=========================

#

# Copyright 2018-2021 Picovoice Inc.

#

# You may not use this file except in compliance with the license. A copy of the license is located in the "LICENSE"

# file accompanying this source.

#

# 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.

#

import argparse

import os

import struct

import wave

from datetime import datetime

from threading import Thread

import pvporcupine

from pvrecorder import PvRecorder

from gpiozero import Buzzer

from time import sleep

buzzer = Buzzer(21)

class PorcupineDemo(Thread):

"""

Microphone Demo for Porcupine wake word engine. It creates an input audio stream from a microphone, monitors it, and

upon detecting the specified wake word(s) prints the detection time and wake word on console. It optionally saves

the recorded audio into a file for further debugging.

"""

def \_\_init\_\_(

self,

access\_key,

library\_path,

model\_path,

keyword\_paths,

sensitivities,

input\_device\_index=None,

output\_path=None):

"""

Constructor.

:param library\_path: Absolute path to Porcupine's dynamic library.

:param model\_path: Absolute path to the file containing model parameters.

:param keyword\_paths: Absolute paths to keyword model files.

:param sensitivities: Sensitivities for detecting keywords. Each value should be a number within [0, 1]. A

higher sensitivity results in fewer misses at the cost of increasing the false alarm rate. If not set 0.5 will

be used.

:param input\_device\_index: Optional argument. If provided, audio is recorded from this input device. Otherwise,

the default audio input device is used.

:param output\_path: If provided recorded audio will be stored in this location at the end of the run.

"""

super(PorcupineDemo, self).\_\_init\_\_()

self.\_access\_key = access\_key

self.\_library\_path = library\_path

self.\_model\_path = model\_path

self.\_keyword\_paths = keyword\_paths

self.\_sensitivities = sensitivities

self.\_input\_device\_index = input\_device\_index

self.\_output\_path = output\_path

def run(self):

"""

Creates an input audio stream, instantiates an instance of Porcupine object, and monitors the audio stream for

occurrences of the wake word(s). It prints the time of detection for each occurrence and the wake word.

"""

keywords = list()

for x in self.\_keyword\_paths:

keyword\_phrase\_part = os.path.basename(x).replace('.ppn', '').split('\_')

if len(keyword\_phrase\_part) > 6:

keywords.append(' '.join(keyword\_phrase\_part[0:-6]))

else:

keywords.append(keyword\_phrase\_part[0])

porcupine = None

recorder = None

wav\_file = None

try:

porcupine = pvporcupine.create(

access\_key=self.\_access\_key,

library\_path=self.\_library\_path,

model\_path=self.\_model\_path,

keyword\_paths=self.\_keyword\_paths,

sensitivities=self.\_sensitivities)

recorder = PvRecorder(device\_index=self.\_input\_device\_index, frame\_length=porcupine.frame\_length)

recorder.start()

if self.\_output\_path is not None:

wav\_file = wave.open(self.\_output\_path, "w")

wav\_file.setparams((1, 2, 16000, 512, "NONE", "NONE"))

print(f'Using device: {recorder.selected\_device}')

print('Listening {')

for keyword, sensitivity in zip(keywords, self.\_sensitivities):

print(' %s (%.2f)' % (keyword, sensitivity))

print('}')

while True:

pcm = recorder.read()

if wav\_file is not None:

wav\_file.writeframes(struct.pack("h" \* len(pcm), \*pcm))

result = porcupine.process(pcm)

if result >= 0:

print('[%s] Detected %s' % (str(datetime.now()), keywords[result]))

buzzer.on()

sleep(1)

buzzer.off()

except pvporcupine.PorcupineInvalidArgumentError as e:

print("One or more arguments provided to Porcupine is invalid: {\n" +

f"\t{self.\_access\_key=}\n" +

f"\t{self.\_library\_path=}\n" +

f"\t{self.\_model\_path=}\n" +

f"\t{self.\_keyword\_paths=}\n" +

f"\t{self.\_sensitivities=}\n" +

"}")

print(f"If all other arguments seem valid, ensure that '{self.\_access\_key}' is a valid AccessKey")

raise e

except pvporcupine.PorcupineActivationError as e:

print("AccessKey activation error")

raise e

except pvporcupine.PorcupineActivationLimitError as e:

print(f"AccessKey '{self.\_access\_key}' has reached it's temporary device limit")

raise e

except pvporcupine.PorcupineActivationRefusedError as e:

print(f"AccessKey '{self.\_access\_key}' refused")

raise e

except pvporcupine.PorcupineActivationThrottledError as e:

print(f"AccessKey '{self.\_access\_key}' has been throttled")

raise e

except pvporcupine.PorcupineError as e:

print(f"Failed to initialize Porcupine")

raise e

except KeyboardInterrupt:

print('Stopping ...')

finally:

if porcupine is not None:

porcupine.delete()

if recorder is not None:

recorder.delete()

if wav\_file is not None:

wav\_file.close()

@classmethod

def show\_audio\_devices(cls):

devices = PvRecorder.get\_audio\_devices()

for i in range(len(devices)):

print(f'index: {i}, device name: {devices[i]}')

def main():

parser = argparse.ArgumentParser()

parser.add\_argument('--access\_key',

help='AccessKey obtained from Picovoice Console (https://console.picovoice.ai/)')

parser.add\_argument(

'--keywords',

nargs='+',

help='List of default keywords for detection. Available keywords: %s' % ', '.join(sorted(pvporcupine.KEYWORDS)),

choices=sorted(pvporcupine.KEYWORDS),

metavar='')

parser.add\_argument(

'--keyword\_paths',

nargs='+',

help="Absolute paths to keyword model files. If not set it will be populated from `--keywords` argument")

parser.add\_argument('--library\_path', help='Absolute path to dynamic library.', default=pvporcupine.LIBRARY\_PATH)

parser.add\_argument(

'--model\_path',

help='Absolute path to the file containing model parameters.',

default=pvporcupine.MODEL\_PATH)

parser.add\_argument(

'--sensitivities',

nargs='+',

help="Sensitivities for detecting keywords. Each value should be a number within [0, 1]. A higher " +

"sensitivity results in fewer misses at the cost of increasing the false alarm rate. If not set 0.5 " +

"will be used.",

type=float,

default=None)

parser.add\_argument('--audio\_device\_index', help='Index of input audio device.', type=int, default=-1)

parser.add\_argument('--output\_path', help='Absolute path to recorded audio for debugging.', default=None)

parser.add\_argument('--show\_audio\_devices', action='store\_true')

args = parser.parse\_args()

if args.show\_audio\_devices:

PorcupineDemo.show\_audio\_devices()

else:

if args.access\_key is None:

raise ValueError("AccessKey (--access\_key) is required")

if args.keyword\_paths is None:

if args.keywords is None:

raise ValueError("Either `--keywords` or `--keyword\_paths` must be set.")

keyword\_paths = [pvporcupine.KEYWORD\_PATHS[x] for x in args.keywords]

else:

keyword\_paths = args.keyword\_paths

if args.sensitivities is None:

args.sensitivities = [0.5] \* len(keyword\_paths)

if len(keyword\_paths) != len(args.sensitivities):

raise ValueError('Number of keywords does not match the number of sensitivities.')

PorcupineDemo(

access\_key=args.access\_key,

library\_path=args.library\_path,

model\_path=args.model\_path,

keyword\_paths=keyword\_paths,

sensitivities=args.sensitivities,

output\_path=args.output\_path,

input\_device\_index=args.audio\_device\_index).run()

if \_\_name\_\_ == '\_\_main\_\_':

main()

=========================