An Introduction to Building Incremental Dialogue Systems in Python

Thilo Michael, Dr. Maike Paetzel-Prüsmann,
Dr. Jana Götze, Prof. Dr. David Schlangen

Technische Universität Berlin
Universität Potsdam

Organizers

Thilo Michael

PhD Candidate at TU Berlin

Maike Paetzel-Prüsmann

Postdoctoral Researcher at University of Potsdam

Jana Götze

Postdoctoral Researcher at University of Potsdam

David Schlangen

Professor at University of Potsdam

Participants

What's it about?

This tutorial is for you if...

- ... you want to get a (deeper) understanding what incremental processing is
- ... you already know what incremental processing is, but you never found a good way to integrate it into your system

After the tutorial you will...

- ... understand how the retico python framework works, what it can do and what features we are still working on
- ... have built your first application in the retico python framework!

Demo Time!

Agenda

09:00 - 09:15	Welcome & Introduction
09:15 - 10:00	retico - An Overview of Incremental Processing and the Framework
10:00 - 10:30	Coffee Break
10:30 - 11:15	Hands-On 1: Building a First Dialogue System (Guided Programming) & Extending the System (Solo Work with Individual Help)
11:15 - 11:50	Hands-On 2: Creating New Modules & Incremental Units (Guided Programming) & Advancing the Dialogue System (Solo Work with Individual Help)
11:50 - 12:00	Advanced Features, Future Work & Farewell

Incremental Processing

A quick intro & some background

What retico is...

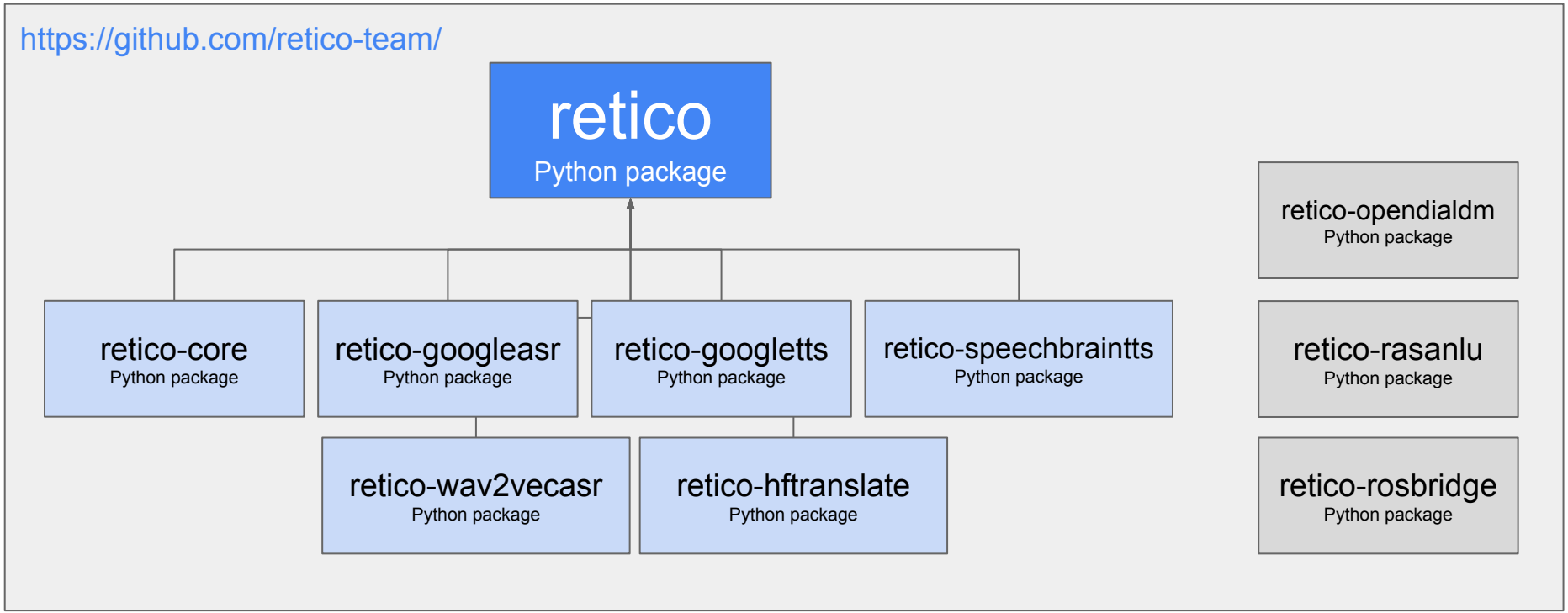
- A framework for incremental dialogue written in python
- A collection of state-of-the-art incremental modules
- An interface for interconnecting incremental systems
- A research tool that is under active development

What retico is not...

- A tool for building dialogue systems (like IBM Watson or Google Dialogflow)
- A finished production-ready project

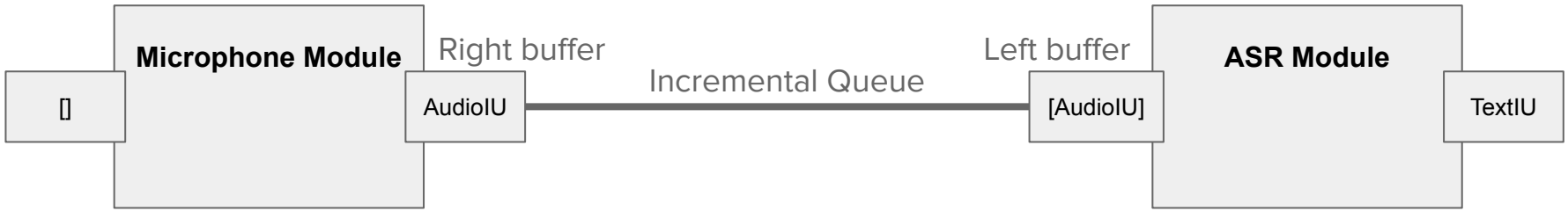
The structure of the retico project

```
from retico import *
```



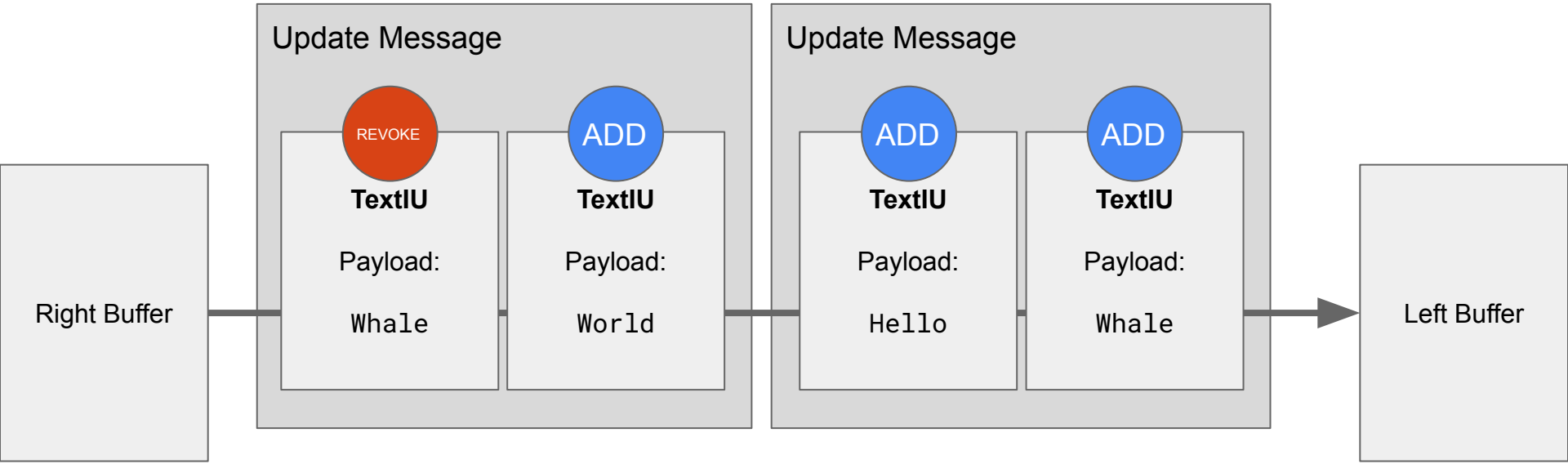
Incremental modules

- All incremental modules inherit from `AbstractIncrementalModule`
- Every module declares a **list of input IU types**, one **output IU type** a **name** and a **description**
- They are connected with the `subscribe` method
  e.g., `microphone_module.subscribe(asr_module)`



Incremental units

- **Incremental units** carry the increments to the connected incremental modules
- Multiple IUs are packaged into **Update Messages**
- Every Incremental Unit has an **Update Type** attached



Creating incremental networks

- Connect the output of a module with the input of another with **`subscribe()`**
- Start the execution of a module with **`run()`**
- Stop the execution of a module with **`stop()`**
- Each incremental module runs in its own *thread*, that means modules run in parallel!

```
from retico import *

modul1 = MyModule1()
modul2 = MyModule2()

modul1.subscribe(modul2)

modul1.run()
modul2.run()

print("Network is running")
input()

modul1.stop()
modul2.stop()
```

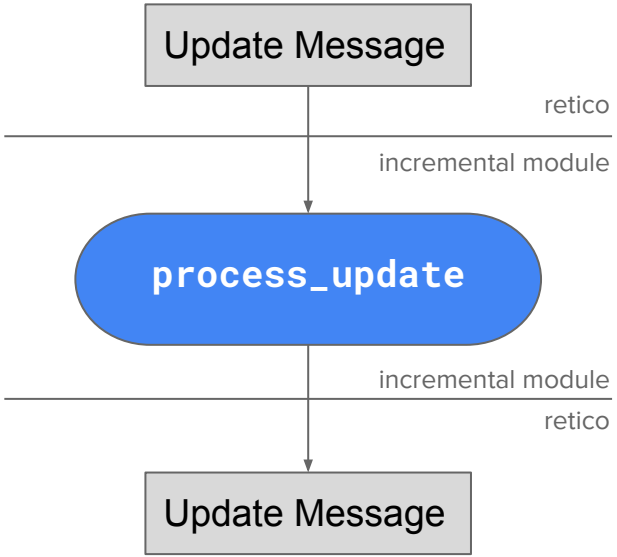
Creating incremental modules

- An incremental module inherits from **`AbstractModule`**
- A module always implements the 4 methods **`name`**, **`description`**, **`input_ius`**, and **`output_iu`**
- An incremental module can process None, 1, or many input_iu types
- An incremental module can process None or exactly 1 output IU type

```
from retico import *

class MyFirstModule(AbstractModule):
    @staticmethod
    def name():
        return "My First Module"

    @staticmethod
    def description():
        return "This module does nothing in particular."

    @staticmethod
    def input_ius():
        return [ius.TextIU, ius.AudioIU, ius.DialogueActIU]

    @staticmethod
    def output_iu():
        return ius.TextIU
```

The process_update method

- Gets called by retico whenever there is a new update message available
- Is responsible for keeping track of the state of the **input**
- Is responsible for producing **new hypotheses**
- Is responsible for keeping track of the state of the **output**
- May return a new update message, containing IUs to be **added, updated, revoked, or committed**



What we need

- Python3, at least 3.10
- The portaudio library
  - For MacOS: **`brew install portaudio ffmpeg`**
  - For Linux (debian-based):
    ```
    sudo apt-get install libasound-dev portaudio19-dev libportaudio2 libportaudiocpp0 ffmpeg
    ```
  - For Windows: Download and install **ffmpeg**
- To install retico:

  ```
  pip3 install --upgrade retico
  ```

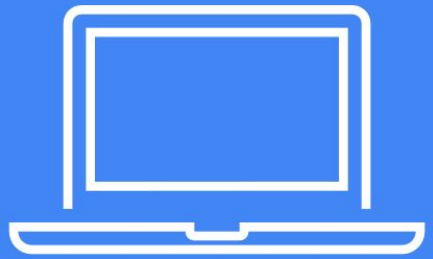
- To check if it worked:

  ```
  import retico
  print(retico.__version__)
  ```

Coffee Break

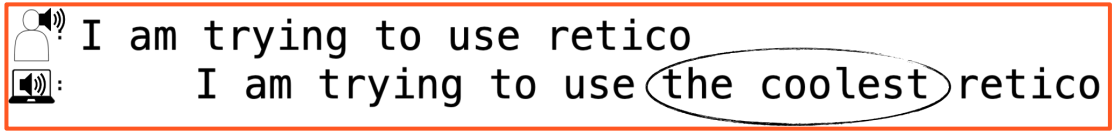
Next up: Hands-On Session
Start: 10:30

Hands-on Part I

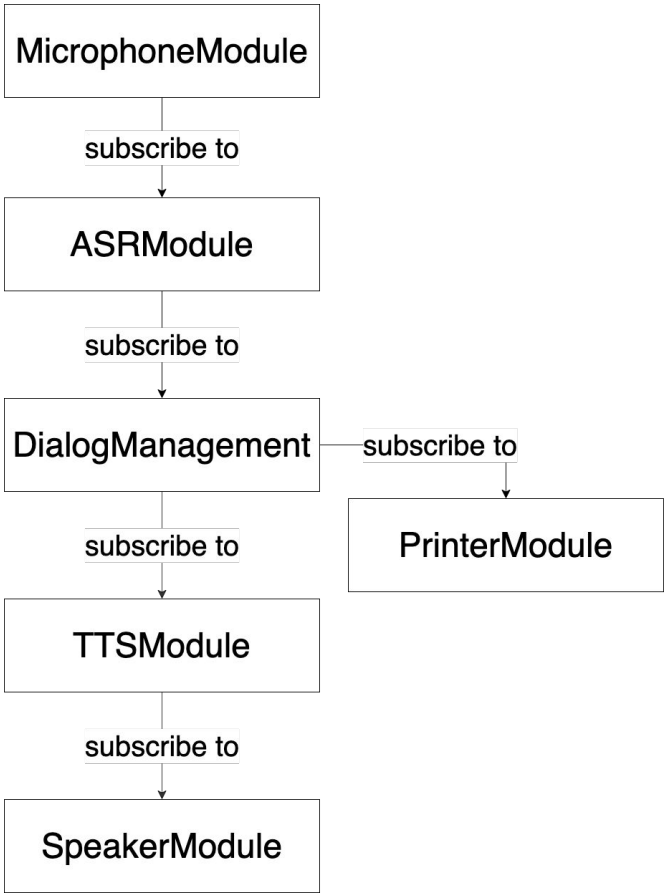


What we'll build

- Network that takes speech as input, replaces/inserts some words, and outputs the result as speech.

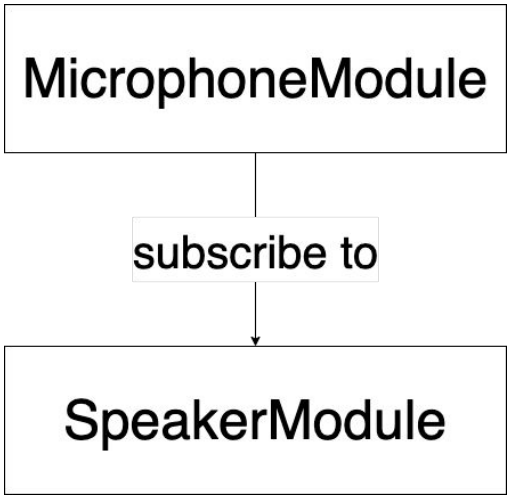


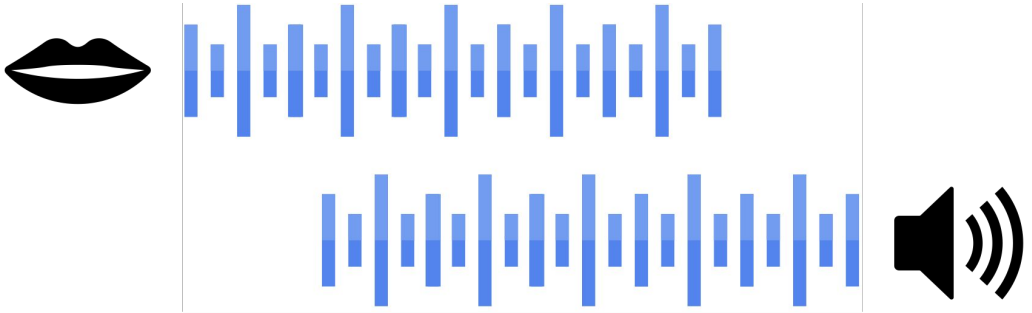
- We will print ASR results to the screen for debugging purposes.
- All the steps: https://github.com/retico-team/konvens-tutorial



Step 1 – Speech in and out

- The first application will be an echo, taking in speech and incrementally playing it back.
- We need two modules:
  one for the input speech stream, one for the output stream.





Step 1 – Speech in and out

- In your favorite editor or IDE: retico_network.py

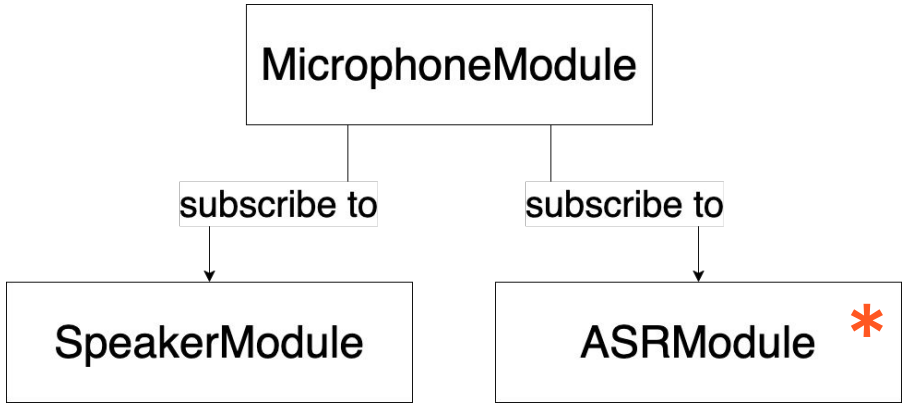
```
1  from retico import *
2
3  microphone = modules.MicrophoneModule()
4  speaker = modules.SpeakerModule()
5
6  microphone.subscribe(speaker)
7
8  run(microphone)
9
10 print("Running the network. Press enter to exit")
11 input()
12
13 stop(microphone)
```

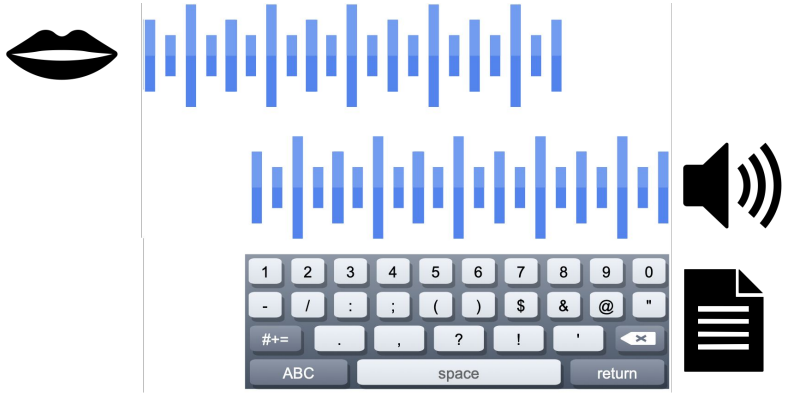
This network can cause a feedback loop, as you output exactly what the microphone records!

Use headphones!!

Step 2 – Speech in and out. And also ASR

- The second application will still do the echo, and will also perform speech recognition.
- We need three modules: one for the input speech stream, one for the output stream, and one for the speech recognition.





Step 2 – Speech in and out. And also ASR

```
1  from retico import *
2
3  microphone = modules.MicrophoneModule()
4  asr = modules.Wav2VecASRModule("en")
5  speaker = modules.SpeakerModule()
6
7  printer_module = modules.TextPrinterModule()
8  asr.subscribe(printer_module)
9
10 microphone.subscribe(speaker)
11 microphone.subscribe(asr)
12
13 run(microphone)
14
15 print("Running the network. Press enter to exit")
16 input()
```

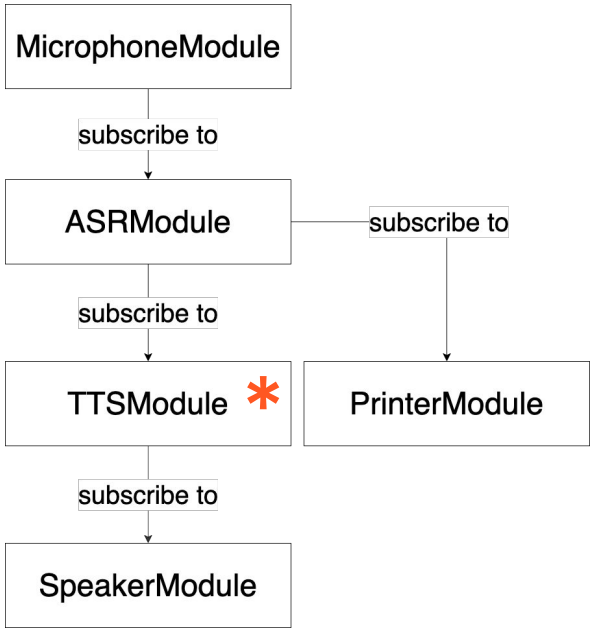
Create asr object for English speech recognition

We need an extra helper module to print ASR results to the terminal.

Add ASR to the network

Step 3 – Adding speech synthesis

- Instead of just playing back the incoming speech, we'll synthesize what the ASR recognized.



Step 3 – Adding speech synthesis

```
1  from retico import *
2
3  microphone = modules.MicrophoneModule(rate=16000)
4  asr = modules.Wav2VecASRModule("en")
5  tts = modules.SpeechBrainTTSModule("en")
6  speaker = modules.SpeakerModule(rate=22050)
7
8  printer_module = modules.TextPrinterModule()
9  asr.subscribe(printer_module)
10
11 # Build the new network
12 # microphone.subscribe(speaker)
13 microphone.subscribe(asr)
14 asr.subscribe(tts)
15 tts.subscribe(speaker)
16
17 run(microphone)
18 print("Running the network. Press enter to exit")
19 input()
20 stop(microphone)
```

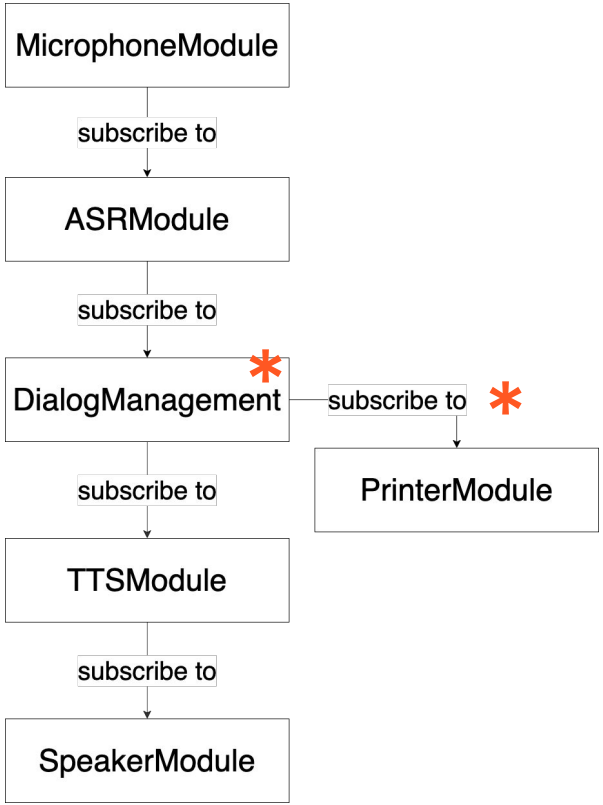
Create the TTS object and set the speech rate for the speaker output

Adjust the network

Step 4 – Adding simple dialog management

- Let’s change some words before synthesis.
- Writing your own module.

- A module that takes text and returns text, replacing every instance of “you” with “I” (or anything else you want to replace).



Step 4 – Adding simple dialog management

```
37  microphone = modules.MicrophoneModule(rate=16000)
38  asr = modules.Wav2VecASRModule(language="en")
39  wordchanger = WordChangerModule(word_map={"you": "I"})
40  printer = modules.TextPrinterModule()
41  tts = modules.SpeechBrainTTSModule(language="en")
42  speaker = modules.SpeakerModule(rate=22050)
43
44  microphone.subscribe(asr)
45  asr.subscribe(wordchanger)
46  wordchanger.subscribe(printer)
47  wordchanger.subscribe(tts)
48  tts.subscribe(speaker)
```

We'll insert a module that takes a mapping as input, specifying what to change.

We'll move the printer module to after the change.

Step 4 – Writing your own module

```
4  class WordChangerModule(AbstractModule):
5      @staticmethod
6      def input_ius():
7          return [ius.TextIU]
8
9      @staticmethod
10     def output_iu():
11         return ius.TextIU
12
13     def __init__(self, word_map, **kwargs):
14         super().__init__(**kwargs)
15         self.word_map = word_map
```

We'll extend the AbstractModule class.

We want text as input and output incremental units.

And we'll add the word replacement mapping as attribute.

Step 4 – Writing your own module

```
def process_update(self, update_message):
    new_update_message = UpdateMessage()
    for incremental_unit, update_type in update_message:
        # replace strings here

        new_update_message.add_iu(incremental_unit, update_type)
    return new_update_message
```

We need to implement this method that takes the update_message and returns a new update message

Step 4 – Writing your own module

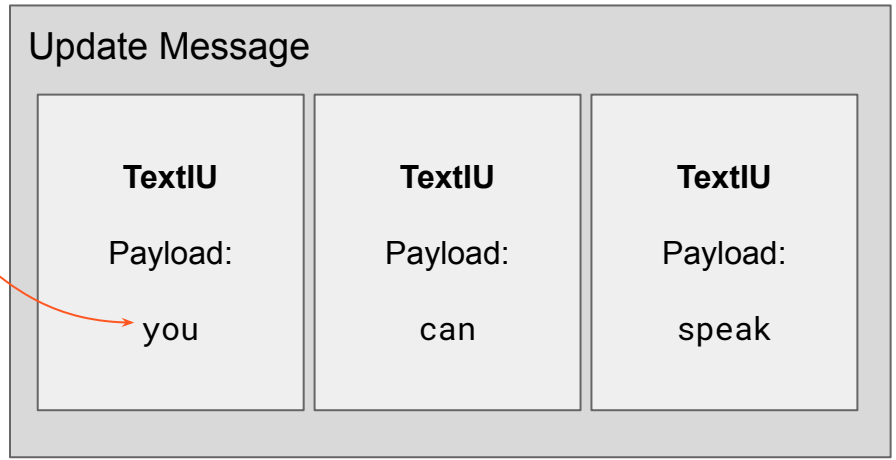
```
def process_update(self, update_message):
    new_update_message = UpdateMessage()
    for incremental_unit, update_type in update_message:
        # replace strings here

        new_update_message.add_iu(incremental_unit, update_type)
    return new_update_message
```

We need to implement this method that takes the update_message and returns a new update message

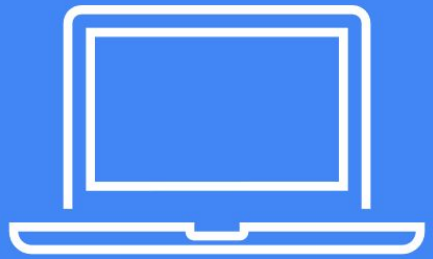
- `incremental_unit` is of type TextIU
- TextIU Payloads are named `.text`

`incremental_unit.text`

- Our replacement dictionary is called `self.word_map`

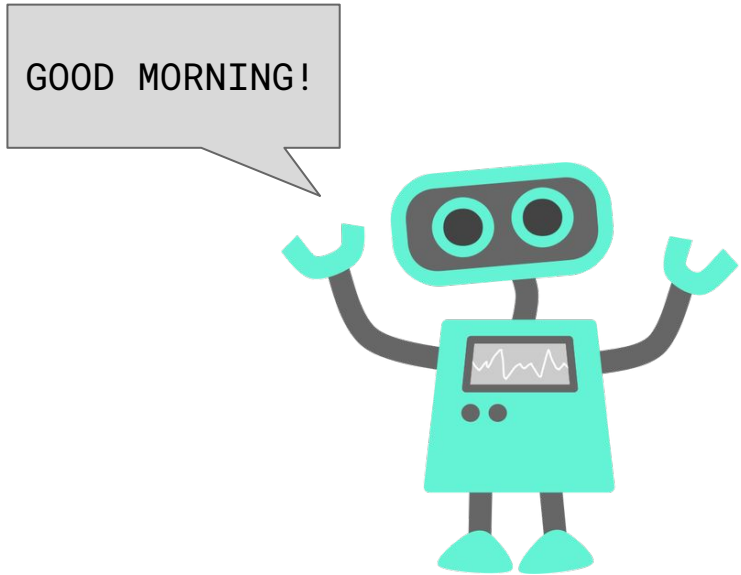


Hands-on Part II



What we'll build

A translation module based on a hugging face transformer network that incrementally translates between different languages



Step 1: HF Translator

- Retico has a helper class called **HFTranslate** that translates non-incrementally between different languages!

```
from retico import *

translator = helper.HFTranslate(from_lang="en",to_lang="de")
print(translator.translate("Hello KONVENS, this is a test."))
```

- Make yourself comfortable with this helper class!

- English to French
- French to English
- English to German
- German to English
- Spanish to English
- English to Spanish
- French to German
- German to French

Step 2: Creating the incremental module

- Input and output will be **`TextIU`**s

- In the initializer we instantiate the HFTranslate module

```
from retico import *

class TranslationModule(AbstractModule):
    @staticmethod
    def name():
        return "Translation Module from the KONVENS 2022 retico tutorial"

    @staticmethod
    def description():
        return "A module that translates between languages."

    @staticmethod
    def input_ius():
        return [ius.TextIU]

    @staticmethod
    def output_iu():
        return ius.TextIU

    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.translator = helper.HFTranslate("en", "de")
```

Step 3: Handling incoming units

```
def process_update(self, update_message):
    for incremental_unit, update_type in update_message:
        if update_type == UpdateType.ADD:
            self.current_input.append(incremental_unit)
        elif update_type == UpdateType.REVOKE:
            self.revoke(incremental_unit)
        elif update_type == UpdateType.COMMIT:
            self.commit(incremental_unit)
```

- **`current_input`** is a list of IUs that are currently being used to form a new output hypothesis (in this case the translation).
- **`revoke()`** flags the incremental unit as revoked and deletes it from the **`current_input`** list
- **`commit()`** flags the incremental unit as committed but does *not* delete it from the **`current_input`** list

Step 4: Constructing the current text

```
current_text = " ".join([iu.text for iu in self.current_input])
current_translation = self.translator.translate(current_text)
```

- The **`current_text`** is constructed by joining together all words in the **`current_input`** list
- With the translator, the complete text is translated into the target language
- **`current_translation`** is the complete translation and needs to be incrementalized depending on the IUs that were already published by the module

Step 5: Turning translations into increments

Example:

current_output

TextIU **The**

TextIU **quick**

TextIU **bright**

Current_translation

The quick brown fox

Expected output:

REVOKE

TextIU **bright**

ADD

TextIU **brown**

ADD

TextIU **fox**

Step 5: Turning translations into increments

```
update_msg, new_tokens = helper.get_text_increment(self, current_translation)
```

Helper function takes as arguments

- the module that contains the current output IUs (stored in **`current_output`**)
- the current translation from which the *difference* should be computed

Helper function returns

- update message with IUs that should be **revoked**
- list of new tokens (words) that should be turned into **incremental units** and **added**
- And it removes the revoked functions from the **`current_output`** of the module

Step 6: Adding new tokens

```
for token in new_tokens:
    new_iu = self.create_iu(grounded_in=incremental_unit)
    new_iu.text = token
    self.current_output.append(new_iu)
    update_msg.add_iu(new_iu, UpdateType.ADD)
```

- Create new IU with the **`create_iu`** function
  - Specify the IU that the new one is *grounded in*
- Set the text to be the new token
- Add the IU to the **`current_output`**
- Add the IU to the update message

Step 7: Cleaning when IUs are committed

```
if self.input_committed():
    for iu in self.current_output:
        self.commit(iu)
        update_msg.add_iu(iu, UpdateType.COMMIT)
    self.current_input = []
    self.current_output = []

return update_msg
```

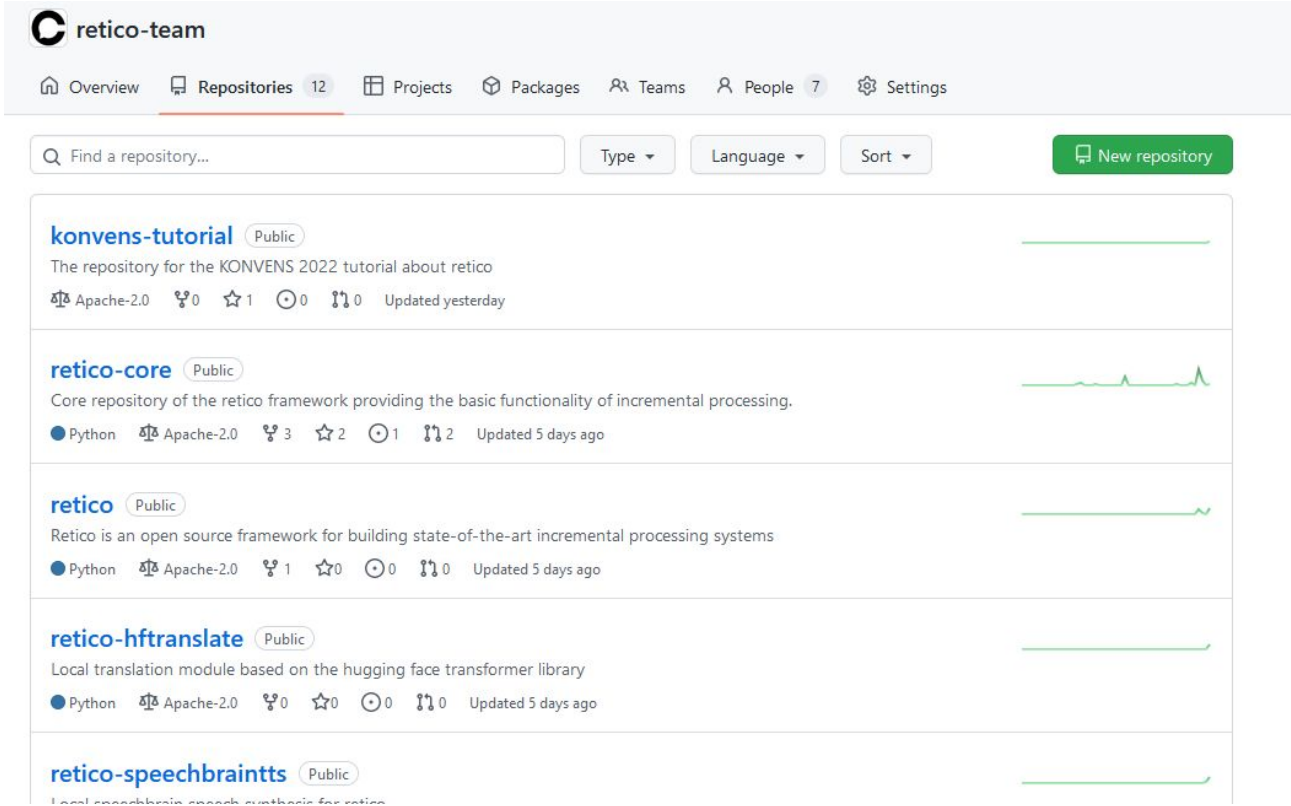
- The **`input_committed`** method checks if all IUs in the **`current_input`** are committed
- All IUs are flagged as committed and added to the update message
- **`current_input`** and **`current_output`** are cleared

Step 8: Creating the network

```
microphone = modules.MicrophoneModule(rate=16000)
asr = modules.Wav2VecASRModule(language="en")
translation = TranslationModule()
printer = modules.TextPrinterModule()

microphone.subscribe(asr)
asr.subscribe(translation)
translation.subscribe(printer)

run(microphone)

print("Network is running")
input()

stop(microphone)
```

Advanced Features & Future Work

retico on GitHub



Advanced Features

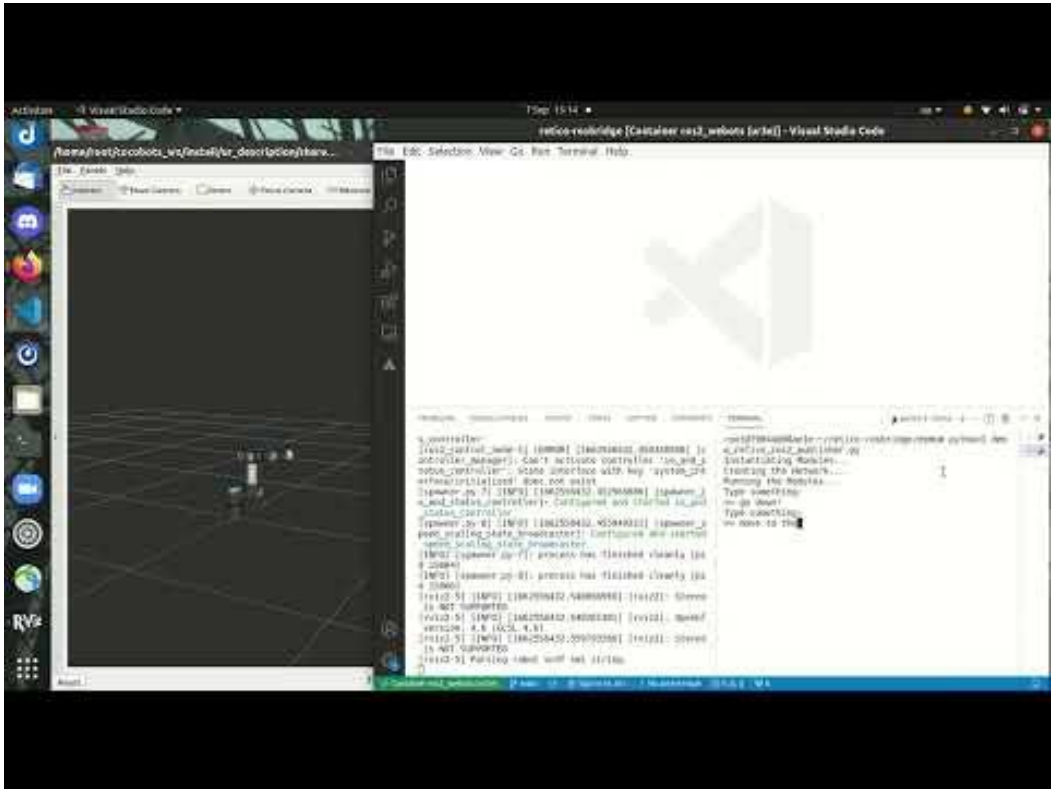
An incremental implementation of Rasa NLU
(Rafla & Kennington, 2019)

Google ASR and TTS System

ZeroMQ for asynchronous messaging between processes

OpenDial Dialogue Management
(Lison & Kennington, 2016)

Integration with Robot Operating System



ROS

Future Plans

Currently under development

- Integration with Microsoft Platform for Situated Intelligence (PSI)
- Adding further sensor input possibilities
  - Webcams
  - RealSense camera
  - Microsoft Kinect
- OpenFace to process camera images

Future Plans

Creating a framework for **teaching** dialogue system development by including

- Tutorials & training videos
- Tasks and sample solutions
- Graphical User Interface

Missing any features?
Have a question?
Need help getting started?

Get in touch on GitHub!

https://github.com/retico-team/retico

Technische Universität Berlin

Universität Potsdam