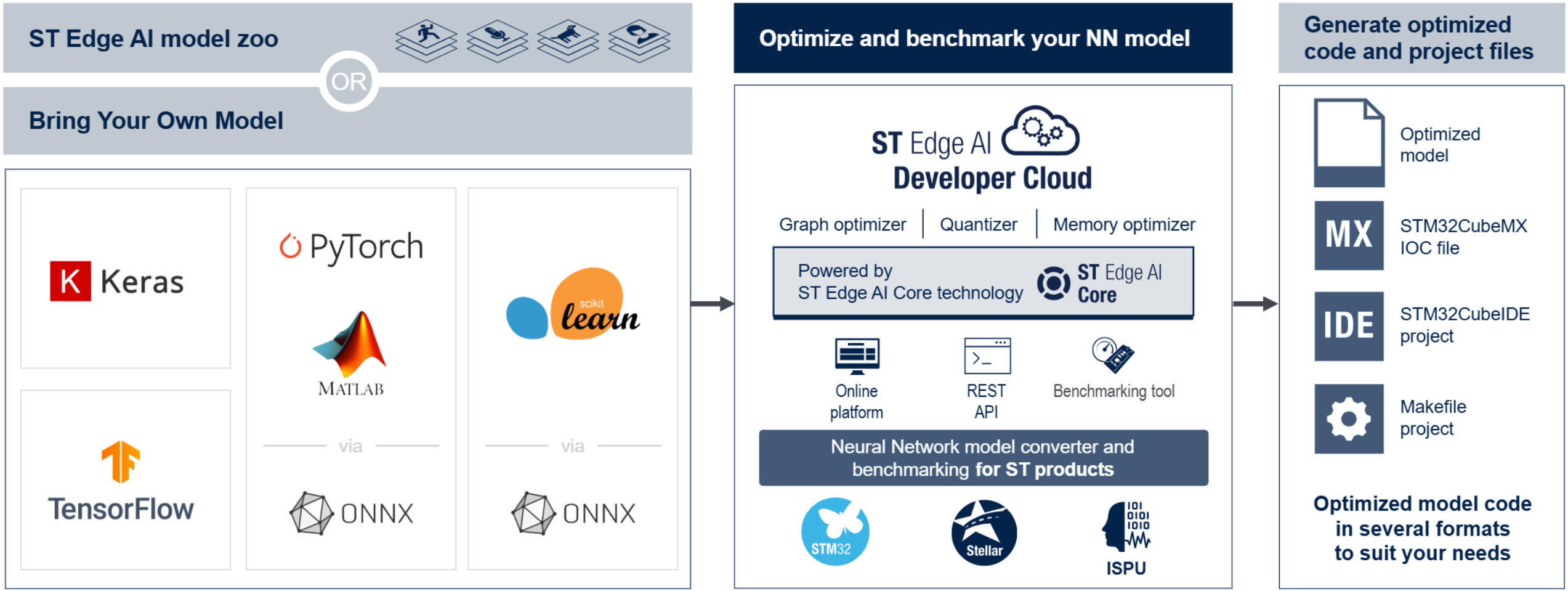






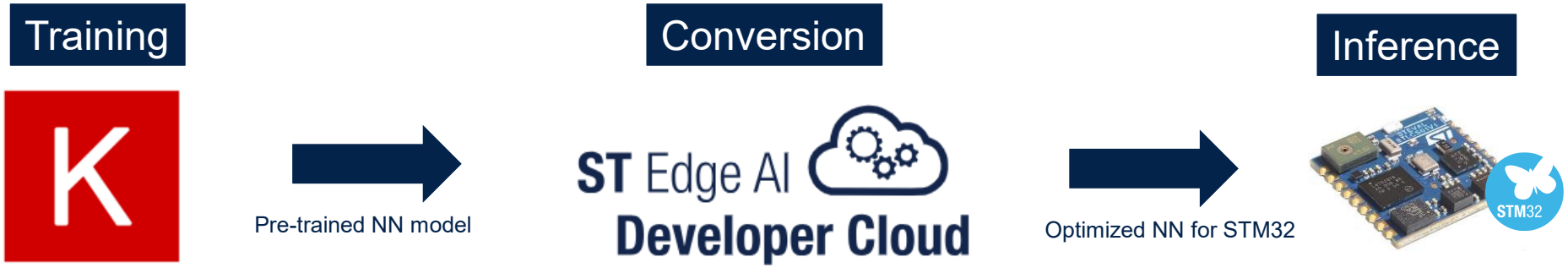
Accuracy = 96,5%

STM32Cube.AI Developer Cloud



[Home - ST Edge AI Developer Cloud]

Deploying of KWS-NN at the deep edge

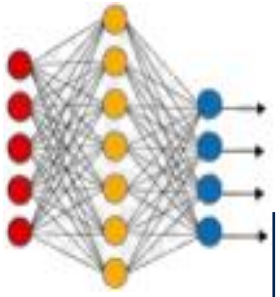


EDGE PLATFORM: ***STM32L4@80MHz, ROM 1MB*** and ***RAM 128KB.***

FLASH	RAM	MACC	Inference Time @ STM32L4
16.5 KB (available 1MB)	**2 KB** (Available 128 KB)	**52.5 K**	**15 ms**

- **FLASH**: Indicates the memory size needed to store weight and bias
- **RAM**: Indicates the size of the memory required to store the runtime calculations
- **MACC**: Reports the complexity of the model in multiply-accumulate operations

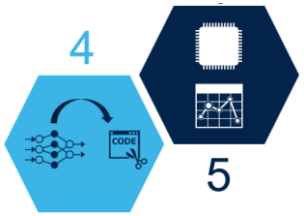
KWS: Neural Network Approach



NN Model

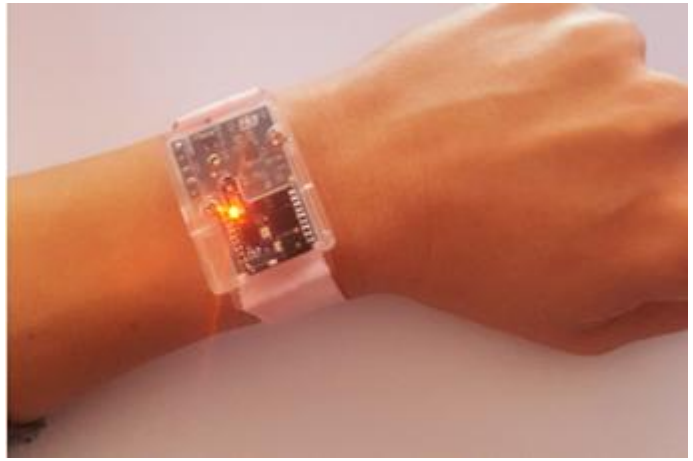
Audio streaming → Advanced Front End → Voice Activity Detector → Feature Extraction → Inference → Post Processing

- Wake up
- Spleep



Operating Mode

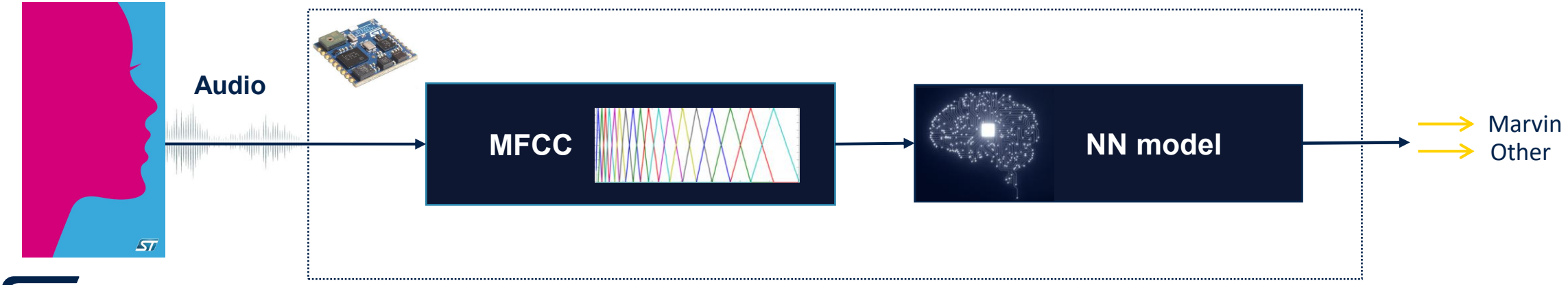
Deep Edge target platform: SensorTile



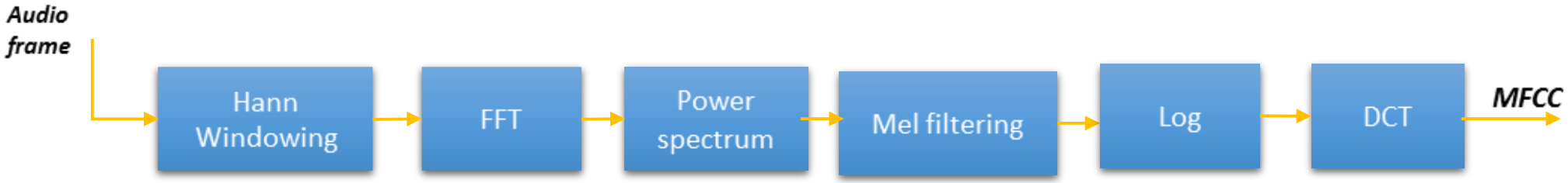
SensorTile as wearable device

- STM32L476 @80MHz ;
- 1MB FLASH;
- 128 KB RAM;
- One digital microphone:

No advanced pre-processing as source localization or beamforming

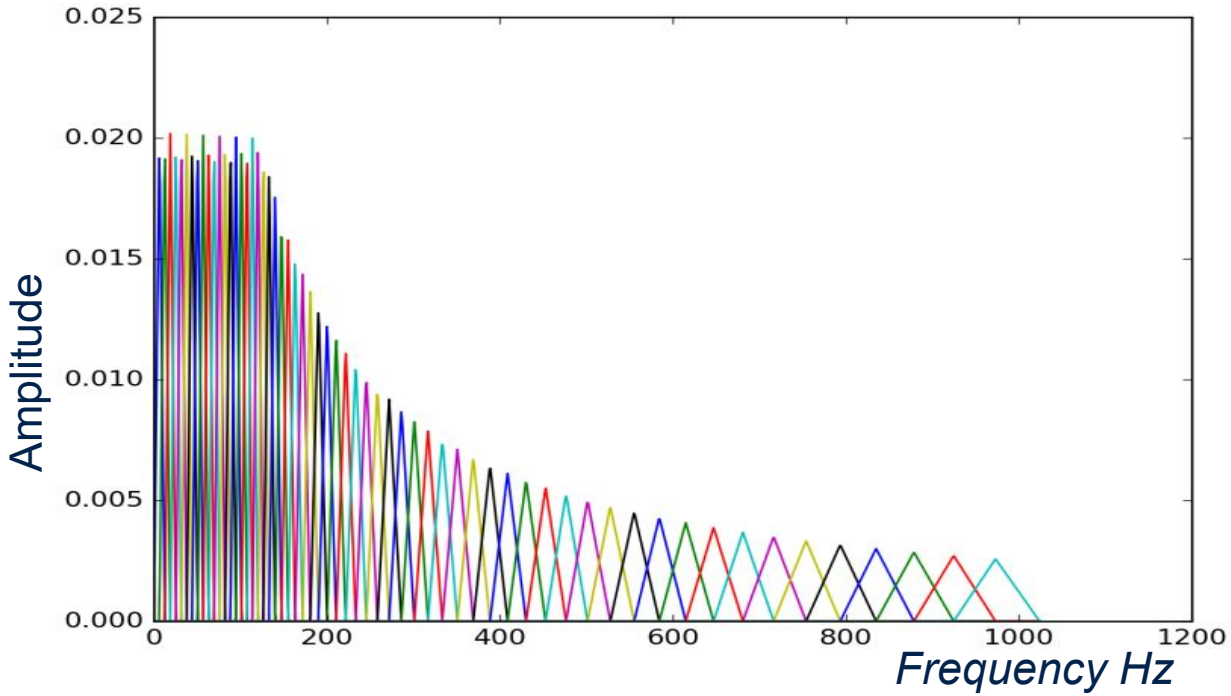


Mel Frequency Cepstral Coefficient



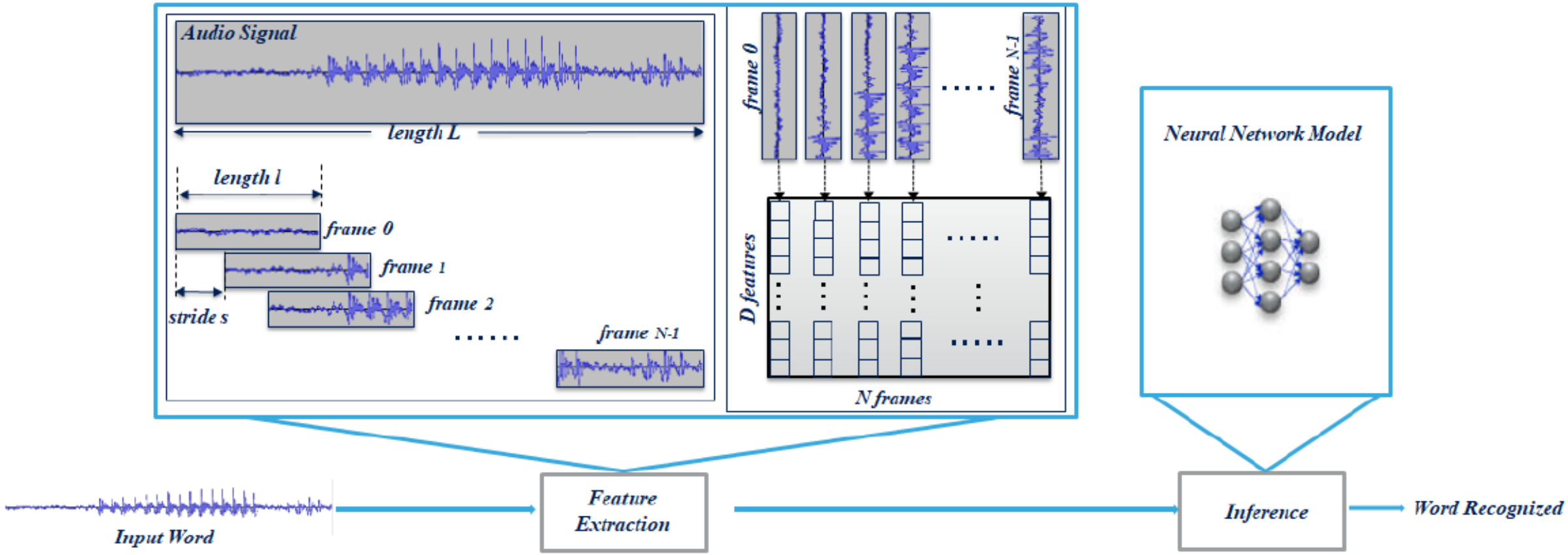
Mel Filterbank:

- Triangular filter that aims to mimic the non-linear human ear perception of sound; our perception of the change in pitch is more sensitive at lower frequencies than at higher frequencies. This means that we are better at detecting differences between lower frequency sounds than those at higher frequencies

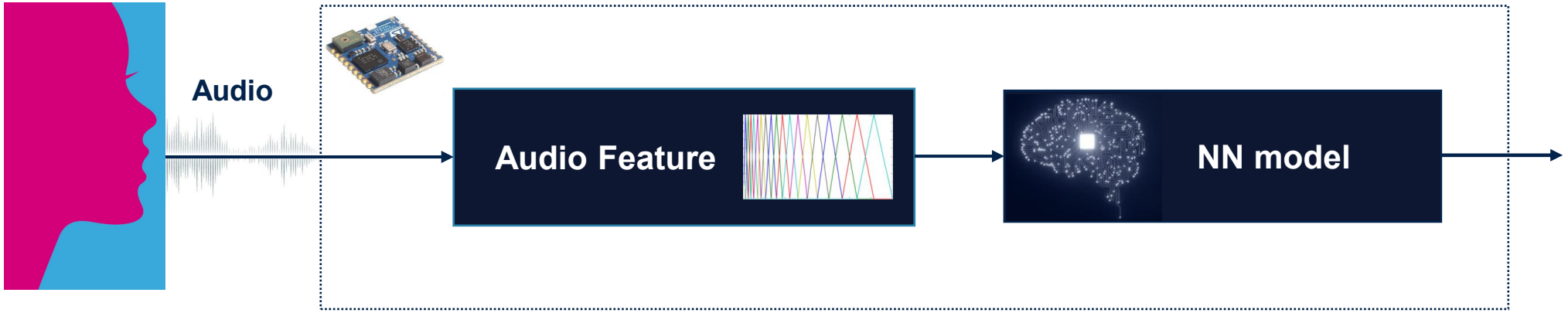


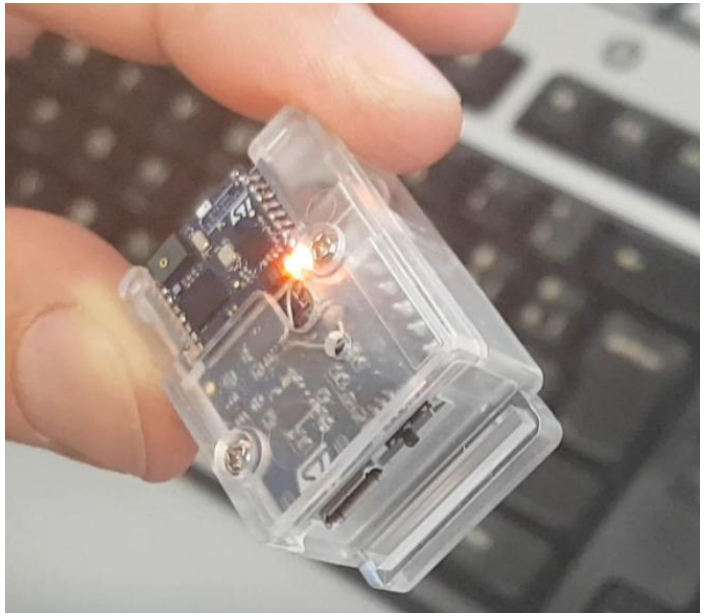
Audio Handling and MFCC

Studying the Effects of Feature Extraction Settings on the Accuracy and memory Requirements of Neural Networks for Keyword Spotting, Berlin 2018



KWS-System at the deep edge



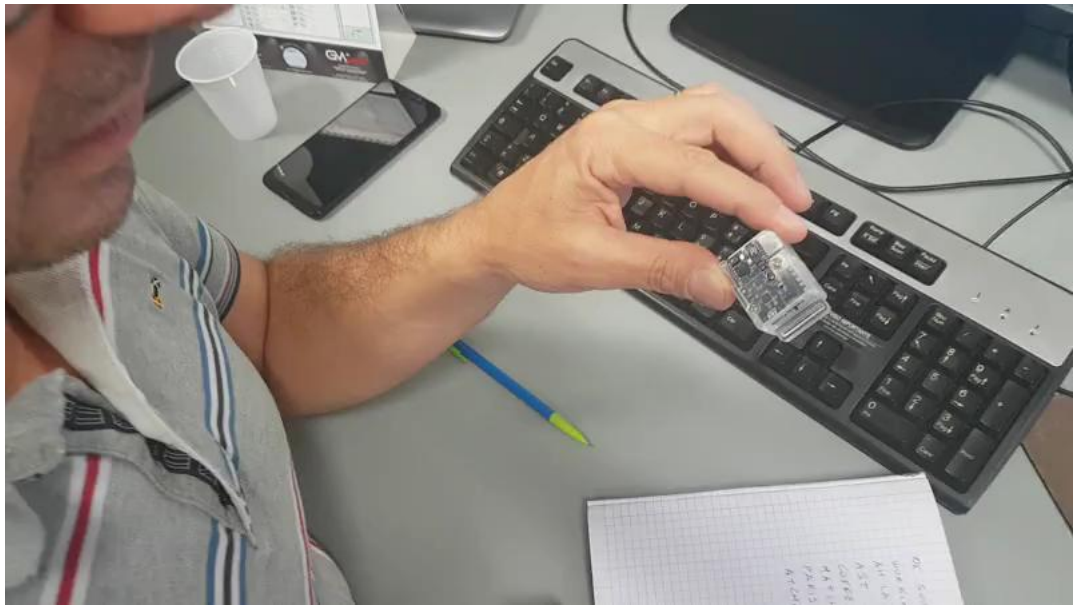


Demo Live

- People not involved in training process tested the KWS application
- The application discriminate in always-on mode the keyword Marvin against other words or continuous speaking or different background noises
- If the keyword Marvin is detected than a red led blinks twice

Testing of KWS@STM32 on real data

Keyword Recognition



KWS integrated with AVS

Trigger for Keyword Spotting (19-CT-0433)

Vocal Commands Recognition with demo @STM32

Vocal Commands Recognition to pilot Robot Movements

- Recognition of vocal commands at the deep edge
- To wakeup the system and to control an action through a vocal command
- To handle simple phrases made of few commands

Vocal Recognition @deep edge

Industrial

- Workers in noisy environments can use voice commands to control machinery or access information without stopping their work or removing safety equipment

Automotive

- Drivers can use voice commands to control navigation systems, media, or make calls, which keeps their focus on driving and enhances safety

Accessibility

- For individuals with disabilities, local voice recognition can enable better interaction with technology, providing more independence

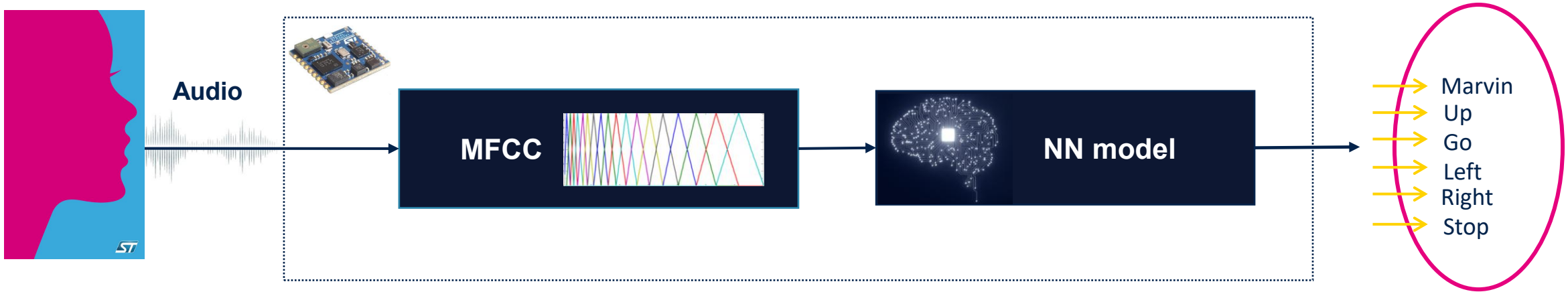
Smart Homes

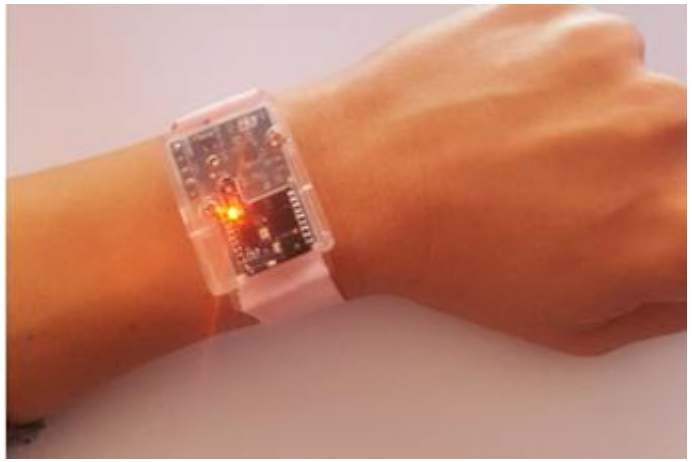
- Edge computing allows for faster and more reliable voice control of smart home devices, even if the internet connection is down

Healthcare

- In remote patient monitoring, voice-based commands can help patients interact with medical devices without physical contact, maintaining hygiene and allowing for quick adjustments in critical care

Vocal Commands Recognition on SensorTile



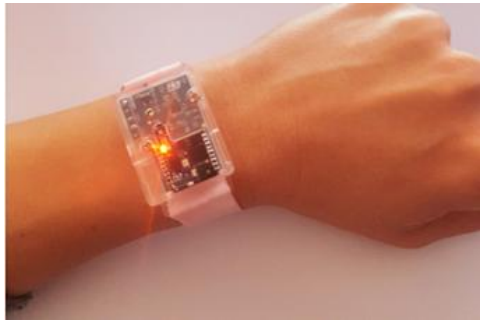


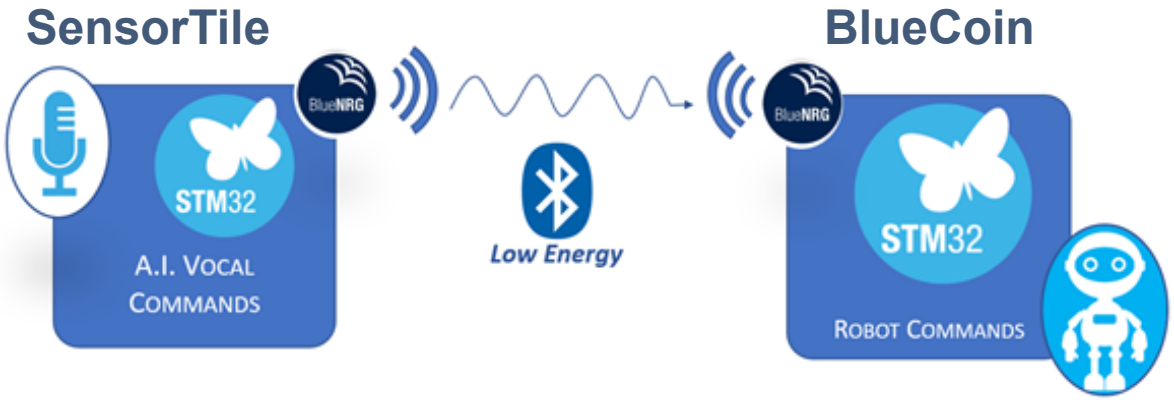
SensorTile as wearable device

- STM32L476 @80MHz ;
- 1MB FLASH;
- 128 KB RAM;
- One digital microphone:

No advanced pre-processing as source localization or beamforming

Vocal Command Recognition to pilot robot movements





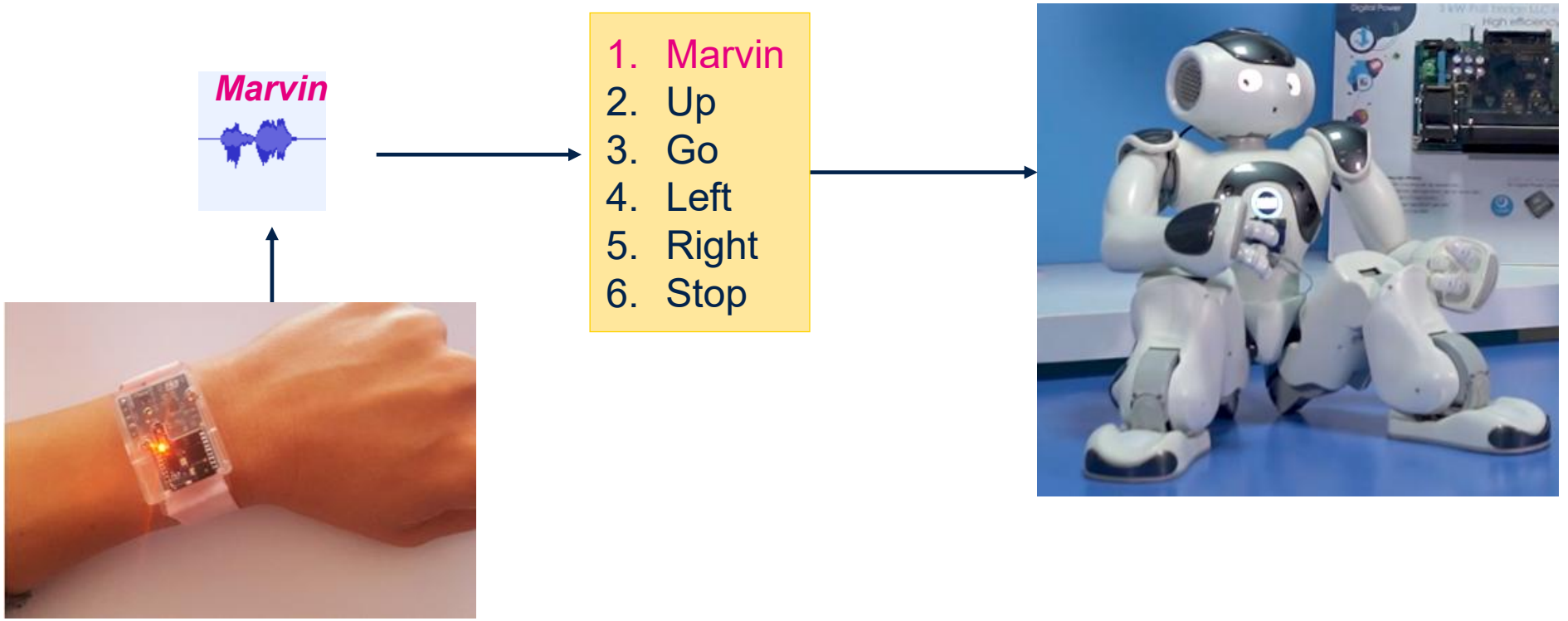
System setup

- Two boards: **SensorTile and BlueCoin**
- Both boards are equipped with the **BlueNRG** chip
- **The two bords communicates via BTLE** by setting of the APIs of the Bluetooth stack

Wake word followed by the command

To actuate a movement of the robot, the system needs to run two consecutive inferences to check the following sequence:

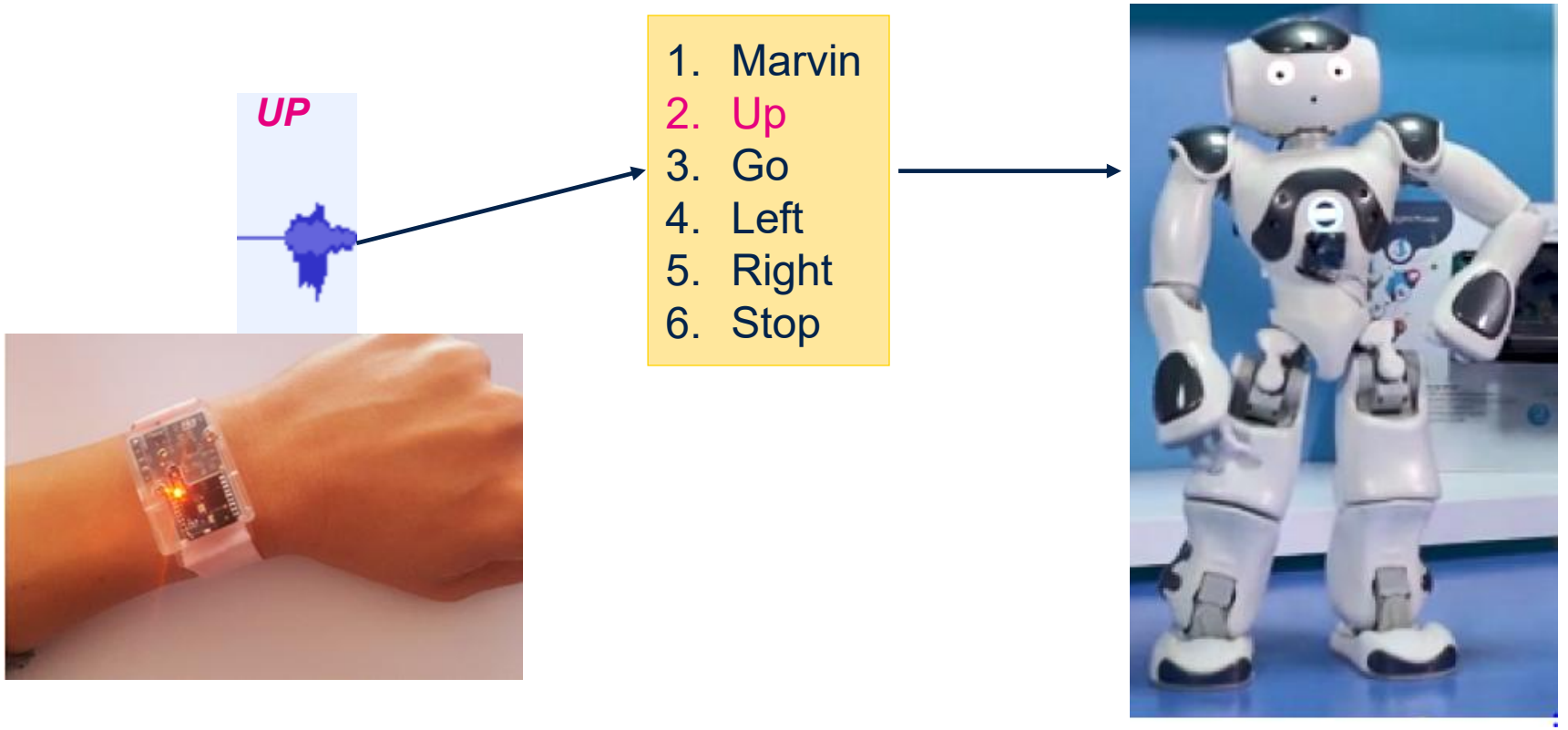
1. The first inference recognizes the command “Marvin” (used as a wake word)
2. The second inference recognizes the pronounced command to actuate robot movements

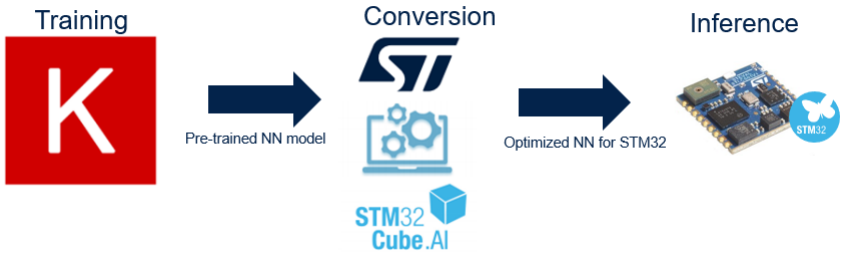


Wake word followed by the command

To actuate a movement of the robot, the system needs to run two consecutive inferences to check the following sequence:

1. The first inference recognizes the command “Marvin” (used as a wake word)
2. The second inference recognizes the pronounced command to actuate robot movements





Profiling on STM32L4

- The proposed RNN is implemented in floating point and, due to the low complexity and low memory requirements, no compression or quantization is necessary

Vocal Commands Recognition profiling: STM32L4 @80MHz, floating point

	FLASH KB	RAM KB	MACC M
Vocal Comm - RNN	295	0.52	0.97

- **FLASH**: Indicates the memory size needed to store weight and bias
- **RAM**: Indicates the size of the memory required to store the runtime calculations
- **MACC**: Reports the complexity of the model in multiply-accumulate operations

Vocal commands recognition@STM32

Vocal Commands to pilot a robot's movements

Deep Edge Vocal commands Recognition enabled by KWS, (20-CT-1051)

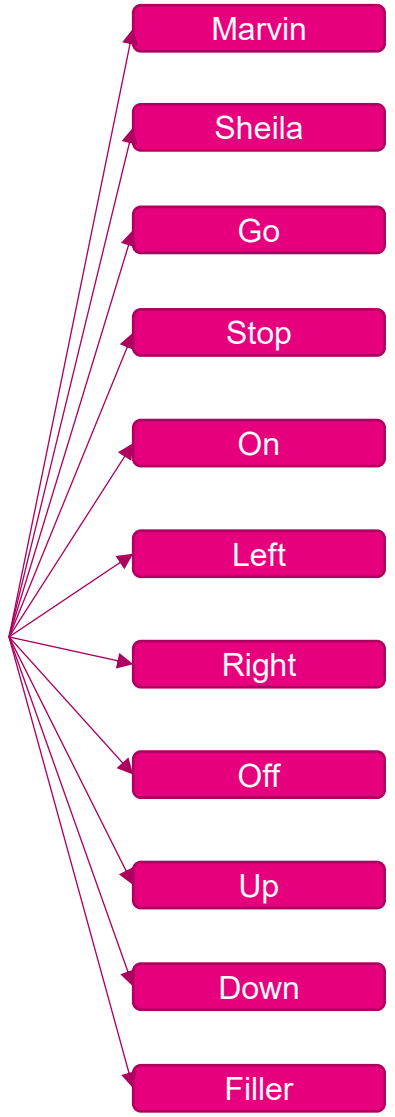
Deep Learning Short Commands Recognition for MCU in robotics applications, Prague 2021

UNIVERSITÀ degli STUDI di CATANIA

Alessandro Lauria

Sviluppo Software per il riconoscimento di Comandi Vocali in Sistemi Embedded

- Relatori:
  - Prof. Corrado Santoro
  - Prof. Giovanni Maria Farinella
- Tutor aziendale:
  - Ing. Ivana Guarneri



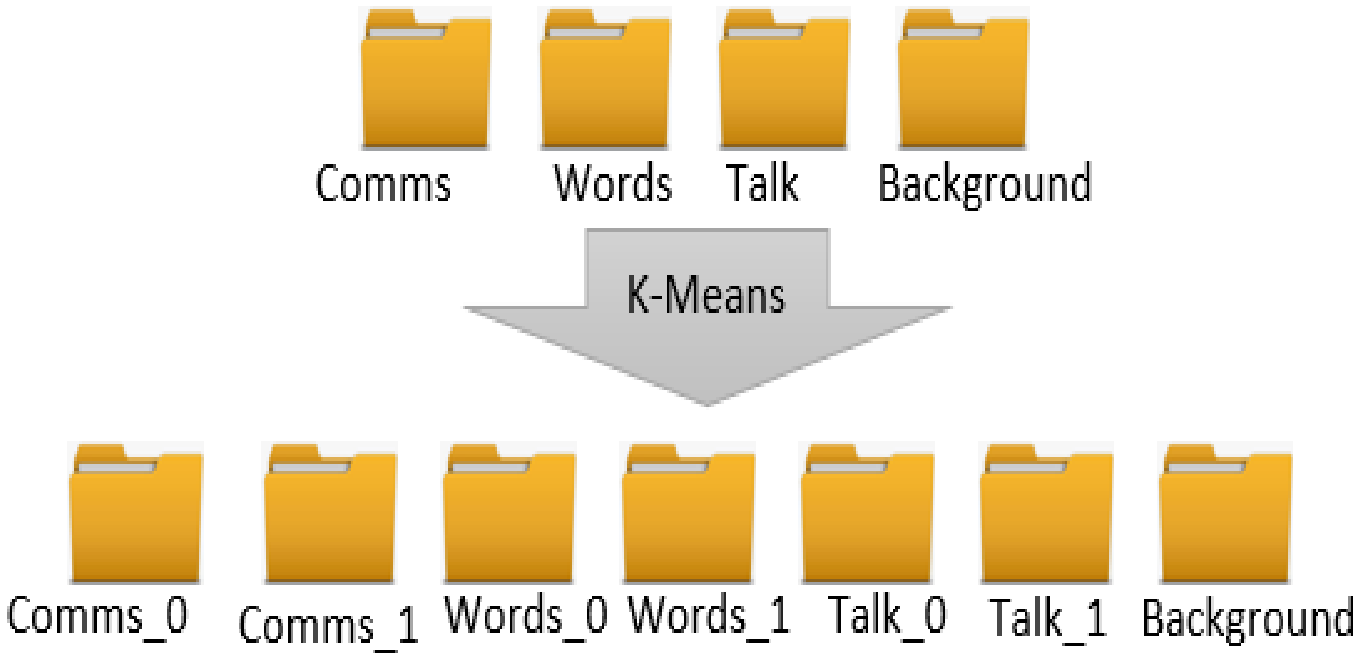
Dataset

The dataset used in this work has been populated by using data from two main repositories:

- **Google Speech Commands Dataset**
  - The Google dataset contains 30 different words for a total of 65.000 audio files of one-second each. Each word has about 2000 samples
- **Hey Snips Dataset**
  - Natural Language benchmark dataset, it contains a large variety of English accents and recording environments. It is composed of about 11K wake-word utterances and 86.5K (~96 hours) negative examples.

Classes	Description
Comms	10 vocal commands of Google dataset: go, stop, left, right, up, down, on, off, Marvin, Sheila
Words	Words different from the 10 selected commands of Google dataset
Talk	Continuous speaking sampled by Snips' dataset
Background	Silence and noise environments sound sampled from youtube

Dataset Processing

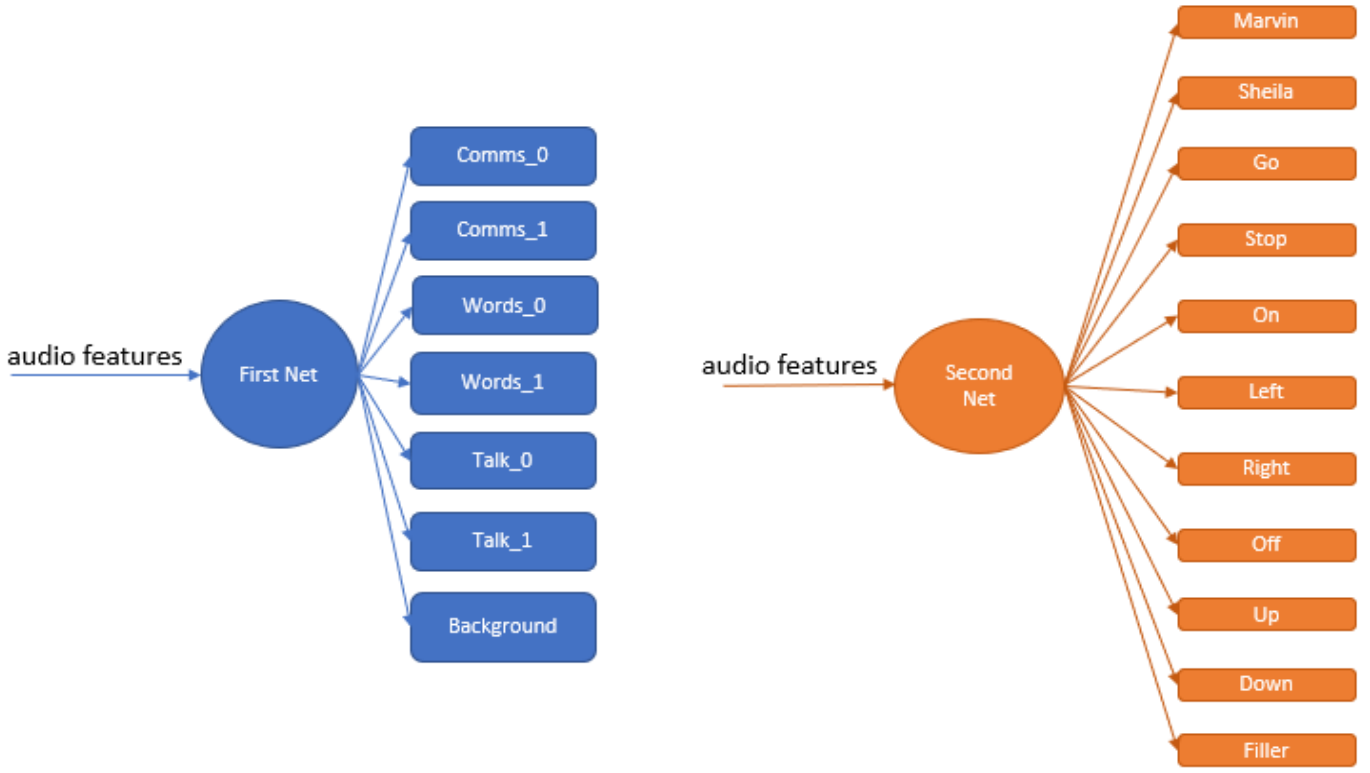


- The classes "***Comms***", "***Words***" and "***Talk***" have been partitioned through a K-Means processing.
- The "***Background***" class has not been processed with K-Means because it includes homogeneous files.

Tiny Networks Pipeline

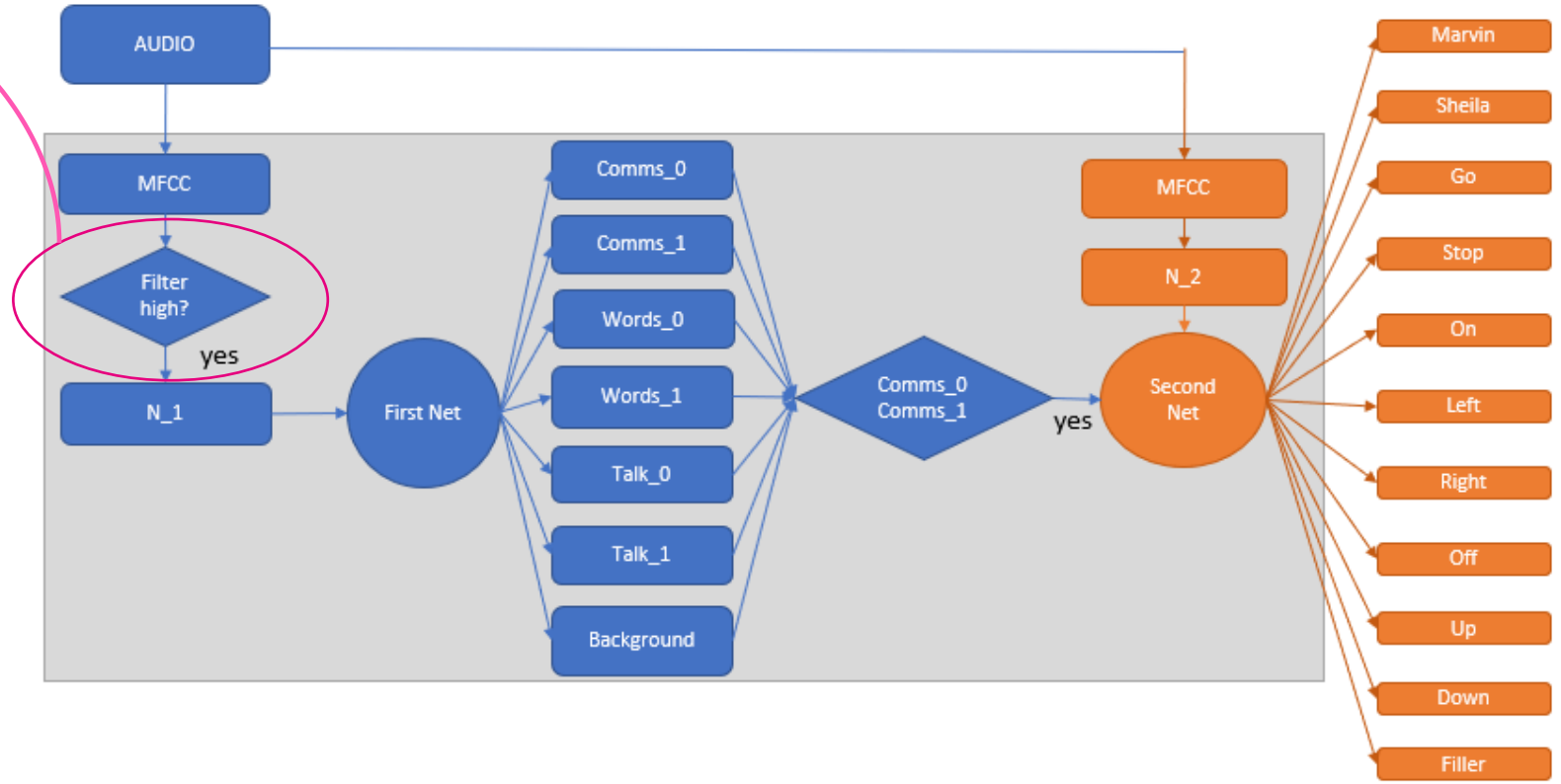
The proposed vocal command recognition system is configured as a ***cascade of two networks***

- The "***First Net***" is trained and tested with 7 classes
- The "***Second Net***" is trained and tested with 11 classes



Vocal command recognition system diagram

- A **filter** is applied to evaluate if there is enough signal energy
- The audio frames are processed by the **First Net** when the output of the filter is high
- The **Second Net** is hence triggered only if the output of the First Net is classified as Comms_0 or Comms_1



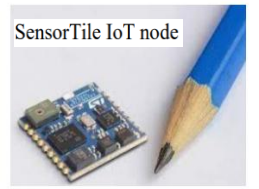
System Performances

- The accuracy of the **Cascade Net** has been compared with a baseline composed by a single network, named **Single Net**.
- The single network has a comparable size to the cascade one.
- The two different systems, requiring the same amount of memory, can be properly compared in terms of accuracy and power-saving capability

APPROACH	MAC	FLASH
Cascade approach	515649	80,55 KB
Single Net	525795	79,68 KB

	Cascade Net	Single Net
Marvin	85 %	64%
Sheila	93 %	81 %
Left	83 %	61%
Right	86 %	65 %
Up	71 %	50 %
Down	75 %	59 %
Go	80 %	48 %
On	78 %	63 %
Off	73 %	74 %
Stop	81%	67 %
Filler	78 %	76%
Avg	80,3 %	70,1 %

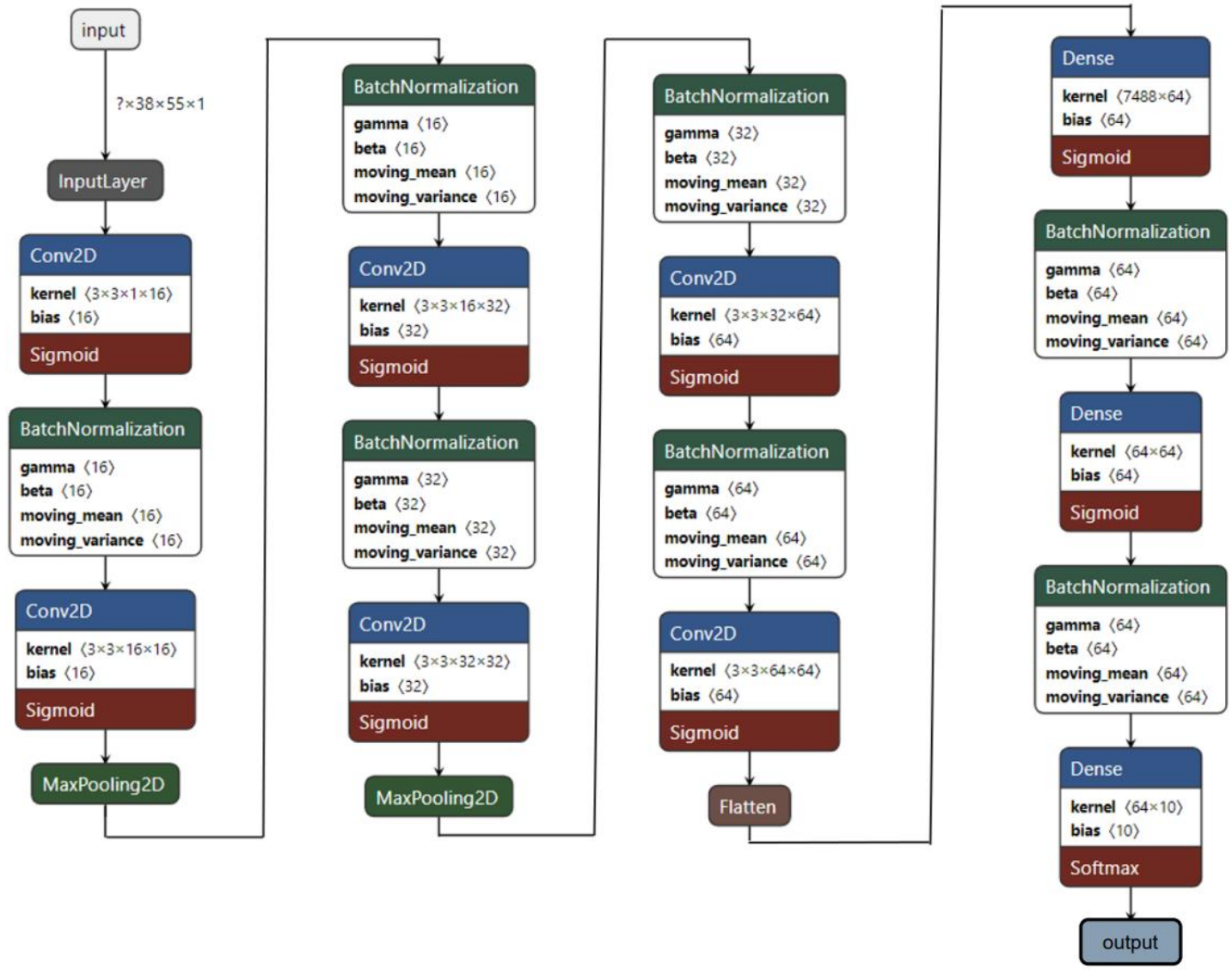
Memory requirements of the cascade system	0,08 MB
SensorTile FLASH	1 MB



Tiny Vocal Commands Recognition on MCU

Quantization for Vocal Commands Recognition @MCU

- The proposed deep NN architecture recognizes 10 vocal commands (Google dataset).
- The network consists of six convolutional layers followed by three fully connected layers



FP32 NN model for Vocal Commands Recognition

FP32 NN

- ACCURACY = 94% on the test set
- FLASH = 2.12 MB
- RAM = 149.20KB

TARGET PLATFORM

- STM32L4 MCU @80MHz
- FLASH = 1MB
- RAM = 320KB

As shown on the left report, the dense layer (id = 14) impacts the most on FLASH requirements

```
Complexity report per layer - macc=20,496,832 weights=2,225,384 act=144,384 ram_io=8,40
--------------------------------------------------------------------------------
id   name                  c_macc                    c_rom                     c_id
--------------------------------------------------------------------------------
0    conv2d                ||                  3.1%  |                   0.0%  [0]
1    batch_normalization   |                   0.3%  |                   0.0%  [1]
2    conv2d_1              ||||||||||||||||    25.3%  |                   0.4%  [2]
4    batch_normalization_1 |                   0.1%  |                   0.0%  [3]
5    conv2d_2              ||||||||            12.3%  |                   0.8%  [4]
6    batch_normalization_2 |                   0.2%  |                   0.0%  [5]
7    conv2d_3              ||||||||||||||      23.9%  |                   1.7%  [6]
9    batch_normalization_3 |                   0.0%  |                   0.0%  [7]
10   conv2d_4              |||||||             10.9%  |                   3.3%  [8]
11   batch_normalization_4 |                   0.1%  |                   0.0%  [9]
12   conv2d_5              ||||||||||||        21.4%  ||                  6.6%  [10]
14   dense                 ||                  2.3%  ||||||||||||||||   86.2%  [11]
14   dense_nl              |                   0.0%  |                   0.0%  [12]
16   dense_1               |                   0.0%  |                   0.7%  [13]
16   dense_1_nl            |                   0.0%  |                   0.0%  [14]
17   batch_normalization_6 |                   0.0%  |                   0.0%  [15]
18   dense_2               |                   0.0%  |                   0.1%  [16]
18   dense_2_nl            |                   0.0%  |                   0.0%  [17]
Creating txt report file C:\Users\guarneri\.stm32cubemx\network_analyze_report.txt
elapsed time (analyze): 2.234s
Analyze complete on AI model
Required Ram or Flash size for network is bigger than available Ram or Flash
```

QKeras: Efficient Neural Network Quantization

- **QKeras** is a Keras-based framework for training neural networks with **quantized weights and activations**.
- It enables reduced precision such as **8-bit**, **4-bit**, **2-bit**, or even **1-bit** representations.
- Benefits:
  - **smaller memory footprint**
  - **lower power consumption**
  - **faster inference**
  - **better suitability for embedded and edge devices**
- Main trade-off: **lower bit-width improves efficiency but may reduce accuracy**.

QKeras helps find the best balance between model accuracy and hardware efficiency

Hybrid Quantization for Vocal Commands Recognition @MCU

QKeras 8-bit aware training quantization

- ACCURACY = 92%
- FLASH = 0.48MB
- RAM = 127.4 K

QKeras hybrid aware training quantization

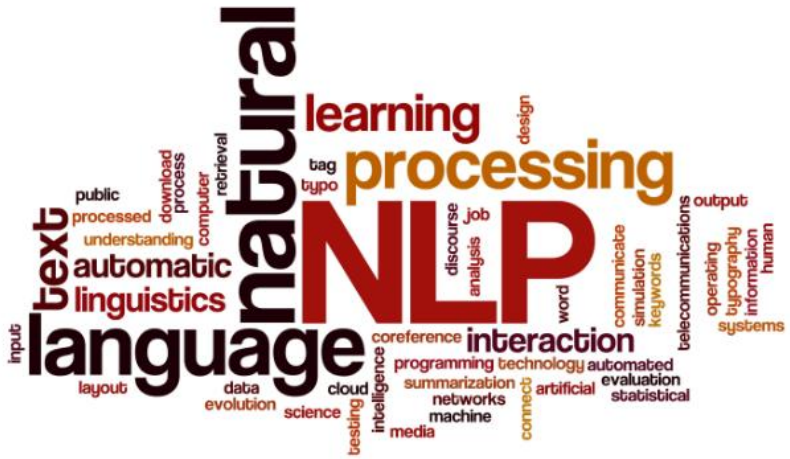
- 8-bit Conv2D layers
- Dense (1-bit) binary layers

TABLE2	ACCURACY %	FLASH MB	RAM KB	MACC M
KWS proposal FP32 (Keras)	94	2.12	149.20	20.5
KWS proposal 8-bit (QKeras)	92	0.48	127.4	10.5
KWS proposal hybrid1-8 bit(QKeras)	90	0.076	127.4	10.5

The binarization allows a further reduction of memory by achieving 90% of accuracy

Transformers for NLP

NLP: from RNN to Transformer

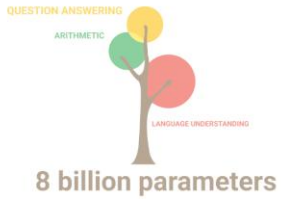


Recurrent Neural Network

- RNNs sequentially process the words of a sentence
- RNNs couldn't parallelize well
- RNNs never really did well at handling long paragraphs

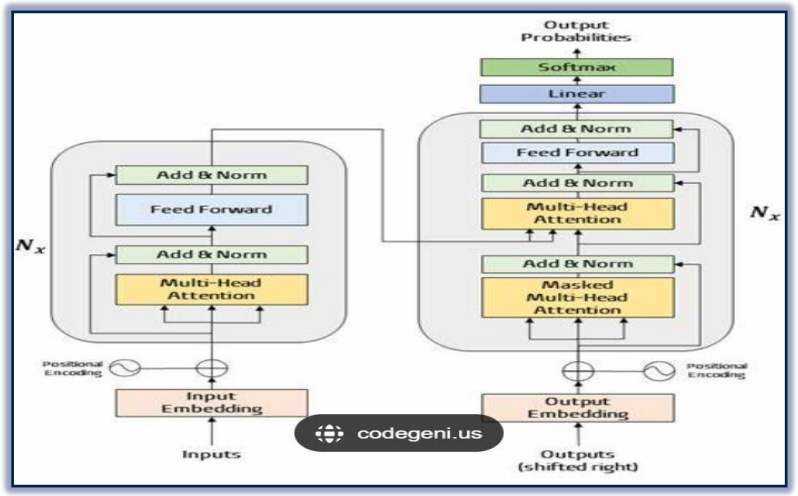
Transformer surpassed RNNs in many NLP tasks

- **Parallelization**: Transformers process entire sequences at once
- **Long-range dependencies**: self-attention determines the importance of words in a sequence capturing long-range dependencies.
- **Scalability**: The Transformer's ability to scale with more data and compute power has led to the creation of very large models like GPT-3 and BERT
- **Flexibility**: The Transformer's architecture is inherently more flexible, which has allowed researchers to adapt and improve upon it in various ways, leading to a rapid evolution of NLP models.



PaLM

Transformer analysis



To study the Transformer architecture by exploring its pros and cons.

To look for use cases by utilizing Transformer models in the fields of NLP and CV.

To understand what's missing in ST Edge AI Core to support them

Transformer architecture

2017

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

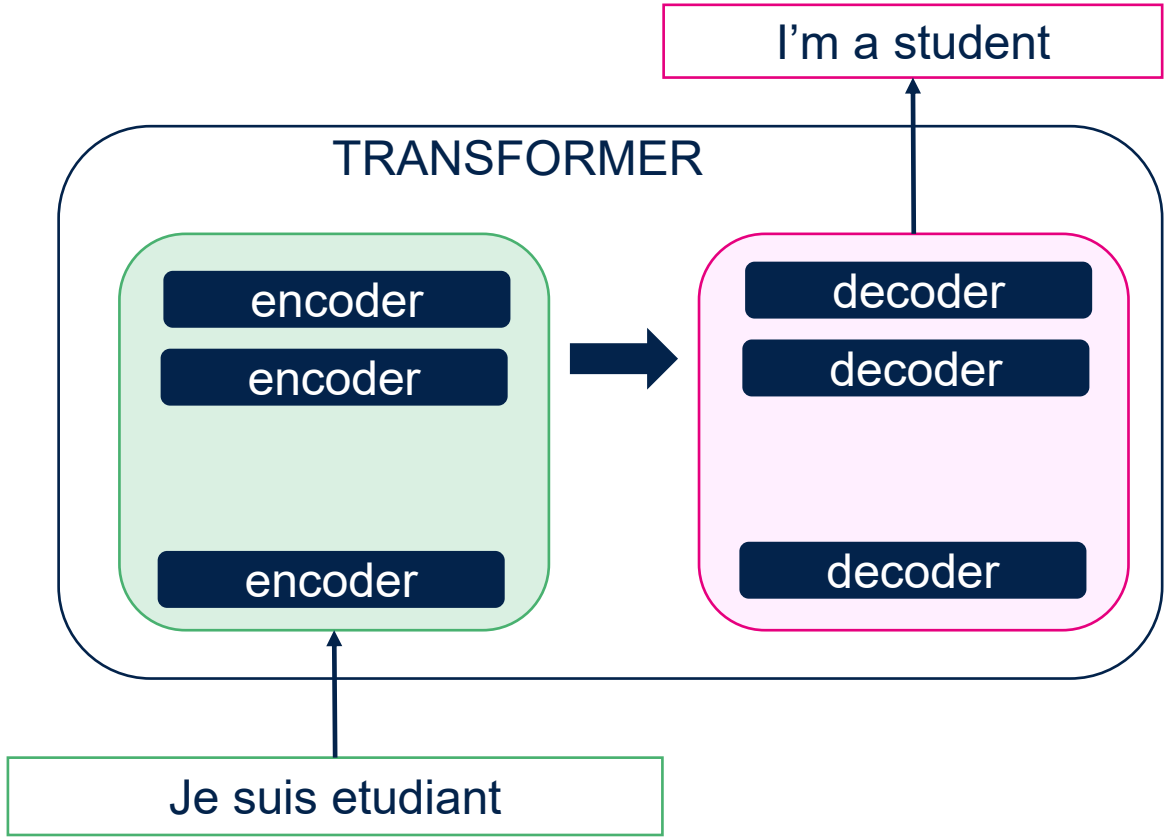
Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

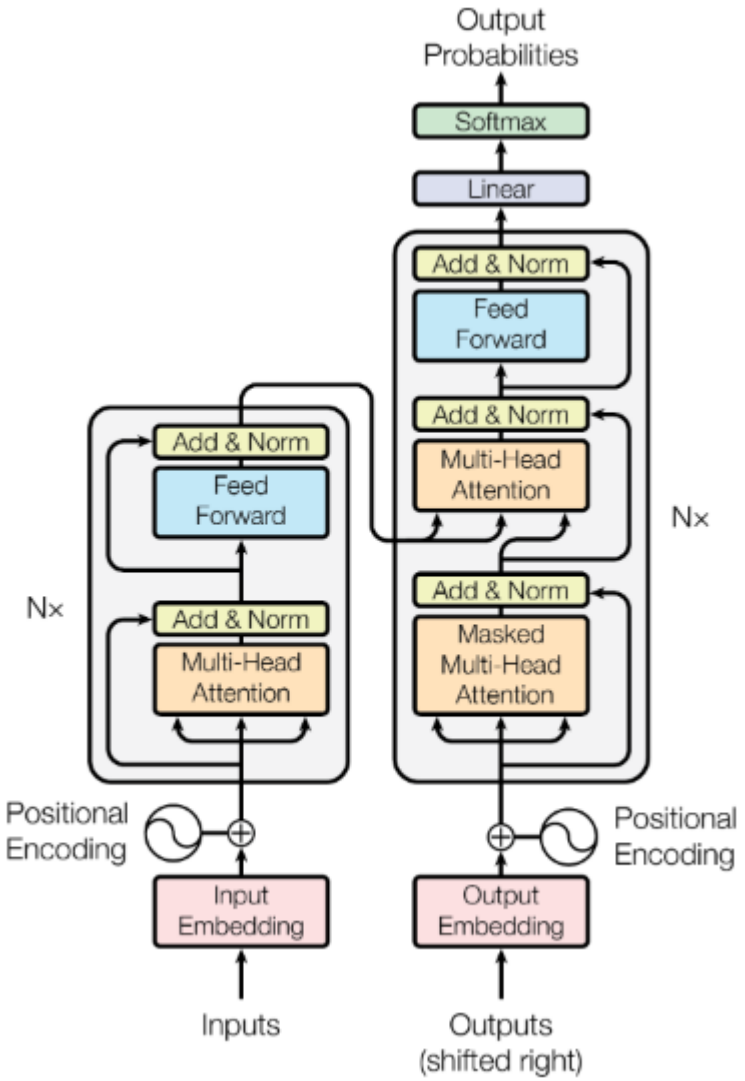
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com



Transformer architecture innovations



Non sequential

- Sentences are processed as a whole, rather than word by word

Positional Encoding

- To inject positional information on the input embeddings

Self-Attention

- To capture relationships between the different elements of the same sentence

The Multi head attention

"The *cat* drank the milk because **it** was hungry"

What does it refer to?

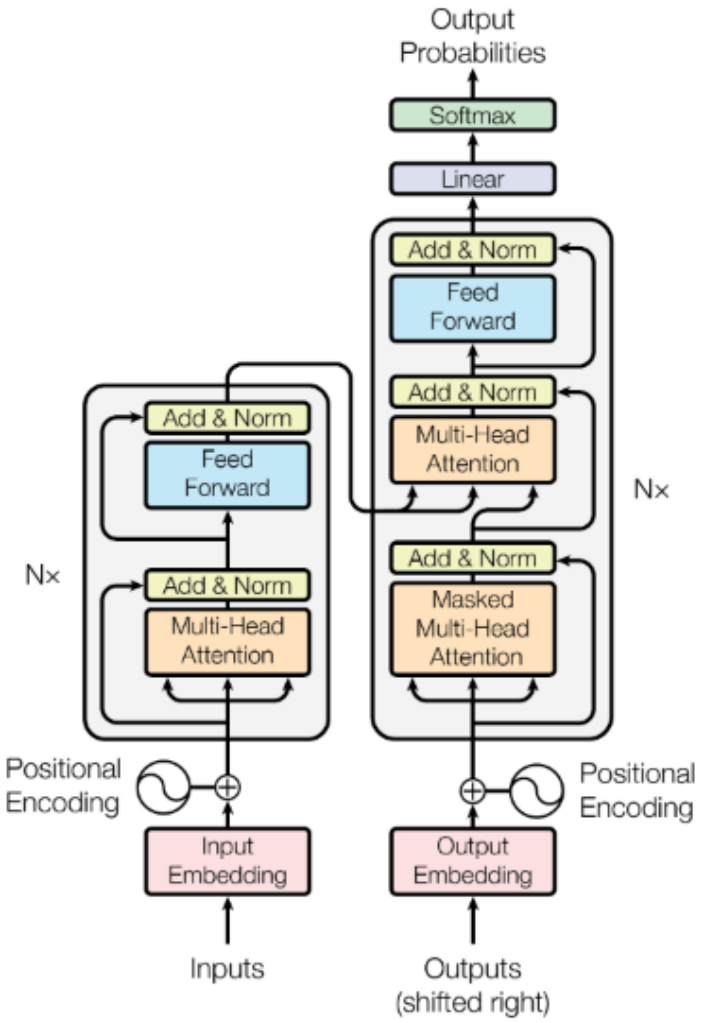
The Transformer uses **Self-Attention** by relating every word in the input sequence to every other word

Self-Attention allows to encode long-range dependencies

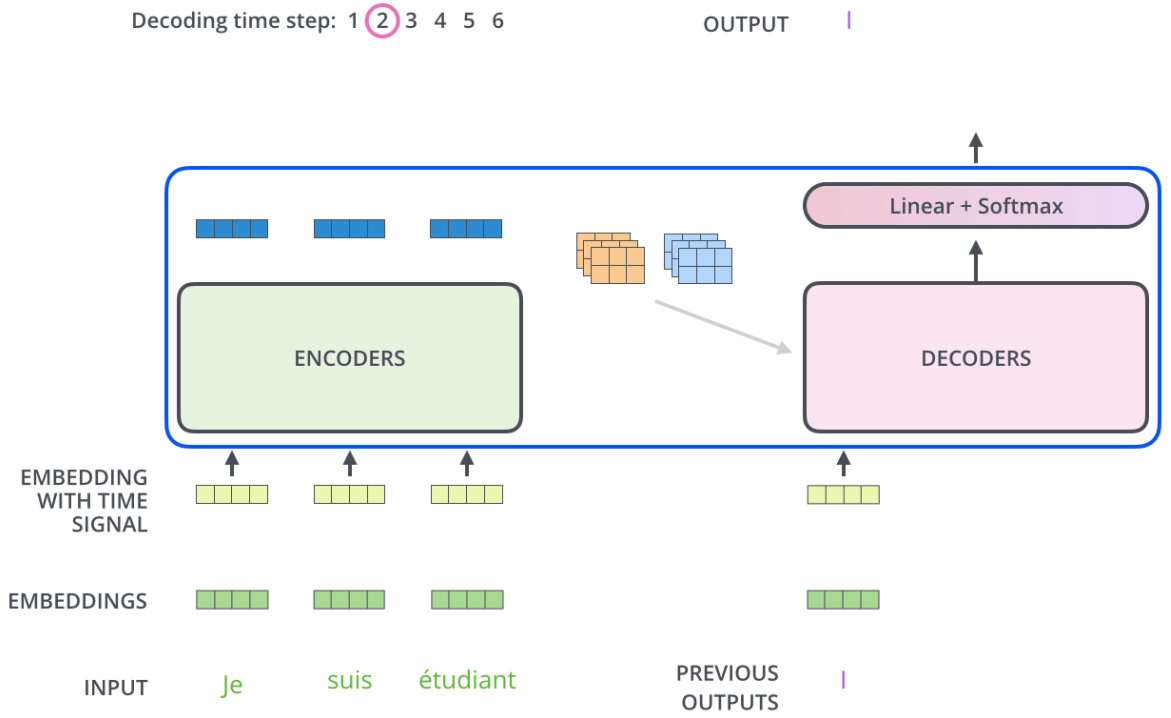
The **Multi-Head Attention** method allows to calculate and capture the relevant information from different heads

Multi-head attention captures the meaning, the nuance in a language

Transformer for language translation



Je suis etudiant → I'm a student

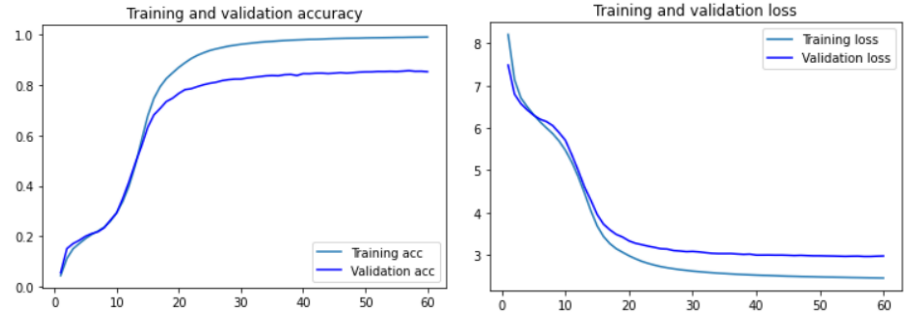


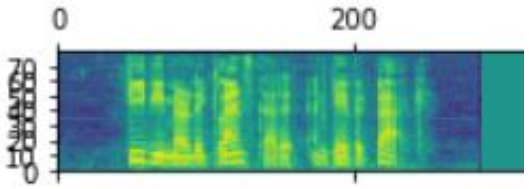
Speech To Text example

Speech to text encoder-decoder Transformer

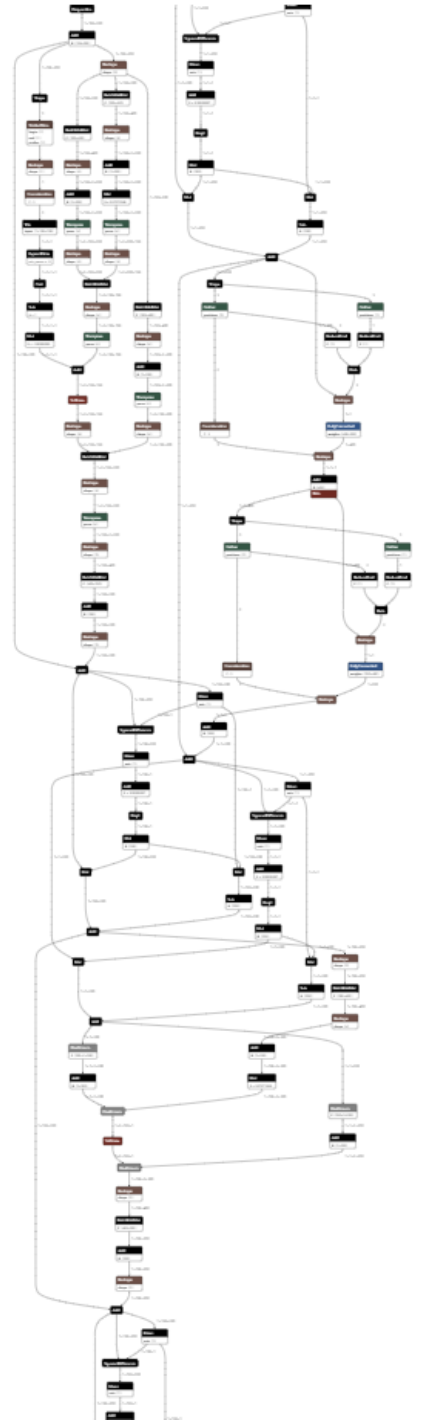
- Trained on a LibriSpeech sub-dataset
- Num Enc = Num Dec = 3
- Num Heads= 8
- Encoding type = subword
- Epochs = 60
- Acc = 85.2%
- Trainable = 30.5 M

```
Model: "transformer"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
inputs (InputLayer)             [(None, None, 80)]   0           []

tf.math.reduce_sum (TFOpLambda  (None, None)         0           ['inputs[0][0]']
)

tf.cast (TFOpLambda)            (None, None)         0           ['tf.math.reduce_sum[0][0]']

dec_inputs (InputLayer)         [(None, None)]       0           []

dense (Dense)                   (None, None, 512)    41472       ['inputs[0][0]']

enc_padding_mask (Lambda)       (None, 1, 1, None)   0           ['tf.cast[0][0]']

encoder (Functional)            (None, None, 512)    9458176     ['dense[0][0]',
                                                                 'enc_padding_mask[0][0]']

look_ahead_mask (Lambda)        (None, 1, None, Non  0           ['dec_inputs[0][0]']
                                e)

dec_padding_mask (Lambda)       (None, 1, 1, None)   0           ['tf.cast[0][0]']

decoder (Functional)            (None, None, 512)    16849920    ['dec_inputs[0][0]',
                                                                 'encoder[0][0]',
                                                                 'look_ahead_mask[0][0]',
                                                                 'dec_padding_mask[0][0]']

outputs (Dense)                 (None, None, 8275)   4245075     ['decoder[0][0]']

==================================================================================================
Total params: 30,594,643
Trainable params: 30,594,643
Non-trainable params: 0
__________________________________________________________________________________________________
```

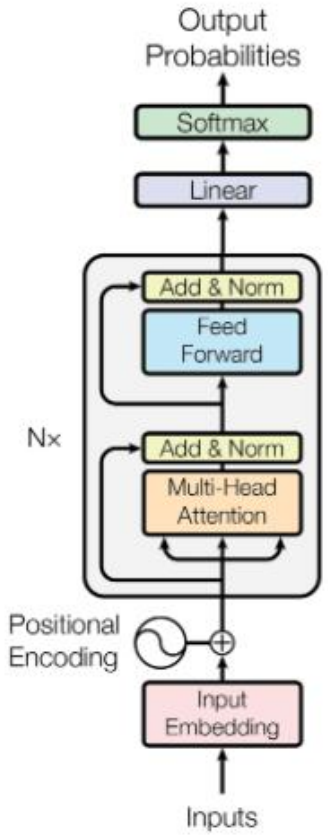




Speech: phoebe merely glanced at it and gave it back
Prediction: heavy nearly glanced at it and gave it back
WER = 0.22



Transformer families for NLP - Encoder Only



BERT

Description

Models that encode the entire input sequence into contextual representations. They are mainly used to “understand” text rather than generate it.

Common Tasks

Text classification, sentiment analysis, semantic search, QA extraction, embeddings

Strengths

Strong text understanding, efficient for classification, excellent contextual representations

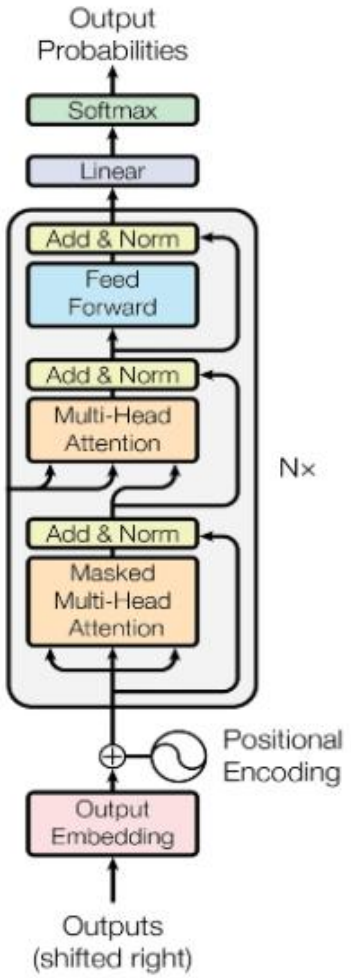
Limitations

Not naturally generative, limited for free-form text generation

Technical notes

Usually trained with “masked language modeling”. Attention is bidirectional

Transformer families for NLP - Dencoder Only



GPT

Description

Models that generate text **autoregressively**, one token at a time, using only the past context.

Common Tasks

Text generation, chatbots, completion, code generation, reasoning

Strengths

Very strong generation ability, simple training objective, scalable

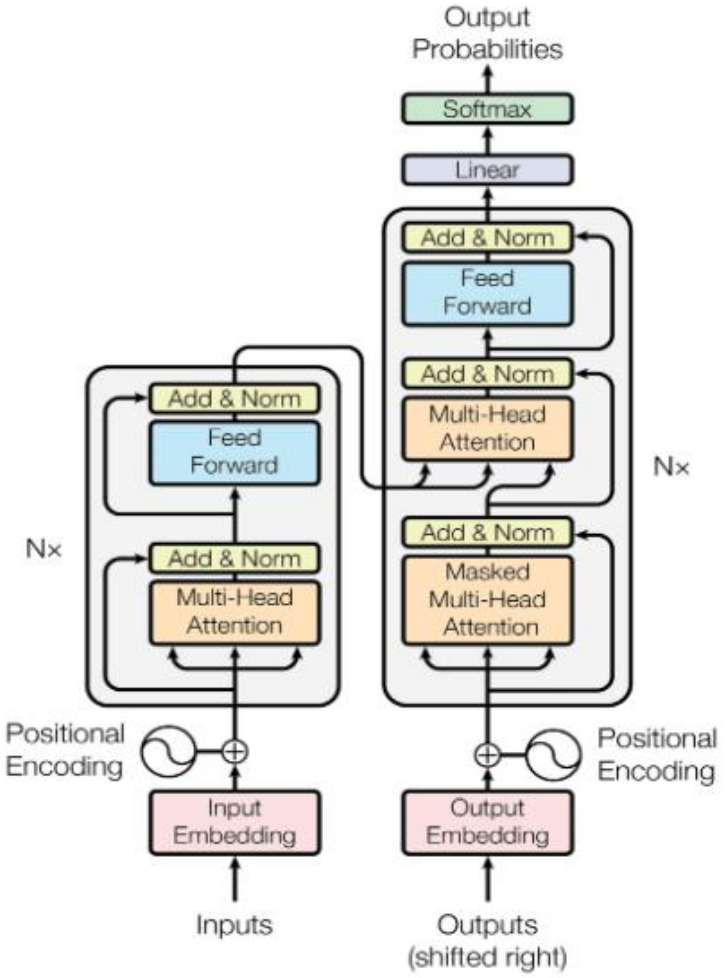
Limitations

Weaker for pure classification unless adapted, can hallucinate

Technical notes

Trained with “causal language modeling”. Uses a causal attention mask

Transformer families for NLP – Encoder Decoder



Description

Models that transform one sequence into another. They are designed for **sequence-to-sequence** tasks.

Common Tasks

Translation, summarization, paraphrasing, QA, text rewriting

Strengths

Very good for transformation tasks, flexible, strong conditional generation

Limitations

More complex architecture, higher compute than single-stack models

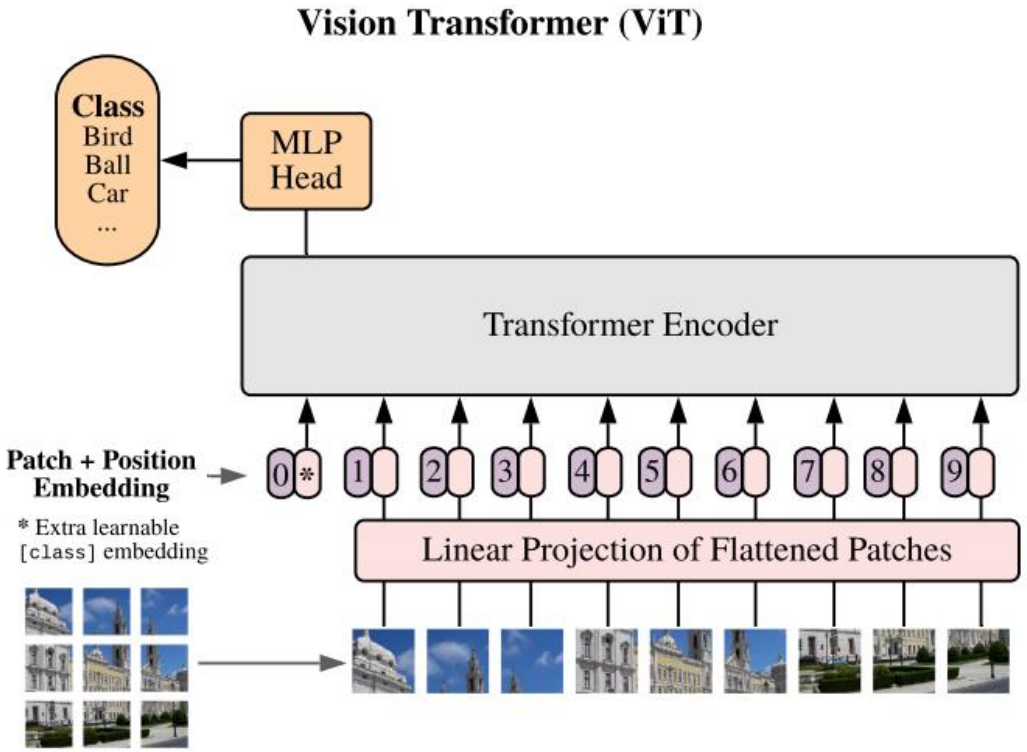
Technical notes

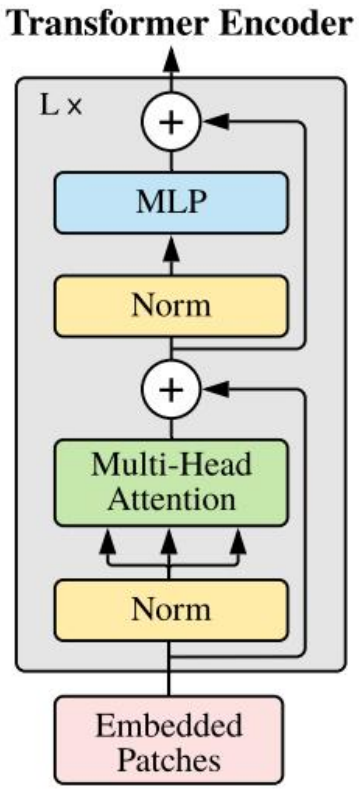
Trained with seq2seq objectives. Uses both self-attention and cross-attention.

Vision Transformer (ViT) for Image Classification

- The ViT divides an image into fixed-size patches and maps each patch to a vector representation using a linear projection. These patch embeddings are then processed by a Transformer encoder.

- Unlike CNNs, ViT directly models image patches with self-attention and performs very well on image classification when trained on large datasets.





Transformers pro

- **High modeling capacity**
  - Effectively captures **long-range dependencies** through self-attention.
  - Produces **context-aware token representations** across the full input sequence.
  - Multi-head attention learns **complementary semantic relationships**.

- **Parallelizable training**
  - Unlike RNNs and LSTMs, Transformer architectures support **parallel computation across tokens during training**.
  - This leads to **higher training throughput** on modern hardware accelerators.

- **Scalable architecture**
  - Performance generally improves with **model size, data scale, and compute budget**.
  - The same backbone can be adapted to many NLP tasks with minimal architectural changes.

- **Strong transfer learning**
  - Pretrained Transformers can be fine-tuned efficiently for downstream tasks.
  - They are effective for **classification, generation, retrieval, and sequence-to-sequence applications**

Transformers cons

- **High computational and memory cost**
  - Self-attention has **quadratic complexity** with respect to sequence length: $\mathcal{O}(n^2)$
  - This makes long-context processing expensive.

- **Large data requirements**
  - High performance typically depends on **massive pretraining datasets** and well-designed tokenization.
  - Training from scratch is **computationally expensive**.

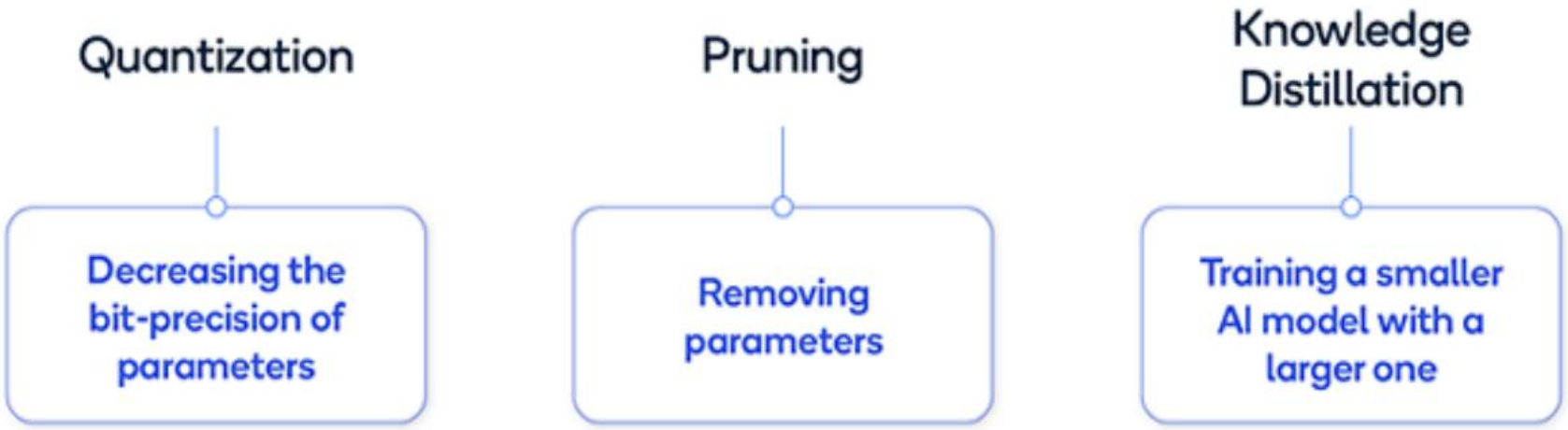
- **Autoregressive decoding**
  - During generation, **each new token depends on previously generated tokens**, which limits parallelization.
  - This **increases latency**, especially for long outputs.

- **Limited edge suitability**
  - Large parameter counts and growing **KV cache** requirements make deployment difficult on **low-resource devices**.
  - Compression techniques such as **quantization, pruning, and distillation** are often necessary.

- **Energy and storage footprint**
  - The combination of model size, activations, and attention states can result in **high memory usage** and **significant energy consumption**.

Transformers for Vocal Commands Recognition @STM32

How to reduce the size of a Transformer?



- **Quantize the model**
  - reduce precision with PTQ* or QAT**; lower memory footprint and energy consumption
- **Prune redundant parameters**
  - remove weights, channels, or attention heads; use structured pruning for hardware efficiency
- **Distill knowledge**
  - transfer knowledge from a large teacher to a smaller student; preserve performance with fewer parameters

PTQ* Post Training Quantization
QAT** Quantization Aware Training

Transformer for Vocal Commands Recognition on MCU

- **Compact Transformer design**
  - reduced depth, width, and attention head count for a smaller model footprint
- **Quantization-driven optimization**
  - evaluate layer support with QKeras
  - apply PTQ/QAT to reduce memory and compute cost
  - analyze layer-wise sensitivity to precision loss
- **MCU deployment constraints**
  - fit within limited SRAM and flash
  - use integer-friendly inference paths
  - target real-time, low-power execution
- **Design trade-off**
  - balance **accuracy**, **latency**, **memory footprint**, and **energy consumption**

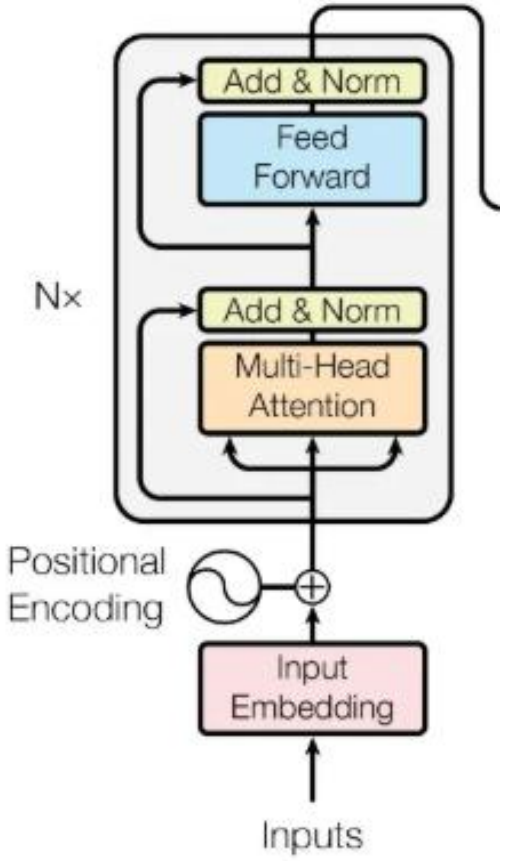
A lightweight encoder-only Transformer for vocal commands classification

Transformer as classifier

- Short audio commands from the Google Speech Commands dataset
- Classification of **10 command classes**

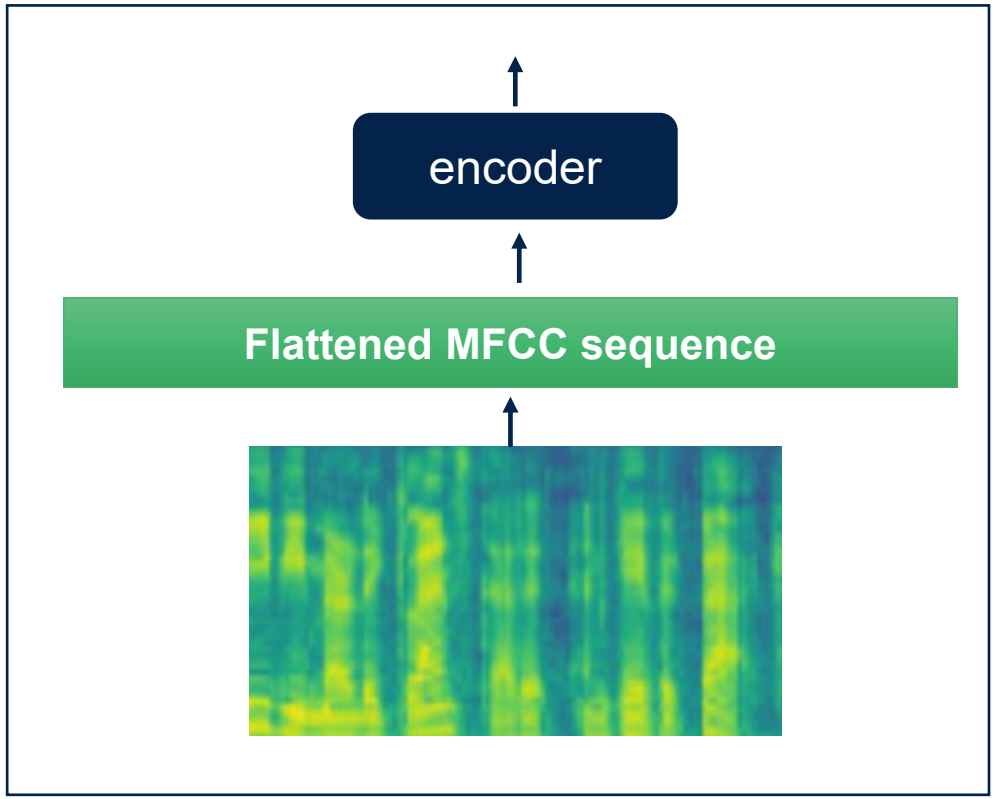
Topics to investigate

- Can **tokenization** be avoided?
- Can **embeddings** be avoided?
- What is the impact on accuracy and efficiency?



- **Encoder-only architecture**
  - no decoder stack

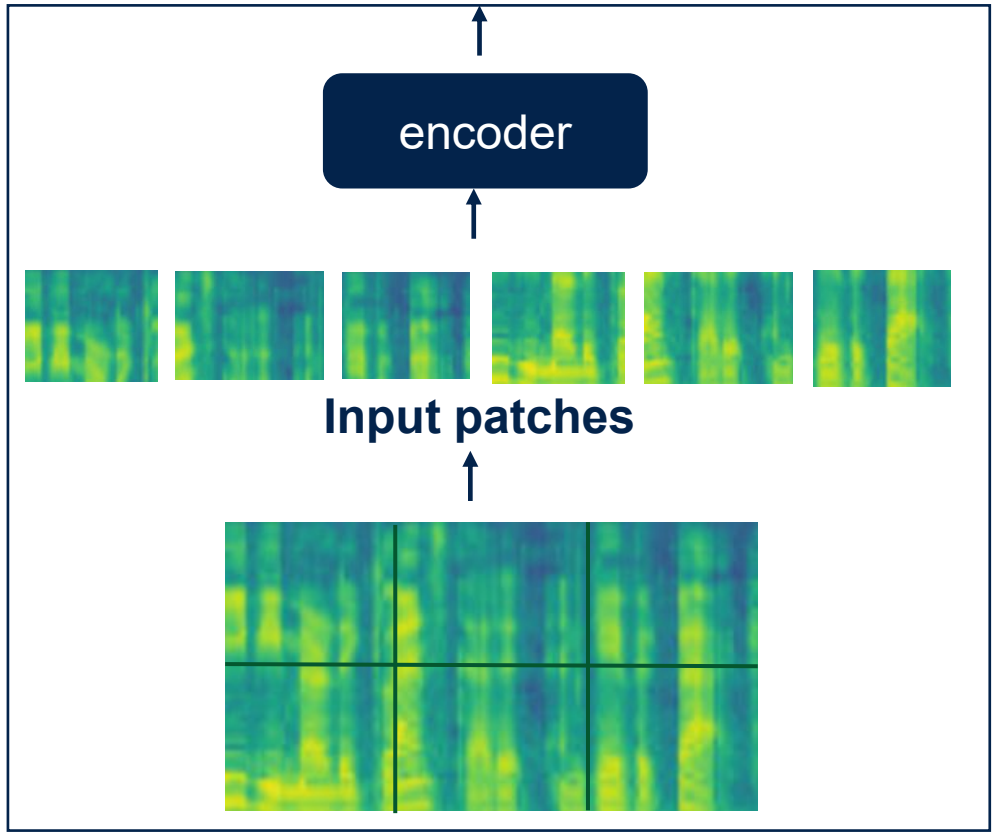
Transformer based architectures for vocal command recognition – Architecture (A)



The MFCC matrix is linearized into a single sequence of values and fed directly to the encoder, without patching or embedding.

Feature	Sequential
Self-attention	No
Embedding	No
Pos. embedding	No

Transformer based architectures for vocal command recognition – Architecture (B)



The MFCC matrix is split into patches, which are then embedded and processed by the encoder, similarly to Vision Transformers

Feature	Patch-based
Self-attention	Yes
Embedding	Yes
Pos. embedding	Yes

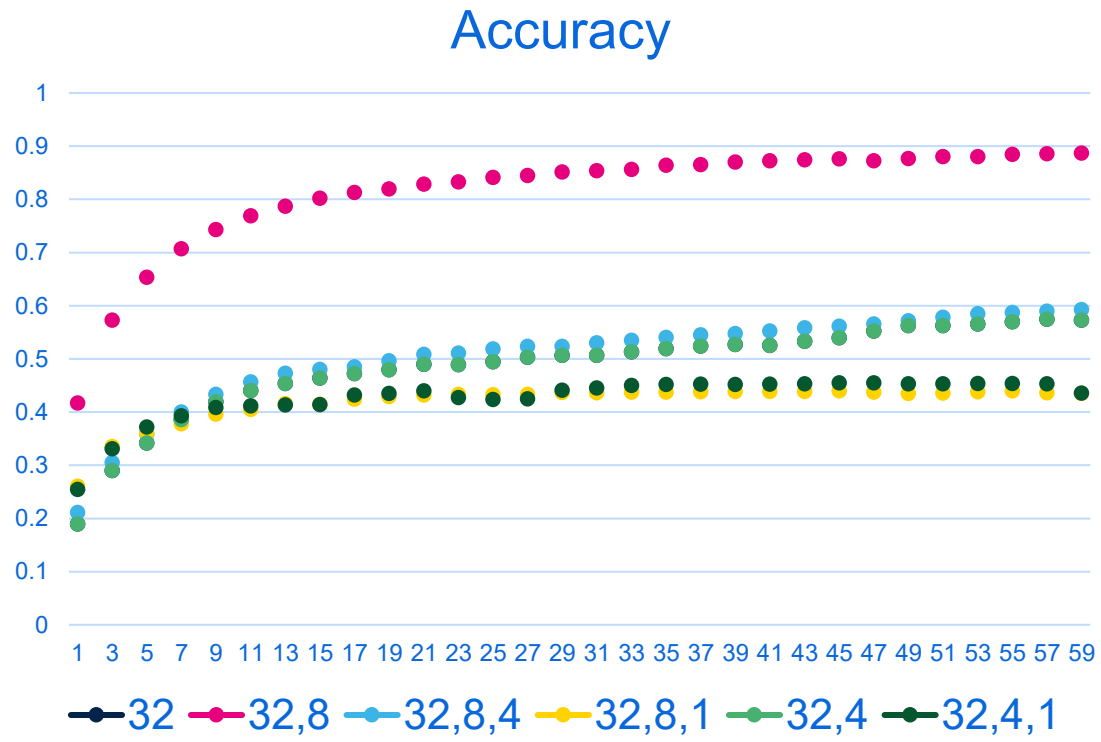
Qkeras for Quantization Aware Training

- QKeras is used to train neural networks with quantized weights and activations
- The objective is to analyze the trade-off between accuracy, trainability, and model size
- Several hybrid quantization schemes are evaluated and compared
- The next slide compares the accuracy and model size achieved by STConformer using several quantization-aware training configurations

STConformer Summary

Bit precision	Acc.	Trainables	Tool
32 bit (TF)	90%	1 MB	TF
• MHA* 32bit • 8bit	89%	< 1 MB	QAT** by QKeras
• MHA* 32bit • 8bit, 4bit	71%	< 1 MB	QAT** by QKeras
• MHA* 32bit • 8bit, 4bit, 1 bit	42%	< 1 MB	QAT** by QKeras
• MHA* 32bit • 4bit	70%	< 1MB	QAT** by QKeras
• MHA* 32bit • 4bit, 1bit	43%	< 1 MB	QAT** by QKeras

MHA*: MultiHeadAttention
QAT**: Quantization Aware Training by QKeras
From CubeAI: Unsupported MHA layer (2023)



Our technology starts with You

Find out more at www.st.com