NeMo and Rasa Application in a Chatbot Environment

Andrea Gurioli, Giovanni Pietrucci, Mario Sessa

Progetto di Natural Language Processing, A.A. 2021/22

andrea.gurioli2@studio.unibo.it, giovanni.pietrucci@studio.unibo.it, mario.sessa@studio.unibo.it

0000984711 - 0000985556 - 0000983529

Abstract: The document aims to explain the infrastructure used in building a personal virtual assistant with vocal speech recognition and synthesizer applied on different modules provided by NVIDIA NeMo and Rasa. We can propose different models to obtain an intelligent agent interactable with a final user. We will discuss the task domain of the built chatbot, called Alysia, the architecture behind her and the synthesis of models used to obtain some sub-task of the system. We can divide the documentation into three different parts: Automatic Speech Recognition (ASR), Natural Language Understanding (NLU) and Text to Speech procedures (TTS).

1. Introduction

NVIDIA NeMo is a conversational AI toolkit built for researchers working on automatic speech recognition (ASR), natural language processing (NLP), and text-to-speech synthesis (TTS). The primary objective of NeMo is to help researchers from industry and academia to reuse prior work (code and pretrained models) and make it easier to create new conversational AI models. With NeMo, we can build models for real-time automated speech recognition (ASR), natural language processing (NLP), and text-to-speech (TTS) applications such as video call transcriptions, intelligent video assistants, and automated call centre support across healthcare, finance, retail, and telecommunications. Our paper will not concentrate only on the NVIDIA NeMo framework applications; NVIDIA NeMo does not guarantee an intent system that can detect a specific type of a sentence using a well-trained NLU model and gives a specific response sentence type. Integration with Rasa can guarantee this function. Rasa is an open-source project which supplies the building blocks for creating virtual assistants. Use Rasa to automate human-to-computer interactions anywhere from websites to social media platforms. It provides a conversation of raw text from user messages into structured data. Parse the user's intent and extract critical key details. Furthermore, Rasa's machine learning-powered dialogue management decides what the assistant should do next, based on the user's message and context from the conversation. It guarantees a built-in integration point for over ten messaging channels and endpoints connecting with databases, APIs, and other data sources. [5]

2. Automatic Speech Recognition

Automatic Speech Recognition or ASR, as it's known in short, is the technology that allows human beings to use their voices to speak with a computer interface in a way that, in its most sophisticated variations, resembles normal human conversation. Actually, it is subject of research and only in the current decade there is relevant improvement of the performances. Automatic Speech Recognition is consider the first part of our project which identify the input management for the vocal system to obtain a good text translation in english language. We can't guarantee an efficient model built by scratch considering the huge amount of data that this kind of procedure needs to obtain considerable accuracy. So, we will introduce the use of an open-source project made by NVIDIA called NeMo ASR. It is a suite of modules entirely dedicated to Automatic Speech Recognition which everyone can use directly from downloading pre-trained models from the NVIDIA Cloud. In this part, we will introduce one of these models which is applied in our practical approach on Alysia. Most state-of-the-art (SOTA) ASR models are extremely large; they tend to have on the order of a few hundred million parameters. This makes them hard to deploy on a large scale given current limitations of devices on the edge. Quartznet model consists of 79 layers and has a total of 18.9 million parameters, with five blocks that repeat fifteen times plus four additional convolutional layers. The model is composed of multiple blocks with residual connections between them, trained with CTC loss. Each block consists of one or more modules with 1D time-channel separable convolutional layers, batch normalization, and ReLU layers. Model achieves near state-of-the-art accuracy on LibriSpeech and Wall Street Journal, while having fewer parameters than all competing models. Neural Modules (NeMo) toolkit makes it easy to use this model for transfer learning or fine tuning. Encoder and Decoder checkpoints trained with NeMo can be

used for fine tuning on new datasets. Dataset used here is the LibriSpeech training dataset with two types of data augmentation techniques: Speed Perturbation and Cutout. Speed Perturbation means additional training samples were created by slowing down or speeding up the original audio data by 10%. Cutout refers to randomly masking out small rectangles out of the spectrogram input as a regularization technique. [1]

2.1. Automatic Speech Recognition Pipeline

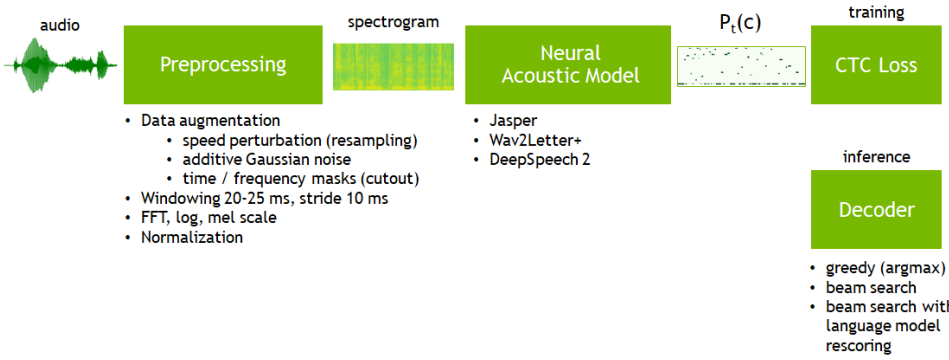

Fig. 1. Automatic Speech Recognition Pipeline Modules

In Figure 1, we can see the Pipeline begins with the Pre-Processing of an Audio File in .wav format. In this part, we apply data augmentation techniques like Speed Perturbation, Additive Gaussian Noise and Cutout for data regularization. Then, we separate audio waves in sections with small intervals using Windowing. Finally, the model applies a Fast Fourier Transform that computes a Discrete Fourier transform (DFT) and converts an audio signal in its frequency representation which is the input of some normalization techniques that guarantee a good Spectogram visualization. [2]

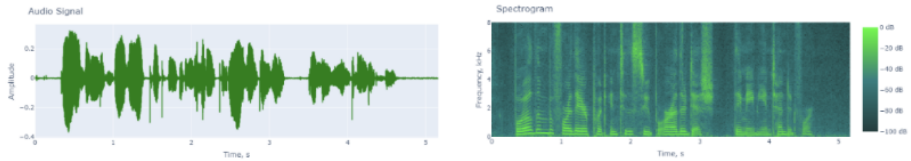

Fig. 2. Original Audio Wave on the left and the Spectogram in output of Pre-Processing on the right

In Figure 2, we can see an example of the input and output of an audio file for the speech ”Boil them before they are put into the soup or other dish they may be intended for”. A spectrogram is generally a visual way of representing the signal strength, or ”loudness”, of a signal over time at various frequencies present in a particular waveform. So, it is a detailed view of audio, representing time, frequency, and amplitude on one graph. Models like Jasper or DeepSpeech 2 from Nemo Models can detect the frequencies and intensities changing, spell some words on the training phase, and discover some design patterns using a spectrogram in input. The generalization of these models can detect a word with a different timbre and tonality thanks to the regularization techniques. As we can observe from Figure 1, we can adopt a mel scale technique in the pre-processing. The mel scale is a scale of pitches that human hearing generally perceives to be equidistant from each other. So, we can define a mel spectrogram to simulate the same perceptions of the human earings and receive an approximated amount of data, as well as our ears do in connection with our brain. As frequency increases, the interval, in hertz, between mel scale values (or simply mels) increases. The name mel derives from melody and indicates that the scale is based on pitch differences. The linear audio spectrogram is ideally suited for applications where all frequencies have equal importance. In contrast, as our case, mel spectrograms are better suited for applications that need to model human hearing perception such as voice dictations systems, vocal analysis, or vocal assistant input management. Generally, mel spectrogram data is suited for use in audio classification applications. A mel spectrogram differs from a linearly scaled audio spectrogram in two ways:

1. A mel spectrogram logarithmically renders frequencies above a certain threshold (the corner frequency). For example, in the linearly scaled spectrogram, the vertical space between 1,000 and 2,000Hz is half the

vertical space between 2,000Hz and 4,000Hz. In the mel spectrogram, the space between those ranges is approximately the same. This scaling is analogous to human hearing, where we find it easier to distinguish between similar low-frequency sounds than similar high-frequency sounds.

2. A mel spectrogram computes its output by multiplying frequency-domain values by a filter bank.

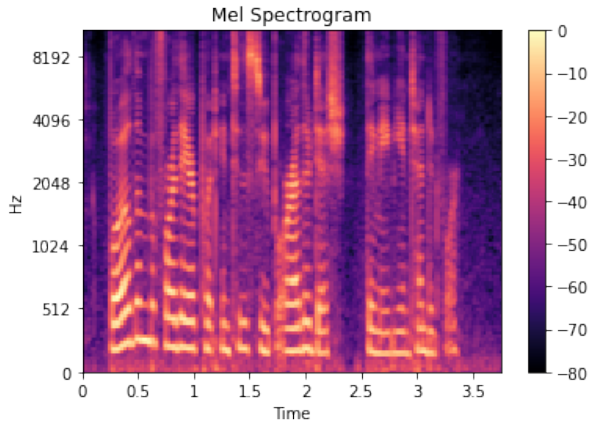

Fig. 3. Mel Spectogram Visualization

After generating the spectrogram, an ASR model receives it as input. The output equals a probabilistic characters diagram that can identify a part of speech from the spectrogram. Unfortunately, we do not know how the characters in the transcript audio align with the audio. It makes training a speech recognizer harder than it might at first seem. That is because it is much harder to adapt every audio file to specific rules maintaining a high level of variability. The solution adopted to the NeMo ASR pipeline and many ASR models which follow the same pattern is the Connectionist Temporal Classification (CTC).

2.1.1. Connectionist Temporal Classification

Connectionist Temporal Classification (CTC) is a type of Neural Network output helpful in tackling sequence problems like handwriting and speech recognition where the timing varies. Using CTC ensures that one does not need an aligned dataset, making the training process more straightforward. A simple token prediction would output a list of tokens that need to be collapsed. That is because a simple model cannot define when a specific character starts or ends correctly. Furthermore, we should avoid collapsing when there are repetitions of the same character type in the sequence. That behaviour is impossible to do respectively for both cases. Within the Connectionist Temporal Classification, the token prediction is extended with a blank token to align the results and obtain an accurate collapsing technique. Using this algorithm, we still predict one token per speech segment, but we use the blank token to determine when the model predictions collapse. In the case of repeating letters, the appearance of a blank token helps their separation, and it should avoid false-positive collapses. CTC Loss is a training procedure representing a milestone in many model configurations in speech recognition domains. CTC loss is in every configuration of NVIDIA NeMo Encoders-Decoders models for Automatic Speech Recognition.

2.1.2. ASR Language Modeling

During the inference phase, one problem that we perceive and usually present on the Automatic Speech Recognition models is the probability of failing a transcription using a different character. Generally, we can apply some techniques of ASR Language Modeling to improve the accuracy of the transcription and minimize the Word Error Rate (WER). NeMo supports the following two approaches to incorporate language models into the ASR models: N-gram language modelling and neural re-scoring. During the project development, we did not have the necessary to apply both methods for our ASR model because the application performance about the speech recognition of vocal chat is acceptable and, integrating with the Beam search using the N-gram language modelling, we obtained an optimal improvement. N-gram LM is trained on text data; then, it is combined with beam search decoding to find the best candidates. The beam search decoders in NeMo support language models trained with KenLM library. An N-gram LM can be used with beam search decoders on the ASR models to produce more accurate candidates. The beam search decoder would incorporate the scores produced by the N-gram LM into its score calculations as the following:

$$S(\theta) = \phi(\theta) + \alpha_b(\theta) \times lm(\theta) + \beta_b \times len(\theta)$$

where θ is the input sequence, $\phi(\theta)$ is the acoustic score which is predicted by the acoustic encoder and $lm(\theta)$ is the one estimated by the LM. Parameter α_b specifies amount of importance to place on the N-gram language model, and β_b is a penalty term to consider the sequence length in the scores. Larger α_b means more importance on the LM and less importance on the acoustic model. Negative values for β_b will give penalty to longer sequences and make the decoder to prefer shorter predictions, while positive values would result in longer candidates.

2.2. *Quartznet*

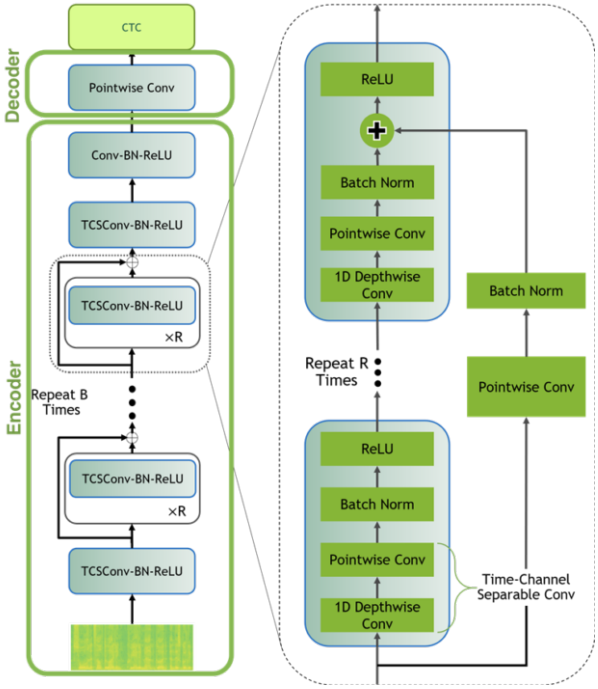

Fig. 4. Quartznet15x3 Model Provided by NeMo ASR Modules

QuartzNet's design is based on the Jasper architecture, which is a convolutional model trained with Connectionist Temporal Classification (CTC) loss [3]. The main novelty in QuartzNet's architecture is that we replaced the 1D convolutions with 1D time-channel separable convolutions, an implementation of depthwise separable convolutions. 1D time-channel separable convolutions can be separated into a 1D depthwise convolutional layer with kernel length K that operates on each channel individually but across K time frames and a pointwise convolutional layer that operates on each time frame independently but across all channels. We can describe the Quartznet layers as follow:

1. Initial layer is a 1D convolutational layer C_1 with strides of 2 followed by a sequence of blocks.
2. Each block B_i is repeated S_i times with residual connections between blocks. Each block has a same baseline module which is repeated R_i times, it contains the following layers:
   (a) A K-sized depthwise convolutional [4] with C_γ channels. Depthwise convolutional is a type of convolution where we apply a single convolutional filter for each input channel. In other words, split the input in different channel and apply the respective filter C_i related to a specific channel i. Then, we stack the convolved outputs together.
   (b) A pointwise convolutional layer [6] which is a type of convolution that uses a 1x1 kernel: a kernel that iterates through every single point. This kernel has a depth of however many channels the input text has. It can be used in conjunction with depthwise convolutions to produce an efficient class of convolutions known as depthwise-separable convolutions.
   (c) A normalization layer which normalize the previous output in an independent batch size.
   (d) ReLU layer which represents the single block module output following the ReLu function.

3. Last part consists in the combination of 3 convolutional layers sequentially with the last one with a dilation rate equals to 2.

Quartznet model was proposed by NVIDIA in 2019 [4], and it is available in the NVIDIA Cloud as a pre-trained Encoder-Decoder model importable in NeMo modules. It is possible to choose different Quartznet configurations based on the number of repeatable blocks B_i and some repeatable baseline modules R_i for each block. The most common configuration has 15 blocks with five repetitions (Quartznet15x5).

2.3. Quartznet Evaluation

The principal metric when it comes to evaluating the Automatic Speech Recognition model is the Word Error Rate (WER) which produces a result related to the substitutions, deletions and correct words values predicted by the ASR:

$$Word\ Error\ Rate = 100 \times \frac{I_n + S_n + D_n}{C_n}$$

Given n the total amount of words of an audio wave in input, to the summary between the number of insertions I_n, a number of substitutions S_n and a number of deletions D_n inversely proportioned to the correct number of words in the transcriptions C_n. The result is evaluated in percentage. The QuartzNet model has a 3.90% value of WER considering the integration with Beam search. Given the less complexity of words usually used to interact with Alysia and the good performances of the Quartznet model, we did not integrate the second available language modelling technique to improve the general performances of the speech recognition.

3. Rasa Framework

Rasa is a tool to build custom AI chatbots using Python and natural language understanding (NLU). Rasa provides a framework for developing AI chatbots that uses natural language understanding (NLU). It also allows the user to train the model and add custom actions. Chatbots built using Rasa deployed on multiple platforms like Facebook Messanger, Microsoft chatbot and other services.

Rasa has two main components:

1. **Rasa NLU (Natural Language Understanding):** Rasa NLU is an open-source natural language processing tool for intent classification (decides what the user is asking), extraction of the entity from the bot in the form of structured data, and helping the chatbot understand what user is saying.

2. **Rasa Core:** a chatbot framework with machine learning-based dialogue management that takes the structured input from the NLU and predicts the following best action using a probabilistic model like LSTM neural network rather than if/else statement. Underneath the hood, it also uses reinforcement learning to improve the prediction of the following best action.

In other words, the Rasa NLU's job is to interpret the input provided by the user in the form of structured data, and Rasa Core's job is to decide the next set of actions performed by the chatbot. Rasa Core and Rasa NLU are independent and can be used separately. [5] In our project, Rasa represents the core of the NLU modules, which defines the intent management to define the following response of the bot and, analyzing the type, can guarantee a related action to perform inside the back-end system. Unfortunately, developing a chatbot with only NeMo framework is a bit hard; that is because NeMo is structured as a task-oriented service that can guarantee optimal performances using their models but, in the case of chatbot building, it does not provide an intent system which is necessary to build a basic chatbot like Alysia.

3.1. Rasa Architecture

This section will analyze the procedure of interest that defines Rasa usage inside the proposed project and the framework's architecture. Rasa's architecture is modular by design. This feature allows easy integration with other systems. For example, Rasa Core can be used as a dialogue manager in conjunction with NLU services other than Rasa NLU. While the code is implemented in Python, both services can expose HTTP APIs to be used easily by projects using other programming languages. We use its HTTP APIs to communicate with a Rasa server deployed in a local host with a current NLU model programmed by scratch using YAML configurations models. Our server is listening on a docker network that can communicate with an external server that provides the forwarding of requests and responses from the front-end. The architecture is based on a dialogue state saved in a tracker object. There is one tracker object per conversation session, which is the only stateful component in the system. There is

also a tracker stores slots and a log of all the events that led to that state and have occurred within a conversation, and the state of a conversation can be reconstructed by replaying all of the events. We can define the procedure behind the architecture in a step-by-step schema using Figure 5.

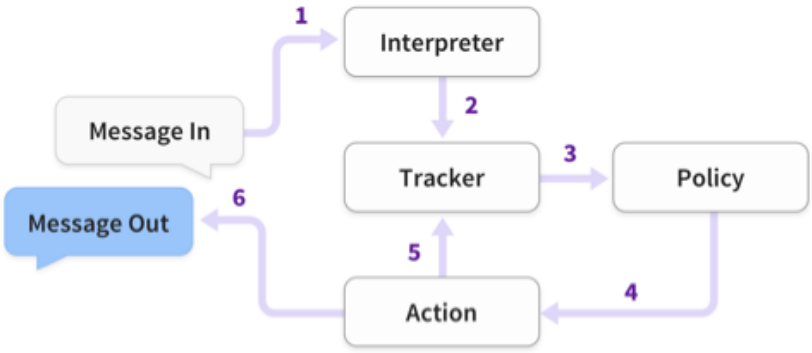

Fig. 5. High-level Architecture Provided by Rasa

Rasa is based on the following steps during the message processing from the front-end:

1. A message is received and passed to an Interpreter (e.g. Rasa NLU) to extract the intent, entities, and any other structured information.
2. The Tracker maintains a conversation state, which is unique for a session. It receives a notification that a new message has been received and prepares the system for processing.
3. The policy receives the current state of the tracker related to the session of the message using the HTTP header.
4. The policy chooses which action to take following processes the intents of the models and defines a level of confidence with the chosen action with the highest value.
5. The tracker logs the chosen action. So, the tracker defines the response type of the Rasa framework to the input message.
6. The action is executed (this may include sending a message to the user). During this step, we can perform a return value using the utter class, which defines a related response message to the input type message previously received by the client.
7. If the predicted action is not 'listen', go back to step 3.

Generally, actions can perform different responses, which should be a simple string text in response to a particular procedure like sending email, changing persistent data in an external database (Rasa provides different configurations to connect its actions modules with external resources like MongoDB or Cassandra Database).

3.2. Rasa Elements Management

This section will introduce some main elements provided by Rasa to manage data and perform previous steps. The first thing to take care of when setting up Rasa is the intent. Intents are the agent's core; they model the basics of the conversation by setting up the response for each question sent by the user. The goal of the Natural Language Understanding (NLU) Model is to predict which intent is going to be triggered. When an intent is triggered, the response from the training model is given to the user. When it comes to specific intents which trigger an action, an entity from the question could be needed. Entities are words extracted by the user's question useful for action purposes. For example, when adding an element for the user's TO-DO list, the question could be 'Add milk to my list. The entity 'milk' will be extracted by the CRF Entity Extractor and given to the chatbot's database to handle the persistence. The last main element for the Rasa Chatbot's architecture is the rules. When it comes to driving a specific type of conversation in a unidirectional way, a rule can be settled, and when the first intent of the rule is triggered, the conversation will be directed to the following intents.

3.3. *Natural Language Processing Pipeline*

The intent prediction and entity extraction are handled by the NLU module, which executes a pipeline based on the tokenization of the question, the featurization of the tokens and the prediction of the intent/entities. For the testing phase, we compared Sparse (Bag of words technique) and Dense (Word embedding) word vectors used on the DIET Classifier, which can handle both intents and entities with an inner CRF Entities Extractor. The comparison

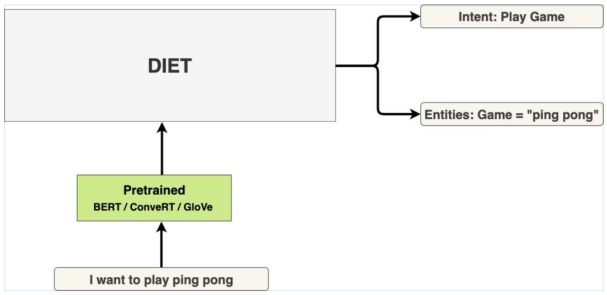

Fig. 6. DIET Classifier For Words Embedding Procedure

is made by settling the model's F1 score, recall, and precision. All the tests are repeated three times because of the not utterly deterministic training phase. The results are given with the actual mean value and the standard deviation from the mean.

Vectors	F1	Recall	Precision
Sparse vectors	0.92 ($\pm$0.05)	**0.90**($\pm$0.03)	**0.90**($\pm$0.06)
Dense vectors	0.89($\pm$0.04)	**0.91**($\pm$0.03)	0.90($\pm$0.05)

As we can see, the results with the sparse vectors are slightly better; this can be caused by the small number of intents that were settled for the model itself. Under these circumstances, we adopted the Sparse vector-based configuration model.

3.4. *Project Tasks and Management*

The proposed project implements different tasks using the NLU modules from Rasa deployed on a model in a Rasa Server instance. Unfortunately, we decided to migrate actions procedures in the external back-end server for performance reasons, so we did not use Rasa Actions Server.

3.4.1. Alysia Task

Alysia provides different tasks similar to the Amazon Alexa setup. We aim to apply some persistent tasks like making a list using external services (MongoDB in our case) or defining a logic status like sending an email procedure that interests a series of intent triggers that defines a story. Implemented tasks are:

- **To-Do List Management:** This can define a To-Do list used as a reminder, but we did not implement it using an alert system connected to a precise timestep like a regular calendar.
- **Sending Email:** Alysia guarantees an email service in which a final user can interact to write or talk about the content of an email. These tasks represent the logic status management example, in which a user can first define the receiver mail then write or perform vocal dictation for the body of the email. This service provides a default email subject and sender email, configured as a Gmail secure email using TTS/SSL protocol on an SMTP channel.
- **Weather System:** Alysia can call a Weather API to perform sky description and environment's temperature about a specific city, which is defined by an entity in the NLU Rasa model.
- **Time Service:** It is a simple service that takes in the output of the current time related to the deployed server.
- **Wikipedia Search:** Defining this task, we tested Rasa integration with external services and NVIDIA NeMo NLP models performances. It uses a BERT-based model provided by NVIDIA NeMo framework trained on Quad 1.0 datasets which dynamically takes an input, provides to search Wikipedia pages using a web scraper based on the part-to-speech technique on the input words, create a JSON with a context defined by our scraper and the question provided by the user input and, finally, submit it to the model to gives in output a response.

3.5. *BERT and Wikipedia Search System*

In this section, we want to detail the Wikipedia Search System. The task's goal of the lookup is to give the correct answer to a user question. In order to achieve this goal, we implemented a BERT system fine-tuned for the SQuAD task given by the NeMo NLP Framework. The language representation model BERT, which stands for Bidirectional Encoder Representations from Transformers, is designed to pre-train deep bidirectional representations from an unlabeled text by jointly conditioning both left and proper context in all layers. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks [7]. The SQuAD fine-tuned BERT can detect the answer's starting point and the ending

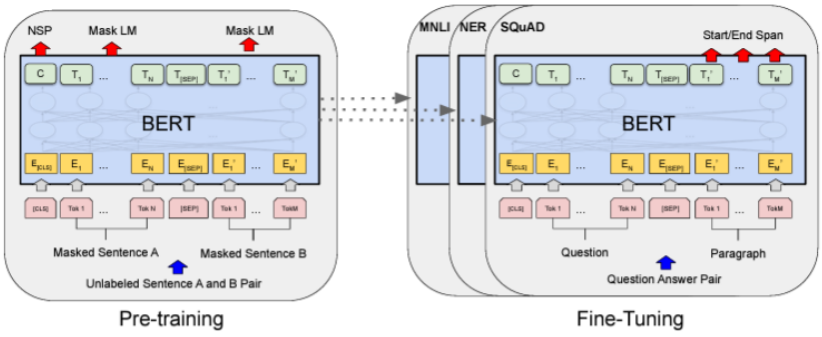

Fig. 7. Deployment Diagram of BERT Model

point in a predefined article. In order to do it, the model needs the question and the article in which it will look for. Having a proper way to search the answer in the article, now, the focus of the problem drifts on the web lookup for the article itself. In order to do that, we circumscribed the problem in Wikipedia's domain. For a meaningful lookup, the question extracted by the entities extractor is processed by applying the Part of Speech tagging and taking the more valuable parts of the question for a Wikipedia lookup. The main parts taken by the pre-processing of the question are pronouns, adjectives, nouns, numeral values and adverbs. After the pre-processing phase, we use the extracted line for the wiki's summary to obtain the summary of Wikipedia's article about the words previously processed. Then the summary is used as an answer for the BERT input. If the initial summary's lookup does not match for an article, Wikipedia's search function is used to obtain a list of correlated subjects. Then we pick up the first element of the list as the input's answer. The NVIDIA NeMo Squad-based NLP model framework returns as output directly the answer to the question.

4. Text To Speech

Speech Synthesis or Text-to-Speech (TTS) involves turning text into human speech. The NeMo TTS collection currently supports two pipelines for TTS:

1. **Two stage pipeline:** First, a model is used to generate a mel spectrogram from a text; called a spectrogram generator. Second, a model is used to generate audio from a mel spectrogram, called vocoder.

2. **The "end-to-end" approach:** That uses one model to generate audio straight from the text.

About the proposed project, the pipeline used to interest the first typology. Combining different models, we could test their performances and choose the best combination. During this phase, we chooses **FastPitch Spectogram Generator** and **HiFinGAN Vocoder**. Unfortunately, these models work well in GPU TPU based systems for flow parallelization provided by CUDA. CUDA is a platform (architecture, programming model, assembly virtual machine, compilation tools...), not just a single programming language. CUDA C is just one of a number of language systems built on this platform (CUDA C, C++, CUDA Fortran, PyCUDA, are others). Furthermore, every model proposed by NeMo uses the Torch tensors. That is because the NeMo toolkit is based on PyTorch Lightning library, a sample library based on Torch and optimized for TPU made by Facebook AI Research in collaboration with academic teams. We will talk about NeMo models about TTS procedures provided by NVIDIA, but, in practice, we use the default Speech Syntethize of Javascript to perform in a no complex environment and guarantee accessibility to no GPU well-performed devices.

4.1. *Spectogram Generator*

The Spectrogram Generator class has two essential functions: A parsing procedure that accepts raw python strings and returns a Torch tensor that represents tokenized text ready to pass to, and a spectrogram generation that accepts a batch of tokenized text and returns a Torch tensor that represents a batch of spectrograms. The spectrogram Generator used provided by NVIDIA NeMo is English FastPitch. FastPitch is a fully-parallel text-to-speech model based on FastSpeech, conditioned on fundamental frequency contours. The model predicts pitch contours during inference. By altering these predictions, the generated speech can be more expressive, better match the semantic of the utterance, and in the end, more engaging to the listener. FastPitch is based on a fully-parallel Transformer architecture, with a much higher real-time factor than Tacotron2 for the mel-spectrogram synthesis of a typical utterance. This model is trained on LJSpeech sampled at 22050Hz, and has been tested on generating female English voices with an American accent. FastPitch is a full feedforward Transformer model that predicts mel spectrograms from raw text. The entire process is parallel, which means that all input letters are processed simultaneously to produce a full mel-spectrogram in a single forward pass. [8] Figure 8 describes the FastPitch Architecture pro-

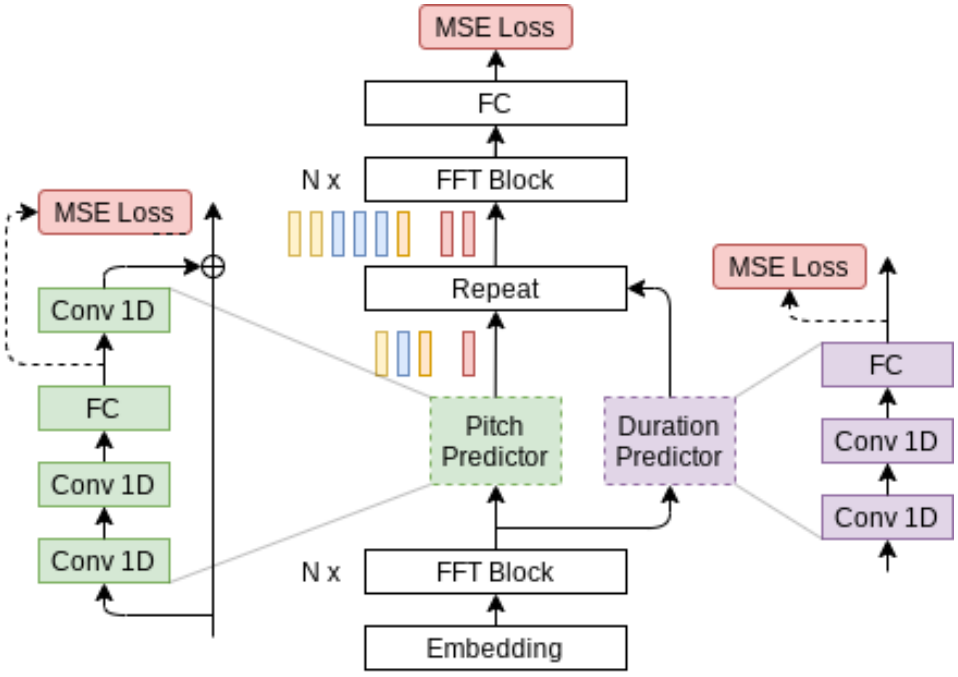

Fig. 8. FastPitch Architecture

posed by Adrian Łancucki [8]. The model comprises a bidirectional Transformer backbone (also known as a Transformer encoder), a pitch predictor, and a duration predictor, which respectively defines a pitch and a duration value of the melody generated in the spectrogram. After passing through the first *n* Transformer blocks, encoding, the signal is augmented with pitch information and discretely upsampled. Then it goes through another set of *n* Transformer blocks to smooth out the upsampled signal and construct a mel spectrogram. It is good to specify that the Transformer blocks computes in parallel and guarantee a fast generation of a well-accurate mel spectrogram. As introduced in the previous section, FastPitch supports multi-GPU and mixed precision training with dynamic loss scaling provided in a library like Apex. During our experiments, we did not use an environment of this type. So, we did not have any dependencies with Apex.

4.2. *HiFiGAN Vocoder*

HiFiGAN is a generative adversarial network (GAN) model that generates audio from mel spectrograms. The generator uses transposed convolutions to upsample mel spectrograms to audio. This model is trained on LJSpeech sampled at 22050Hz, and has been tested on generating female English voices with an American accent. [9] Its architecture consists of one generator and two discriminators: which are multi-scale and multi-period discriminators. The generator and discriminators are trained adversarially, along with two additional losses for improving training stability and model performance. We can resume the model as follow:

1. **Generator:** It is a fully connected convolutional neural network that gives in input the mel spectrogram and upsamples in transported convolutions until the length of the output sequence matches with the temporal

resolution of a raw waveform. Every transported convolution has a multi-receptive field fusion (MFS)

2. **Multi-Receptive Field Fusion:** It observes the different patterns in parallel concerning the various length of the raw waveform. It returns the sum of the multiple residual blocks output. To adapt the matching of a specific length, we left some adjustable parameters in the Generator; the hidden dimension h_u, kernel sizes k_u of the transposed convolutions, kernel sizes k_r, and dilation rates D_r of MRF modules can be regulated to match one's requirement in a trade-off between synthesis efficiency and sample quality.

3. **Discrimators:** They are GANs that identifies long-term dependencies is the key for modelling realistic speech audio. HiFiGAN uses two different discriminators:

   (a) **Multi-Period Discriminator:** It is composed of sub-discriminators with a stack of convolutional layers with different stride values and terminates with a ReLu layer. This discriminator aims to find implicit structure in the audio input considering different period values for each sub-discriminators to avoid the overlap analysis.

   (b) **Multi-Scale Discriminator** It is another GAN introduced in the MelGAN model [10]. It is a mixture of 3 sub-discriminators. Their composition is a stack of strode and grouped convolutional layers with ReLu as an activation function to discover hidden patterns on different parts of raw audio files.

   This model is one of the current state-of-art architecture of the Speech synthesis procedure. One of the main features of HiFiGAN is the optimal performance in terms of synthesis speed which defines an excellent point to justify his usage in real-time applications like our chatbot.

5. Project Architecture

In this section, we presents the final architecture of the proposed project. Introducing Automatic Speech Recognition and Text-To-Speech pipelines, used NeMo models and Rasa NLU configurations, we can assemble each part in a well-formatted and efficient architecture. Figure 9. shows the communication with differents model presented

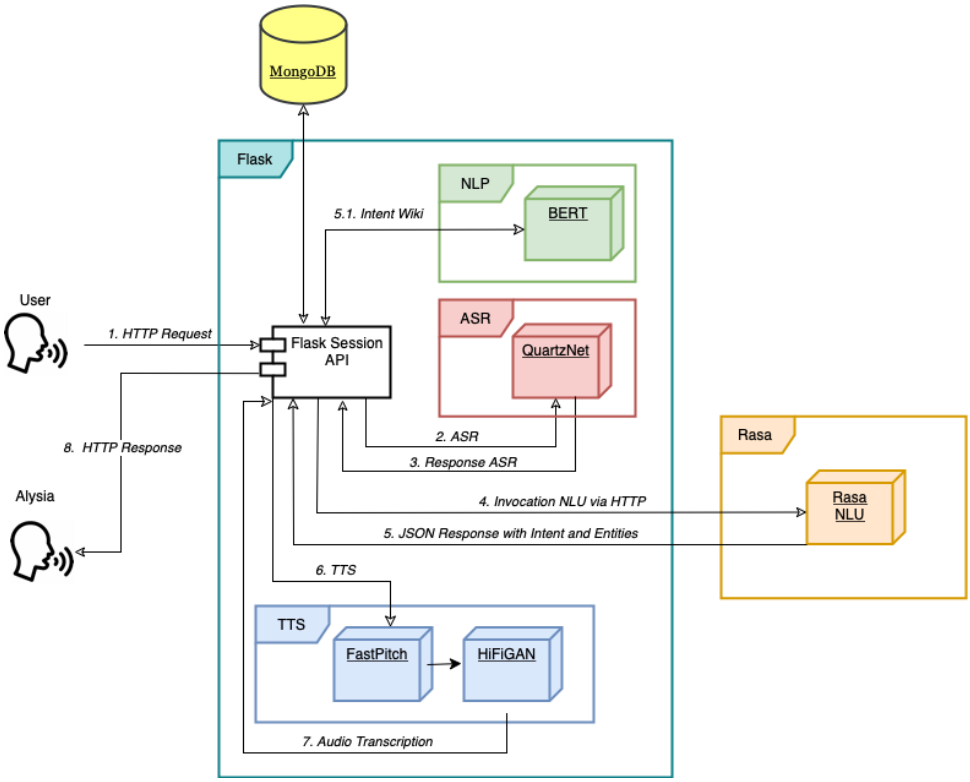

Fig. 9. Deployment Diagram of the Proposed Project

in the final chatbot. Alysia's architecture is based on a client-server design pattern. Principal components are NeMo models about Automatic Speech Recognition and Text-To-Speech recently described and even one Natural Language Processing model based on the BERT structure for the Wikipedia Search as defined in section 3.4.1.; Alysia

provides a writing and vocal communication thanks to two buttons in the main interface of the application. Differences between the two functionalities are the Text-to-Speech usage. In the writing communication TTS modules are not invoked. A positive aspects about the TTS is the irrelevant differences in terms of speed between writing and vocal communication. This aspect is possible thanks by the HiFiGAN model which has optimal performance in synthesize speed. NeMo models are deployed on a Flask Server which load from the web the pre-trained models using NeMo APIs. In the architecture, we use a second server: Rasa Server. It is deployed on a Docker container with an Rasa NLU model for the intent management. Flask and Rasa communicate directly in HTTP. In the final architecture, Flask invokes Rasa's parse function to define an intent given a question from the front-end. If the intent chosen by the highest confidency is Wiki, Flask invokes a BERT model provided by NVIDIA NeMo to manage the search engine on Wikipedia and gives in ouput a response of a general-pourpose question. If the intent is different from the Wikipedia's type, Flask performs a dispatched procedure to execute some actions related to the intent semantic. An external Mongo database stores persistent data about the To-Do tasks options, remaining tasks works with session data, external API or third-protocols like Weather APIs or SMTP for the email sending. Javascript front-end prepare data visualization or audio playing to let users interact with Alysia in a real-time environment.

6. Conclusions

NeMo addresses many issues often encountered in developing DL applications by transferring best practices from software engineering. It operates with a higher-level abstraction, the neural module, and introduces a neural type system capable of semantic checks. It also comes with collections of pre-built modules for conversational AI for Automatic Speech Recognition, Natural Language Processing and Text-to-Speech using state-of-art configurations. [11] Furthermore, its integration with external conversational frameworks like Rasa, NVIDIA Riva or Google Dialogflow lets us use it for many different applications. Adaptive configurations, easy deployment and high abstraction, are the three main features. In recent years, NVIDIA has integrated NeMo with new features and incorporates the NeMo toolkit in NVIDIA Jarvis and NVIDIA Riva. These innovative frameworks improve performances of NeMo, simplify configurations and installation setup and improve performances on GPU and TPU usage. In conclusion, we can affirm that NeMo could be a good starting point for research in conversational AI optimization and, as said in the NeMo official paper, the NVIDIA AI Research section will guarantee in the following years additional improvements.

References

1. *Automatic Speech Recognition: A Deep Learning Approach, Dong Yu, Li Deng, Springer Edition, 2015*
2. *Audio Augmentation for Speech Recognition, Tom Ko, Vijayaditya Peddint2, Daniel Povey, Sanjeev Khudanpur, 2015*
3. *Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks." Graves, Alex et al., 2006*
4. *QuartzNet: Deep Automatic Speech Recognition with 1D Time-Channel Separable Convolutions, Samuel Kriman, Stanislav Beliaev, Boris Ginsburg, Jocelyn Huang, Oleksii Kuchaiev, Vitaly Lavrukhin, Ryan Leary, Jason Li, Yang Zhang, 2019*
5. *Rasa: Open Source Language Understanding and Dialogue Management, Tom Bocklisch, Joey Faulkner, Nick Pawlowski, Alan Nichol, 2017*
6. *Pointwise Convolutional Neural Networks, Binh-Son Hua and Minh-Khoi Tran and Sai-Kit Yeung, 2018*
7. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova, 2019*
8. *FastPitch: Parallel Text-to-speech with Pitch Prediction, Adrian Łańcucki, 2021*
9. *HiFi-GAN: Generative Adversarial Networks for Efficient and High Fidelity Speech Synthesis, Jungil Kong, Jaehyeon Kim, Jaekyoung Bae, 2020*
10. *MelGAN: Generative Adversarial Networks for Conditional Waveform Synthesis, Kundan Kumar, Rithesh Kumar, Thibault de Boissiere, Lucas Gestin, Wei Zhen Teoh, Jose Sotelo, Alexandre de Brebisson, Yoshua Bengio, Aaron Courville, 2019*
11. *NeMo: a toolkit for building AI applications using Neural Modules, Oleksii Kuchaiev, Jason Li, Huyen Nguyen, Oleksii Hrinchuk, Ryan Leary, Boris Ginsburg, Samuel Kriman, Stanislav Beliaev, Vitaly Lavrukhin, Jack Cook, Patrice Castonguay, Mariya Popova, Jocelyn Huang, Jonathan M. Cohen. Santa Clara, NVIDIA, 2019*