Embedded ChatBot

Project of a tool for interaction with everyday objects

Elisa DELHOMMÉ
3A-SIA

2021-2022, ENSEA

Surpervised by Sylvain Reynal

Contents

Abstract

This project consists of the creation of a tool enabling people with difficulties or disabilities (e.g. blindness) to interact with everyday objects. The user will thus be able to know the location or status of the desired object. This study focuses only on the software aspect of the tool, and not on the recovery of the surrounding data.

A first simple model (dealing with sine prediction) and its embedding on the board are studied in order to understand the process and operation of a model on the board.

Thereafter, the choice of the chatbot model in order to carry out this project was required, taking into account the important and non-negligible software and hardware constraints. The detailed implementation of the model on the electronic board, from its conversion into an embedded language to the necessary adjustments for the perfect functioning of the tool with the adapted processing, was also developed in the project.

1 Introduction - Context of the project

The work developed here is carried out within the framework of a project to help and assist people with difficulties or disabilities in their daily tasks. More specifically, we expect the user to be able to **interact with the objects**. These would be equipped with sensors and it would be possible to know both their precise **location** and their **state**. For example, a blind person could interrogate the tool in order to know if the coffee pot is correctly positioned on its base or if it is full or if it needs to be refilled.

The specifications of the tool are then defined as follows:

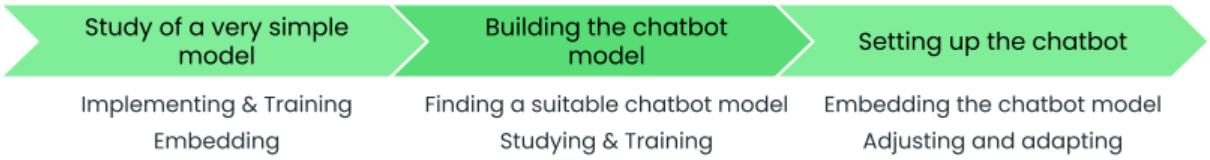
Parts	Speech recognition	**Chatbot**	Speech synthesis
Goal	Listening to and understanding the user	**Return of the adapted answer**	Voice output of the answer
Objective	The tool must be able to hear what the user says and render it in writing	**The tool must be able to understand the user and provide a relevant answer.**	The tool must be able to render a written sentence vocally.

In all cases, the difficulty and the challenge lies in embedding the developed models on an STM32 board. Therefore, the following study focuses on this task rather than on the development of prediction models.

The first part, the speech recognition part, was studied by Samane HABIBI. The following study will only concern my own work about the chatbot part.

My concrete objective is then to be able to enter a textual message (with the keyboard, the part of oral restitution being carried out by another student), and that the STM32 card can provide him an answer (displayed on the screen for instance).

Here is a brief visualization of the three different stages of this work.



2 State of the art - Review of existing chatbots

Chatbots have been around since the 1960s, and have evolved a lot since then, especially in recent years.

What is a chatbot ?

A chatbot is a software robot that can converse with an individual or consumer through an automated conversation service that can be performed through choice trees or through an ability to process natural language.

The very first chatbot

The first chatbot was born in 1966 from the work of Joseph Weizenbaum, a professor at MIT. The program was designed to mimic human conversation. The **ELIZA** chatbot works by transmitting the words users type to a computer, then matching them with a list of possible scripted responses. This simulates a psychotherapist.

The program worked by association of ideas but lacked artificial intelligence. However, Weizenbaum was troubled by the reaction of users. He expected ELIZA to be a mere caricature of human conversation, but users were confiding their deepest thoughts. Experts said chatbots would be indistinguishable from humans in a few years.

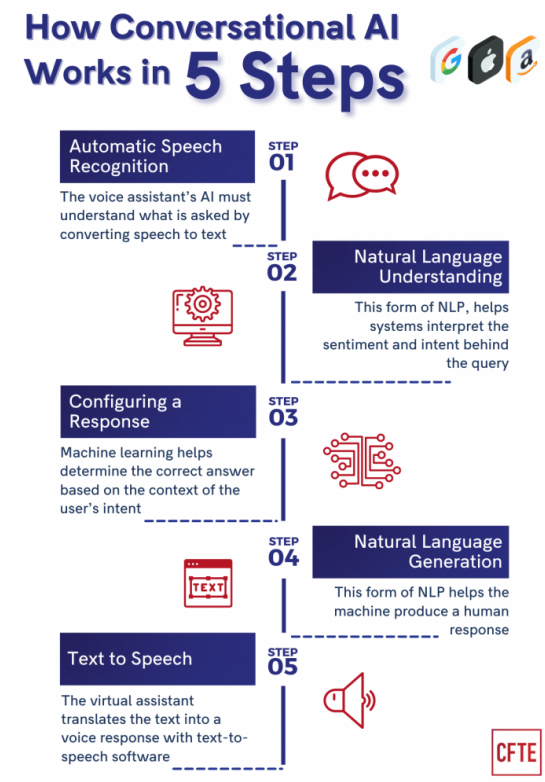
Precursors of today's chatbots

The first chatbots played for many the role of doctors or psychologists, and evolved little by little until they integrated AI and resembled everyday people (with an identity, hobbies etc.), such as the **A.L.I.C.E.** (1995) or **SmarterChild** (2001) chatbots that preceded the popular Siri.

The new generation

The latest chatbots are the ones we know and are very efficient. One of them is **Siri**, Apple's chatbot for iOS since 2010. It can be used in writing or orally depending on the user's desire. Unlike its predecessors, it can respond in a textual way but also with various media such as images or audio.

The most recent are **Google Assistant**, **Cortana** or **Alexa**, all born in the 2010s. They all work with a voice recognition system and are able to provide information or recommendations to the user. Nowadays, chatbots are more like personal assistants than the psychotherapists of their beginning.



These chatbots all work on the same general principle. The chatbot, in order to generate the most suitable answer, concerns steps 2 to 4:

- Analysis and search for meaning in the user's words
- Generation of an adequate answer
- Formulation similar to that of a human being

Most of them use NLP (Natural Language Processing) processes with recurrent layers of neurons to operate.

3 Preliminary test with a simpler task (in cooperation with Pierre Chouteau)

In order to better understand the embedding of a Deep Learning model on the electronic board, the first study deals with the case of a very simple model that will require little processing. More precisely, it concerns the study of the prediction of the value of the sine of a number based on the article and tutorial of Shawn Hymel [1].

3.1 Test model

The developed model is extremely simple: it consists of three dense layers and immediately predicts the estimated value.

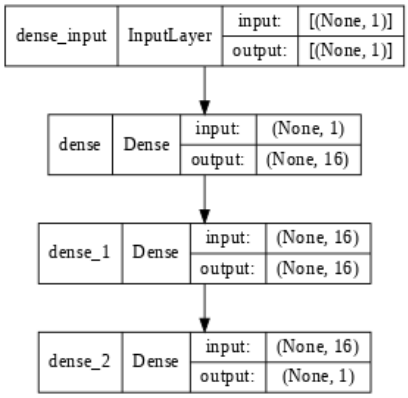
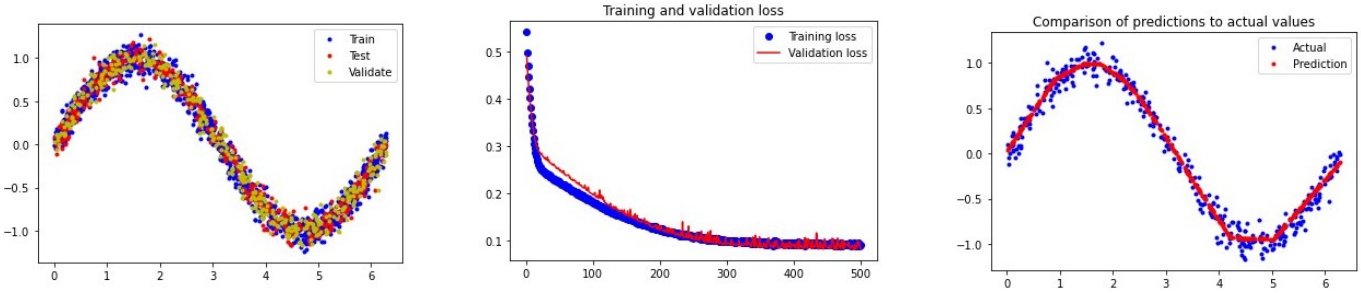


Figure 1: Sine prediction model

Training, test and validation data are generated manually, training is done on 500 epochs and is quite efficient.



(a) Training data (b) Training history (c) Results on test data

Figure 2: Information about the model

Exporting the model as an .h5 file is then necessary in order to embed it.

3.2 Introduction to X-Cube-AI - Embedding on the electronic board

What is X-Cube-AI and why do we need it

X-Cube-AI is an extension package of STM32CubeIDE with automatic conversion of pre-trained Artificial Intelligence algorithms (including Neural Network and Machine Learning models), and integration of generated optimized library into the user's project. It allows us to generate a model with Python code, then to convert it into C code in order to be able to run it on the board. [2]

Generating the C code

Using the h5 file generated earlier, the C code can be generated simply by selecting the X-Cube-AI extension (v.5.1.2) and by adding the corresponding model within the CubeIDE interface. A more detailed tutorial is available on the project GitHub [3].

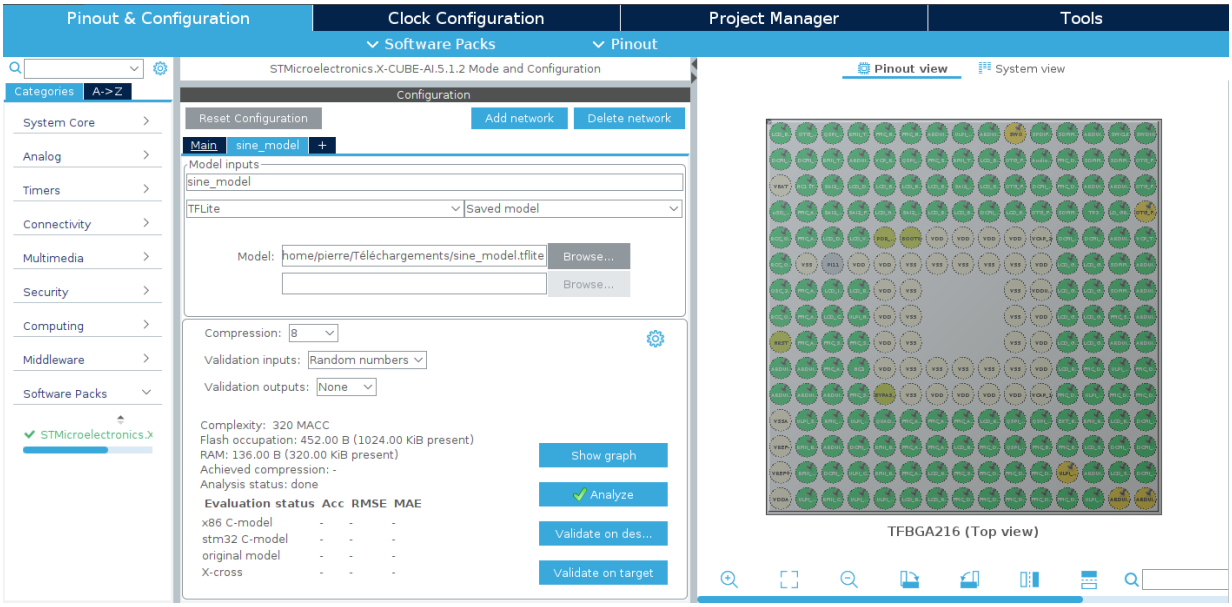
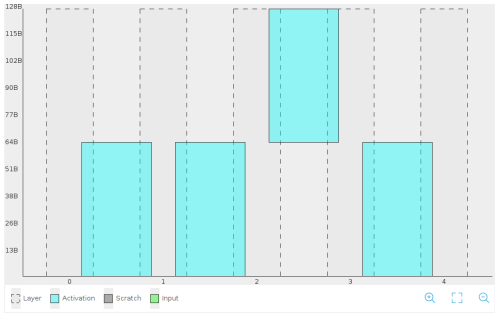


Figure 3: CubeIDE interface : code generation

This interface also provides information about the model, such as memory usage, with details by layer. It also allows validation to be performed on the desktop or target, in a preliminary way without having to add the model to the code.



(a) Memory distribution per layer

```
Using TensorFlow backend.
model_name        : sine_model
model_hash        : 184e97180e56be3a44c6742602736483
input             : serving_default_dense_input0 [1 items, 4 B, ai_float, FLOAT32, (1,)]
inputs (total)    : 4 B
output            : dense_2 [1 items, 4 B, ai_float, FLOAT32, (1,)]
outputs (total)   : 4 B
params #          : 321 items (1.25 KiB)
macc              : 320
weights (ro)      : 452 B (452 B) (-64.80%)
activations (rw)  : 128 B (128 B)
ram (total)       : 136 B (136 B) = 128 + 4 + 4

id  layer (type)                output shape      param #     connected to            macc          rom
0   serving_default_dense_input0 (Input) (1,)
    dense_0 (Dense)              (16,)             32          serving_default_dense_in 16            128
    nl_0 (Nonlinearity)          (16,)                         dense_0                 16
1   dense_1 (Dense)              (16,)             272         nl_0                    256           256 (c)
    nl_1 (Nonlinearity)          (16,)                         dense_1                 16
2   dense_2 (Dense)              (1,)              17          nl_1                    16            68
sine_model p=321(1.25 KBytes) macc=320 rom=452 Bytes (-64.80%) ram=128 B io_ram=8 B

Complexity per-layer - macc=320 rom=452
id      layer (type)                macc                                    rom
0       dense_0 (Dense)             ||                                5.0%  |||||||||||||||   28.3%
0       nl_0 (Nonlinearity)         ||                                5.0%  |                  0.0%
1       dense_1 (Dense)             ||||||||||||||||||||||||||||||||  80.0%  ||||||||||||||||||||||||||||||  56.6%
1       nl_1 (Nonlinearity)         ||                                5.0%  |                  0.0%
2       dense_2 (Dense)             ||                                5.0%  |||||||          15.0%
```

(b) More details about used memory

Results

Once the model has been generated in C language by the software, adjustments are often necessary (only in the `USER CODE` parts, which allow the generation to keep if redone), in particular to achieve pre- and post-processing of the data. In the case of sine prediction, a value is entered and the model returns the predicted sine value, no superficial adjustments are needed. Below are some tests to check that the model runs correctly on the board.

test	float	0
y_val	float	0.0594997816

(a) Sine prediction of 0

test	float	1.57075
y_val	float	0.932305694

(b) Sine prediciton of $\pi/2$

test	float	3.1415
y_val	float	-0.0227158722

(c) Sine prediciton of π

test	float	4.71225023
y_val	float	-1.05025923

(d) Sine prediciton of $3\pi/2$

Figure 5: Tests with different values in order to verify the correct operation of the model

4 Implementation of the chatbot

The choice of the chatbot model was a first challenge to overcome. My first intention was to develop a model proposed by PyTorch [4] according to the following architecture:

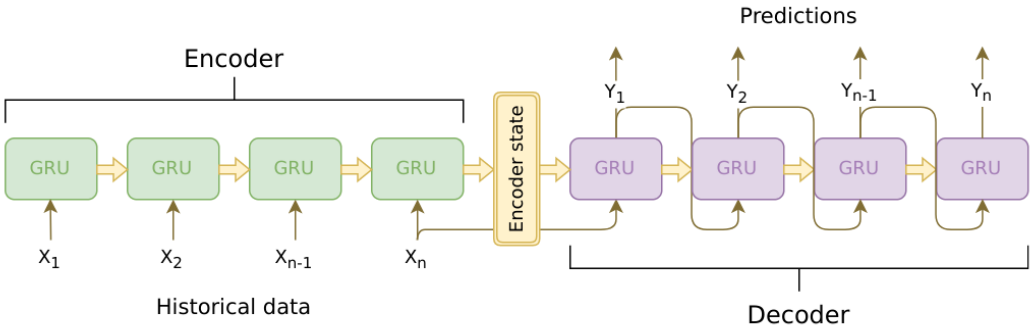


Figure 6: First model considered

This is a Seq2Seq model which has the possibility of taking a variable-length sequence as an input and return a variable-length sequence as an output. It uses Gated Recurrent Units (GRU) which have a short term memory and have a possibility of forgetting and have less parameters than a Long Short-Term Memory (LSTM) cell. They are efficient on NLP tasks like here. The model was then trained on the *Cornell Movie-Dialogs Corpus* and gave pretty good results.
However, the previous preliminary step allowed me to learn about the types of layers created by X-Cube-AI and thus those that can be used in a model.Unfortunately, the model presented above is too complex to be embedded with the version of X-Cube-AI used: the recurrent layers such as GRU here are processed but not the Embedding layers.

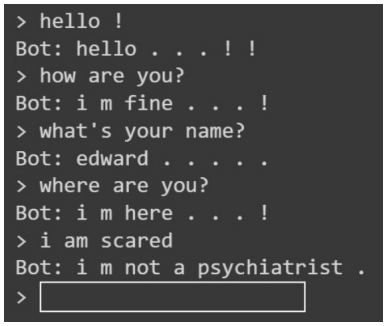


Figure 7: Example of a conversation with the PyTorch chatbot

Another model is then required. All models using a recurrent layer such as an LSTM or a GRU are preceded by an Embedding layer. Since the manual creation of such a layer is too complex and time consuming, a simpler model is required.

4.1 The chatbot model

The previously stated constraints then lead to the choice of the simpler model below, composed only of fully connected layers [5]. It is built with Tensorflow 2.0.0 and Keras 2.3.1. These versions (earlier than the current ones) are required to allow further processing by X-Cube-AI.

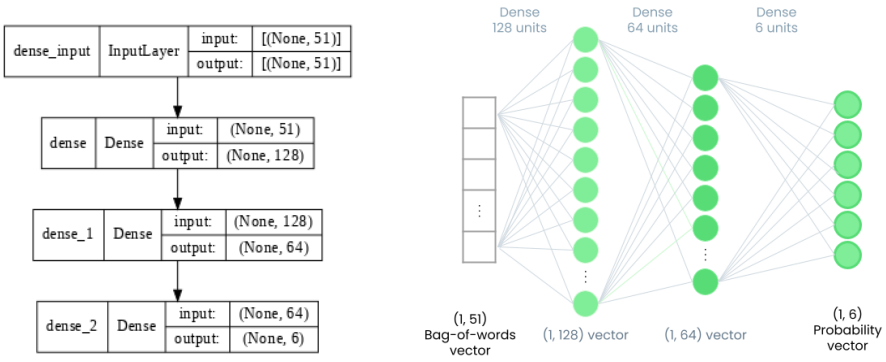


Figure 8: Model architecture

Training data

The training data are grouped in a dictionary forming a JSON file of intentions. The advantage of this format is the possibility of enriching the dictionary according to the desired application: in the case studied, sentences such as "Is the coffee machine full?" or "Where is the vacuum cleaner?" can be introduced with potential answers.

Dictionaries are labeled with a tag, classifying the type of information carried by the user's words, such as the greeting tag:

```
{"tag": "greeting",
 "patterns": ["Hello", "La forme?", "yo", "Salut", "ça roule?"],
 "responses": ["Salut à toi!", "Hello", "Comment vas tu?", "Salutations!", "Enchanté"],
},
```

From this dictionary, a list of meaningful words is extracted before the model. The objective of the model is then to predict the class of the sentence sent by the user of the tool.

What happens precisely

In order to find the suitable answer to the user's sentence, it is necessary to encode it. To do that, a reference words list is created from the JSON file with all the meaningful words. Then, the bag-of-words method is used in order to output a similarity vector (the same length as the reference list) between the sentence and the reference list. This will allow to predict a class and choose a response in the dictionary.

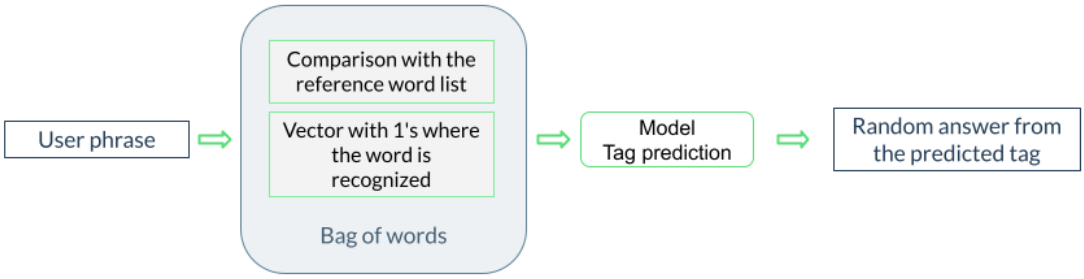


Figure 9: How the response process works

4.2 Embedding of the chatbot on the electronic board and adjustments

Embedding on the board is done in the same way as for the simple sine prediction model.

The difficulty here lies in the processing of the user's message in order to be processed by the model. Indeed, the model takes as input a vector composed of 0 and 1 representing the sentence, and returns a vector of probability of belonging to the classes. It is therefore necessary to introduce, in the C project on CubeIDE generated by X-Cube-AI:

- the pre-processing:
  - creation of the list of meaningful words
  - bag-of-words encoding of the message
- the post-processing:
  - getting the maximum probability and the corresponding tag
  - choosing a random answer from the appropriate ones

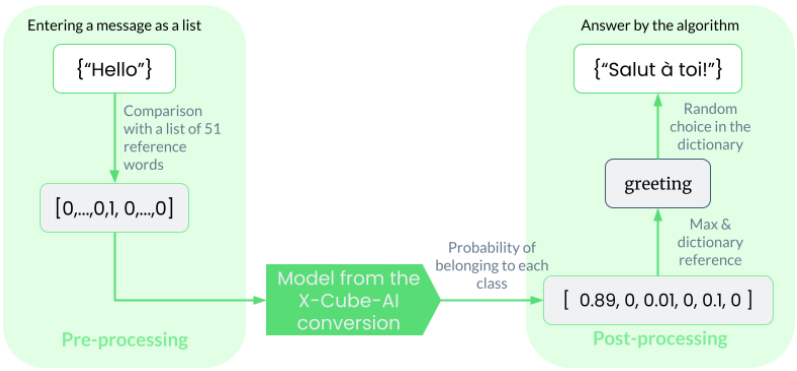


Figure 10: Pre and post-processing to be done in C on the board

The data processing then requires the switching of the data dictionary from Python to C language in order to allow the correct running of the response process.

The model performs well on STM32: it runs and is able to provide an answer to the user. However, it always predict the same class for the user's input message, which leads to an answer always of the same class whatever the initial subject. The input message is encoded correctly to enter the model. To continue, we would need to determine if the problem is with the conversion of the template or the code to retrieve its output.

Furthermore, the benefit of this simple model is the low consumption of its operation: about 4% RAM usage and 14% of the flash memory.

5 Reflection on ethics & environmental impact

It would be interesting to question the ethical question raised by this project in order to validate its real usefulness without harming in one way or another.

The project presented here responds to a real need: to help people with disabilities to carry out their daily tasks, every day and at any time.

It is important to check that the tool developed does not require too many resources, on the calculation servers for example.

In this case, the model is trained only once on GPU, during its design. Once the model is trained, it is implemented on a card like the STM32 as developed previously. There, it will only perform a calculation on the card itself and will never need resources such as a GPU. It would be ideal to study the performance of the embedded chatbot, more precisely the energy required, in order to reduce energy consumption but also to avoid the user having to change the batteries too regularly. The low memory resource required to run the model is a first good point.

6 Conclusion

This project is interested in artificial intelligence, but especially in the possibility of its embedding, a current and increasingly popular problem. The embedding of a trained model must be done in an embeddable language, unlike the model trained in python language most often, but also face various constraints that artificial intelligence on GPU is exempt from.

The challenge of boarding the model was met, despite a small final flaw. To continue, it would be wise to solve the last problem raised, namely to determine whether the prediction problem comes from the conversion of the model or from the code that retrieves its output. The chatbot would then be fully functional.

Moreover, the chatbot project is part of a useful project to help the population and could be pursued in particular thanks to the enrichment of the dictionary (addition of examples, more targeted vocabulary), a complexification of the model allowing better performances and finally, why not a display of the answer on the screen of the STM32 card.

Another very rewarding prospect would be to succeed in using the first model considered, with recurrent cells such as GRUs, through adjustments in order to have a more robust model, resistant to misspellings or able to answer on a larger field of possibility. The choice of the simplicity of the model was however in line with the main objective of embedding the model on the map, more than in the study of the robustness of this one taking into account the time allowed.

Bibliography

[1] Shawn Hymel. *Intro to TinyML Part 1: Training a Model for Arduino in TensorFlow*. URL: https://www.digikey.com/en/maker/projects/intro-to-tinyml-part-1-training-a-model-for-arduino-in-tensorflow/8f1fc8c0b83d417ab521c48864d2a8ec (visited on 11/2020).

[2] ST. *AI expansion pack for STM32CubeMX*. URL: https://www.st.com/en/embedded-software/x-cube-ai.html (visited on 10/2021).

[3] Elisa DELHOMMÉ. *CHATBOT*. URL: https://github.com/elisadelh/CHATBOT.

[4] Matthew Inkawhich. *Chatbot Tutorial*. URL: https://pytorch.org/tutorials/beginner/chatbot_tutorial.html#define-models (visited on 10/2021).

[5] Kurtis Pykes. *Build A Simple Chatbot In Python With Deep Learning*. URL: https://towardsdatascience.com/a-simple-chatbot-in-python-with-deep-learning-3e8669997758 (visited on 12/2021).