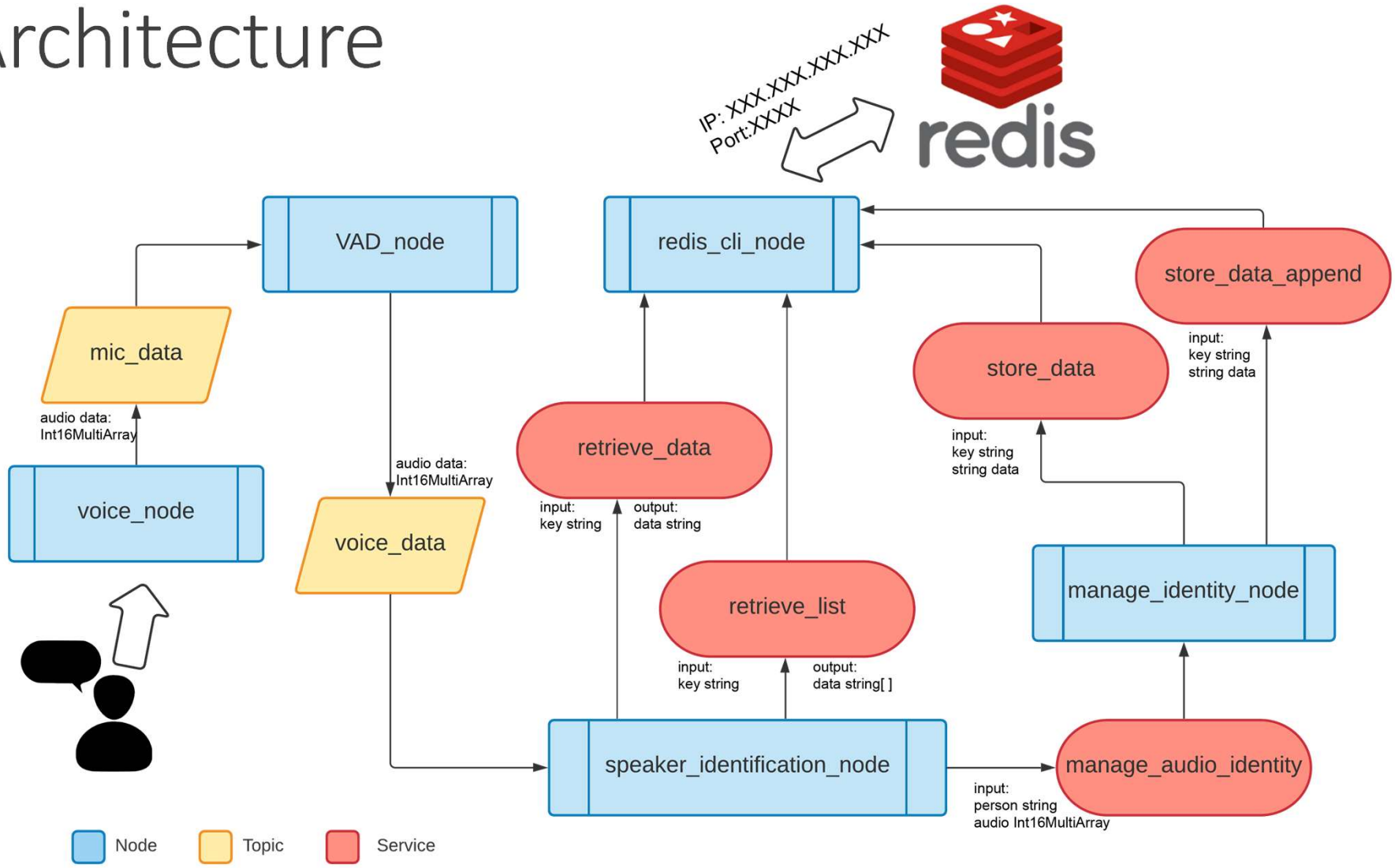
Cognitive Robotics Contest

- PICARIELLO EMILIO
- VALITUTTO ANDREA

Architecture



Main Ideas

Publisher/subscriber architecture:

Main task is tought to be deployed in a real-time scenario and because this project works with a stream of audio data it seems appropriate to use a publisher subscriber as main architecture:

- Pub/sub has been used for main operations as audio capture, VAD, Speaker Recognition. For the last two using a service architecture would slow down elaboration and results would be available for just one node per time, so using pub/sub may allow other nodes to use such data when needed and helping lowering coupling between them
- An event based architecture like pub/data is a better way to handle a continuos flow of audio data

Service communication:

Not every node uses a pub/sub type of data exchange

- Services are used for operations that do not have to run constantly
- They are used when is needed to answer in a different way to a different client request, for example handling Redis requests

Main Ideas

Audio elaborations:

- Audio is captured and filtered, taking in account only audio with energy greater than a certain threshold representing ambient noise, this creates a stream of audio organized different audio clusters
- Audio is elaborated in VAD task, via a SED, so that the arriving cluster of audio is filtered leaving only human activity, in other words audio is rearranged to filter out any non human data, via splitting and then bonding again in a same audio sample
- Audio is finally used in final identity recognition

With this method, SED has not to handle with continuous data but with clusters. Using this, an audio sample could be used even with some sort of sudden not human noise, because is meant to be removed without splitting original track in two different pieces to be handled separately

Support Database

Redis:

- It is an open source in-memory data structure store, used as a database, cache, and message broker
- It provides data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs, geospatial indexes, and streams

Why using it:

- It is used to store speaker identities and audio without saving them directly in a local directory, achieving a more secure way to store data and a faster way to retrieve it
- It is simple to export all data from a machine to another, only dump file is needed
- A sql database seems to be excessive because not so many different types of data are to be stored, mongoDB uses a cluster online and to improve portability is not a good idea to ask for a mandatory internet connection
- Remarkable disadvantage is having an harder manual editability of stored data but, other than for debug and develop purpose it is not a big deal for final user

How Redis is used

Redis stores data in multiple ways, mainly using a key per single or multiple data values.
Those are the more convenient ones for the actual implementation:

- one key per single value
- one key for list of values

Identities:

All identities are saved in a json file which is saved as a string, key = «identities», value = json stringified. In this Json file identity name, cache_id (to retrieve audio samples) are written.

Audio sample:

All audio samples are saved as stringified bytes, inside a list where key is given by identity cache id. That id uses full name + first encounter date, example, if Mario Rossi is met on 12 of May 2021 at 10:32:01 it is given «mario_rossi_20210512103201». All audio samples for each identity are stored in this way:
key = cache_id, data = [audio1,…,audioN]

How Redis is used

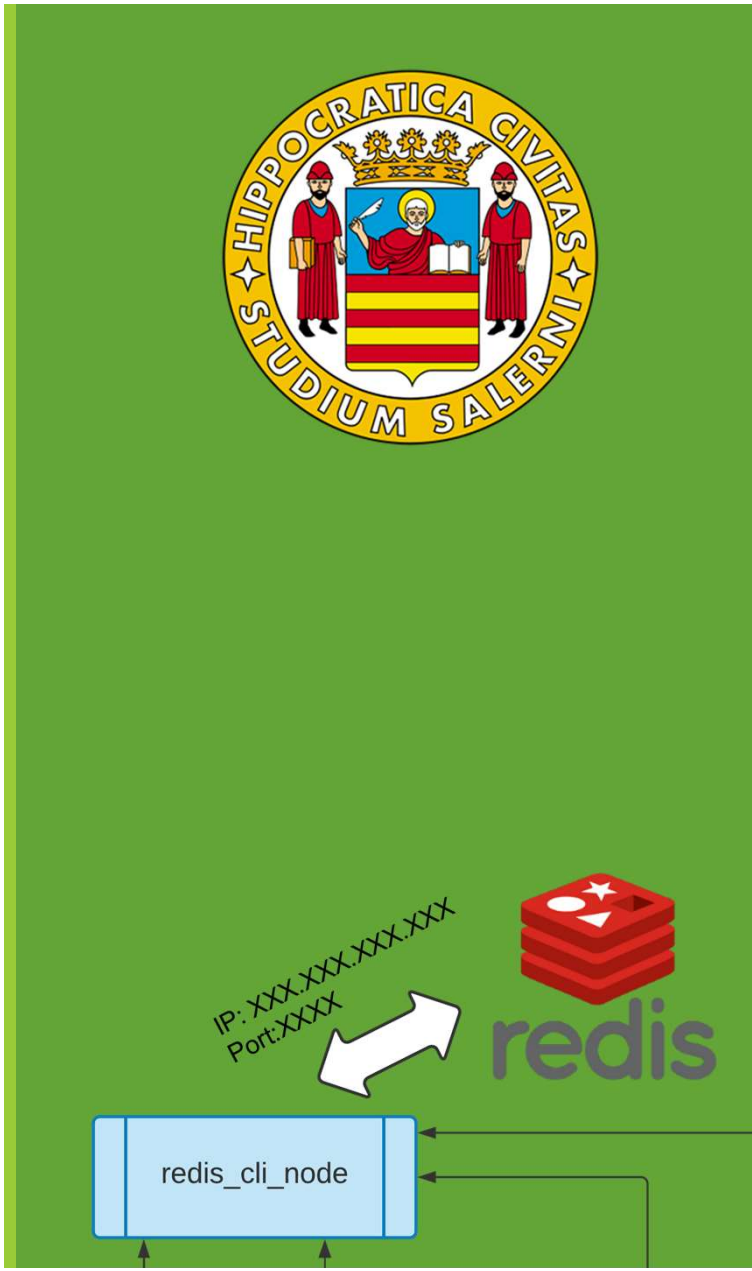
Example database:

key	value
identities	{[«cache_id»: «mario_rossi_20210512103201», «name»: «Mario Rossi»],[],…,[]}
mario_rossi_20210512103201	[stringified audio 1, stringified audio 2, stringified audio 3, stringified audio 4]
luigi_bianchi_20210512103201	[stringified audio 1, stringified audio 2, stringified audio 3]

Node Description (1/5)

Redis CLI Node:

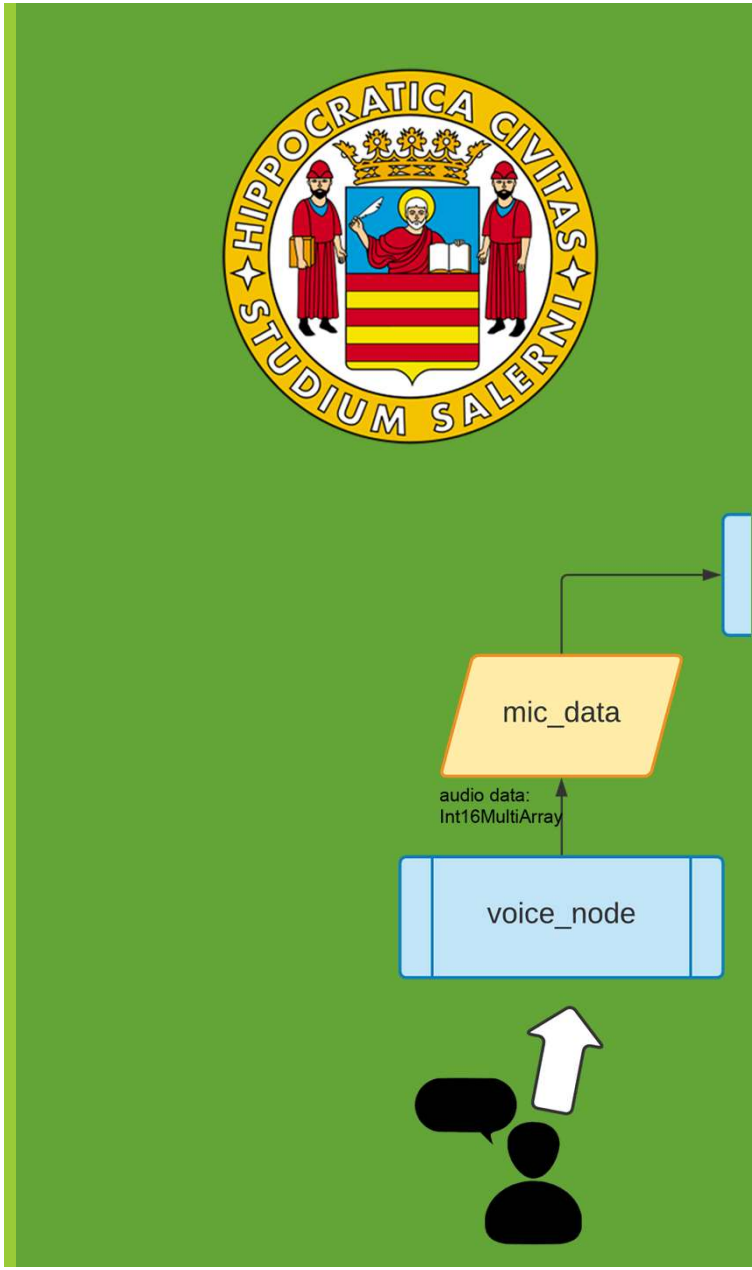
- This node uses a command-line interface (CLI) python library and it works as a wrapper, implementing store and retrieval commands as ros services
- Services available:
  - '*store_data_append*' to append some data to a list of a given key
  - '*retrieve_list*' to get all data available from a list of a given key
  - '*store_data*' to store data in a record with a given key
  - '*retrieve_data*' to retrieve data from a record of given key



Node Description (2/5)

Voice Node:

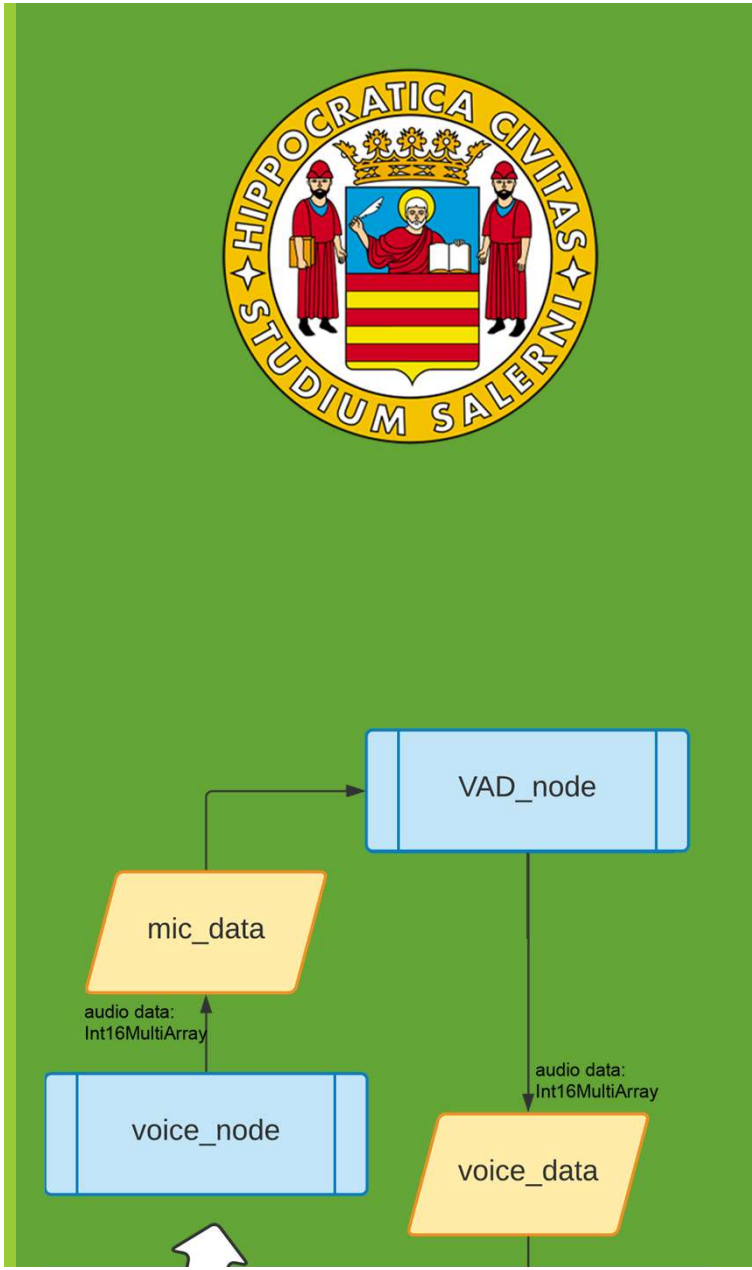
- This node handles audio capture from internal microphone of pc
- Sample rate is 16000
- If pc has not an internal microphone it uses its primary microphone
- It is calibrated on ambient noise, if energy of incoming data is higher than calibration threshold it publishes it on topic '*mic_data*' as a Int16MultiArray
- It uses publisher/subscriber communication



Node Description (3/5)

Voice Activity Detection Node:

- This node implements VAD task from incoming audio from '*mic_data*'
- VAD is implemented using a sound event detection net adapted to this task.
- Sound Event detection (SED) classifier is YAMNet, implement by Google. It is employed on mobilenet and implemented in keras.
- Original array is filtered leaving only voice portions and given as output on topic '*voice_data*'

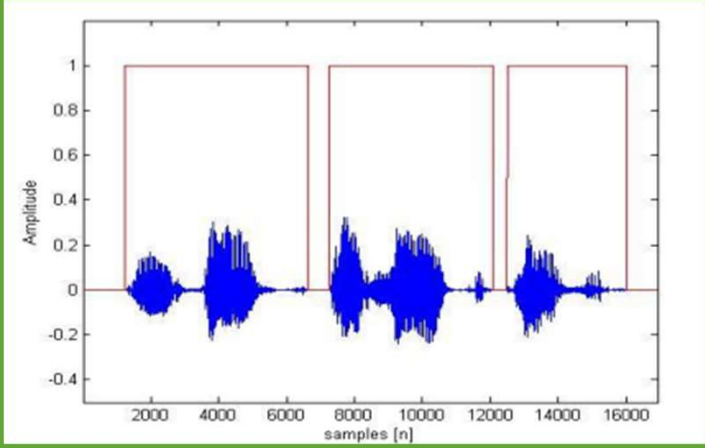


Voice Activity Detection (VAD) task

Common approaches:

- Energy thresholding, accomplished choosing an energy threshold high enough to ignore ambient noise (chosen by calibration). Every chunk of signal is assumed as voice activity only if higher than said threshold. It is a lightweight approach, but it does not check if it is really a human voice or just a loud sound event
- Probabilistical approach, most common solution is Google's WebRTC VAD which uses a Gaussian Mixture Model to discriminate voice from noise on pretrained noise and voice features. It is a lightweight approach, with better performance than energy thresholding but it suffers from many false positives and it is not implemented in Python, only a wrapper is available but it is poor documented and difficult to use
- Machine Learning:
  - Ad hoc Neural Network, those are scarce and usually poor documented/implemented.
  - Adapting a Sound Event Detection Network, this solution gives the opportunity to choose a pretrained network from a more vastly documented and implemented task. Also it makes it possible to check if that is a human voice and not a loud sound event.

the one chosen



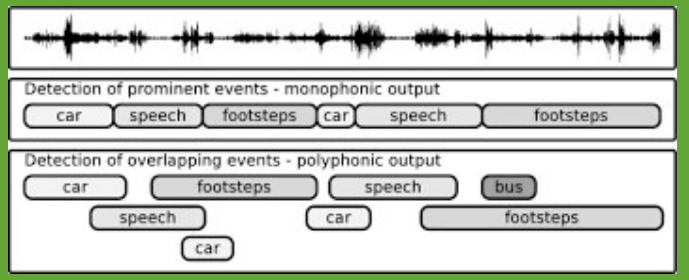
Sound Event Detection (SED) task

YAMNet:

- YAMNet is a pretrained CRNN, it is a state of art for SED task and uses 521 audio classes based on AudioSet-Youtube dataset
- Employed with Mobilenet, it is optimized to be used in embedded applications
- On the 20,366-segment AudioSet eval set, over the 521 included classes, the balanced average d-prime is 2.318, balanced mAP is 0.306, and the balanced average lwlrap is 0.393
- By and large it is an optimal solution, it performs excellent results and it is built to be quite lightweigth

Implementation:

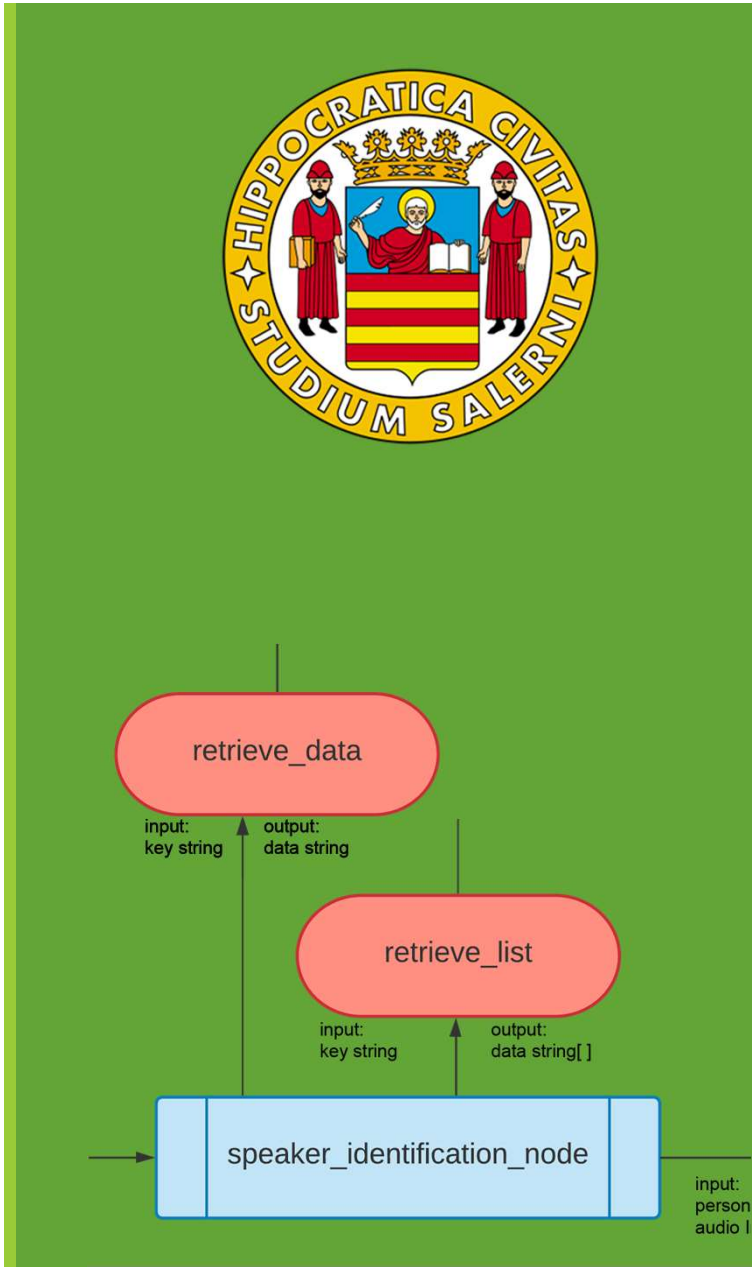
- All neural network has been implemented as it is, its output has been adapted for VAD problem, considering first 31 tags (es. »speech»,»laughter»,»scream») as vocal activity and all the others as not vocal activity
- Main concern was real-time performance, but thanks to mobilenet employment, it is feasible to be used even in background, so we decided to use it as it is and not trying to search for a more lightweigth neural network or a not machine learning solution



Node Description (4/5)

Speaker Identification Node:

- This node implements Speaker Reidentification task on the incoming audio Int16MultiArray from '*voice_data*' topic
- It recreates an audio embedding file every time it is run, every time a new identity is added or when asked to via service '*create_audio_embedding*'
- Embedding creation uses '*retrieve_data*' and '*retrieve _list*' to gather identities and their audio samples to use for its task
- It uses DeepSpeaker, one of the best neural network for this task
- Features are extracted from incoming audio and compared via cosine similarity to ones present in embeddings file. Most similiar one represents predicted identity. If score doesn't go beyond a certain threshold it is thought to be a new identity. If score goes beyond threshold but not so much, it is considered to be of low reliability
- When identity is not recognized or it's score is not reliable it calls service '*manage_audio_indentity*' to ask final user if they want to add new identity/audio to Redis



Threshold evaluation

Threshold has been chosen empirically, trying multiple different situations and input conditions. Sintetically those are the most important case scenarios in relation to score for a true identity:

Actual audio recorded to recognize identity with	Audio samples saved for that identity	Recognition score
Clean audio	< 4 audio	0.64 – 0.83
High background noise	< 4 audio	0.56 – 0.67
Clean audio	> 4 audio	> 0.72
High background noise	> 4 audio	> 0.65

It has then been chosen a threshold of 0.55

Speaker Reidentification task

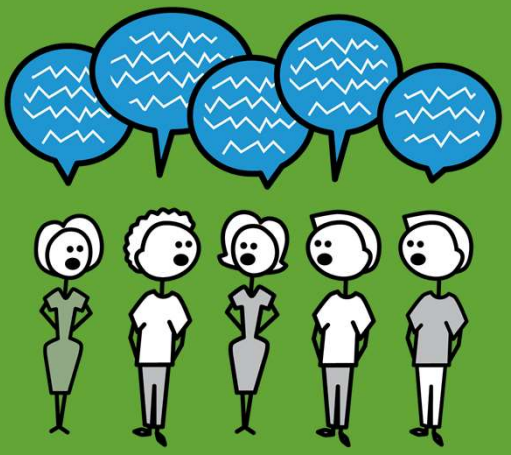
Possible Neural Networks:

Main evaluation metrics is EER (equal error rate), model and implementation. Those are the available and implemented neural networks (with their paper's specifications)

Name	Model	EER	Implementation
Siamese Capsule Network	Thin-ResNet34 + Siamese Capsule	3.14%	PyTorch
AutoSpeech	ResNet-34	11.99%	PyTorch
SincNet	SincNet-Raw	0.51%	PyTorch
Deep Speaker	ResCNN	1.83%	Keras, Pytorch

Best two are SincNet and Deep Speaker in terms of performance. It has been chosen to go with Deep Speaker, sacrifing that 1.3% of EER preferring a keras implementation over a pytorch one, mainly for two reasons:

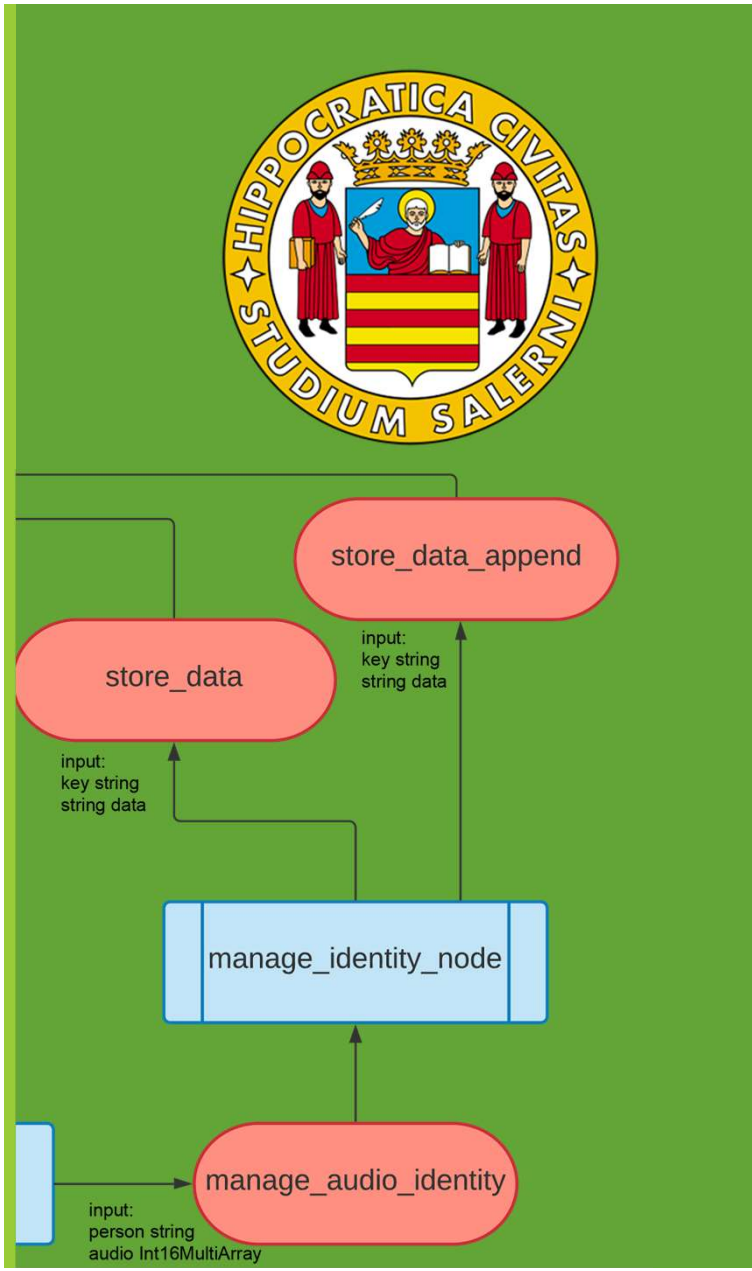
- more programming experience with keras
- keras is the most used in machine learning, using this library provides more support and more portability



Node Description (5/5)

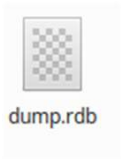
Manage Identity Node:

- This node is an user oriented interface to communicate via terminal and let they decide if to add a new identity or audio when that is a viable option
- It is a server of '*manage_audio_indentity*' service, which, called by reindentificator node, is used in two cases:
  1. ReID node believes that it has never met the speaker according to its incoming audio, so perhaps a new idenity is needed to be added
  2. ReID node associated an existing identity with a very low score, so perhaps a new audio is needed to be appended for that identity to improve performance
- When called it askes user to answer yes or no, to tell if they want to add new resource, it gives user 5 second to answer, if exceeded a negative response is assumed
- To store a new audio it uses '*store_data_append*' service
- When it is a new identity it gives user plenty of time to choose a name, after that it uses '*store_data*' and '*store_data_append*' to store in Redis



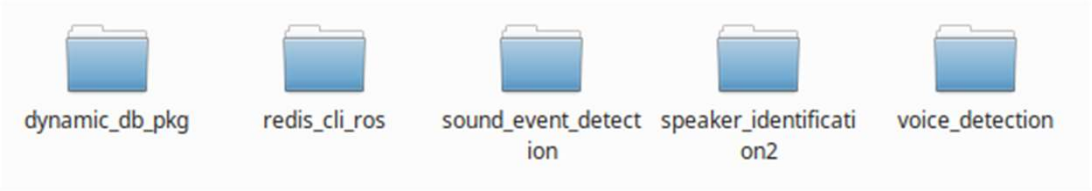
Workspace Organization

Where redis has been installed data will be saved in a dump file like the one in there:



With just this file all data could be exported in a different a different system very easily

Inside src every node has its package:

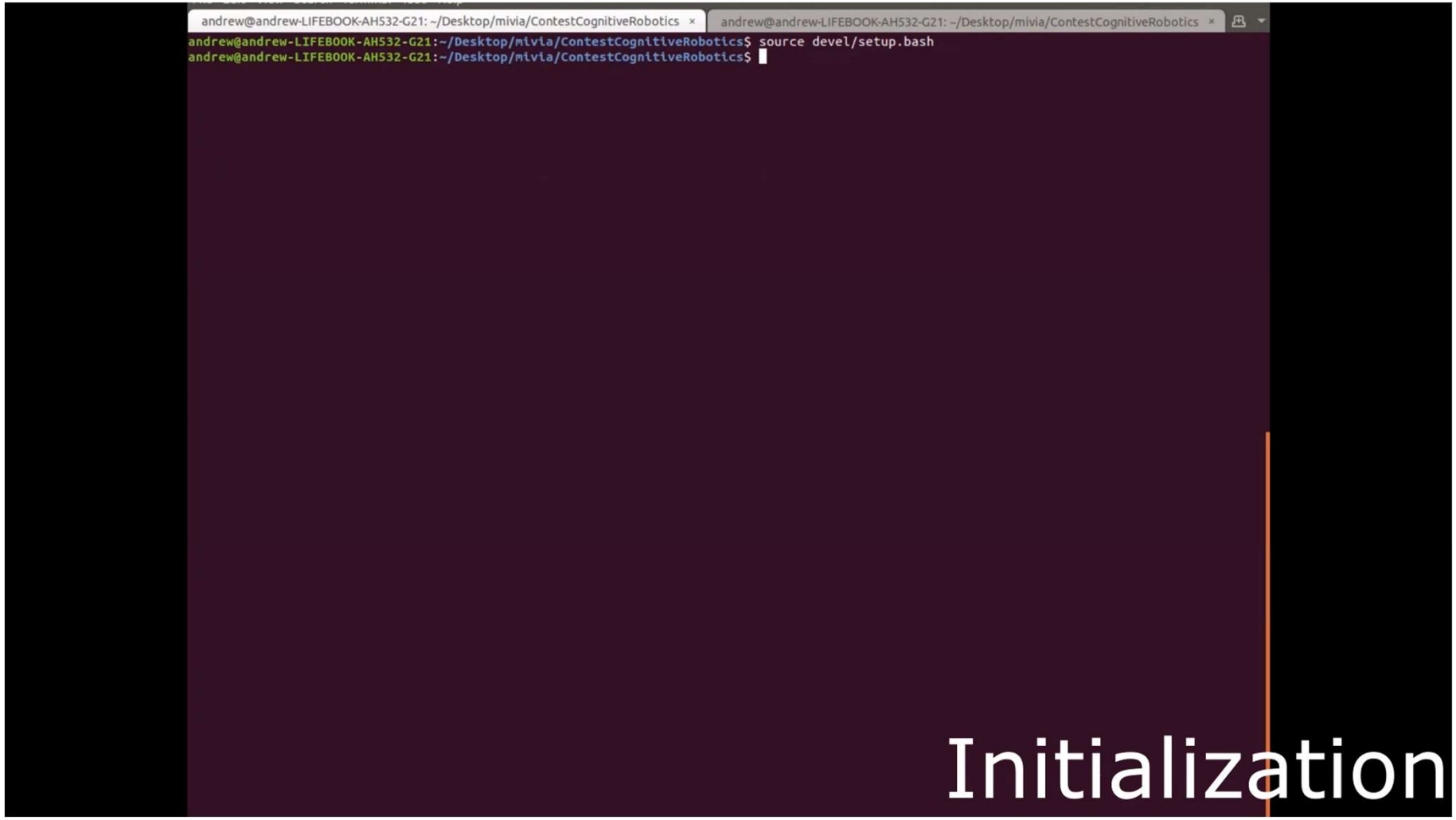


dynamic_db_pkg contains Manage Identity Node
redis_cli_ros contains Redis CLI Node
sound_event_detection contains Voice Activity Detection Node
speaker_identification2 contains Speaker Identification Node
voice_detection contains Manage Identity Node

Possible future developments

- Deleting an identity and its record when said person is not encountered after a chosen span of time (i.e. 1 week, 1 month)
- '*create_audio_embedding*' has no current client implementation, it has been decided to leave it available for a future node that periodically decides to create a new embedding
- Manage identity node could be substituted/modified to use an advanced user interface or something else
- Using identificator's unused topic 'output_id' for other tasks
- Deploying this model in a real world scenario analyzing false and true positive rates and fine tune score threshold on it

Demo video


andrew@andrew-LIFEBOOK-AH532-G21: ~/Desktop/mivia/ContestCognitiveRobotics
andrew@andrew-LIFEBOOK-AH532-G21: ~/Desktop/mivia/ContestCognitiveRobotics
andrew@andrew-LIFEBOOK-AH532-G21:~/Desktop/mivia/ContestCognitiveRobotics$ source devel/setup.bash
andrew@andrew-LIFEBOOK-AH532-G21:~/Desktop/mivia/ContestCognitiveRobotics$
Initialization