Riva End-to-end User Guide

spoonnvidia

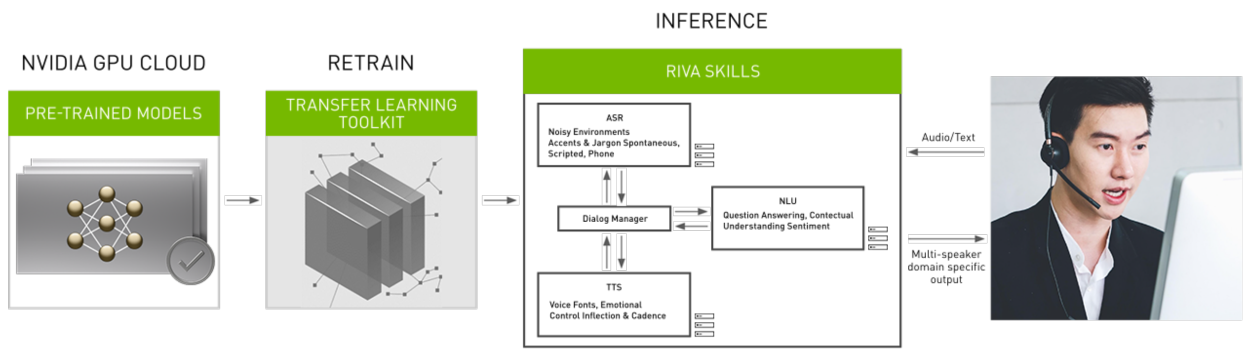
Oct 11, 2021

RIVA

CHAPTER

ONE

RIVA - GETTING STARTED



NVIDIA Riva is a GPU-accelerated SDKs which we can use to build conversational AI applications using deep learning. Riva platform offers power features including

1. Complete set of tools to build conversational AI combining services of speech recognition (ASR), Natural Language Understanding/Processing (NLU/NLP), and text speech_synthesis (TTS)
2. Complete workflow to build, train, and deploy models in Riva using NVIDIA TAO, NVIDIA NeMo.
3. Optimised deployed models to deliver greater performance and latency.
4. Client integration using gRPC API

1.1 Prerequisites

Note: Follow below to launch Riva server.

1. follow CLI Install Guide to set up NGC Catalog API
2. Download Riva quick start scripts

```
ngc registry resource download-version "nvidia/riva/riva_quickstart:1.5.0-beta"
```

3. Initialise and start Riva

```
cd riva_quickstart_v1.5.0-beta
bash riva_init.sh
bash riva_start.sh
```

If `bash riva_start.sh` was successfully executed, we will see

```
Waiting for Riva server to load all models...retrying in 10 seconds
Waiting for Riva server to load all models...retrying in 10 seconds
Waiting for Riva server to load all models...retrying in 10 seconds
Riva server is ready...
```

That's it! Riva server is set up.

Hint: For checking Riva Available services, run `docker logs riva-speech`, and you should see output similar to below outputs:

```
+-----------------------------------------------+---------+--------+
| Model                                         | Version | Status |
+-----------------------------------------------+---------+--------+
| riva-asr                                      | 1       | READY  |
| riva-asr-ctc-decoder-cpu-streaming            | 1       | READY  |
| riva-asr-feature-extractor-streaming          | 1       | READY  |
| riva-asr-voice-activity-detector-ctc-streaming | 1       | READY  |
| riva-trt-riva-asr-am-streaming                | 1       | READY  |
+-----------------------------------------------+---------+--------+
```

Note: Configurations of Riva Server (e.g. models, services, ports) can be configured in `config.sh`. See *Riva - Server Configurations* for more details.

To stop running Riva server, run

```
bash riva_stop.sh
```

1.2 Sample service usage

You might start a container with sample clients for each service

```
bash riva_start_client.sh
```

Try different services using provided notebooks

```
jupyter notebook --ip=0.0.0.0 --allow-root --notebook-dir=/work/notebooks
```

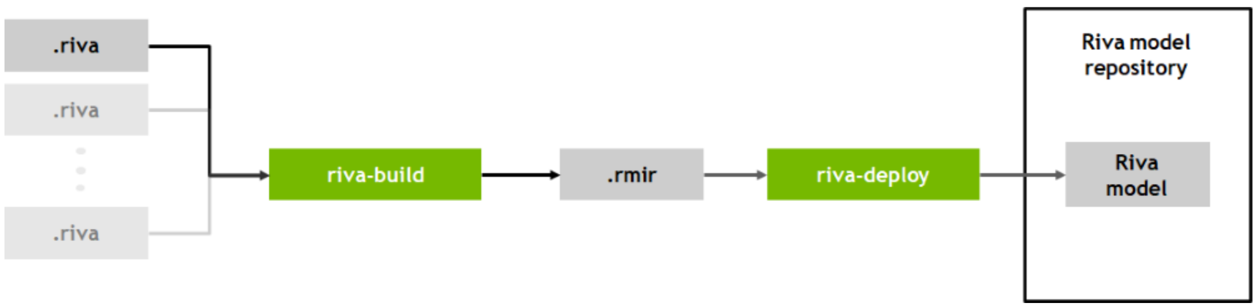
See also:

- NVIDIA Riva documentation
- Stanford CS224N: NLP with Deep Learning

CHAPTER

TWO

RIVA - SERVICEMAKER



After exporting the models using TAO or Nemo, *Riva ServiceMaker* could be used to deploy exported models in Riva server.

Note: See *Riva - Custom Model* for more details of exporting custom AI models in `.riva` formats for deployment.

2.1 riva-build

`riva-build` tool takes 1 or more `.riva` models as input and outputs a `.rmir` (intermediate format) file.

Syntax of `riva-build`

```
riva-build <task-name> --decoder_type=<decoder> output-dir-for-rmir/model.rmir:key dir-for-riva/model.riva:key
```

Table 1: Currently supported tasks (Riva services) by riva-build tool

Tasks	Corresponding services
`speech_recognition`	ASR
`speech_synthesis`	TTS
`qa`	Question answering
`token_classification`	Token level classification
`intent_slot`	Joint intent
`text_classification`	-
`punctuation`	-

Note: See Riva Build for more details.

2.2 riva-deploy

riva-deploy tool takes 1 or more .rmir files and a Riva model repo as inputs.

Syntax of riva-deploy

```
riva-deploy /servicemaker-dev/<rmir_filename>:<encryption_key> /data/models
```

Note: See Riva Deploy for more details.

CHAPTER

THREE

RIVA - SERVER CONFIGURATIONS

Configure Riva Server properties by modifying `config.sh` from Riva quick start scripts

Note: Refer *Riva - Getting Started* to download Riva Quick Start scripts.

3.1 Sample config.sh

```
# Copyright (c) 2021, NVIDIA CORPORATION.  All rights reserved.
#
# NVIDIA CORPORATION and its licensors retain all intellectual property
# and proprietary rights in and to this software, related documentation
# and any modifications thereto.  Any use, reproduction, disclosure or
# distribution of this software and related documentation without an express
# license agreement from NVIDIA CORPORATION is strictly prohibited.

# Enable or Disable Riva Services
service_enabled_asr=true
service_enabled_nlp=true
service_enabled_tts=true

# Specify one or more GPUs to use
# specifying more than one GPU is currently an experimental feature, and may result in␣
↪undefined behaviours.
gpus_to_use="device=0"

# Specify the encryption key to use to deploy models
MODEL_DEPLOY_KEY="tlt_encode"

# Locations to use for storing models artifacts
#
# If an absolute path is specified, the data will be written to that location
# Otherwise, a docker volume will be used (default).
#
# riva_init.sh will create a `rmir` and `models` directory in the volume or
# path specified.
#
# RMIR ($riva_model_loc/rmir)
# Riva uses an intermediate representation (RMIR) for models
```

(continues on next page)

(continued from previous page)

```
# that are ready to deploy but not yet fully optimized for deployment. Pretrained
# versions can be obtained from NGC (by specifying NGC models below) and will be
# downloaded to $riva_model_loc/rmir by `riva_init.sh`
#
# Custom models produced by NeMo or TLT and prepared using riva-build
# may also be copied manually to this location $(riva_model_loc/rmir).
#
# Models ($riva_model_loc/models)
# During the riva_init process, the RMIR files in $riva_model_loc/rmir
# are inspected and optimized for deployment. The optimized versions are
# stored in $riva_model_loc/models. The riva server exclusively uses these
# optimized versions.
riva_model_loc="riva-model-repo"

# The default RMIRs are downloaded from NGC by default in the above $riva_rmir_loc directory
# If you'd like to skip the download from NGC and use the existing RMIRs in the $riva_rmir_loc
# then set the below $use_existing_rmirs flag to true. You can also deploy your set of custom
# RMIRs by keeping them in the riva_rmir_loc dir and use this quickstart script with the
# below flag to deploy them all together.
use_existing_rmirs=false

# Ports to expose for Riva services
riva_speech_api_port="50051"
riva_vision_api_port="60051"

# NGC orgs
riva_ngc_org="nvidia"
riva_ngc_team="riva"
riva_ngc_image_version="1.4.0-beta"
riva_ngc_model_version="1.4.0-beta"

# Pre-built models listed below will be downloaded from NGC. If models already exist in $riva-rmir
# then models can be commented out to skip download from NGC

########## ASR MODELS ##########

models_asr=(
### Punctuation model
    # "${riva_ngc_org}/${riva_ngc_team}/rmir_nlp_punctuation_bert_base:${riva_ngc_model_version}"

### Citrinet-1024 Streaming w/ CPU decoder, best latency configuration
    # "${riva_ngc_org}/${riva_ngc_team}/rmir_asr_citrinet_1024_asrset1p7_streaming:${riva_ngc_model_version}"

### Citrinet-1024 Streaming w/ CPU decoder, best throughput configuration
#    "${riva_ngc_org}/${riva_ngc_team}/rmir_asr_citrinet_1024_asrset1p7_streaming_throughput:${riva_ngc_model_version}"
```

(continues on next page)

(continued from previous page)

```

### Citrinet-1024 Offline w/ CPU decoder,
    # "${riva_ngc_org}/${riva_ngc_team}/rmir_asr_citrinet_1024_asrset1p7_offline:${riva_ngc_model_version}"

### Jasper Streaming w/ CPU decoder, best latency configuration
#    "${riva_ngc_org}/${riva_ngc_team}/rmir_asr_jasper_english_streaming:${riva_ngc_model_version}"

### Jasper Streaming w/ CPU decoder, best throughput configuration
#    "${riva_ngc_org}/${riva_ngc_team}/rmir_asr_jasper_english_streaming_throughput:${riva_ngc_model_version}"

###  Jasper Offline w/ CPU decoder
#    "${riva_ngc_org}/${riva_ngc_team}/rmir_asr_jasper_english_offline:${riva_ngc_model_version}"

### Quarztnet Streaming w/ CPU decoder, best latency configuration
#    "${riva_ngc_org}/${riva_ngc_team}/rmir_asr_quartznet_english_streaming:${riva_ngc_model_version}"

### Quarztnet Streaming w/ CPU decoder, best throughput configuration
#    "${riva_ngc_org}/${riva_ngc_team}/rmir_asr_quartznet_english_streaming_throughput:${riva_ngc_model_version}"

### Quarztnet Offline w/ CPU decoder
#    "${riva_ngc_org}/${riva_ngc_team}/rmir_asr_quartznet_english_offline:${riva_ngc_model_version}"

### Jasper Streaming w/ GPU decoder, best latency configuration
#    "${riva_ngc_org}/${riva_ngc_team}/rmir_asr_jasper_english_streaming_gpu_decoder:${riva_ngc_model_version}"

### Jasper Streaming w/ GPU decoder, best throughput configuration
#    "${riva_ngc_org}/${riva_ngc_team}/rmir_asr_jasper_english_streaming_throughput_gpu_decoder:${riva_ngc_model_version}"

### Jasper Offline w/ GPU decoder
#    "${riva_ngc_org}/${riva_ngc_team}/rmir_asr_jasper_english_offline_gpu_decoder:${riva_ngc_model_version}"
)

########## NLP MODELS ##########

models_nlp=(
### Bert base Punctuation model
    # "${riva_ngc_org}/${riva_ngc_team}/rmir_nlp_punctuation_bert_base:${riva_ngc_model_version}"

### BERT base Named Entity Recognition model fine-tuned on GMB dataset with class labels LOC, PER, ORG etc.
    # "${riva_ngc_org}/${riva_ngc_team}/rmir_nlp_named_entity_recognition_bert_base:${riva_ngc_model_version}"
```

(continues on next page)

(continued from previous page)

```

### BERT Base Intent Slot model fine-tuned on weather dataset.
    # "${riva_ngc_org}/${riva_ngc_team}/rmir_nlp_intent_slot_bert_base:${riva_ngc_model_version}"

### BERT Base Question Answering model fine-tuned on Squad v2.
    # "${riva_ngc_org}/${riva_ngc_team}/rmir_nlp_question_answering_bert_base:${riva_ngc_model_version}"

### Megatron345M Question Answering model fine-tuned on Squad v2.
#    "${riva_ngc_org}/${riva_ngc_team}/rmir_nlp_question_answering_megatron:${riva_ngc_model_version}"

### Bert base Text Classification model fine-tuned on 4class (weather, meteorology, personality, nomatch) domain model.
    # "${riva_ngc_org}/${riva_ngc_team}/rmir_nlp_text_classification_bert_base:${riva_ngc_model_version}"
)

########## TTS MODELS ##########

models_tts=(
#    "${riva_ngc_org}/${riva_ngc_team}/rmir_tts_fastpitch_hifigan_ljspeech:${riva_ngc_model_version}"
#   "${riva_ngc_org}/${riva_ngc_team}/rmir_tts_tacotron_waveglow_ljspeech:${riva_ngc_model_version}"
)

NGC_TARGET=${riva_ngc_org}
if [[ ! -z ${riva_ngc_team} ]]; then
NGC_TARGET="${NGC_TARGET}/${riva_ngc_team}"
else
team="\"\""
fi

# define docker images required to run Riva
image_client="nvcr.io/${NGC_TARGET}/riva-speech-client:${riva_ngc_image_version}"
image_speech_api="nvcr.io/${NGC_TARGET}/riva-speech:${riva_ngc_image_version}-server"

# define docker images required to setup Riva
image_init_speech="nvcr.io/${NGC_TARGET}/riva-speech:${riva_ngc_image_version}-servicemaker"

# daemon names
riva_daemon_speech="riva-speech"
riva_daemon_client="riva-client"
```

3.1.1 enable/ disable Riva services

Specify *line 10-12* to enable or disable Riva services.

```
service_enabled_asr=true
service_enabled_nlp=true
service_enabled_tts=true
```

3.1.2 encryption key

Specify *line 19* (consistency with the encryption key used to export models)

```
MODEL_DEPLOY_KEY="tlt_encode"
```

3.1.3 Riva model location

Specify *line 43* to configure the path of Riva model location

```
riva_model_loc="riva-model-repo"
```

Note: See *Riva - ServiceMaker* for details of how to use `riva-build` and `riva-deploy` tools to deploy custom models in Riva model repository.

3.1.4 ports config

Specify *line 53-54* to configure the exposed ports for Riva services

```
riva_speech_api_port="50051"
riva_vision_api_port="60051"
```

CHAPTER

FOUR

RIVA - SPEECH RECOGNITION

Automatic Speech Recognition (ASR) is the first step of building a conversational AI.

- Offline Recognition
- Streaming Recognition

CHAPTER

FIVE

RIVA - SPEECH RECOGNITION - MODEL ARCHITECTURES

5.1 Jasper

5.2 QuartzNet

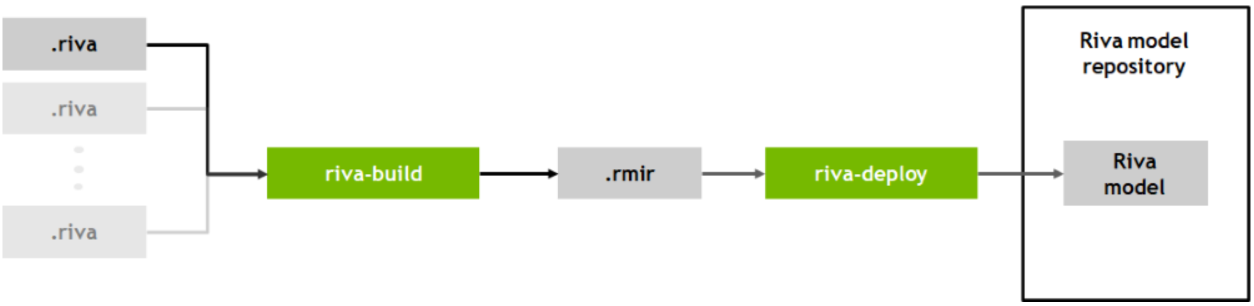
5.3 Citrinet

CHAPTER

SIX

RIVA - SPEECH RECOGNITION - DEPLOY ASR MODEL

Riva ServiceMaker will be used to deploy ASR `.riva` models in Riva server.



Note: Refer to *Riva - ServiceMaker* to see basic usage of `riva-build` and `riva-deploy`.

6.1 riva-build

Simple `riva-build` pipeline for speech recognition (ASR) models. See ASR model pipeline configuration for more details.

```
riva-build speech_recognition \
        /servicemaker-dev/<rmir_filename>:<encryption_key>  \
        /servicemaker-dev/<riva_filename>:<encryption_key> \
        --name=<pipeline_name> \
        --decoder_type=greedy \
        --acoustic_model_name=<acoustic_model_name>
```

- `speech_recognition`: task of `riva-build`. See *Riva - ServiceMaker* for more details.

Sample `riva-build` tool usage for ASR models.

```
docker run --rm --gpus 1 \
        -v $MODEL_LOC:/tao \
        -v $RMIR_LOC:/riva \
        $RIVA_SM_CONTAINER -- \
        riva-build speech_recognition --decoder_type=greedy /riva/asr.rmir:$KEY /tao/
↪$MODEL_NAME:$KEY
```

where

- `MODEL_LOC`: path of directory exported `.riva` model
- `RMIR_LOC`: path of output `.rmir` file
- `RIVA_SM_CONTAINER`: name of Riva ServiceMaker image.
- `MODEL_NAME`: name of exported `.riva` model
- `KEY`: encryption_key used in exporting `.riva` model.

6.2 riva-deploy

Sample `riva-deploy` tool usage for ASR models.

```
docker run --rm --gpus 1 \
        -v $RIVA_MODEL_LOC:/data \
        $RIVA_SM_CONTAINER -- \
        riva-deploy -f  /data/rmir/asr.rmir:$KEY /data/models/
```

where

- `RIVA_MODEL_LOC`: path of Riva model repository.

CHAPTER

SEVEN

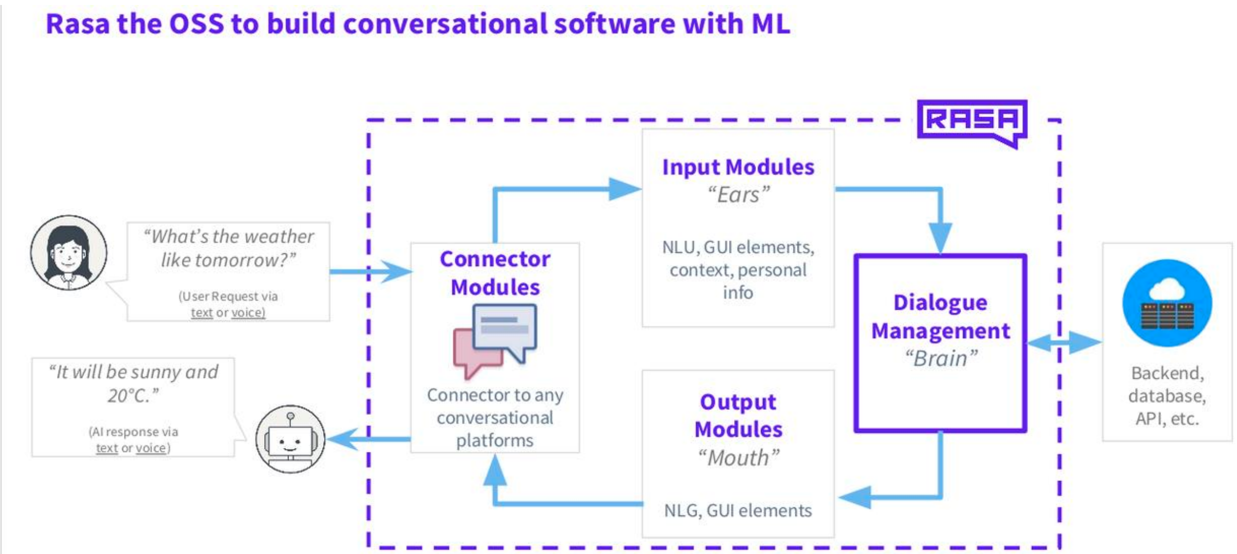
RIVA - NATURAL LANGUAGE PROCESSING

- Text Classification
- Token Classification (Named Entity Recognition)
- Joint Intent and Slots
- Question Answering (Extractive)
- Punctuation and Capitalization

CHAPTER

EIGHT

RIVA - RASA



8.1 Prerequisites

Create and activate a virtual environment for Rasa

```
python3 -m venv rasa_envs
source rasa_envs/bin/activate
```

Download rasa open source

```
pip install rasa
```

8.2 Sample Chatbot

Activate the rasa environment, and run

```
rasa init
```

Follow the instructions to create sample chatbot directory, train models, start Rasa server, and start chatting with the chatbot.

Simply type in `/stop` or `Ctrl+C` to exit the Rasa server.

Note:

- To start Rasa server in shell, run `rasa shell`
- To interact with Rasa Chatbot for improving its correctness on responses or predictions, run `rasa interactive`. Hit `Ctrl+C` to end the interactive session, and you might export the new data for `nlu.yml`, `domains.yml`, and `stories.yml`

8.2.1 Chatbot structure

```
.
├── actions
│   ├── __init__.py
│   ├── __pycache__
│   │   ├── __init__.cpython-38.pyc
│   │   └── actions.cpython-38.pyc
│   └── actions.py
├── config.yml
├── credentials.yml
├── data
│   ├── nlu.yml
│   ├── rules.yml
│   └── stories.yml
├── domain.yml
├── endpoints.yml
├── models
│   └── 20210909-224920.tar.gz
└── tests
    └── test_stories.yml
```

See also:

Useful learning sources:

- Rasa for beginners
- Rasa Advanced Workshop - Deployment
- Rasa Advanced Workshop - Custom Actions, Forms, & Responses
- Rasa Certification Workshop

CHAPTER

NINE

RIVA - RASA - NLU COMPONENT

9.1 Sample nlu.yml

```
version: "2.0"

nlu:
- intent: greet
examples: |
    - hey
    - hello
    - hi
    - hello there
    - good morning
    - good evening
    - moin
    - hey there
    - let's go
    - hey dude
    - goodmorning
    - goodevening
    - good afternoon

- intent: goodbye
examples: |
    - good afternoon
    - cu
    - good by
    - cee you later
    - good night
    - bye
    - goodbye
    - have a nice day
    - see you around
    - bye bye
    - see you later

- intent: affirm
examples: |
    - yes
    - y
```

(continues on next page)

(continued from previous page)

```
    - indeed
    - of course
    - that sounds good
    - correct

- intent: deny
examples: |
    - no
    - n
    - never
    - I don't think so
    - don't like that
    - no way
    - not really

- intent: mood_great
examples: |
    - perfect
    - great
    - amazing
    - feeling like a king
    - wonderful
    - I am feeling very good
    - I am great
    - I am amazing
    - I am going to save the world
    - super stoked
    - extremely good
    - so so perfect
    - so good
    - so perfect

- intent: mood_unhappy
examples: |
    - my day was horrible
    - I am sad
    - I don't feel very well
    - I am disappointed
    - super sad
    - I'm so sad
    - sad
    - very sad
    - unhappy
    - not good
    - not very good
    - extremly sad
    - so saad
    - so sad

- intent: bot_challenge
examples: |
    - are you a bot?
```

(continues on next page)

(continued from previous page)

```
    - are you a human?
    - am I talking to a bot?
    - am I talking to a human?
```

Note:

- after adding/ removing intents, update `intents` in `domain.yml` (See *Riva - Rasa - Domain*).

Note:

- after adding/ removing intents, you may want to add/ remove/ update corresponding responses for such intents. To do that, update `responses` in `domain.yml` (See *Riva - Rasa - Domain*).

CHAPTER

TEN

RIVA - RASA - STORIES (DIALOG MANAGER)

Stories (Dialog Manager) are defined to train Dialog Manager model in order to inform Rasa chatbots what to do next (or next next, next next next, . . .)

10.1 Sample stories.yml

```
version: '2.0'

stories:
    - story: happy path
      steps:
          - intent: greet
          - action: utter_greet
          - intent: mood_great
          - action: utter_happy

    - story: sad path 1
      steps:
          - intent: greet
          - action: utter_greet
          - intent: mood_unhappy
          - action: utter_cheer_up
          - action: utter_did_that_help
          - intent: affirm
          - action: utter_happy

    - story: sad path 2
      steps:
          - intent: greet
          - action: utter_greet
          - intent: mood_unhappy
          - action: utter_cheer_up
          - action: utter_did_that_help
          - intent: deny
          - action: utter_goodbye
```

CHAPTER

ELEVEN

RIVA - RASA - ACTIONS

CHAPTER

TWELVE

RIVA - RASA - DOMAIN

12.1 Sample domain.yml

```
version: "2.0"

intents:
- greet
- goodbye
- affirm
- deny
- mood_great
- mood_unhappy
- bot_challenge

responses:
utter_greet:
- text: "Hey! How are you?"

utter_cheer_up:
- text: "Here is something to cheer you up:"
    image: "https://i.imgur.com/nGF1K8f.jpg"

utter_did_that_help:
- text: "Did that help you?"

utter_happy:
- text: "Great, carry on!"

utter_goodbye:
- text: "Bye"

utter_iamabot:
- text: "I am a bot, powered by Rasa."

session_config:
session_expiration_time: 60
carry_over_slots_to_new_session: true
```

CHAPTER

THIRTEEN

RIVA - RASA - CONFIGURATIONS

CHAPTER

FOURTEEN

RIVA - RASA - MODEL TRAINING

After updating some components of Rasa, train/ retrain the models before running Rasa.

```
rasa train
```

CHAPTER

FIFTEEN

RIVA - RASA - CREDENTIALS

CHAPTER SIXTEEN

RIVA - RASA - ENDPOINTS

CHAPTER

SEVENTEEN

RIVA - RASA - CONVERSATION TEST

CHAPTER

EIGHTEEN

RIVA - SPEECH SYNTHESIS

- Synthesize
- SynthesizeOnline

CHAPTER

NINETEEN

RIVA - SPEECH SYNTHESIS - MODEL ARCHITECTURES

Table 1: Available models for Speech Synthesis

Tasks	Models
Mel Spectrogram Generator	Tacotron 2
Mel Spectrogram Generator	Glow-TTS
Mel Spectrogram Generator	TalkNet
Mel Spectrogram Generator	FastPitch
Mel Spectrogram Generator	FastSpeech2
Audio Generator	WaveGlow
Audio Generator	SqueezeWave
Audio Generator	UniGlow
Audio Generator	MelGAN
Audio Generator	HiFiGAN
Audio Generator	Griffin-Lim
End-to-end model	FastPitch_HifiGan_E2E
End-to-end model	FastSpeech2_HifiGan_E2E

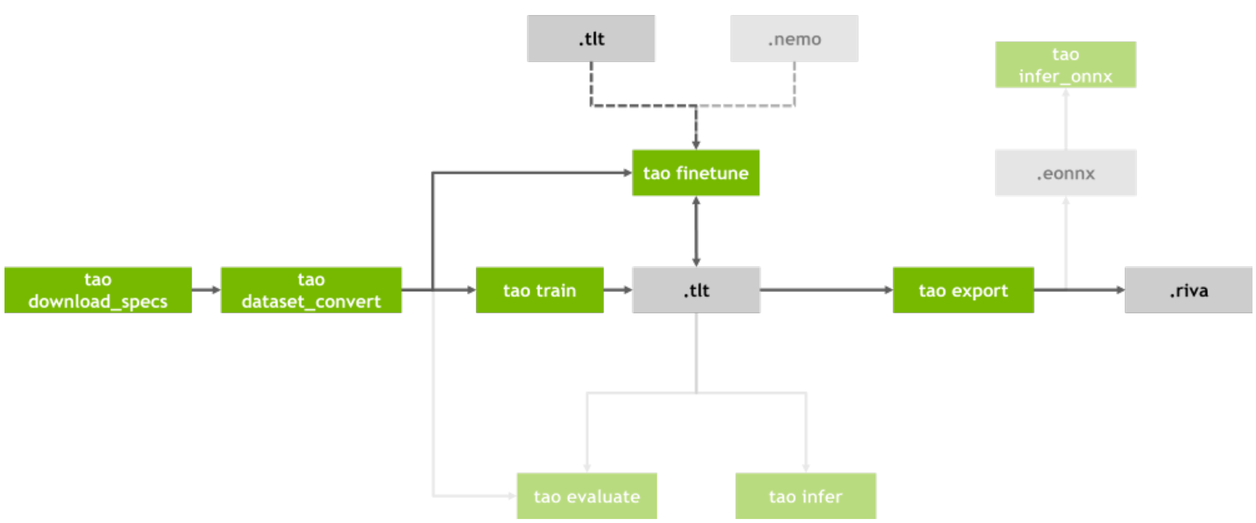
CHAPTER

TWENTY

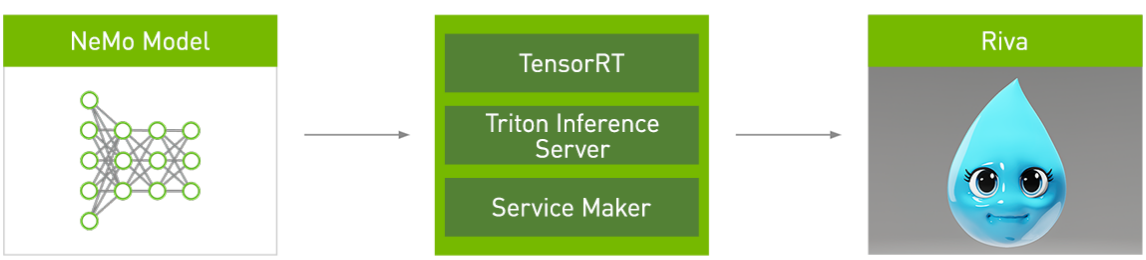
RIVA - CUSTOM MODEL

To train, export AI models, and then deploy the models in Riva server, we will use NVIDIA TAO and NVIDIA NeMo. Both training frameworks enable the transformation of custom models into `.riva` models that can be deployed in Riva.

20.1 TAO



20.2 NeMo



20.3 Export models

The exported models in `.riva` format can be deployed in Riva with Riva ServiceMaker.

Note: See *Riva - ServiceMaker* to deploy TAO- or NeMo-trained models.

See also:

- NeMo guide
- NeMo Tutorials

CHAPTER

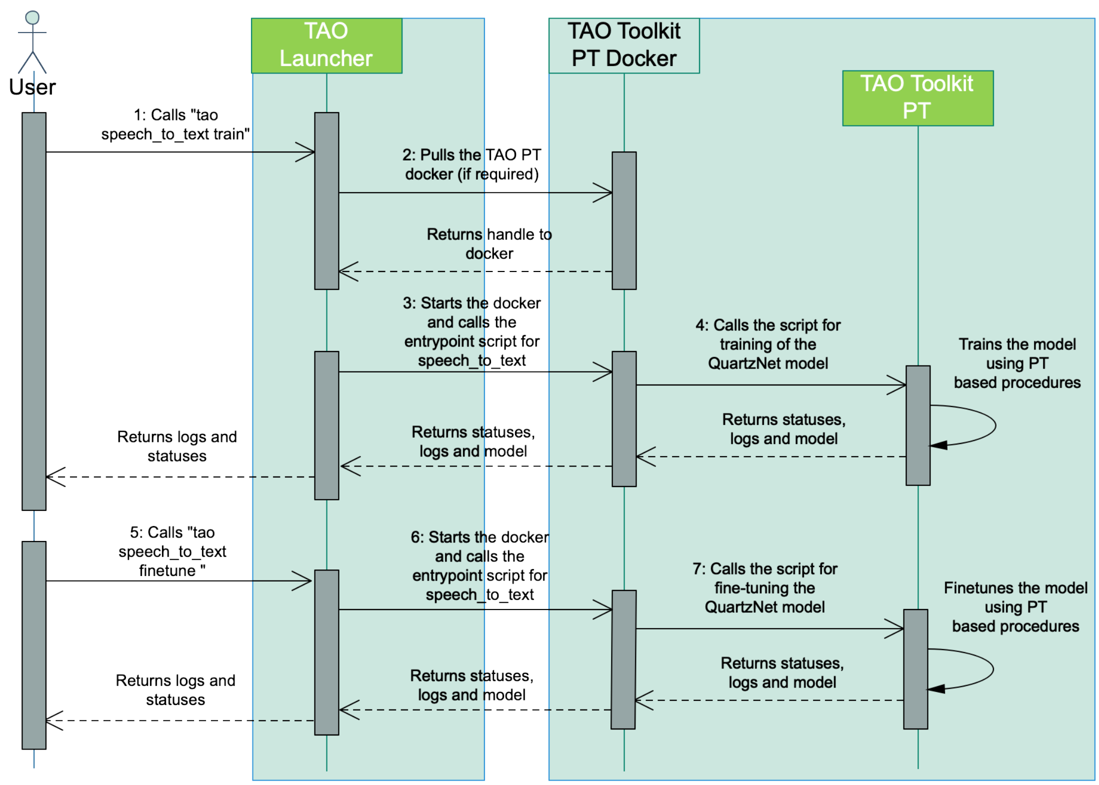
TWENTYONE

RIVA - CUSTOM MODEL - TAO

21.1 TAO Launcher

```
tao <task> <subtask> <args>
```

RIVA - CUSTOM MODEL - TAO - SPEECH-TO-TEXT



22.1 Usage

The structure used to perform `speech_to_text` task

```
tao speech_to_text <subtask>
```

22.1.1 Subtasks

The subtasks for `speech_to_text` include *dataset_convert, evaluate, export, finetune, infer, infer_onnx, train, download_specs*

Sample task *download_specs*

```
!tao speech_to_text download_specs \
    -o $SPECS_DIR/speech_to_text \
    -r $RESULTS_DIR
```

Sample task *export*

```
# SPECS_DIR/speech_to_text/export.yaml
# Name of the .tlt EFF archive to be loaded/model to be exported.
restore_from: trained-model.tlt

# Set export format: ONNX | RIVA
export_format: RIVA

# Output EFF archive containing ONNX.
export_to: exported-model.riva
```

```
!tao speech_to_text export \
    -r $RESULTS_DIR/quartznet/export \
    -k $KEY \
    -e $SPECS_DIR/speech_to_text/export.yaml \
    -g 1 \
    -m $MODELS_DIR/speechtotext_english_quartznet.tlt \
    export_format=RIVA \
    export_to=asr-model.riva
```

22.1.2 Arguments

Table 1: TAO speech-to-text task arguments

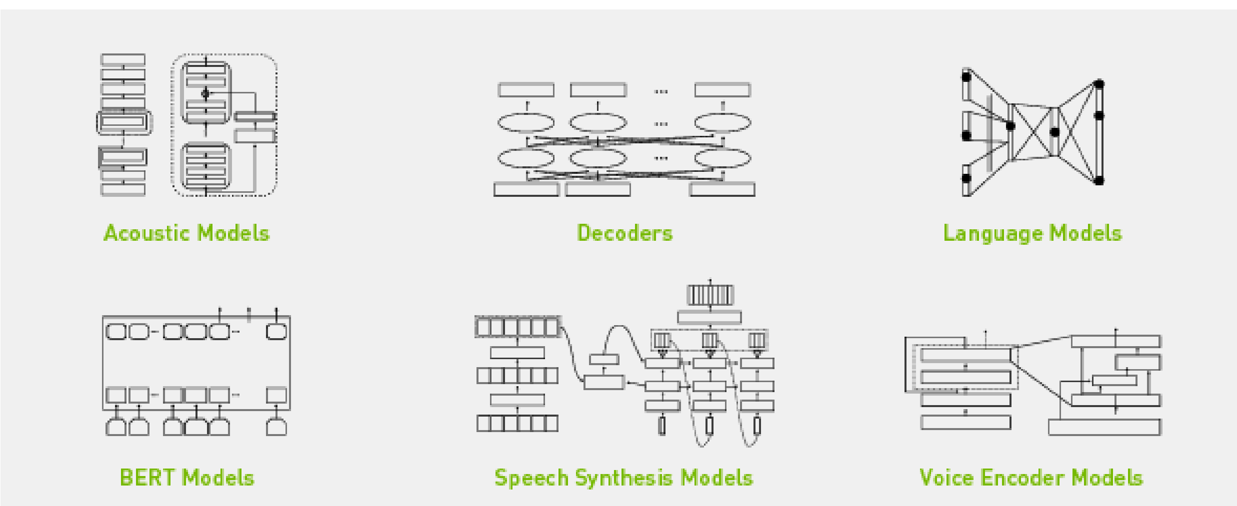
Arguments	Details
subtask	choose the task from *Subtasks*
-r	path of results directory
-k	your NGC API key
-e	spec file of a specific subtask
-g	number of gpu to use
-m	path of pretrained ASR model
-o	specify path for downloading spec files
additional arguments	override values in spec file

CHAPTER

TWENTYTHREE

RIVA - CUSTOM MODEL - NEMO

NVIDIA NeMo (Neural Module) is built on PyTorch and PyTorch lightning.



23.1 Prerequisites

- NGC API Key

23.2 Getting started

1. Clone NVIDIA NeMo Github folder

```
git clone https://github.com/NVIDIA/NeMo
```

2. Create PyTorch container

```
docker run --gpus all -it --rm -v <nemo_github_folder>:/NeMo --shm-size=8g \
-p 8888:8888 -p 6006:6006 --ulimit memlock=-1 --ulimit \
stack=67108864 --device=/dev/snd nvcr.io/nvidia/pytorch:21.05-py3
```

3. Install NeMo

```
cd /NeMo
./reinstall.sh
```

See also:

- NeMo Tutorials
- ASR with NeMo

CHAPTER

TWENTYFOUR

RIVA - CUSTOM MODEL - NEMO - DATA MANIFEST - ASR

The standardized manifest (a json file storing all metadata of a given dataset) format for NeMo is that

- each line corresponds to one sample of audio
- Number of lines equals to number of samples

A sample of a line in the manifest

```
{"audio_filepath": "path/to/audio.wav", "duration": 3.45, "text": "transcript of audio.
↪wav"}
```

CHAPTER

TWENTYFIVE

RIVA - CUSTOM MODEL - NEMO - ASR - MANDARIN

This sample implements ASR of Mandarin using Nemo.

Note:

- If you wish to skip the following steps to set up the environment to train mandarin ASR model, a container is prepared which you can launch and start training immediately:

```
docker pull nvcr.io/nvidian/sae/mandarin-asr-spoon:1.0.1
```

- To launch the container, run:

```
docker run --gpus all -it --rm --shm-size=8g \
-p 8888:8888 -p 6006:6006 --ulimit memlock=-1 --ulimit \
stack=67108864 --device=/dev/snd nvcr.io/nvidian/sae/mandarin-asr-spoon:1.0.1
```

- Finally, run the training:

```
python3 main.py --e <train-epoch> --g <no-of-gpus>
```

25.1 0. Launch PyTorch container with NeMo

Follow *Getting started* to set up container for the sample ASR Mandarin implementation.

Then start a jupyter notebook inside the container

```
jupyter notebook --ip=0.0.0.0 --allow-root
```

Access the jupyter notebook by the URI *http://<your-host-ip>:8888/tree*. Create a notebook, and so you are ready for next step.

25.2 1. Prepare data manifest

25.2.1 1.1 Download Aishell-1 dataset

Download AISHELL-1 dataset from Openslr. AISHELL-1 dataset is an open-sourced dataset contaning 178 hours of Mandarin utterances.

```
!wget https://www.openslr.org/resources/33/data_aishell.tgz
!tar zxvf data_aishell.tgz
```

Retrieve the audio files

```python
import os
import tarfile
import shutil
from tqdm import tqdm

# remove manifests if already exists
os.chdir("/NeMo/data_aishell/wav")
if delete_previous:
    shutil.rmtree("train")
    shutil.rmtree("test")
    shutil.rmtree("dev")

# untar the audio files if you have not done this
wav_tars = os.listdir(os.getcwd())
for wav_tar in tqdm(wav_tars):
    current_tar = os.path.join(os.getcwd(), wav_tar)
    tar = tarfile.open(current_tar)
    tar.extractall('.')
    tar.close()
```

25.2.2 1.2 Build data manifest

Data manifest are files containing all metadata of audio required by NeMo.

```python
import librosa
import json

def build_aishell_manifest(transcript_path, manifest_path, wav_path):
    with open(transcript_path, "r") as fin:
        with open(manifest_path, "w") as fout:

            transcript = fin.readlines()

            for ref in tqdm(transcript):
                s_num = ref[6:11]
                text = ref[11 + 5:].strip()
                audio_file = "BAC009" + ref[6:16] + ".wav"
                audio_path = os.path.join(wav_path, s_num, audio_file)
```

(continues on next page)

(continued from previous page)

```
                if os.path.exists(audio_path):

                    duration = librosa.core.get_duration(filename=audio_path)

                    metadata = {
                        "audio_filepath": audio_path,
                        "duration": duration,
                        "text": text
                    }

                    json.dump(metadata, fout, ensure_ascii=False)
                    fout.write('\n')


# build training manifest
print("building training manifest")
transcript_path = "/NeMo/data_aishell/transcript/aishell_transcript_v0.8.txt"
train_manifest_path = "/NeMo/data_aishell/train_manifest.json"
train_wav_path = "/NeMo/data_aishell/wav/train"
build_aishell_manifest(transcript_path, train_manifest_path, train_wav_path)
print("training manifest prepared at {}".format(train_manifest_path))

# build validation manifest
print("\nbuilding validation manifest")
transcript_path = "/NeMo/data_aishell/transcript/aishell_transcript_v0.8.txt"
valid_manifest_path = "/NeMo/data_aishell/valid_manifest.json"
valid_wav_path = "/NeMo/data_aishell/wav/test"
build_aishell_manifest(transcript_path, valid_manifest_path, valid_wav_path)
print("training manifest prepared at {}".format(valid_manifest_path))

# build testing manifest
print("\nbuilding testing manifest")
transcript_path = "/NeMo/data_aishell/transcript/aishell_transcript_v0.8.txt"
test_manifest_path = "/NeMo/data_aishell/test_manifest.json"
test_wav_path = "/NeMo/data_aishell/wav/dev"
build_aishell_manifest(transcript_path, test_manifest_path, test_wav_path)
print("testing manifest prepared at {}".format(test_manifest_path))
```

See also:

see *Riva - Custom Model - NeMo - Data Manifest - ASR* for more about manifest formatting.

25.3 2. Prepare training configuration

25.3.1 2.1 Load QuartzNet 15x5 zh default config

The model we are using to train with Aishell-1 is QuartzNet 15x5 zh. We could start with a default config file `quartznet_15x5_zh.yaml` at `/NeMo/examples/asr/conf/quartznet/`.

```
from omegaconf import OmegaConf
import copy
```

(continues on next page)

(continued from previous page)

```
try:
    from ruamel.yaml import YAML
except ModuleNotFoundError:
    from ruamel_yaml import YAML

config_path = "/NeMo/examples/asr/conf/quartznet/quartznet_15x5_zh.yaml"

yaml = YAML(typ='safe')
with open(config_path) as f:
    zh_qn_params = yaml.load(f)

print(OmegaConf.to_yaml(zh_qn_params))
```

Note: The config file is very lengthy as it contains Chinese lexicons.

25.3.2 2.2 Update manifest path

Update the path to data manifests that we just created

```
zh_qn_params["model"]["train_ds"]["manifest_filepath"] = train_manifest_path
zh_qn_params["model"]["validation_ds"]["manifest_filepath"] = valid_manifest_path
```

25.3.3 2.3 Update num_workers

Update num_workers of data loaders

```
num_workers = 40
zh_qn_params["model"]["train_ds"]["num_workers"] = num_workers
zh_qn_params["model"]["validation_ds"]["num_workers"] = num_workers
```

25.3.4 2.4 Update logging directory

We will create a results folder `/NeMo/data_aishell/results` where our training logs and output `.nemo` model will be stored.

```
!mkdir data_aishell/results
```

Update the results path in the configuration.

```
results_path = "/NeMo/data_aishell/results"
zh_qn_params["exp_manager"]["exp_dir"] = results_path
```

25.4 3. Train QuartzNet 15x5 zh

After the configuration is ready, we move on to model training session. Define `pytorch_lightning.Trainer` and `nemo_asr.models.EncDecCTCModel` instances:

```
from omegaconf import DictConfig
import pytorch_lightning as pl
import nemo
import nemo.collections.asr as nemo_asr

trainer = pl.Trainer(checkpoint_callback=False, logger=False, gpus=4, max_epochs=50,␣
↪accelerator="ddp")

zh_qn = nemo_asr.models.EncDecCTCModel(cfg=DictConfig(zh_qn_params['model']),␣
↪trainer=trainer)
```

Also, we apply `exp_manager` to apply the logging path of training.

```
from nemo.utils.exp_manager import exp_manager
exp_manager(trainer, zh_qn_params.get("exp_manager", None))
```

However, `pl.Trainer` using an accelerator `ddp` cannot be done in interactive environment (notebook).

> **Warning:** If you attempt to use `ddp` in the notebook, you will be likely warned:
>
> ```
> AttributeError: Can't pickle local object 'FilterbankFeatures.__init__.<locals>.
> ↪<lambda>'
> ```

Thus, let's create a python script `main.py` to start the training and save our result `model.nemo`. To do so, go to jupyter notebook menu, open a terminal, create `main.py`, and paste the following script.

```
import librosa
import json
from omegaconf import OmegaConf
import copy
from omegaconf import DictConfig
import pytorch_lightning as pl
import nemo
import nemo.collections.asr as nemo_asr
from nemo.utils.exp_manager import exp_manager
try:
    from ruamel.yaml import YAML
except ModuleNotFoundError:
    from ruamel_yaml import YAML

def main():
    # set manifest path
    train_manifest_path = "/NeMo/data_aishell/train_manifest.json"
    valid_manifest_path = "/NeMo/data_aishell/valid_manifest.json"

    # load QuartzNet 15x5 zh default config
    config_path = "/NeMo/examples/asr/conf/quartznet/quartznet_15x5_zh.yaml"
```

(continues on next page)

(continued from previous page)

```
    yaml = YAML(typ='safe')
    with open(config_path) as f:
        zh_qn_params = yaml.load(f)

    # update manifest path
    zh_qn_params["model"]["train_ds"]["manifest_filepath"] = train_manifest_path
    zh_qn_params["model"]["validation_ds"]["manifest_filepath"] = valid_manifest_path

    # update num_workers
    num_workers = 40
    zh_qn_params["model"]["train_ds"]["num_workers"] = num_workers
    zh_qn_params["model"]["validation_ds"]["num_workers"] = num_workers

    # update training logs path
    results_path = "/NeMo/data_aishell/results"
    zh_qn_params["exp_manager"]["exp_dir"] = results_path

    # Initialise trainer
    trainer = pl.Trainer(checkpoint_callback=False, logger=False, gpus=4, max_epochs=50,␣
↪accelerator="ddp")
    zh_qn = nemo_asr.models.EncDecCTCModel(cfg=DictConfig(zh_qn_params['model']),␣
↪trainer=trainer)

    # training logs export
    exp_manager(trainer, zh_qn_params.get("exp_manager", None))

    # Start training
    zh_qn.summarize()
    trainer.fit(zh_qn)

    # Save trained model
    zh_qn.save_to("/NeMo/data_aishell/results/model.nemo")


if __name__ == '__main__':
    main()
```

Then, in the terminal, execute `main.py` to start the training process

```
python3 main.py
```

25.5 4. Visualise Training Progress

The training logs are saved at `/NeMo/data_aishell/results/QuartzNet15x5`, and we use TensorBoard for visualisation using a custom Python environment with TensorBoard installed.

Create a python environment

```
python3 -m venv venv
```

Install tensorboard

```
pip install tensorboard
```

Launch tensorboard

```
cd /NeMo/data_aishell/results/
tensorboard --bind_all --logdir QuartzNet15x5
```

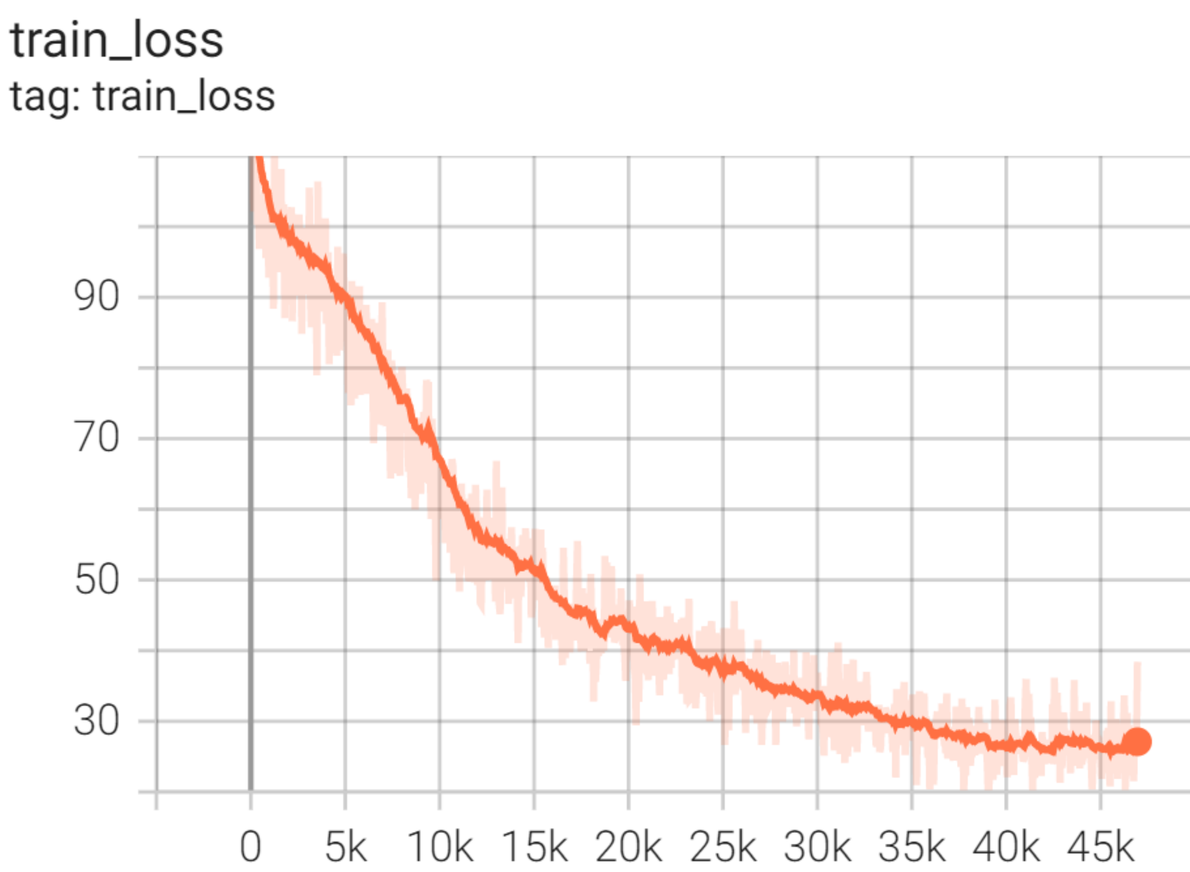
Warning: If you use tensorboard that comes with NeMo package install script `reinstall.sh` (as of Sep 2021), that is:

```
tensorboard --bind_all --logdir QuartzNet15x5/
```

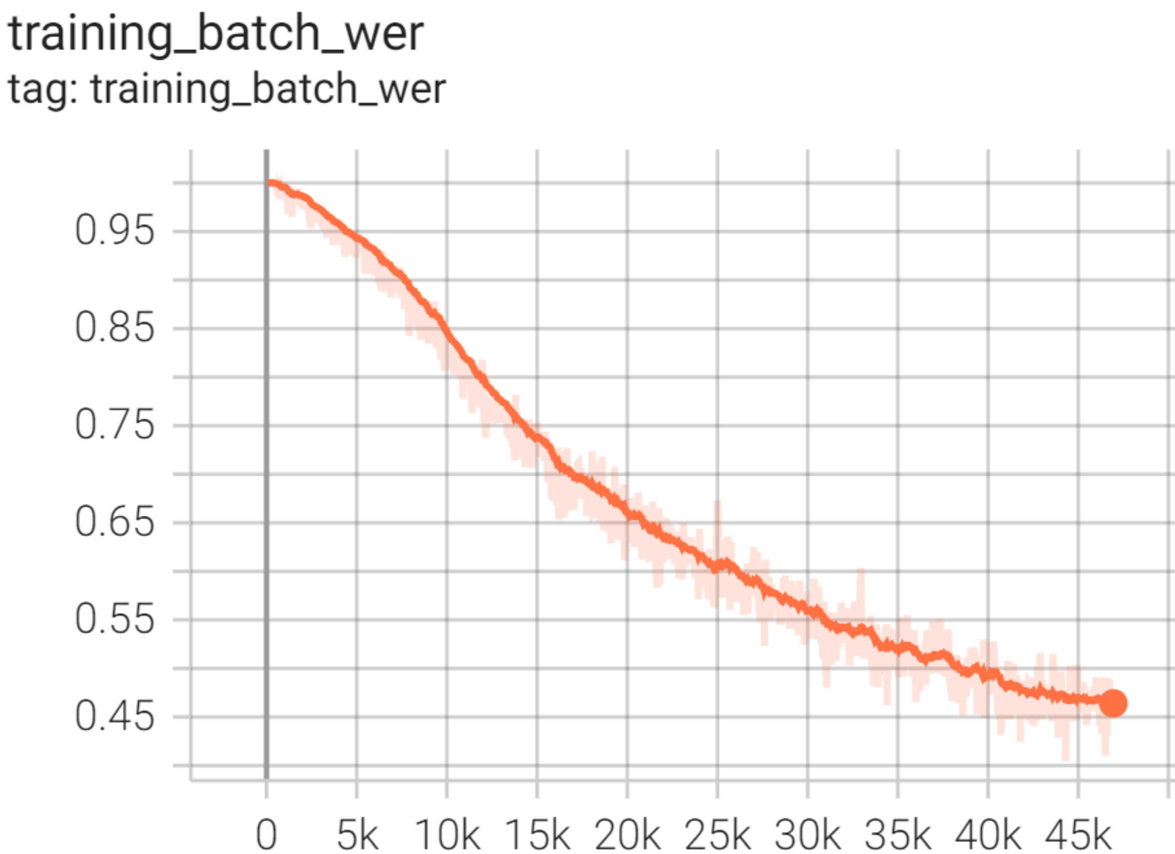
you will meet the following error:

```
ValueError: Duplicate plugins for name projector
```

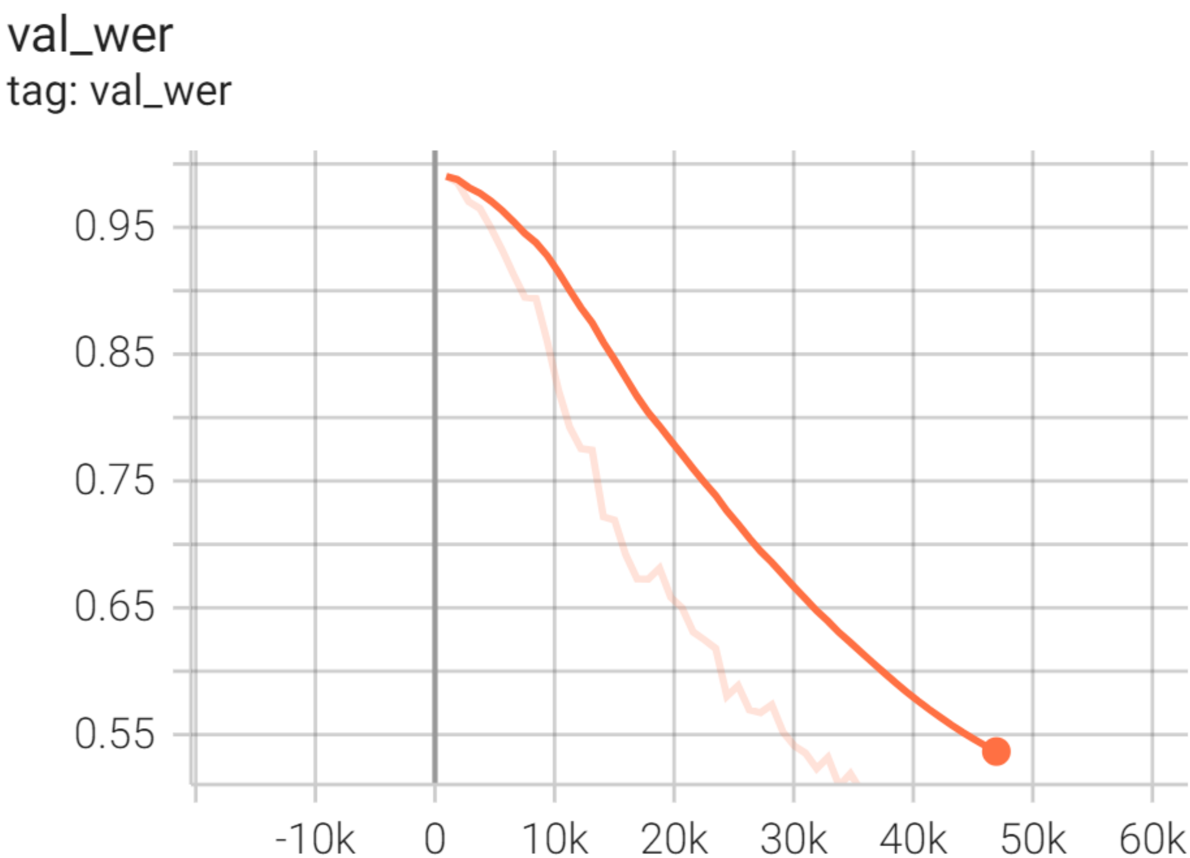
25.5.1 Training loss



25.5.2 Training batch WER



25.5.3 Validation WER



CHAPTER

TWENTYSIX

RIVA - CUSTOM MODEL - NEMO - DATA FORMAT - NLP (INTENT-SLOT)

It's required to convert data into specific NeMo format when training NLP models for *Joint Intent and Slot Classification*. The model training requires necessary input files for *intent* training and *slot* training (`dict.intents.csv`, `dict.slots.csv`, `train.tsv`, `train_slots.tsv`, `test.tsv`, `test_slots.tsv`) which are expected to store in a directory. The path of data directory needs to be updated in the training config file.

26.1 Expected structure

```
|--<target_data_dir>
        |-- dict.intents.csv
        |-- dict.slots.csv
        |-- train.tsv
        |-- train_slots.tsv
        |-- test.tsv
        |-- test_slots.tsv
```

26.2 Sample dict.intents.csv

Suppose we have N intents

```
#intent #label
interest        # 0
transportation  # 1
sleep           # 2
...
intentN         # N-1
```

26.3 Sample dict.slots.csv

Suppose we have N slots

```
#slot   #label
interest_organisation   # 0
transportation_mathod   # 1
sleep_bahaviour         # 2
...
O                       # N-1
```

26.4 Sample train.tsv

For every first line of any `train.tsv`, we have a header line

```
sentence <tab> label
```

and, for rest of the lines follows `<sentence> <tab> <label>` where `<label>` refers to label in `dict.intents.csv`

```
I love NVIDIA <tab> <label>
I go to company by bus <tab> <label>
I sleep late everyday <tab> <label>
...
```

26.5 Sample train_slots.tsv

There is no header line for any `train_slots.tsv`. For each line of the file, each token of the line is presented a label to slots (defined in `dict.slots.csv`).

```
# there is space between labels
N-1 N-1 0
N-1 N-1 N-1 N-1 N-1 1
N-1 N-1 2 N-1
...
```

26.6 Sample test.tsv

Similar with `train.tsv`

26.7 Sample test_slots.tsv

Similar with `train_slots.tsv`

See also:

- Joint_Intent_and_Slot_Classification.ipynb

CHAPTER

TWENTYSEVEN

RIVA - CUSTOM MODEL - NEMO - DATA FORMAT - NLP (NER)

See also:

- Token_Classification_Named_Entity_Recognition.ipynb

CHAPTER

TWENTYEIGHT

RIVA - CUSTOM MODEL - NEMO - DATA FORMAT - NLP (TEXT CLASSIFICATION)

See also:

- Text_Classification_Sentiment_Analysis.ipynb

CHAPTER

TWENTYNINE

RIVA - CUSTOM MODEL - NEMO - TTS - INFERENCE

29.1 Two-stage Approach

29.1.1 Download and load models

```
import soundfile as sf
from nemo.collections.tts.models.base import SpectrogramGenerator, Vocoder

## spectrogram generators
spec_generator = SpectrogramGenerator.from_pretrained(model_name="tts_en_fastpitch").
↪cuda()
# spec_generator = SpectrogramGenerator.from_pretrained(model_name="tts_en_tacotron2").
↪cuda()
# spec_generator = SpectrogramGenerator.from_pretrained(model_name="tts_en_glowtts").
↪cuda()
# spec_generator = SpectrogramGenerator.from_pretrained(model_name="tts_en_tts_en_talknet
↪").cuda()
# spec_generator = SpectrogramGenerator.from_pretrained(model_name="tts_en_fastspeech2").
↪cuda()

## Audio generators
vocoder = Vocoder.from_pretrained(model_name="tts_hifigan").cuda()
# vocoder = Vocoder.from_pretrained(model_name="tts_waveglow").cuda()
# vocoder = Vocoder.from_pretrained(model_name="tts_squeezewave").cuda()
# vocoder = Vocoder.from_pretrained(model_name="tts_uniglow").cuda()
# vocoder = Vocoder.from_pretrained(model_name="tts_melgan").cuda()

# TBA for griffin-lim
```

Note:

- See *Riva - Speech Synthesis - Model Architectures* for details of models

29.1.2 Convert to audio

```
parsed = spec_generator.parse("You can type your sentence here to get nemo to produce␣
↪speech.")
spectrogram = spec_generator.generate_spectrogram(tokens=parsed)
audio = vocoder.convert_spectrogram_to_audio(spec=spectrogram)
```

Note: Mel Spectrogram generators have two helper functions:

- `parse()``: Accepts raw python strings and returns a torch.tensor that represents tokenized text
- `generate_spectrogram()`: Accepts a batch of tokenized text and returns a torch.tensor that represents a batch of spectrograms

Vocoder have just one helper function:

- `convert_spectrogram_to_audio()`: Accepts a batch of spectrograms and returns a torch.tensor that represents a batch of raw audio

29.1.3 Save Audio

```
sf.write("speech.wav", audio.to('cpu').detach().numpy()[0], 22050)
```

29.2 End-to-end Approach

29.2.1 Download and load models

```
import soundfile as sf
from nemo.collections.tts.models.base import SpectrogramGenerator, Vocoder,␣
↪TextToWaveform

e2e_model = TextToWaveform.from_pretrained("tts_en_e2e_fastpitchhifigan").cuda()
# e2e_model = TextToWaveform.from_pretrained("tts_en_e2e_fastspeech2hifigan").cuda()
```

29.2.2 Convert to audio

```
parsed = e2e_model.parse("You can type your sentence here to get nemo to produce speech.
↪")
audio = e2e_model.convert_text_to_waveform(tokens=parsed)[0]
```

Note: End-to-end models have two helper functions:

- `parse()`: Accepts raw python strings and returns a torch.tensor that represents tokenized text
- `convert_text_to_waveform()`: Accepts a batch of tokenized text and returns a torch.tensor that represents a batch of raw audio

29.2.3 Save Audio

```
sf.write("speech.wav", audio.to('cpu').detach().numpy()[0], 22050)
```

CHAPTER

THIRTY

RIVA - CUSTOM MODEL - NEMO - DATA MANIFEST - TTS

See *Riva - Custom Model - NeMo - Data Manifest - ASR* to build data manifest for training TTS models.

CHAPTER

THIRTYONE

RIVA - CUSTOM MODEL - NEMO - TTS - MANDARIN

This sample simplements the sample training Mel spectrogram generator model Tacotron2 using Mandarin dataset.

See Original paper

CHAPTER

THIRTYTWO

RIVA - GRPC API

Riva's services are exposed using gRPC to maximise its compatibility with different software infrastructure and easier integrations.

32.1 riva_asr.proto

```
// Copyright 2019 Google LLC.
// Copyright (c) 2021, NVIDIA CORPORATION. All rights reserved.
//
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//
//     http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.
//

syntax = "proto3";

package nvidia.riva.asr;

option cc_enable_arenas = true;
option go_package = "nvidia.com/riva_speech";

import "riva_audio.proto";

/*
* The RivaSpeechRecognition service provides two mechanisms for converting speech to text.
*/
service RivaSpeechRecognition {
    // Recognize expects a RecognizeRequest and returns a RecognizeResponse. This request will block
```

(continues on next page)

(continued from previous page)

```
    // until the audio is uploaded, processed, and a transcript is returned.
    rpc Recognize(RecognizeRequest) returns (RecognizeResponse) {}
    // StreamingRecognize is a non-blocking API call that allows audio data to be fed to the server in
    // chunks as it becomes available. Depending on the configuration in the StreamingRecognizeRequest,
    // intermediate results can be sent back to the client. Recognition ends when the stream is closed
    // by the client.
    rpc StreamingRecognize(stream StreamingRecognizeRequest) returns (stream StreamingRecognizeResponse) {}
}


/*
* RecognizeRequest is used for batch processing of a single audio recording.
*/
message RecognizeRequest {
    // Provides information to recognizer that specifies how to process the request.
    RecognitionConfig config = 1;
    // The raw audio data to be processed. The audio bytes must be encoded as specified in
    // `RecognitionConfig`.
    bytes audio = 2;
}


/*
* A StreamingRecognizeRequest is used to configure and stream audio content to the
* Riva ASR Service. The first message sent must include only a StreamingRecognitionConfig.
* Subsequent messages sent in the stream must contain only raw bytes of the audio
* to be recognized.
*/
message StreamingRecognizeRequest {
    // The streaming request, which is either a streaming config or audio content.
    oneof streaming_request {
        // Provides information to the recognizer that specifies how to process the
        // request. The first `StreamingRecognizeRequest` message must contain a
        // `streaming_config`  message.
        StreamingRecognitionConfig streaming_config = 1;
        // The audio data to be recognized. Sequential chunks of audio data are sent
        // in sequential `StreamingRecognizeRequest` messages. The first
        // `StreamingRecognizeRequest` message must not contain `audio` data
        // and all subsequent `StreamingRecognizeRequest` messages must contain
        // `audio` data. The audio bytes must be encoded as specified in
        // `RecognitionConfig`.
        bytes audio_content = 2;
    }
}

// Provides information to the recognizer that specifies how to process the request
```

(continues on next page)

(continued from previous page)

```
message RecognitionConfig {
    // The encoding of the audio data sent in the request.
    //
    // All encodings support only 1 channel (mono) audio.
    AudioEncoding encoding = 1;

    //  Sample rate in Hertz of the audio data sent in all
        // `RecognizeAudio` messages.
    int32 sample_rate_hertz = 2;

    // Required. The language of the supplied audio as a
        // [BCP-47](https://www.rfc-editor.org/rfc/bcp/bcp47.txt) language tag.
        // Example: "en-US".
                // Currently only en-US is supported
    string language_code = 3;

    // Maximum number of recognition hypotheses to be returned.
    // Specifically, the maximum number of `SpeechRecognizeAlternative` messages
    // within each `SpeechRecognizeResult`.
    // The server may return fewer than `max_alternatives`.
    // If omitted, will return a maximum of one.
    int32 max_alternatives = 4;

        // The number of channels in the input audio data.
        // ONLY set this for MULTI-CHANNEL recognition.
        // Valid values for LINEAR16 and FLAC are `1`-`8`.
        // Valid values for OGG_OPUS are '1'-'254'.
        // Valid value for MULAW, AMR, AMR_WB and SPEEX_WITH_HEADER_BYTE is only `1`.
        // If `0` or omitted, defaults to one channel (mono).
        // Note: We only recognize the first channel by default.
        // To perform independent recognition on each channel set
        // `enable_separate_recognition_per_channel` to 'true'.
        int32 audio_channel_count = 7;

        // If `true`, the top result includes a list of words and
        // the start and end time offsets (timestamps) for those words. If
        // `false`, no word-level time offset information is returned. The default is
        // `false`.
    bool enable_word_time_offsets = 8;

                // If 'true', adds punctuation to recognition result hypotheses.
        // The default 'false' value does not add punctuation to result hypotheses.
        bool enable_automatic_punctuation = 11;

        // This needs to be set to `true` explicitly and `audio_channel_count` > 1
        // to get each channel recognized separately. The recognition result will
        // contain a `channel_tag` field to state which channel that result belongs
        // to. If this is not true, we will only recognize the first channel. The
        // request is billed cumulatively for all channels recognized:
        // `audio_channel_count` multiplied by the length of the audio.
        bool enable_separate_recognition_per_channel = 12;

```

(continues on next page)

(continued from previous page)

```
                // Which model to select for the given request. Valid choices: Jasper, Quartznet
        string model = 13;

    // The verbatim_transcripts flag enables or disable inverse text normalization.
    // 'true' returns exactly what was said, with no denormalization.
    // 'false' applies inverse text normalization, also this is the default
    bool verbatim_transcripts = 14;

    // Custom fields for passing request-level
    // configuration options to plugins used in the
    // model pipeline.
    map<string, string> custom_configuration = 24;


}

// Provides information to the recognizer that specifies how to process the request
message StreamingRecognitionConfig {
                // Provides information to the recognizer that specifies how to process the request
    RecognitionConfig config = 1;

                // If `true`, interim results (tentative hypotheses) may be
    // returned as they become available (these interim results are indicated with
    // the `is_final=false` flag).
    // If `false` or omitted, only `is_final=true` result(s) are returned.
    bool interim_results = 2;
}

// The only message returned to the client by the `Recognize` method. It
// contains the result as zero or more sequential `SpeechRecognitionResult`
// messages.
message RecognizeResponse {
    // Sequential list of transcription results corresponding to
        // sequential portions of audio. Currently only returns one transcript.
    repeated SpeechRecognitionResult results = 1;
}

// A speech recognition result corresponding to the latest transcript
message SpeechRecognitionResult {

    // May contain one or more recognition hypotheses (up to the
    // maximum specified in `max_alternatives`).
    // These alternatives are ordered in terms of accuracy, with the top (first)
    // alternative being the most probable, as ranked by the recognizer.
    repeated SpeechRecognitionAlternative alternatives = 1;

    // For multi-channel audio, this is the channel number corresponding to the
    // recognized result for the audio from that channel.
    // For audio_channel_count = N, its output values can range from '1' to 'N'.
    int32 channel_tag = 2;
```

(continues on next page)

(continued from previous page)

```

    // Length of audio processed so far in seconds
    float audio_processed = 3;
}

// Alternative hypotheses (a.k.a. n-best list).
message SpeechRecognitionAlternative {
    // Transcript text representing the words that the user spoke.
    string transcript = 1;

    // The non-normalized confidence estimate. A higher number
    // indicates an estimated greater likelihood that the recognized words are
    // correct. This field is set only for a non-streaming
    // result or, of a streaming result where `is_final=true`.
    // This field is not guaranteed to be accurate and users should not rely on it
    // to be always provided.
    float confidence = 2;

    // A list of word-specific information for each recognized word. Only populated
    // if is_final=true
    repeated WordInfo words = 3;
}

// Word-specific information for recognized words.
message WordInfo {
    // Time offset relative to the beginning of the audio in ms
    // and corresponding to the start of the spoken word.
    // This field is only set if `enable_word_time_offsets=true` and only
    // in the top hypothesis.
    int32 start_time = 1;

    // Time offset relative to the beginning of the audio in ms
    // and corresponding to the end of the spoken word.
    // This field is only set if `enable_word_time_offsets=true` and only
    // in the top hypothesis.
    int32 end_time = 2;

    // The word corresponding to this set of information.
    string word = 3;
}


// `StreamingRecognizeResponse` is the only message returned to the client by
// `StreamingRecognize`. A series of zero or more `StreamingRecognizeResponse`
// messages are streamed back to the client.
//
// Here are few examples of `StreamingRecognizeResponse`s
//
// 1. results { alternatives { transcript: "tube" } stability: 0.01 }
//
// 2. results { alternatives { transcript: "to be a" } stability: 0.01 }
//
```

(continues on next page)

(continued from previous page)

```
// 3. results { alternatives { transcript: "to be or not to be"
//                             confidence: 0.92 }
//              alternatives { transcript: "to bee or not to bee" }
//              is_final: true }
//

message StreamingRecognizeResponse {

    // This repeated list contains the latest transcript(s) corresponding to
    // audio currently being processed.
                // Currently one result is returned, where each result can have multiple
                // alternatives
    repeated StreamingRecognitionResult results = 1;
}

// A streaming speech recognition result corresponding to a portion of the audio
// that is currently being processed.
message StreamingRecognitionResult {
    // May contain one or more recognition hypotheses (up to the
    // maximum specified in `max_alternatives`).
    // These alternatives are ordered in terms of accuracy, with the top (first)
    // alternative being the most probable, as ranked by the recognizer.
    repeated SpeechRecognitionAlternative alternatives = 1;

    // If `false`, this `StreamingRecognitionResult` represents an
    // interim result that may change. If `true`, this is the final time the
    // speech service will return this particular `StreamingRecognitionResult`,
    // the recognizer will not return any further hypotheses for this portion of
    // the transcript and corresponding audio.
    bool is_final = 2;

    // An estimate of the likelihood that the recognizer will not
    // change its guess about this interim result. Values range from 0.0
    // (completely unstable) to 1.0 (completely stable).
    // This field is only provided for interim results (`is_final=false`).
    // The default of 0.0 is a sentinel value indicating `stability` was not set.
    float stability = 3;

    // For multi-channel audio, this is the channel number corresponding to the
    // recognized result for the audio from that channel.
    // For audio_channel_count = N, its output values can range from '1' to 'N'.
    int32 channel_tag = 5;

    // Length of audio processed so far in seconds
    float audio_processed = 6;
}
```

32.2 riva_nlp.proto

```
// Copyright (c) 2021, NVIDIA CORPORATION.  All rights reserved.
//
// NVIDIA CORPORATION and its licensors retain all intellectual property
// and proprietary rights in and to this software, related documentation
// and any modifications thereto.  Any use, reproduction, disclosure or
// distribution of this software and related documentation without an express
// license agreement from NVIDIA CORPORATION is strictly prohibited.

syntax = "proto3";

package nvidia.riva.nlp;

option cc_enable_arenas = true;
option go_package = "nvidia.com/riva_speech";

/* Riva Natural Language Services implement generic and task-specific APIs.
* The generic APIs allows users to design
* models for arbitrary use cases that conform simply with input and output types
* specified in the service. As an explicit example, the ClassifyText function
* could be used for sentiment classification, domain recognition, language
* identification, etc.
* The task-specific APIs can be used for popular NLP tasks such as
* intent recognition (as well as slot filling), and entity extraction.
*/

service RivaLanguageUnderstanding {

    // ClassifyText takes as input an input/query string and parameters related
    // to the requested model to use to evaluate the text. The service evaluates the
    // text with the requested model, and returns one or more classifications.
    rpc ClassifyText(TextClassRequest) returns (TextClassResponse) {}

    // ClassifyTokens takes as input either a string or list of tokens and parameters
    // related to which model to use. The service evaluates the text with the requested
    // model, performing additional tokenization if necessary, and returns one or more
    // class labels per token.
    rpc ClassifyTokens(TokenClassRequest) returns (TokenClassResponse) {}

    // TransformText takes an input/query string and parameters related to the
    // requested model and returns another string. The behavior of the function
    // is defined entirely by the underlying model and may be used for
    // tasks like translation, adding punctuation, augment the input directly, etc.
    rpc TransformText(TextTransformRequest) returns (TextTransformResponse) {}

    // AnalyzeEntities accepts an input string and returns all named entities within
    // the text, as well as a category and likelihood.
    rpc AnalyzeEntities(AnalyzeEntitiesRequest) returns (TokenClassResponse) {}

    // AnalyzeIntent accepts an input string and returns the most likely
    // intent as well as slots relevant to that intent.
```

(continues on next page)

(continued from previous page)

```
    //
    // The model requires that a valid "domain" be passed in, and optionally
    // supports including a previous intent classification result to provide
    // context for the model.
    rpc AnalyzeIntent(AnalyzeIntentRequest) returns (AnalyzeIntentResponse) {}

    // PunctuateText takes text with no- or limited- punctuation and returns
    // the same text with corrected punctuation and capitalization.
    rpc PunctuateText(TextTransformRequest) returns (TextTransformResponse) {}

    // NaturalQuery is a search function that enables querying one or more documents
    // or contexts with a query that is written in natural language.
    rpc NaturalQuery(NaturalQueryRequest) returns (NaturalQueryResponse) {}
}

// NLPModelParams is a metadata message that is included in every request message
// used by the Core NLP Service and is used to specify model characteristics/requirements
message NLPModelParams {
    // Requested model to use. If unavailable, the request will return an error
    string model_name = 1;
}

// TextTransformRequest is a request type intended for services like TransformText
// which take an arbitrary text input
message TextTransformRequest {
    // Each repeated text element is handled independently for handling multiple
    // input strings with a single request
    repeated string text = 1;
    uint32 top_n = 2; //
    NLPModelParams model = 3;
}

// TextTransformResponse is returned by the TransformText method. Responses
// are returned in the same order as they were requested.
message TextTransformResponse {
    repeated string text = 1;
}

// TextClassRequest is the input message to the ClassifyText service.
message TextClassRequest {
    // Each repeated text element is handled independently for handling multiple
    // input strings with a single request
    repeated string text = 1;

    // Return the top N classification results for each input. 0 or 1 will return top␣
↪class, otherwise N.
    // Note: Current disabled.
    uint32 top_n = 2;
    NLPModelParams model = 3;
}

// Classification messages return a class name and corresponding score
```

(continues on next page)

(continued from previous page)

```
message Classification {
    string class_name = 1;
    float score = 2;
}

// Span of a particular result
message Span {
    uint32 start = 1;
    uint32 end = 2;
}

// ClassificationResults contain zero or more Classification messages
// If the number of Classifications is > 1, top_n > 1 must have been
// specified.
message ClassificationResult {
    repeated Classification labels = 1;
}

// TextClassResponse is the return message from the ClassifyText service.
message TextClassResponse {
    repeated ClassificationResult results = 1;
}

// TokenClassRequest is the input message to the ClassifyText service.
message TokenClassRequest {
    // Each repeated text element is handled independently for handling multiple
    // input strings with a single request
    repeated string text = 1;

    // Return the top N classification results for each input. 0 or 1 will return top class, otherwise N.
    // Note: Current disabled.
    uint32 top_n = 3;
    NLPModelParams model = 4;
}

// TokenClassValue is used to correlate an input token with its classification results
message TokenClassValue {
    string token = 1;
    repeated Classification label = 2;
    repeated Span span = 3;
}

// TokenClassSequence is used for returning a sequence of TokenClassValue objects
// in the original order of input tokens
message TokenClassSequence {
    repeated TokenClassValue results = 1;
}

// TokenClassResponse returns a single TokenClassSequence per input request
message TokenClassResponse {
    repeated TokenClassSequence results = 1;
```

(continues on next page)

(continued from previous page)

```
}

// AnalyzeIntentContext is reserved for future use when we may send context back in a
// a variety of different formats (including raw neural network hidden states)
message AnalyzeIntentContext {
    // Reserved for future use
}

// AnalyzeIntentOptions is an optional configuration message to be sent as part of
// an AnalyzeIntentRequest with query metadata
message AnalyzeIntentOptions {
    // Optionally provide context from previous interactions to bias the model's prediction
    oneof context {
        string previous_intent = 1;
        AnalyzeIntentContext vectors = 2;
    }
    // Optional domain field. Domain must be supported otherwise an error will be returned.
    // If left blank, a domain detector will be run first and then the query routed to the
    // appropriate intent classifier (if it exists)
    string domain = 3;

    // Optional language field. Assumed to be "en-US" if not specified.
    string lang = 4;
}

// AnalyzeIntentRequest is the input message for the AnalyzeIntent service
message AnalyzeIntentRequest {
    // The string to analyze for intent and slots
    string query = 1;
    // Optional configuration for the request, including providing context from previous turns
    // and hardcoding a domain/language
    AnalyzeIntentOptions options = 2;
}

// AnalyzeIntentResponse is returned by the AnalyzeIntent service, and includes information
// related to the query's intent, (optionally) slot data, and its domain.
message AnalyzeIntentResponse {
    // Intent classification result, including the label and score
    Classification intent = 1;
    // List of tokens explicitly marked as filling a slot relevant to the intent, where the
    // tokens may not exactly match the input (based on the recombined values after tokenization)
    repeated TokenClassValue slots = 2;
    // Returns the inferred domain for the query if not hardcoded in the request. In the case where
    // the domain was hardcoded in AnalyzeIntentRequest, the returned domain is an exact match to the
```

(continues on next page)

(continued from previous page)

```
    // request. In the case where no domain matches the query, intent and slots will be unset.
    //
    // DEPRECATED, use Classification domain field.
    string domain_str = 3;

    // Returns the inferred domain for the query if not hardcoded in the request. In the case where
    // the domain was hardcoded in AnalyzeIntentRequest, the returned domain is an exact match to the
    // request. In the case where no domain matches the query, intent and slots will be unset.
    Classification domain = 4;
}

// AnalyzeEntitiesOptions is an optional configuration message to be sent as part of
// an AnalyzeEntitiesRequest with query metadata
message AnalyzeEntitiesOptions {
    // Optional language field. Assumed to be "en-US" if not specified.
    string lang = 4;
}

// AnalyzeEntitiesRequest is the input message for the AnalyzeEntities service
message AnalyzeEntitiesRequest {
    // The string to analyze for intent and slots
    string query = 1;
    // Optional configuration for the request, including providing context from previous turns
    // and hardcoding a domain/language
    AnalyzeEntitiesOptions options = 2;
}

message NaturalQueryRequest {
    // The natural language query
    string query = 1;

    // Maximum number of answers to return for the query. Defaults to 1 if not set.
    uint32 top_n = 2;

    // Context to search with the above query
    string context = 3;
}

message NaturalQueryResult {
    // text which answers the query
    string answer = 1;
    // Score representing confidence in result
    float score = 2;
}

message NaturalQueryResponse {
    repeated NaturalQueryResult results = 1;
```

(continues on next page)

(continued from previous page)

```
}
```

32.3 riva_tts.proto

```
// Copyright (c) 2021, NVIDIA CORPORATION.  All rights reserved.
//
// NVIDIA CORPORATION and its licensors retain all intellectual property
// and proprietary rights in and to this software, related documentation
// and any modifications thereto.  Any use, reproduction, disclosure or
// distribution of this software and related documentation without an express
// license agreement from NVIDIA CORPORATION is strictly prohibited.


syntax = "proto3";

package nvidia.riva.tts;

option cc_enable_arenas = true;
option go_package = "nvidia.com/riva_speech";

import "riva_audio.proto";

service RivaSpeechSynthesis {
    // Used to request speech-to-text from the service. Submit a request containing the
    // desired text and configuration, and receive audio bytes in the requested format.
    rpc Synthesize(SynthesizeSpeechRequest) returns (SynthesizeSpeechResponse) {}

    // Used to request speech-to-text returned via stream as it becomes available.
    // Submit a SynthesizeSpeechRequest with desired text and configuration,
    // and receive stream of bytes in the requested format.
    rpc SynthesizeOnline(SynthesizeSpeechRequest) returns (stream SynthesizeSpeechResponse) {}
}

message SynthesizeSpeechRequest {
    string text = 1;
    string language_code = 2;
    // audio encoding params
    AudioEncoding encoding = 3;
    int32 sample_rate_hz = 4;
    // voice params
    string voice_name = 5;
}

message SynthesizeSpeechResponse {
    bytes audio = 1;
}

/*
*
```

(continues on next page)

(continued from previous page)

```
*/
```

32.4 riva_audio.proto

```
// Copyright (c) 2021, NVIDIA CORPORATION.  All rights reserved.
// NVIDIA CORPORATION and its licensors retain all intellectual property
// and proprietary rights in and to this software, related documentation
// and any modifications thereto.  Any use, reproduction, disclosure or
// distribution of this software and related documentation without an express
// license agreement from NVIDIA CORPORATION is strictly prohibited.


syntax = "proto3";

package nvidia.riva;

option cc_enable_arenas = true;


/*
 * AudioEncoding specifies the encoding of the audio bytes in the encapsulating␣
↪message.
 */
enum AudioEncoding {
        // Not specified.
        ENCODING_UNSPECIFIED = 0;

        // Uncompressed 16-bit signed little-endian samples (Linear PCM).
        LINEAR_PCM = 1;

        // `FLAC` (Free Lossless Audio
        // Codec) is the recommended encoding because it is
        // lossless--therefore recognition is not compromised--and
        // requires only about half the bandwidth of `LINEAR16`. `FLAC` stream
        // encoding supports 16-bit and 24-bit samples, however, not all fields in
        // `STREAMINFO` are supported.
        FLAC = 2;

        // 8-bit samples that compand 14-bit audio samples using G.711 PCMU/mu-law.
        MULAW = 3;

        // 8-bit samples that compand 13-bit audio samples using G.711 PCMU/a-law.
        ALAW = 20;
}
```

32.5 health.proto

```
// Copyright (c) 2021, NVIDIA CORPORATION.  All rights reserved.
//
// NVIDIA CORPORATION and its licensors retain all intellectual property
// and proprietary rights in and to this software, related documentation
// and any modifications thereto.  Any use, reproduction, disclosure or
// distribution of this software and related documentation without an express
// license agreement from NVIDIA CORPORATION is strictly prohibited.


//
//Based on gRPC health check protocol - more details found here:
//https://github.com/grpc/grpc/blob/master/doc/health-checking.md
//

syntax = "proto3";
option go_package = "nvidia.com/riva_speech";

package grpc.health.v1;


option cc_enable_arenas = true;

service Health {
rpc Check(HealthCheckRequest) returns (HealthCheckResponse);
rpc Watch(HealthCheckRequest) returns (stream HealthCheckResponse);
}

message HealthCheckRequest {
    string service = 1;
}

message HealthCheckResponse {
enum ServingStatus {
    UNKNOWN = 0;
    SERVING = 1;
    NOT_SERVING = 2;
}
ServingStatus status = 1;
}
```

See also:

- gRPC API
- Python API
- gRPC & Protocol Buffers

CHAPTER

THIRTYTHREE

RIVA - PYTHON API - SAMPLE SERVICES

This session illustrates the samples that demonstrate how Riva clients use Riva services through Riva Speech API server. To facilate the understanding of the usage of Riva Speech API, you might read Python API

33.1 Prerequisites

To use the Riva Speech API from the client side, install `riva-speech` Python wheel from Riva Quick Start scripts (Riva 1.5.0 is used in this demo).

```
pip install riva_api-1.5.0-beta-py3-none-any.whl
```

Note: Refer *Riva - Getting Started* to download Riva Quick Start scripts.

After installing `riva_api-1.5.0-beta-py3-none-any.whl`, run `riva_start_client.sh` to start a client container

```
bash riva_start_client.sh
```

You may then host a jupyter server, and create notebooks to use the sample services.

```
jupyter notebook --allow-root --ip=0.0.0.0
```

33.2 ASR - Recognize

1. Import necessary libraries

```
import grpc
import librosa
import io
import IPython.display as ipd
import riva_api.riva_asr_pb2 as rasr
import riva_api.riva_asr_pb2_grpc as rasr_srv
import riva_api.riva_audio_pb2 as ra
```

2. Establish connections between client and server

```
channel = grpc.insecure_channel("localhost:50051")
server = rasr_srv.RivaSpeechRecognitionStub(channel)
```

3. Read audio

```
path = "/work/wav/sample.wav"
audio, sr = librosa.core.load(path, sr=None)
with io.open(path, 'rb') as fh:
        content = fh.read()
```

Note:

- Besides `.wav` audio files, Riva also accept `.alaw`, `.mulaw`, `.flac` formatted files with single channel.

4. Create a `RecognizeRequest`

```
req = rasr.RecognizeRequest()
req.audio = content
req.config.encoding = ra.AudioEncoding.LINEAR_PCM
req.config.sample_rate_hertz = sr
req.config.language_code = "en-US"
req.config.max_alternatives = 1
req.config.enable_automatic_punctuation = True
req.config.audio_channel_count = 1
```

See also:

- *riva_asr.proto*

```
message RecognizeRequest {
  // Provides information to recognizer that specifies how to process the request.
  RecognitionConfig config = 1;
  // The raw audio data to be processed. The audio bytes must be encoded as specified in
  // `RecognitionConfig`.
  bytes audio = 2;
}
```

```
message RecognitionConfig {
    // The encoding of the audio data sent in the request.
    //
    // All encodings support only 1 channel (mono) audio.
    AudioEncoding encoding = 1;

    //  Sample rate in Hertz of the audio data sent in all
    // `RecognizeAudio` messages.
    int32 sample_rate_hertz = 2;

    // Required. The language of the supplied audio as a
    // [BCP-47](https://www.rfc-editor.org/rfc/bcp/bcp47.txt) language tag.
    // Example: "en-US".
    // Currently only en-US is supported
    string language_code = 3;
```

(continues on next page)

(continued from previous page)

```
    // Maximum number of recognition hypotheses to be returned.
    // Specifically, the maximum number of `SpeechRecognizeAlternative` messages
    // within each `SpeechRecognizeResult`.
    // The server may return fewer than `max_alternatives`.
    // If omitted, will return a maximum of one.
    int32 max_alternatives = 4;

    // The number of channels in the input audio data.
    // ONLY set this for MULTI-CHANNEL recognition.
    // Valid values for LINEAR16 and FLAC are `1`-`8`.
    // Valid values for OGG_OPUS are '1'-'254'.
    // Valid value for MULAW, AMR, AMR_WB and SPEEX_WITH_HEADER_BYTE is only `1`.
    // If `0` or omitted, defaults to one channel (mono).
    // Note: We only recognize the first channel by default.
    // To perform independent recognition on each channel set
    // `enable_separate_recognition_per_channel` to 'true'.
    int32 audio_channel_count = 7;

  // If `true`, the top result includes a list of words and
    // the start and end time offsets (timestamps) for those words. If
    // `false`, no word-level time offset information is returned. The default is
    // `false`.
    bool enable_word_time_offsets = 8;

    // If 'true', adds punctuation to recognition result hypotheses.
    // The default 'false' value does not add punctuation to result hypotheses.
    bool enable_automatic_punctuation = 11;

    // This needs to be set to `true` explicitly and `audio_channel_count` > 1
    // to get each channel recognized separately. The recognition result will
    // contain a `channel_tag` field to state which channel that result belongs
    // to. If this is not true, we will only recognize the first channel. The
    // request is billed cumulatively for all channels recognized:
    // `audio_channel_count` multiplied by the length of the audio.
    bool enable_separate_recognition_per_channel = 12;

    // Which model to select for the given request. Valid choices: Jasper, Quartznet
    string model = 13;

    // The verbatim_transcripts flag enables or disable inverse text normalization.
    // 'true' returns exactly what was said, with no denormalization.
    // 'false' applies inverse text normalization, also this is the default
    bool verbatim_transcripts = 14;

    // Custom fields for passing request-level
    // configuration options to plugins used in the
    // model pipeline.
    map<string, string> custom_configuration = 24;


}
```

See also:

- *riva_audio.proto*

```
/*
  * AudioEncoding specifies the encoding of the audio bytes in the encapsulating message.
*/
enum AudioEncoding {
    // Not specified.
    ENCODING_UNSPECIFIED = 0;

    // Uncompressed 16-bit signed little-endian samples (Linear PCM).
    LINEAR_PCM = 1;

    // `FLAC` (Free Lossless Audio
    // Codec) is the recommended encoding because it is
    // lossless--therefore recognition is not compromised--and
    // requires only about half the bandwidth of `LINEAR16`. `FLAC` stream
    // encoding supports 16-bit and 24-bit samples, however, not all fields in
    // `STREAMINFO` are supported.
    FLAC = 2;

    // 8-bit samples that compand 14-bit audio samples using G.711 PCMU/mu-law.
    MULAW = 3;

    // 8-bit samples that compand 13-bit audio samples using G.711 PCMU/a-law.
    ALAW = 20;
}
```

5. Remote procedure call `Recognize`

```
response = server.Recognize(req)
```

See also:

- `Recognize` remote procedure call is defined as:

```
rpc Recognize(RecognizeRequest) returns (RecognizeResponse) {}
```

See also:

- *riva_asr.proto*

```
message RecognizeResponse {
    // Sequential list of transcription results corresponding to
    // sequential portions of audio. Currently only returns one transcript.
    repeated SpeechRecognitionResult results = 1;
}
```

```
// A speech recognition result corresponding to the latest transcript
message SpeechRecognitionResult {

  // May contain one or more recognition hypotheses (up to the
  // maximum specified in `max_alternatives`).
  // These alternatives are ordered in terms of accuracy, with the top (first)
```

(continues on next page)

(continued from previous page)

```
  // alternative being the most probable, as ranked by the recognizer.
  repeated SpeechRecognitionAlternative alternatives = 1;

  // For multi-channel audio, this is the channel number corresponding to the
  // recognized result for the audio from that channel.
  // For audio_channel_count = N, its output values can range from '1' to 'N'.
  int32 channel_tag = 2;

  // Length of audio processed so far in seconds
  float audio_processed = 3;
}
```

```
// Alternative hypotheses (a.k.a. n-best list).
message SpeechRecognitionAlternative {
  // Transcript text representing the words that the user spoke.
  string transcript = 1;

  // The non-normalized confidence estimate. A higher number
  // indicates an estimated greater likelihood that the recognized words are
  // correct. This field is set only for a non-streaming
  // result or, of a streaming result where `is_final=true`.
  // This field is not guaranteed to be accurate and users should not rely on it
  // to be always provided.
  float confidence = 2;

  // A list of word-specific information for each recognized word. Only populated
  // if is_final=true
  repeated WordInfo words = 3;
}
```

6. Print the response from Riva server

```
print(response)
```

```
Full Response Message:
results {
  alternatives {
    transcript: "What is natural language processing? "
    confidence: 1.0
  }
  channel_tag: 1
  audio_processed: 4.800000190734863
}
```

33.3 NLP - Core Services

1. Import necessary libraries

```
import grpc
import riva_api.riva_nlp_pb2 as rnlp
import riva_api.riva_nlp_pb2_grpc as rnlp_srv
```

2. Establish connections between client and server

```
channel = grpc.insecure_channel('localhost:50051')
server = rnlp_srv.RivaLanguageUnderstandingStub(channel)
```

3. Select one of core NLP services below to continue.

Note:

- Notice for every core NLP service, `NLPModelParams model` metadata (name of models to be used) is included in every request to specify model characteristics/ requirements.
- *riva_nlp.proto*

```
// NLPModelParams is a metadata message that is included in every request message
// used by the Core NLP Service and is used to specify model characteristics/requirements
message NLPModelParams {
  // Requested model to use. If unavailable, the request will return an error
  string model_name = 1;
}
```

- You can use `docker logs riva-speech` to check which models are available for services.

33.3.1 TransformText

4. Create a `TextTransformRequest` (string -> string)

```
req = rnlp.TextTransformRequest()
req.model.model_name = "riva_punctuation"
req.text.append("add punctuation to this sentence")
req.text.append("do you have any NVDA stocks")
req.text.append("i buy one apple four oranges and two watermelons "
                "for my after-dinner it's going to be very cool")
```

See also:

- *riva_nlp.proto*

```
message TextTransformRequest {
  // Each repeated text element is handled independently for handling multiple
  // input strings with a single request
  repeated string text = 1;
  uint32 top_n = 2; //
  NLPModelParams model = 3;
}
```

5. remote procedure call `TransformText`

```
nlp_resp = server.TransformText(req)
```

Note:

- `TransformText` remote procedure call is defined as:

```
rpc TransformText(TextTransformRequest) returns (TextTransformResponse) {}
```

- *riva_nlp.proto*

```
// TextTransformResponse is returned by the TransformText method. Responses
// are returned in the same order as they were requested.
message TextTransformResponse {
  repeated string text = 1;
}
```

6. Print the response from Riva server

```
print(nlp_resp)
```

```
text: "Add punctuation to this sentence."
text: "Do you have any NVDA stocks?"
text: "I buy one apple, four oranges and two watermelons for my after-dinner It\'s going␣
↪to be very cool."
```

33.3.2 ClassifyText

4. Create a `TextClassRequest` (text -> label)

```
request = rnlp.TextClassRequest()
request.model.model_name = "riva_text_classification_domain"
request.text.append("Is it going to snow in Burlington, Vermont tomorrow night?")
request.text.append("What causes rain?")
request.text.append("What is your favorite season?")
```

See also:

- *riva_nlp.proto*

```
// TextClassRequest is the input message to the ClassifyText service.
message TextClassRequest {
  // Each repeated text element is handled independently for handling multiple
  // input strings with a single request
  repeated string text = 1;

  // Return the top N classification results for each input. 0 or 1 will return top␣
↪class, otherwise N.
  // Note: Current disabled.
  uint32 top_n = 2;
```

(continues on next page)

(continued from previous page)

```
  NLPModelParams model = 3;
}
```

5. Remote procedure call ClassifyText

```
ct_response = riva_nlp.ClassifyText(request)
```

Note:

- ClassifyText remote procedure call is defined as:

```
rpc ClassifyText(TextClassRequest) returns (TextClassResponse) {}
```

- *riva_nlp.proto*

```
message TextClassResponse {
  repeated ClassificationResult results = 1;
}
```

```
// ClassificationResults contain zero or more Classification messages
// If the number of Classifications is > 1, top_n > 1 must have been
// specified.
message ClassificationResult {
  repeated Classification labels = 1;
}
```

```
// Classification messages return a class name and corresponding score
message Classification {
  string class_name = 1;
  float score = 2;
}
```

6. Print the response from the Riva Server

```
print(ct_response.results)
```

```
[labels {
  class_name: "weather"
  score: 0.9975590109825134
}
, labels {
  class_name: "meteorology"
  score: 0.984375
}
, labels {
  class_name: "personality"
  score: 0.984375
}
]
```

33.3.3 ClassifyTokens

4. Create a `TokenClassRequest` (token -> label)

```
req = rnlp.TokenClassRequest()
req.model.model_name = "riva_ner"
req.text.append("Jensen Huang is the CEO of NVIDIA Corporation, "
            "located in Santa Clara, California")
```

See also:

- *riva_nlp.proto*

```
// TokenClassRequest is the input message to the ClassifyText service.
message TokenClassRequest {
  // Each repeated text element is handled independently for handling multiple
  // input strings with a single request
  repeated string text = 1;

  // Return the top N classification results for each input. 0 or 1 will return top␣
↪class, otherwise N.
  // Note: Current disabled.
  uint32 top_n = 3;
  NLPModelParams model = 4;
}
```

5. Remote procedure call `ClassifyTokens`

```
resp = server.ClassifyTokens(req)
```

Note:

- `ClassifyTokens` remote procedure call is defined as:

```
rpc ClassifyTokens(TokenClassRequest) returns (TokenClassResponse) {}
```

- *riva_nlp.proto*

```
// TokenClassResponse returns a single TokenClassSequence per input request
message TokenClassResponse {
  repeated TokenClassSequence results = 1;
}
```

```
// TokenClassSequence is used for returning a sequence of TokenClassValue objects
// in the original order of input tokens
message TokenClassSequence {
  repeated TokenClassValue results = 1;
}
```

```
// TokenClassValue is used to correlate an input token with its classification results
message TokenClassValue {
  string token = 1;
  repeated Classification label = 2;
```

(continues on next page)

(continued from previous page)

```
  repeated Span span = 3;
}
```

```
// Span of a particular result
message Span {
  uint32 start = 1;
  uint32 end = 2;
}
```

6. Print the response from the Riva server

```
print(resp)
```

```
results {
  results {
    token: "jensen huang"
    label {
      class_name: "PER"
      score: 0.9972559809684753
    }
    span {
      end: 11
    }
  }
  results {
    token: "nvidia corporation"
    label {
      class_name: "ORG"
      score: 0.9613900184631348
    }
    span {
      start: 27
      end: 44
    }
  }
  results {
    token: "santa clara"
    label {
      class_name: "LOC"
      score: 0.9977059960365295
    }
    span {
      start: 58
      end: 68
    }
  }
  results {
    token: "california"
    label {
      class_name: "LOC"
      score: 0.9961509704589844
```

(continues on next page)

(continued from previous page)

```
    }
    span {
      start: 71
      end: 80
    }
  }
}
results {
  results {
    token: "hku"
    label {
      class_name: "ORG"
      score: 0.830374002456665
    }
    span {
      start: 26
      end: 28
    }
  }
}
```

33.4 NLP - AnalyzeIntent

1. Import necessary libraries

```python
import grpc
import riva_api.riva_nlp_pb2_grpc as rnlp_srv
import riva_api.riva_nlp_pb2 as rnlp
```

2. Establish connections between client and server

```python
channel = grpc.insecure_channel('localhost:50051')
server = rnlp_srv.RivaLanguageUnderstandingStub(channel)
```

3. Create a `AnalyzeIntentRequest`

```python
req = rnlp.AnalyzeIntentRequest(query="How is the weather today in Hong Kong")
```

See also:

- *riva_nlp.proto*

```
// AnalyzeIntentRequest is the input message for the AnalyzeIntent service
message AnalyzeIntentRequest {
  // The string to analyze for intent and slots
  string query = 1;
  // Optional configuration for the request, including providing context from previous␣
↪turns
  // and hardcoding a domain/language
  AnalyzeIntentOptions options = 2;
}
```

4. Remote procedure call AnalyzeIntent

```
resp = server.AnalyzeIntent(req)
```

Note:

- AnalyzeIntent remote procedure call is defined as:

```
rpc AnalyzeIntent(AnalyzeIntentRequest) returns (AnalyzeIntentResponse) {}
```

See also:

- *riva_nlp.proto*

```
// AnalyzeIntentResponse is returned by the AnalyzeIntent service, and includes␣
↪information
// related to the query's intent, (optionally) slot data, and its domain.
message AnalyzeIntentResponse {
  // Intent classification result, including the label and score
  Classification intent = 1;
  // List of tokens explicitly marked as filling a slot relevant to the intent, where the
  // tokens may not exactly match the input (based on the recombined values after␣
↪tokenization)
  repeated TokenClassValue slots = 2;
  // Returns the inferred domain for the query if not hardcoded in the request. In the␣
↪case where
  // the domain was hardcoded in AnalyzeIntentRequest, the returned domain is an exact␣
↪match to the
  // request. In the case where no domain matches the query, intent and slots will be␣
↪unset.
  //
  // DEPRECATED, use Classification domain field.
  string domain_str = 3;

  // Returns the inferred domain for the query if not hardcoded in the request. In the␣
↪case where
  // the domain was hardcoded in AnalyzeIntentRequest, the returned domain is an exact␣
↪match to the
  // request. In the case where no domain matches the query, intent and slots will be␣
↪unset.
  Classification domain = 4;
}
```

5. Print the response from Riva server:

```
print(resp)
```

```
intent {
  class_name: "weather.weather"
  score: 0.9965819716453552
}
slots {
  token: "today"
```

(continues on next page)

(continued from previous page)

```
  label {
    class_name: "weatherforecastdaily"
    score: 0.8901410102844238
  }
}
slots {
  token: "hong"
  label {
    class_name: "weatherplace"
    score: 0.7496770024299622
  }
}
slots {
  token: "kong"
  label {
    class_name: "weatherplace"
    score: 0.5112429857254028
  }
}
domain_str: "weather"
domain {
  class_name: "weather"
  score: 0.9970700144767761
}
```

33.5 TTS - SynthesizeSpeech

1. Import necessary libraries

```
import grpc
import numpy as np
import IPython.display as ipd
import riva_api.riva_tts_pb2 as rtts
import riva_api.riva_tts_pb2_grpc as rtts_srv
import riva_api.audio_pb2 as ra
```

2. Establish connections between client and server

```
channel = grpc.insecure_channel("localhost:50051")
server = rtts_srv.RivaSpeechSynthesisStub(channel)
```

3. Create a `SynthesizeSpeechRequest`

```
req = rtts.SynthesizeSpeechRequest()
req.text = "Buy Nvidia stocks, You will get rich"
req.language_code = "en-US"
req.encoding = ra.AudioEncoding.LINEAR_PCM
req.sample_rate_hz = 22050
req.voice_name = "ljspeech"
```

See also:

- *riva_tts.proto*

```
message SynthesizeSpeechRequest {
    string text = 1;
    string language_code = 2;
    // audio encoding params
    AudioEncoding encoding = 3;
    int32 sample_rate_hz = 4;
    // voice params
    string voice_name = 5;
}
```

4. Remote procedure call SyntheSize

```
resp = server.Synthesize(req)
```

Note:

- SyntheSize remote procedure call is defined as:

```
rpc Synthesize(SynthesizeSpeechRequest) returns (SynthesizeSpeechResponse) {}
```

```
message SynthesizeSpeechResponse {
    bytes audio = 1;
}
```

---

5. Convert buffer to numpy array and listen

```
audio_samples = np.frombuffer(resp.audio, dtype=np.float32)
ipd.Audio(audio_samples, rate=22050)
```

CHAPTER

THIRTYFOUR

RIVA - PYTHON CLIENT

34.1 asr_client.py

```
import argparse
import wave
import sys
import grpc
import time
import riva_api.audio_pb2 as ra
import riva_api.riva_asr_pb2 as rasr
import riva_api.riva_asr_pb2_grpc as rasr_srv


def get_args():
    parser = argparse.ArgumentParser(description="Streaming transcription via Riva AI␣
↪Services")
    parser.add_argument("--server", default="localhost:50051", type=str, help="URI to␣
↪GRPC server endpoint")
    parser.add_argument("--audio-file", required=True, help="path to local file to stream
↪")
    parser.add_argument(
        "--show-intermediate", action="store_true", help="show intermediate transcripts␣
↪as they are available"
    )
    return parser.parse_args()


def listen_print_loop(responses, show_intermediate=False):
    num_chars_printed = 0
    idx = 0
    for response in responses:
        idx += 1
        if not response.results:
            continue

        result = response.results[0]
        if not result.alternatives:
            continue

        transcript = result.alternatives[0].transcript
```

(continues on next page)

(continued from previous page)

```
        if show_intermediate:
            overwrite_chars = ' ' * (num_chars_printed - len(transcript))

            if not result.is_final:
                sys.stdout.write(">> " + transcript + overwrite_chars + '\r')
                sys.stdout.flush()

                num_chars_printed = len(transcript) + 3

            else:
                print("## " + transcript + overwrite_chars + "\n")
                num_chars_printed = 0
        else:
            if result.is_final:
                print(f"## {transcript.encode('utf-8')}\n")
                sys.stdout.buffer.write(transcript.encode('utf-8'))

CHUNK = 1024
args = get_args()
wf = wave.open(args.audio_file, 'rb')

channel = grpc.insecure_channel(args.server)
client = rasr_srv.RivaSpeechRecognitionStub(channel)


config = rasr.RecognitionConfig(
    encoding=ra.AudioEncoding.LINEAR_PCM,
    sample_rate_hertz=wf.getframerate(),
    language_code="en-US",
    max_alternatives=1,
    enable_automatic_punctuation=True,
)
streaming_config = rasr.StreamingRecognitionConfig(config=config, interim_results=True)

# read data
def generator(w, s):
    yield rasr.StreamingRecognizeRequest(streaming_config=s)
    d = w.readframes(CHUNK)
    while len(d) > 0:
        yield rasr.StreamingRecognizeRequest(audio_content=d)
        d = w.readframes(CHUNK)


responses = client.StreamingRecognize(generator(wf, streaming_config))
listen_print_loop(responses, show_intermediate=args.show_intermediate)
```

34.2 Riva ASR service

```
python3 $RIVA_QS/asr_client.py --audio-file $path
```

CHAPTER

THIRTYFIVE

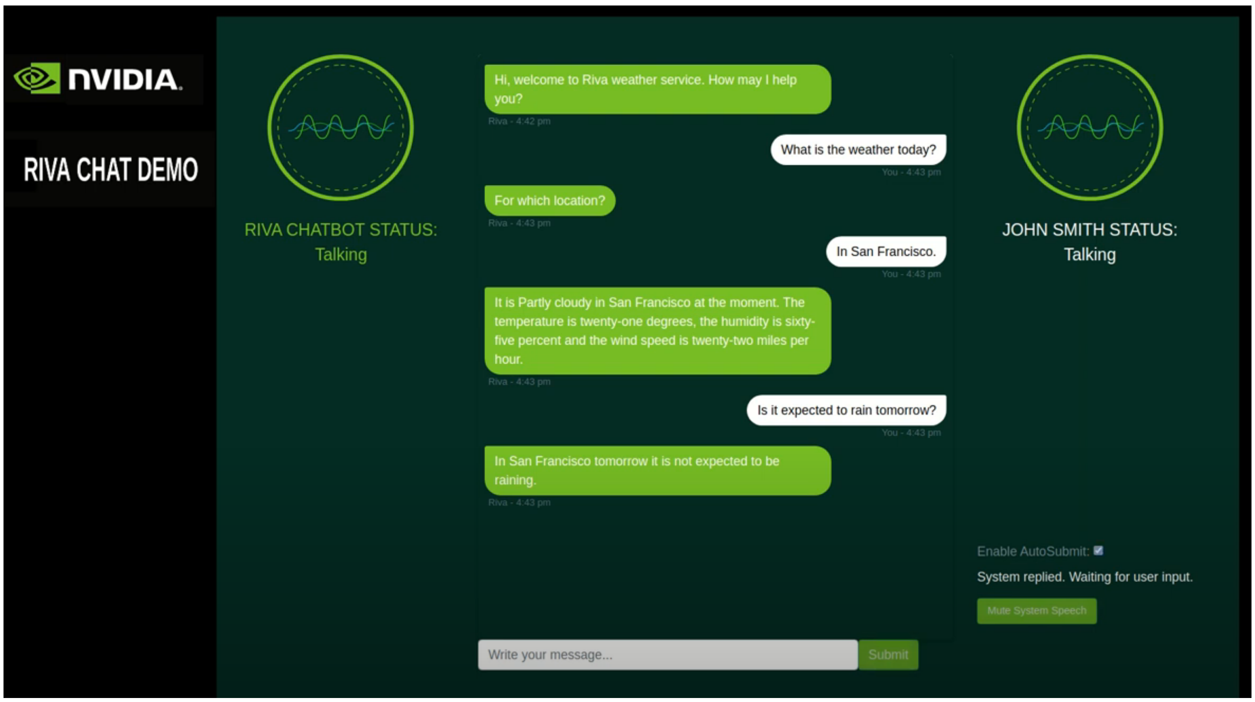
RIVA - STREAMING ASR

```
python3 $RIVA_QS/examples/riva_streaming_asr_client.py  --input-file $path
```

CHAPTER THIRTYSIX

RIVA - VIRTUAL ASSISTANT

This sample implementation of conversational AI describes how to build Virtual Assistant using NGC sample image.



To see more details, click

- Riva - Sample Virtual Assistant

36.1 Start Riva services

Follow *Riva - Getting Started* and start Riva services

```
bash riva_start.sh
```

36.2 Download sample image

Make sure your NGC API is configured before pulling the sample image

```
docker pull nvcr.io/nvidia/riva/riva-speech-client:1.4.0-beta-samples
```

36.3 Run Riva service

First we create a container from the sample image we just pulled from NGC.

```
docker run  -it --rm -p 8009:8009 nvcr.io/nvidia/riva/riva-speech-client:1.4.0-beta-
↪samples /bin/bash
```

Edit `riva_config` in `samples/virtual-assistant/config.py`:

```
riva_config = {
    "RIVA_SPEECH_API_URL": "<riva-host-ip>:50051",
    "ENABLE_QA": "QA unavailable in this VA version. Coming soon",
    "WEATHERSTACK_ACCESS_KEY": "<weather-api-key>",
    "VERBOSE": True  # print logs/details for diagnostics
}
```

- `RIVA_SPEECH_API_URL`: Replace the IP & port with your hosted Riva endpoint
- `WEATHERSTACK_ACCESS_KEY`: Get your access key at here

36.4 Start Virtual Assistant app

After configuring `config.py`, we can now start the web application of Virtual Assistant.

```
python3 main.py
```

Visit *https://<riva-host-ip>:8009/rivaWeather* to use Virtual Assistant.

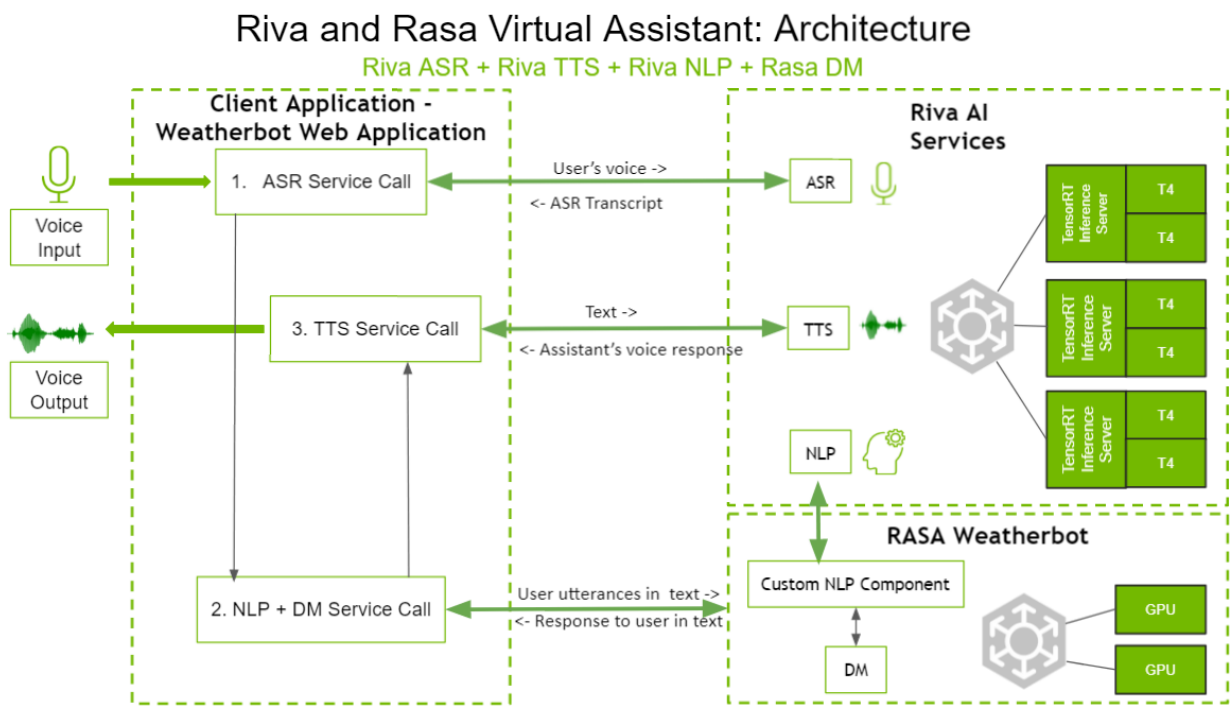
CHAPTER

THIRTYSEVEN

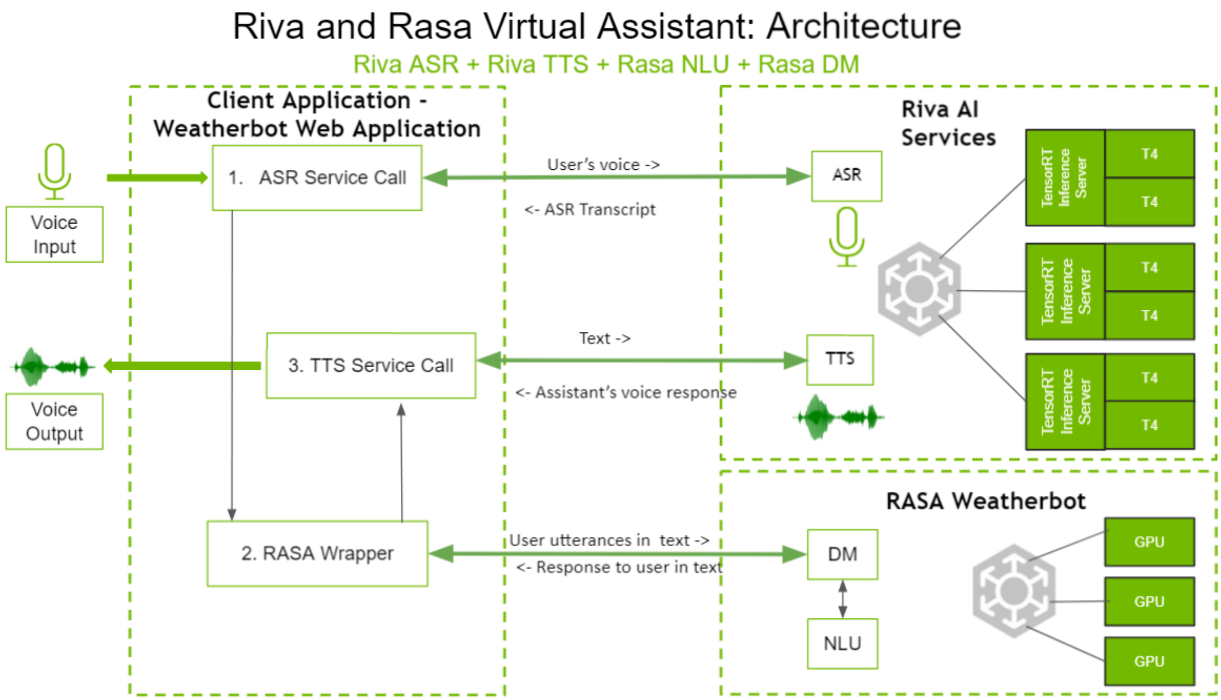
RIVA - VIRTUAL ASSISTANT - RASA

This sample implementation of conversational AI describes how to build Virtual Assistant with Rasa dialogue manager hosted.

Option 1: Riva ASR + Riva TTS + Riva NLP + Rasa dialog manager



Option 2: Riva ASR + Riva TTS + Rasa NLU + Rasa dialog manager



37.1 Overview of RVA Rasa

```
.
|-- README.md
|-- config.py
|-- rasa-riva-weatherbot-webapp
|   |-- client
|   |   `-- webapplication
|   |       |-- cert.pem
|   |       |-- key.pem
|   |       |-- server
|   |       |   |-- __init__.py
|   |       |   `-- server.py
|   |       |-- start_web_application.py
|   |       `-- ui
|   |           |-- README.md
|   |           |-- img
|   |           |   |-- Rivadm.png
|   |           |   `-- User.png
|   |           |-- index.html
|   |           |-- script.js
|   |           |-- static
|   |           |   |-- stylesheets
|   |           |   |   `-- index.css
|   |           |   `-- svg_files
|   |           |       |-- another_sample.svg
|   |           |       |-- circle.svg
|   |           |       |-- logo.svg
```

(continues on next page)

(continued from previous page)

```
|   |           |       |-- logo_sample.svg
|   |           |       |-- nv_logo.svg
|   |           |       |-- nv_logo_1.svg
|   |           |       |-- nvidia_logo.svg
|   |           |       |-- nvidia_name.svg
|   |           |       |-- riva_name.png
|   |           |       |-- sample_.svg
|   |           |       |-- speech.svg
|   |           |       `-- speech_logo.svg
|   |           `-- style.css
|   |-- main.py
|   `-- riva
|       |-- __init__.py
|       |-- asr
|       |   |-- __init__.py
|       |   `-- asr.py
|       |-- chatbot
|       |   |-- __init__.py
|       |   |-- chatbot.py
|       |   `-- chatbots_multiconversations_management.py
|       |-- rasa
|       |   |-- __init__.py
|       |   `-- rasa.py
|       `-- tts
|           |-- __init__.py
|           |-- tts.py
|           `-- tts_stream.py
`-- rasa-weatherbot
    |-- __init__.py
    |-- __pycache__
    |   |-- __init__.cpython-37.pyc
    |   `-- actions.cpython-37.pyc
    |-- actions
    |   |-- __init__.py
    |   |-- actions.py
    |   `-- weather.py
    |-- components.py
    |-- config
    |   |-- config_rasanlp.yml
    |   `-- config_rivanlp.yml
    |-- data
    |   |-- nlu_rasanlp.yml
    |   |-- nlu_rivanlp.yml
    |   |-- rules_rasanlp.yml
    |   |-- rules_rivanlp.yml
    |   |-- stories_rasanlp.yml
    |   `-- stories_rivanlp.yml
    |-- domain
    |   |-- domain_rasanlp.yml
    |   `-- domain_rivanlp.yml
    |-- endpoints.yml
    |-- riva
```

(continues on next page)

(continued from previous page)

```
    |   |-- __init__.py
    |   `-- nlp
    |       |-- __init__.py
    |       `-- nlp.py
    `-- tests
        `-- conversation_tests.md
```

37.1.1 RVA-Rasa Pipeline

Rasa Action Server & Rasa Server

Riva & Web application

rasa-riva-weatherbot-webapp/main.py:

```
import os, sys

root_folder = (os.path.abspath(os.path.join(os.path.dirname(__file__), os.path.pardir)))
sys.path.append(root_folder)

from config import client_config

if __name__ == '__main__':
    if client_config["CLIENT_APPLICATION"] == "WEBAPPLICATION":
        from client.webapplication.start_web_application import start_web_application
        start_web_application()
```

37.1.2 Rasa Server & Rasa Action

37.2 Prerequisites

1. Pull Riva Sample container

```
docker pull nvcr.io/nvidia/riva/riva-speech-client:1.5.0-beta-samples
```

2. Prepare network configuration files

Create a Riva config config.py, and replace RASA_API_URL with the local machine ip you use to host Rasa server.

> **Warning:** Please use local machine ip (not 0.0.0.0/ 127.0.0.1/ localhost)

```
# ==============================================================================
# Copyright (c) 2020, NVIDIA CORPORATION. All rights reserved.
#
# The License information can be found under the "License" section of the
# README.md file.
# ==============================================================================
```

(continues on next page)

(continued from previous page)

```
client_config = {
    "CLIENT_APPLICATION": "WEBAPPLICATION", # Default and only config value for this␣
↪version
    "PORT": 5555, # The port your flask app will be hosted at
    "DEBUG": False, # When this flag is set, the UI displays detailed riva data
    "VERBOSE": True  # print logs/details for diagnostics
}

riva_config = {
    "RIVA_SPEECH_API_URL": "0.0.0.0:50051", # Replace the IP port with your hosted RIVA␣
↪endpoint
    "WEATHERSTACK_ACCESS_KEY": "8a82ce50a069bfaaa3db4427ecca723d",  # Get your access␣
↪key at - https://weatherstack.com/
    "VERBOSE": True  # print logs/details for diagnostics
}

rasa_config = {
    "VERBOSE": False, # Print logs/details for diagnostics
    "RASA_API_URL": "http://<rasa-host-ip>:5005", # Replace the IP & Port with the rasa-
↪weatherbot's IP & Port
}

asr_config = {
    "VERBOSE": False, # Print logs/details for diagnostics
    "SAMPLING_RATE": 16000, # The Sampling Rate for the audio input file. The only value␣
↪currently supported is 16000
    "LANGUAGE_CODE": "en-US", # The language code as a BCP-47 language tag. The only␣
↪value currently supported is "en-US"
    "ENABLE_AUTOMATIC_PUNCTUATION": True, # Enable or Disable punctuation in the␣
↪transcript generated. The only value currently supported by the chatbot is True␣
↪(Although Riva ASR supports both True & False)
}

tts_config = {
    "VERBOSE": True, # Print logs/details for diagnostics
    "SAMPLE_RATE": 22050, # The speech is generated at this sampling rate. The only␣
↪value currently supported is 22050
    "LANGUAGE_CODE": "en-US", # The language code as a BCP-47 language tag. The only␣
↪value currently supported is "en-US"
    "VOICE_NAME": "ljspeech", # The voice name for the speech generated. The only value␣
↪currently supported is "ljspeech"
}

rivanlp_config = {
    "VERBOSE": False, # Print logs/details for diagnostics
    "NLU_FALLBACK_THRESHOLD": 0.3 # When Intent's confidence/score is less than this␣
↪value, intent is set to nlu_fallback
}
```

3. Create a Riva config `endpoints.yml`

```
# ==============================================================================
# Copyright (c) 2020, NVIDIA CORPORATION. All rights reserved.
#
# The License information can be found under the "License" section of the
# README.md file.
# ==============================================================================

# This file contains the different endpoints your bot can use.

# Server where the models are pulled from.
# https://rasa.com/docs/rasa/user-guide/configuring-http-api/#fetching-models-from-a-
↪server/

#models:
#  url: http://my-server.com/models/default_core@latest
#  wait_time_between_pulls:  10   # [optional](default: 100)

# Server which runs your custom actions.
# https://rasa.com/docs/rasa/core/actions/#custom-actions/

action_endpoint:
    url: "http://0.0.0.0:5055/webhook"
# Tracker store which is used to store the conversations.
# By default the conversations are stored in memory.
# https://rasa.com/docs/rasa/api/tracker-stores/

#tracker_store:
#    type: redis
#    url: <host of the redis instance, e.g. localhost>
#    port: <port of your redis instance, usually 6379>
#    db: <number of your database within redis, e.g. 0>
#    password: <password used for authentication>
#    use_ssl: <whether or not the communication is encrypted, default false>

#tracker_store:
#    type: mongod
#    url: <url to your mongo instance, e.g. mongodb://localhost:27017>
#    db: <name of the db within your mongo instance, e.g. rasa>
#    username: <username used for authentication>
#    password: <password used for authentication>

# Event broker which all conversation events should be streamed to.
# https://rasa.com/docs/rasa/api/event-brokers/

#event_broker:
#  url: localhost
#  username: username
#  password: password
#  queue: queue
```

3. Create a local dir `cfg_dir` and store `config.py` and `endpoints.yml` to `cfg_dir`

37.3 1. Start Riva Server

Follow *Riva - Getting Started* to start enable Riva services.

37.4 2. Start Rasa Action Server

2.1 Create Riva Sample container

```
docker run -it --rm -p 5055:5055 -v $(pwd)/cfg_dir:/workspace/cfg nvcr.io/nvidia/riva/
↪riva-speech-client:1.5.0-beta-samples /bin/bash
```

2.2 Replace `/workspace/samples/virtual-assistant-rasa/config.py` with `/workspace/cfg_dir/config.py`

2.3 Replace `/workspace/samples/virtual-assistant-rasa/rasa-weatherbot/endpoints.yml` with `/workspace/cfg_dir/endpoints.yml`

2.4 Activate the Rasa Python environment

```
. /pythonenvs/rasa/bin/activate
```

2.5 Navigate to the `/workspace/samples/virtual-assistant-rasa/rasa-weatherbot` directory and Start Rasa Action server

```
cd /workspace/samples/virtual-assistant-rasa/rasa-weatherbot
rasa run actions --actions actions
```

37.5 3. Start Rasa Server

Open another terminal and create Riva sample container

3.1 Create Riva Sample container

```
sudo docker run -it --rm -p 5005:5005 -v $(pwd)/cfg_dir:/workspace/cfg nvcr.io/nvidia/
↪riva/riva-speech-client:1.5.0-beta-samples /bin/bash
```

3.2 Replace `/workspace/samples/virtual-assistant-rasa/config.py` with `/workspace/cfg_dir/config.py`

3.3 Replace `/workspace/samples/virtual-assistant-rasa/rasa-weatherbot/endpoints.yml` with `/workspace/cfg_dir/endpoints.yml`

3.4 Activate the Rasa Python environment.

```
. /pythonenvs/rasa/bin/activate
```

3.5 Navigate to the `/workspace/samples/virtual-assistant-rasa/rasa-weatherbot` directory and Run the Rasa training

```
cd /workspace/samples/virtual-assistant-rasa/rasa-weatherbot
```

If you use Riva NLP:

```
rasa train -c config/config_rivanlp.yml \
    -d domain/domain_rivanlp.yml \
    --out models/models_rivanlp/ \
    --data data/nlu_rivanlp.yml \
    data/rules_rivanlp.yml \
    data/stories_rivanlp.yml
```

Or you use Rasa NLU:

```
rasa train -c config/config_rasanlp.yml \
    -d domain/domain_rasanlp.yml \
    --out models/models_rasanlp/ \
    --data data/nlu_rasanlp.yml \
    data/rules_rasanlp.yml data/stories_rasanlp.yml
```

3.6 Start the Rasa Server

- If you use Riva NLP:

```
rasa run -m models/models_rivanlp/ --enable-api \
    --log-file out.log --endpoints endpoints.yml
```

- If you use Rasa NLU:

```
rasa run -m models/models_rasanlp/ --enable-api \
    --log-file out.log --endpoints endpoints.yml
```

37.6 4. Start RVA Rasa Server

4.1 Create Riva Sample container

```
sudo docker run -it --rm -p 5555:5555 -v $(pwd)/cfg_dir:/workspace/cfg nvcr.io/nvidia/
↪riva/riva-speech-client:1.5.0-beta-samples /bin/bash
```

4.2 Replace `/workspace/samples/virtual-assistant-rasa/config.py` with `/workspace/cfg_dir/config.py`

4.3 Replace `/workspace/samples/virtual-assistant-rasa/rasa-weatherbot/endpoints.yml` with `/workspace/cfg_dir/endpoints.yml`

4.4 Activate Chatbot Python environment

```
. /pythonenvs/client/bin/activate
```

4.5 Navigate to the `/workspace/samples/virtual-assistant-rasa/rasa-riva-weatherbot-webapp` directory and Start Chatbot Client server

```
python3 main.py
```

37.7 5. Open RVA Rasa

Browse *https://[riva chatbot server host IP]:5555/rivaWeather*.

CHAPTER

THIRTYEIGHT

GUIDE DESCRIPTION

End-to-end Riva development from custom inference engines to custom Riva deployment.

CHAPTER

THIRTYNINE

AUTHOR

Season POON

Position: machine learning intern @ NVIDIA

Email: spoon@nvidia.com