TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN
LỚP CỬ NHÂN TÀI NĂNG

BÙI TRUNG HẢI – PHẠM NGỌC TUẤN

XÂY DỰNG ỨNG DỤNG TRỢ LÝ ẢO CHO MÁY TÍNH SỬ DỤNG GOOGLE SPEECH API

KHÓA LUẬN TỐT NGHIỆP CỬ NHÂN CNTT

TP. HCM, 2017

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN
LỚP CỬ NHÂN TÀI NĂNG

BÙI TRUNG HẢI – 1312165
PHẠM NGỌC TUẤN – 1312669

XÂY DỰNG ỨNG DỤNG TRỢ LÝ ẢO CHO MÁY TÍNH SỬ DỤNG GOOGLE SPEECH API

KHÓA LUẬN TỐT NGHIỆP CỬ NHÂN CNTT

GIÁO VIÊN HƯỚNG DẪN

TS. NGÔ MINH NHỰT

KHÓA 2013 - 2017

NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....

TpHCM, ngày ….. tháng …… năm ……

Giáo viên hướng dẫn

[Ký tên và ghi rõ họ tên]

NHẬN XÉT CỦA GIÁO VIÊN PHẢN BIỆN

....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....

Khóa luận đáp ứng yêu cầu của Khóa luận cử nhân CNTT.

TpHCM, ngày ….. tháng …… năm ……

Giáo viên phản biện

[Ký tên và ghi rõ họ tên]

LỜI CẢM ƠN

Với lòng biết ơn sâu sắc, trước hết chúng em xin chân thành cảm ơn quý Thầy Cô khoa Công Nghệ Thông Tin - trường đại học Khoa Học Tự Nhiên, những người đã ân cần giảng dạy, xây dựng cho em một nền tảng kiến thức vững chắc để chúng em có thể thực hiện khóa luận này.

Đặc biệt, chúng em xin gửi lời tri ân sâu sắc đến Thầy Ngô Minh Nhựt. Thầy đã rất tận tâm, nhiệt tình hướng dẫn và chỉ bảo chúng em trong suốt quá trình thực hiện luận văn. Nếu không có sự giúp đỡ tận tình của thầy, chúng em chắc chắn không thể hoàn thành luận văn.

Cuối cùng, chúng con xin cảm ơn ba mẹ đã sinh thành, nuôi dưỡng, và dạy dỗ để chúng con có được thành quả như ngày hôm nay. Ba mẹ luôn là nguồn động viên, nguồn sức mạnh hết sức lớn lao mỗi khi chúng con gặp khó khăn trong cuộc sống.

Để hoàn thành luận văn này là tất cả những cố gắng, nỗ lực của chúng em. Tuy nhiên, sẽ không thể tránh khỏi những thiếu sót, kính mong nhận được sự cảm thông và giúp đỡ của quý Thầy Cô và các bạn.

TP. Hồ Chí Minh, 7/2017

Bùi Trung Hải

Phạm Ngọc Tuấn

ĐỀ CƯƠNG CHI TIẾT

Tên Đề Tài: Xây dựng ứng dụng trợ lý ảo cho máy tính sử dụng Google Speech API
Giáo viên hướng dẫn: ThS. Ngô Minh Nhựt
Thời gian thực hiện: từ ngày 15/12/2016 đến ngày 15/07/2017
Sinh viên thực hiện: Bùi Trung Hải – 1312165, Phạm Ngọc Tuấn - 1312669
Loại đề tài: Phát triển hệ thống, nghiên cứu thuật toán

Nội Dung Đề Tài: Tìm hiểu các phương pháp xử lý và tương tác với tín hiệu âm thanh, tiếng nói. Nghiên cứu hệ thống Natural Language Understanding đơn giản. Áp dụng vào xây dựng ứng dụng trợ lý ảo cho máy tính.

Nội dung chi tiết của đề tài bao gồm:

- Xây dựng ứng dụng trợ lí ảo tương tác bằng giọng nói tiếng Anh cho máy tính
- Các vấn đề quan tâm:
    - Xử lý tín hiệu số (âm thanh, tiếng nói), hệ thống chuyển đổi tiếng nói thành văn bản.
    - Tương tác audio I/O, hệ thống chuyển đổi văn bản thành tiếng nói.
    - Giao thức truyền nhận dữ liệu REST API,…
    - Hệ thống Natural Language Understanding hiệu quả để xác định ý muốn của người dùng
    - Hệ thống chạy đa nhiệm
    - Tối ưu hóa hệ thống

- Các thành phần cơ bản của hệ thống:
    - Module thu âm từ microphone
    - Module nhận dạng từ khóa wake up
    - Module chuyển đổi tiếng nói thành văn bản
    - Module chuyển đổi văn bản thành hành động.
    - Module chuyển đổi văn bản thành tiếng nói
- Ứng dụng thử nghiệm sẽ hỗ trợ các tính năng:
    - Thông báo giờ hiện tại
    - Dự báo thời tiết trong ngày
    - Phát nhạc
    - Trả lời các câu hỏi Wh-question
    - Chào hỏi cơ bản

Kế Hoạch Thực Hiện:

- 15/12/2016 – 14/01/2017: Khảo sát, tìm hiểu về các thư viện python phục vụ cho việc tương tác và xử lý tín hiệu âm thanh.

- 15/01/2017 – 14/02/2017: Thiết kế các thành phần của hệ thống.

- 15/02/2017 – 14/03/2017: Cài đặt và thử nghiệm các thành phần: thu âm, nhận dạng từ khóa wake up, chuyển giọng nói thành văn bản, chuyển văn bản thành giọng nói.

- 15/03/2017 – 14/04/2017: Tìm hiểu và xây dựng hệ thống Natural Language Understanding.

- 15/04/2017 – 14/05/2017: Cài đặt các chức năng mà ứng dụng hỗ trợ.

- 15/05/2017 – 31/05/2017: Ráp nối tất cả các thành phần, tiến hành thử nghiện và hoàn thiện hệ thống.

- 01/06/2017 – 28/06/2017: Viết và hoàn thiện luận văn.

Xác nhận của GVHD	**Ngày 28 tháng 06 năm 2017** **SV Thực hiện**

MỤC LỤC

DANH MỤC HÌNH ẢNH

DANH MỤC BẢNG

TÓM TẮT KHÓA LUẬN

Trong xu hướng công nghệ hiện nay, vai trò của các trợ lý ảo ngày càng trở nên quan trọng. Các hãng công nghệ lớn thay nhau tung ra những trợ lý ảo của riêng mình tích hợp trên các thiết bị di động: Siri của Apple, Cortana của Microsoft, Google Assistant của Google, Alexa của Amazon,... Chức năng của các trợ lý ảo này ngày càng được mở rộng, từ những chức năng đơn giản như tra cứu, hỏi đáp, đến những chức năng cao hơn như quản lý lịch, gọi điện thoại, dẫn đường, điều khiển các thiết bị khác,... Khóa luận này có mục đích tạo ra một trợ lý ảo có khả năng chạy được trên nhiều nền tảng hệ điều hành khác nhau trên máy tính cá nhân.

Nhận diện giọng nói là một trong những thành phần quan trọng nhất của một trợ lý ảo. Nhiều công ty và nhóm nghiên cứu lớn nhỏ đã nghiên cứu và đưa ra các bộ toolkit cũng như API cho việc nhận diện giọng nói, trong đó một trong những API có chất lượng được đánh giá tốt nhất là Google Speech API của gã khổng lồ công nghệ Google. Do đó, chúng tôi muốn tận dụng chất lượng của Google Speech API để tạo nên một trợ lý ảo có độ chính xác cao về nhận diện giọng nói.

Kết quả sơ bộ mà khóa luận đạt được là tạo ra một trợ lý ảo có thể chạy trên các hệ điều hành phổ biến trên máy tính cá nhân như Windows, Linux, Mac. Trợ lý ảo có những chức năng cơ bản của một trợ lý ảo như hỏi đáp, tra cứu thông tin, trả lời các câu hỏi về thời gian, thời tiết, ngoài ra còn có thể phát nhạc theo yêu cầu và chào hỏi ở mức độ đơn giản.

Chương 1

Mở đầu

Nội dung của chương 1 giới thiệu tổng quan về đề tài, nêu ra mục tiêu của khóa luận, và cấu trúc nội dung của luận văn.

1.1 Tổng quan về đề tài

1.1.1 Giới thiệu về trợ lý ảo

Trợ lý ảo là một phần mềm trên máy tính hoặc thiết bị di động có khả năng hỗ trợ người dùng thực hiện nhiều loại công việc, nhận lệnh từ người dùng dưới dạng ngôn ngữ tự nhiên, thường là giọng nói. Nhờ khả năng nhận lệnh và phản hồi qua giọng nói, người dùng có thể ra lệnh cho trợ lý ảo mà không cần phải thao tác bằng tay trên thiết bị. Điều đó sẽ giúp tăng tính hiệu quả và tạo ra sự tự nhiên trong giao tiếp giữa người và máy, tạo ra những kênh tương tác mới khác với truyền thống, mang đến cho người dùng những trải nghiệm mới mẻ và thứ vị hơn khi sử dụng những thiết bị công nghệ.

Chức năng của các trợ lý ảo rất phong phú và đa dạng, từ những chức năng bình thường như hỏi đáp, tra cứu thông tin, bật nhạc, tìm kiếm trong danh bạ, quản lý lịch, đặt báo thức,... cho đến những chức năng đặc biệt như chơi game, trò chuyện, điều khiển các thiết bị trong gia đình, thậm chí là mua sắm, đặt chỗ nhà hàng, đặt vé máy bay,...

1.1.2 Khảo sát thị trường trợ lý ảo

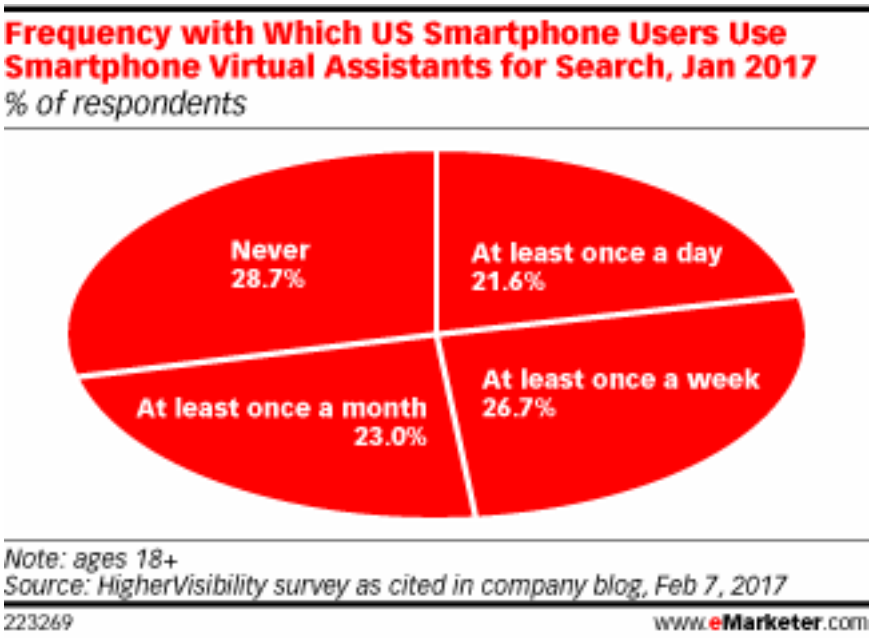
Tính đến hiện tại, gần như tất cả các hãng công nghệ lớn đều đã tung ra trợ lý ảo của riêng mình:

- Apple với Siri, hoạt động trên các thiết bị của Apple như iPhone, iPad, iPod Touch, Mac và Apple TV.
- Microsoft với Cortana, hoạt động trên các phiên bản mới của Windows như Windows 10, Windows 10 Mobile, Windows Phone 8.1, và các thiết bị khác như Microsoft Band, Xbox One.
- Amazon với Alexa, hoạt động trên loa thông minh Amazon Echo.
- Google với Google Assistant, hoạt động trên các thiết bị Android, loa thông minh Google Home và ứng dụng nhắn tin Allo.
- Gần đây, Samsung đã ra mắt trợ lý ảo của mình mang tên Bixby, chạy trên các dòng điện thoại Samsung Galaxy.
- Facebook cũng đã công bố trợ lý ảo của mình mang tên M, sẽ ra mắt trong năm 2017 trên các ứng dụng Facebook và Facebook Messenger.

Trong xu hướng đó, số lượng người dùng và tần suất sử dụng của các trợ lý ảo đang ngày càng gia tăng. Theo một khảo sát thực hiện vào tháng 01/2017 trên các người dùng smartphone tại Mỹ[5], có gần 27% người được hỏi nói rằng họ dùng trợ lý ảo ít nhất một lần mỗi tuần, và khoảng 22% người dùng sử dụng trợ lý ảo hàng ngày. Trong khi đó, có 28.7% số người được hỏi chưa bao giờ sử dụng trợ lý ảo.

Khi được hỏi về lý do sử dụng trợ lý ảo, 1/3 số người được hỏi nói rằng tìm kiếm bằng trợ lý ảo dễ hơn tìm kiếm bằng tay. Khoảng 1/4 số người được khảo sát nói rằng họ không thể gõ chữ trên smartphone, hoặc không thể nhìn rõ trên smartphone, hoặc vì tìm kiếm bằng trợ lý ảo nhanh hơn tìm kiếm bằng tay. Tuy nhiên, lý do lớn nhất mà nhiều người dùng sử dụng trợ lý ảo là lái xe. Có đến hơn một nửa số người được hỏi cho biết họ sử dụng trợ lý ảo trong khi lái xe.

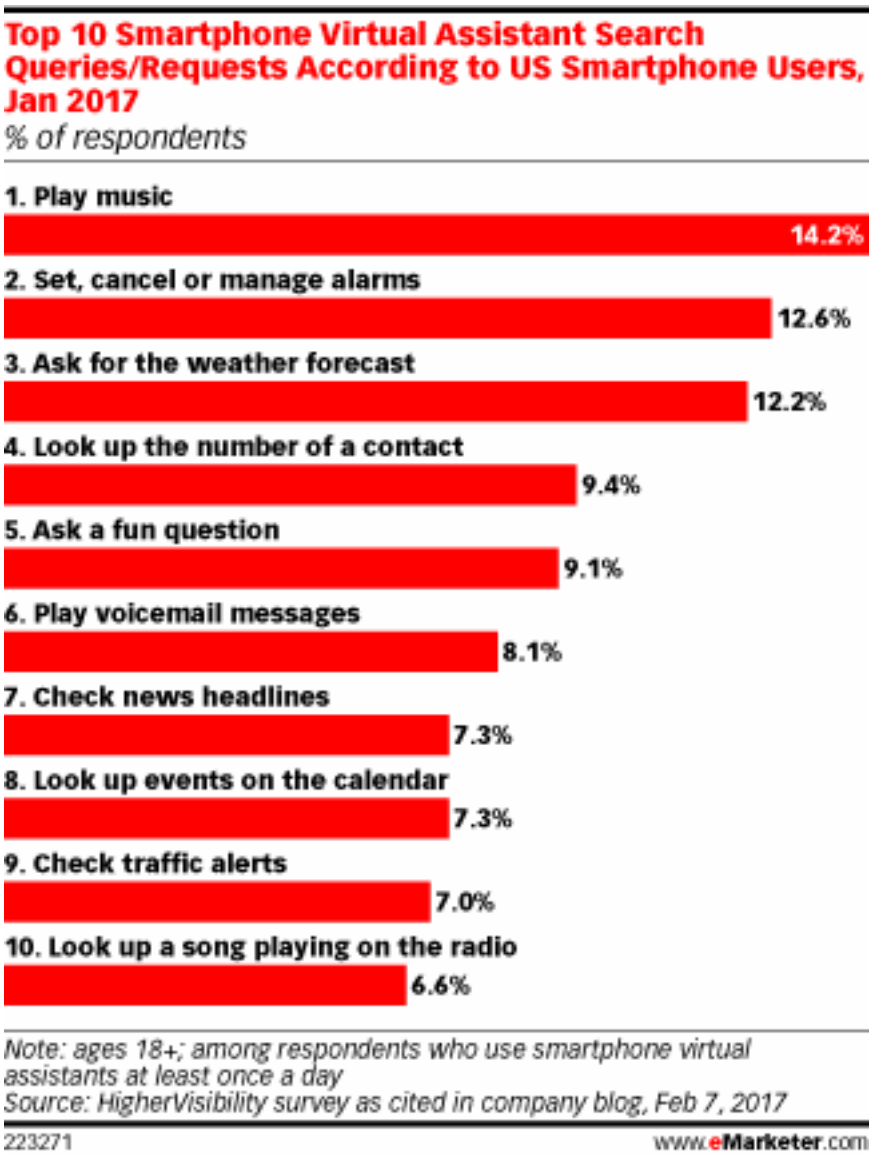
Nhìn vào hình 1.2 có thể thấy phần đông người dùng sử dụng trợ lý ảo với mục đích bật nhạc, quản lý báo thức, hỏi thông tin dự báo thời tiết. Ngoài ra, họ còn dùng



Hình 1.1: Tần suất người dùng smartphone ở Mỹ sử dụng trợ lý ảo, tháng 01/2017[5]

trợ lý ảo để tìm kiếm số điện thoại trong danh bạ, hỏi câu hỏi vui, bật các tin nhắn thoại hoặc xem tin tức.

Như vậy có thể nói các trợ lý ảo đang ngày càng góp một phần quan trọng trong cuộc sống của những người dùng công nghệ. Không chỉ tạo ra một trải nghiệm mới, các trợ lý ảo còn giúp tiết kiệm thời gian và công sức. Ngoài ra, các trợ lý ảo còn có thể giúp được những người khuyết tật hoặc người cao tuổi có thể tiếp cận với các thiết bị công nghệ một cách dễ dàng hơn.



Hình 1.2: Những chức năng được sử dụng nhiều nhất trên trợ lý ảo theo người dùng smartphone ở Mỹ, tháng 01/2017[5]

1.2 Mục tiêu của khóa luận

Nhận thấy các trợ lý ảo nêu trong phần trước đa phần chỉ hoạt động trên một số nền tảng nhất định của hãng làm ra chúng, và chủ yếu dành cho các thiết bị di động, chúng tôi thực hiện khóa luận này với mục đích chính là tạo ra một ứng dụng trợ lý ảo có khả năng chạy trên các nền tảng hệ điều hành phổ biến trên máy tính cả nhân như Windows, Linux, Mac, với các chứng năng cơ bản:

- Hỏi đáp, tra cứu thông tin
- Hỏi giờ, thời tiết
- Phát nhạc
- Trò chuyện đơn giản

Các mục tiêu nhỏ:

- Ứng dụng có khả năng tự kích hoạt khi người dùng gọi tên
- Nhận biết chính xác câu nói của người dùng
- Phản hồi một cách chính xác và tự nhiên
- Thời gian xử lý ngắn (tính từ lúc người dùng hoàn thành câu nói đến lúc ứng dụng bắt đầu phản hồi).

1.3 Nội dung luận văn

Nội dung của luận văn sẽ gồm những phần sau:

- Chương 1: *Mở đầu*: Giới thiệu tổng quan về đề tài, mục tiêu của khóa luận và cấu trúc nội dung của luận văn.
- Chương 2: *Tổng quan về tín hiệu âm thanh, tiếng nói. Thư viện PyAudio*: Giới thiệu tổng quan về tín hiệu âm thanh, tiếng nói, các thành phần của âm thanh, cách lưu trữ âm thanh trong máy tính, các thông số của file âm thanh; giới thiệu về thư viện PyAudio: chức năng, cách cài đặt, cách sử dụng.

- Chương 3: *Speech to Text*: Giới thiệu về bài toán Speech to Text, các ứng dụng, các vấn đề cần giải quyết trong Speech to Text; giới thiệu về các thư viện pocketsphinx và Google Speech to Text: chức năng, cách cài đặt, cách sử dụng, ưu và nhược điểm, vai trò của các thư viện đó trong hệ thống.

- Chương 4: *Text to Speech*: Giới thiệu về bài toán Text to Speech, các ứng dụng; giới thiệu về Google Text to Speech và iSpeech: chức năng, cách sử dụng, ưu và nhược điểm, vai trò trong hệ thống.

- Chương 5: *Intent Classification*: Giới thiệu về bài toán Intent Classification, các thuật toán để giải quyết bài toán này, các ứng dụng; giới thiệu về thư viện Rasa NLU: chức năng, cách cài đặt, cách sử dụng, vai trò trong hệ thống.

- Chương 6: *Ứng dụng Alexa*: Giới thiệu tổng quan về ứng dụng Alexa, các module trong chương trình, luồng hoạt động giữa các module, các chức năng của ứng dụng.

- Chương 7: *Kết luận và Hướng phát triển*: Nếu ra các kết quả đạt được của khóa luận, các hạn chế và hướng phát triển.

Chương 2

Tín hiệu âm thanh, tiếng nói. Thư viện PyAudio

Nội dung chương 2 sẽ giới thiệu tổng quan về âm thanh và tiếng nói, cách âm thanh được lưu trữ trên máy tính, các ứng dụng của âm thanh và tiếng nói. Chương 2 cũng sẽ giới thiệu về chức năng, cách cài đặt, cách sử dụng cũng như các ưu nhược điểm của thư viện PyAudio.

2.1 Tổng quan

2.1.1 Tổng quan về âm thanh

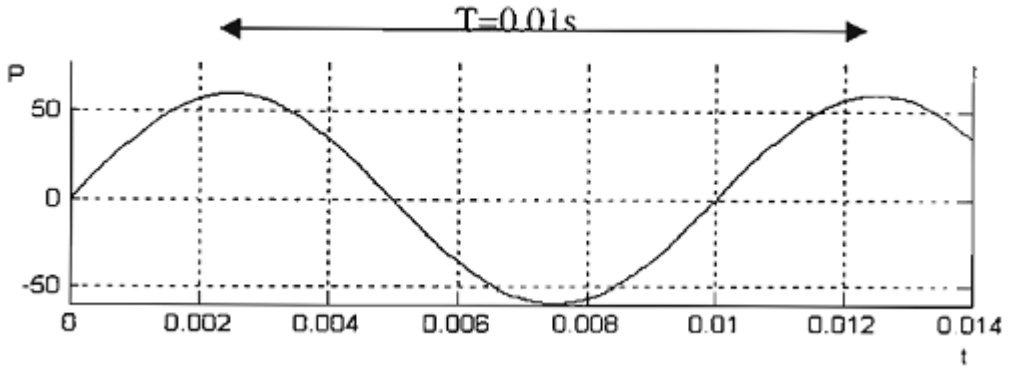
Trong vật lý, âm thanh được định nghĩa là các giao động cơ học lan truyền thông qua các phương tiện truyền dẫn như: không khí, nước, chất rắn,... Các giao động cơ học đó còn được gọi là sóng âm. Sóng âm được tạo ra do có sự biến đổi về áp suất theo thời gian. Vận tốc lan truyền trong không khí của sóng âm vào khoảng 343.2 m/s.

Đối với con người, âm thanh là những giao động có thể được cảm nhận thông qua thính giác. Các giao động khi lan truyền trong không khí sẽ va đập và làm rung màng nhĩ, qua đó não bộ sẽ thu được tín hiệu âm thanh. Con người có thể nghe được âm thanh có tần số từ 16 Hz đến 20 kHz. Âm thanh có tần số cao hơn 20 kHz gọi là siêu âm, âm thanh có tần số thấp hơn 16 Hz gọi là hạ âm.

Cách đơn giản nhất để biểu diễn âm thanh là dưới dạng sóng sin với trục x là thời gian, trục y là áp xuất:

$$P = A\sin(2\pi ft) \tag{2.1}$$

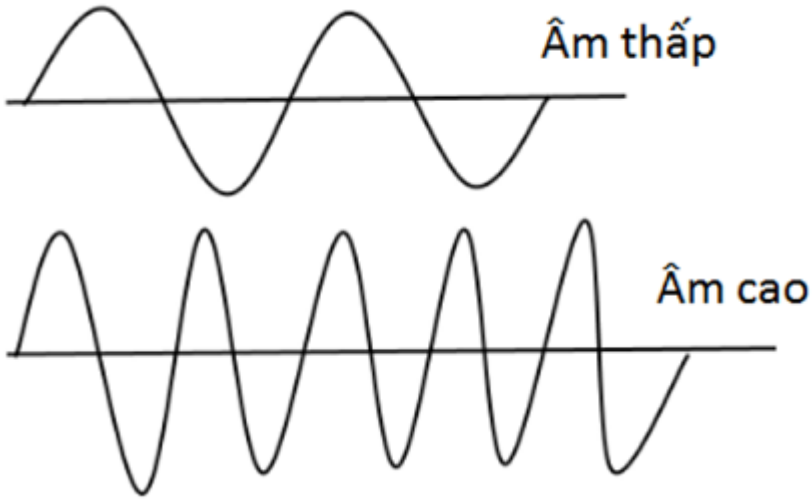
Trong đó: P là áp suất, đơn vị là decibels (dB) hoặc pascals
A là biên độ của sóng, đơn vị là decibels (dB)
t là thời gian, đơn vị là giây (s)
f là tần số, đơn vị là hertz (Hz)



Hình 2.1: Sóng sin có biên độ 60 dB, tần số 100 Hz

Âm thanh có một số đặc trưng cơ bản như: độ cao, độ mạnh, độ dài và âm sắc.

- **Độ cao** (Pitch): âm thanh luôn có một độ cao nhất định. Độ cao của âm thanh phụ thuộc vào tần số của sóng âm. Tần số càng lớn thì âm thanh càng cao, tần số càng bé thì âm thanh càng trầm.
- **Độ mạnh** (Intensity): hay còn gọi là độ to của âm thanh. Độ mạnh của âm thanh phụ thuộc vào biên độ sóng âm. Biên độ càng lớn thi cường độ âm càng mạnh, biên độ càng bé thì cường độ âm càng yếu.
- **Độ dài** (Duration): là thời gian kéo dài của sóng âm.
- **Âm sắc** (Timbre): âm sắc là một đặc trưng sinh lý của âm, giúp phân biết âm thanh do các nguồn khác nhau phát ra. Âm sắc liên quan mật thiết với đồ thị giao động âm.



Hình 2.2: Minh họa độ cao (Pitch) của âm

2.1.2 Tổng quan về tiếng nói

Trong tự nhiên, âm thanh bao gồm nhiều loại được tạo ra từ nhiều nguồn khác nhau:

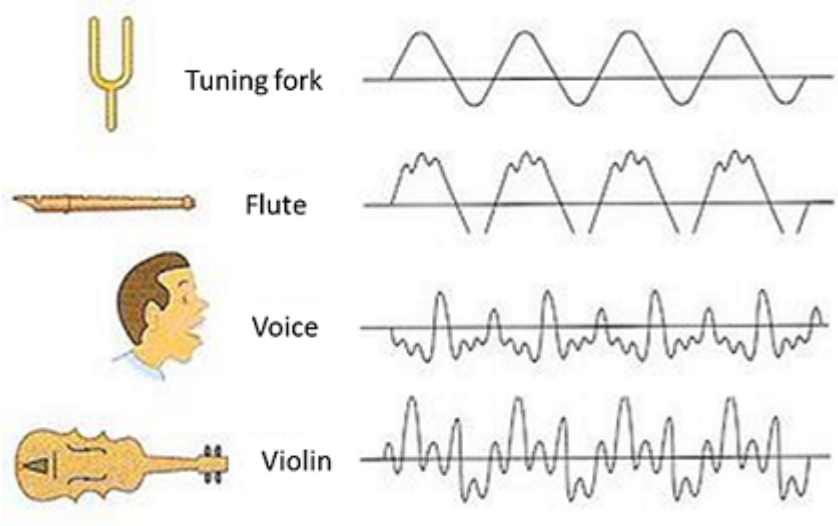
- **Âm nhạc**: âm thanh được phát ra từ các nhạc cụ.
- **Tiếng kêu**: được phát ra từ các loại động vật. Ví dụ: cá heo (1-164 kHz).
- **Tiếng động**: âm thanh phát ra từ sự va chạm giữa hai vật.
- **Tiếng ồn**: là những âm thanh không mong muốn.
- **Tiếng nói**: là những âm thanh được phát ra từ miệng con người

Ta có thể phân loại tiếng nói dựa theo thanh:

- **Âm hữu thanh**: là âm khi phát ra có sự dao động của đôi dây thanh quản.
- **Âm vô thanh**: phát ra khi đôi dây thanh quản không dao động. Thí dụ phần cuối của phát âm English, chữ sh cho ra âm xát.

Hoặc theo âm:

- **Nguyên âm**: là âm phát ra có thể kéo dài. Tất cả nguyên âm đều là âm hữu thanh, nghĩa là tuần hoàn và khá ổn định trong một đoạn thời gian vài chục ms.



Hình 2.3: Minh họa âm sắc (Timbre) của âm

- **Phụ âm**: là âm chỉ phát ra một nhát, không kéo dài được. Có phụ âm hữu thanh và phụ âm vô thanh.

Tiếng nói đóng vai trò quan trọng trong hoạt động động giao tiếp giữa con người, nó là phương tiện giao tiếp nhanh, tiện lợi và phổ biến nhất.

2.2 Lưu trữ âm thanh trong máy tính

Tất cả âm thanh mà chúng ta nghe được trong tự nhiên đều tồn tại dưới dạng sóng âm, là các sóng cơ học tuần hoàn liên tục analog. Trong khi đó, máy tính xử lý và lưu trữ thông tin dưới dạng các xung điện tử rời rạc digital. Vì vậy, để có thể lưu trữ và xử lý tín hiệu âm thanh trên máy tính, ta phải mô phỏng sóng âm bằng những mẫu rời rạc. Việc mô phỏng được đặc trưng bởi các thông số:

- **Mẫu (Sample) là gì**: là đơn vị âm thanh nhỏ nhất được lưu trong máy tính. Để có các xung điện tử rời rạc, ta cần lấy mẫu nhiều lần từ tín hiệu analog. Mỗi mẫu là giá trị biên độ của sóng âm tại thời điểm lấy mẫu.

- **Tần số lấy mẫu (Sample Rate)**: là số lần lấy mẫu trên một giây, đơn vị là Hz. Tần số lấy mẫu càng cao, tín hiệu số thu được càng chính xác.

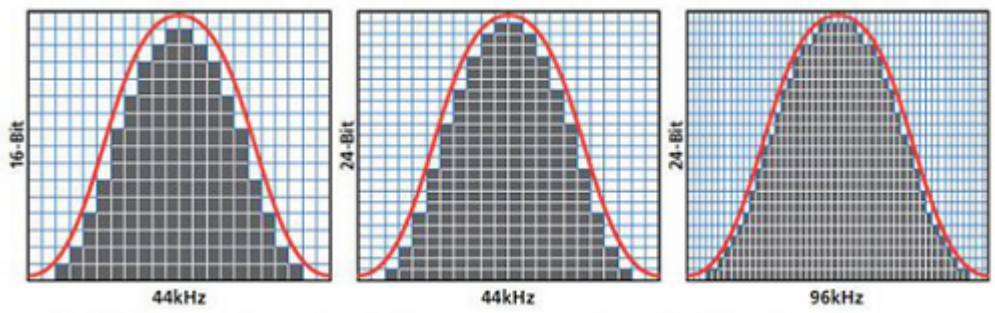
- **Độ dày bit (BitDepth)**: để lưu lại dưới dạng số, mỗi mẫu được biểu diễn bằng

một lượng bit dữ liệu nhất định gọi là BitDepth. BitDepth càng lớn âm thanh càng sắc nét, trung thực.

- **Kênh (Channel)**: Bằng thuật toán, tín hiệu số có thể được chia thành nhiều kênh để khi nghe bằng hệ thống âm thanh vòm sẽ tạo ra cảm giác thật nhất.

Một trong những phương pháp chuyển đổi tín hiệu analog sang digital phổ biến nhất hiện nay là Pulse-Code Modulation (PCM). Kỹ thuật PCM bao gồm 3 bước:

- **Lấy mẫu (Sampling)**: quá trình rời rạc hoá tín hiệu analog đầu vào theo tần số lấy mẫu f. Ví dụ f = 44100 Hz, ta sẽ lấy mẫu 44100 lần trong một giây.

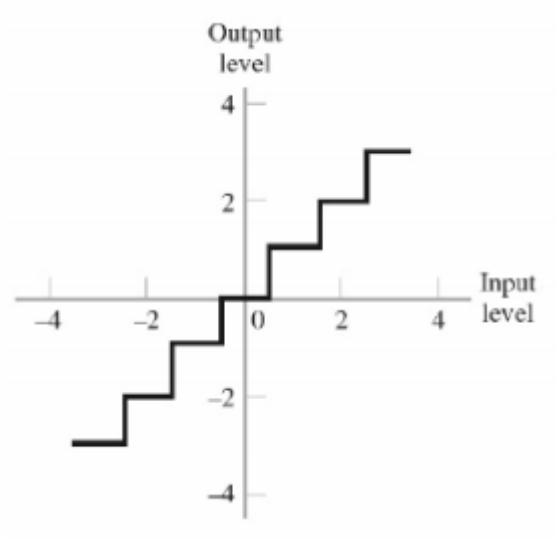


Hình 2.4: Minh họa tín hiệu analog được lấy mẫu theo nhiều tần số khác nhau

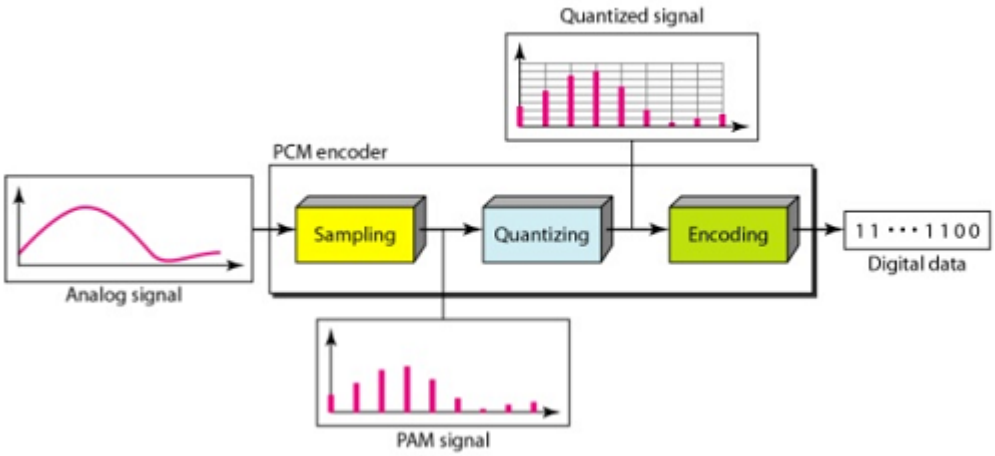
- **Lượng tử hóa (Quantization)**: tín hiệu analog sau khi được lấy mẫu thì mỗi mẫu có thể có vô số các giá trị. Thay vì sử dụng giá trị mẫu chính xác, giá trị mẫu được thay bằng giá trị gần nhất trong M giá trị cho phép. Các giá trị lượng tử được gọi là các mức lượng tử, nếu các mức lượng tử cách đều nhau gọi là lượng tử đều, ngược lại gọi là lượng tử không đều. Số mức lượng tử M phụ thuộc vào số khả năng mà BitDepth có thể biểu diễn.

- **Mã hóa (Encoding)**: Là quá trình biến đổi giá trị các mẫu sau khi lượng tử thành các từ mã dài n bit (n là BitDepth).

PCM cùng với những biến thể của nó là nền tảng cho âm thanh số. PCM sẽ sẽ hình thành một dạng sóng, sóng này ít nhiều có thể được chạy ngay bởi một bộ xử lý tín hiệu số. Trong khi hầu hết các định dạng khác khi thao tác với âm thanh thì cần thông qua các thuật toán điều khiển nên phải giải mã chúng khi sử dụng.



Hình 2.5: Minh họa quá trình lượng tử hóa



Hình 2.6: Sơ đồ khối mô phỏng phương pháp PCM

2.2.1 Lưu trữ không nén (uncompressed)

Các tín hiệu số thu được thông qua PCM sẽ được lưu trữ nguyên bản, không qua bất kỳ phương pháp nén hay sửa đổi.

- **Ưu điểm**: khi lưu trữ dưới dạng không nén, tín hiệu âm thanh có thể được chạy ngay bởi một bộ xử lý tín hiệu số mà không cần thông qua các thuật toán giải mã.
- **Khuyết điểm**: kích thước tập tin âm thanh sẽ rất lớn làm hao phí không gian lưu trữ và băng thông đường truyền.

Các định dạng âm thanh tiêu biểu cho phương pháp lưu trữ không nén là: WAV, AIFF,...

2.2.2 Lưu trữ nén

Các tín hiệu số thu được thông qua PCM sẽ được nén thông qua các thuật toán khác nhau.

- **Ưu điểm**: kích thước tập tin âm thanh nhỏ. Tiết kiệm không gian lưu trữ và băng thông đường truyền.
- **Khuyết điểm**: chất lượng âm thanh có thể bị giảm sút tùy vào thuật toán nén. Tập tin âm thanh muốn thao tác được phải thông qua quá trình giải mã

Nén không mất mát (Lossless compression)

Các tín hiệu âm thanh gốc sẽ được nén mà không mất dữ liệu và đảm bảo chất lượng ban đầu sau khi giải nén.

Để làm được điều này, cần nhờ đến nhiều thuật toán nén khác nhau. Tuy nhiên, ý tưởng chung của các thuật toán đều là tìm ra quy luật lặp của dữ liệu, sau đó tìm 1 cách hiển thị khác tối ưu hơn, tốn ít dữ liệu hơn. Ví dụ thay vì lưu chuỗi "aaaa bbb cccccc" ta sẽ chuyển thành chuỗi "a4 b3 c6" ít tốn dữ liệu hơn.

Tập tin âm thanh được nén bằng phương pháp này sẽ có tỉ lệ nén không cao, vào khoảng 1/2 đến 1/3 dung lượng âm thanh gốc.

Tiêu biểu cho phương pháp nén không mất mát là các định dạng âm thanh: FLAC, ALAC, APE,...

Nén có mất mát (Lossy compression)

Các tín hiệu âm thanh khi được nén sẽ bị loại bỏ đi các thành phần không quan trọng nhằm giảm tối đa kích thước lưu trữ. Mỗi thuật toán nén sẽ có những tiêu chí khác nhau để chọn lựa các thành phần âm thanh sẽ bị bỏ.Ví dụ, ngưỡng nghe của tai người là những âm thanh có tần số từ 14 Hz đến 20 kHz, như vậy thuật toán sẽ bỏ bớt đi các âm thanh có tần số nằm ngoài khoảng đó.

Bên cạnh đó, khi giải mã tập tin âm thanh bị nén, các thuật toán sẽ tạo ra các âm thanh giả nhằm lấp vào các phần đã bị bỏ bớt đi. Hệ quả của việc này là bạn thường nghe các âm thanh méo mó. Các tập tin nhạc được nén với tỉ lệ càng cao thì sự méo tiếng càng nhiều. Bạn sẽ rất dễ dàng nhận ra sự khác biệt khi nghe hai tập tin nhạc gốc và nhạc nén bị mất mát dữ liệu.

Tập tin âm thanh được nén bằng phương pháp này sẽ có tỉ lệ nén rất cao, lên đến 1/10 dung lượng âm thanh gốc.

Tiêu biểu cho phương pháp nén không mất mát là các định dạng âm thanh: MP3, AAC, WMA,...

2.3 Ứng dụng

Hiện nay âm thanh, tiếng nói đã được nghiên cứu và ứng dụng rộng rãi trong nhiều lĩnh vực của cuộc sống như: truyền thông, âm nhạc, chế tạo sonar,...

Trong luận văn này, tiếng nói giữ vai trò quan trọng là phương tiện tương tác chính giữa người dùng và ứng dụng. Người dùng sử dụng tiếng nói để ra lệnh cho ứng dụng, và ứng dụng sử dụng tiếng nói để phản hồi kết quả cho người dùng.

2.4 Thư viện PyAudio

2.4.1 Tổng quan

PyAudio là thư viện mã nguồn mở hỗ trợ người dùng thao tác với âm thanh trên máy tính một cách dễ dàng. PyAudio được viết bằng Python dựa trên thư viện PortAudio. PyAudio có khả năng hoạt động và cài đặt dễ dàng trên đa nền tảng: Windows, Mac OS, và Linux . Tính tới tháng 06/2017 phiên bản mới nhất của PyAudio là 0.2.11.

2.4.2 Chức năng

PyAudio cũng cấp tính năng giúp người dùng dễ dàng thu và phát âm thanh từ dữ liệu thô dưới dạng từng sample. PyAudio có thể hoạt động ở chế độ đồng bộ (Synchronous) hoặc không đồng bộ (Asynchronous).

2.4.3 Cài đặt

- **Windows**: `python -m pip install pyaudio`
- **Mac OS**:

```
brew install portaudio
pip install pyaudio
```

- **Linux**: `pip install pyaudio`
- **Build từ source**: https://people.csail.mit.edu/hubert/git/pyaudio.git

2.4.4 Cách sử dụng

```
"""PyAudio Example: Play a wave file."""
import pyaudio
import wave
import sys
CHUNK = 1024

if len(sys.argv) < 2:
   print("Plays a wave file.\n\nUsage: %s filename.wav" % sys.argv[0])
   sys.exit(-1)

wf = wave.open(sys.argv[1], 'rb')

# instantiate PyAudio (1)
p = pyaudio.PyAudio()

# open stream (2)
stream = p.open(format=p.get_format_from_width(wf.getsampwidth()),
                channels=wf.getnchannels(),
                rate=wf.getframerate(),
                output=True)
```

```
# read data
data = wf.readframes(CHUNK)

# play stream (3)
while len(data) > 0:
   stream.write(data)
   data = wf.readframes(CHUNK)

# stop stream (4)
stream.stop_stream()
stream.close()

# close PyAudio (5)
p.terminate()
```

Đoạn code trên minh họa cách sử dụng thư viện PyAudio để phát âm thanh từ một file WAV. Chúng tôi sẽ dùng nó để hướng dẫn cách sử dụng đơn giản thư viện PyAudio

- Để sử dụng thư viện PyAudio, trước hết cần khởi tạo PyAudio bằng cách sử dụng câu lệnh **pyaudio.PyAudio()** (1). Câu lệnh này sẽ khởi tạo thư viện PortAudio bên dưới.

- Để có thể thu âm hoặc phát audio, ta mở các stream tương ứng bằng câu lệnh **pyaudio.PyAudio.open()** (2). Input stream để thu âm, output stream để phát audio. Một số tham số cần chú ý trong câu lệnh này:

  - **format**: là định dạng được sử dụng khi encoding. Định dạng này phụ thuộc vào số lượng bit dùng để mã hóa một mẫu (BitDepth).
  - **chanels**: số lượng kênh của tập tin âm thanh cần phát hoặc số lượng kênh của âm thanh cần thu âm.
  - **rate**: là tần số lấy mẫu (Sample Rate)
  - **output**: giá trị này bằng **true** thì đây là stream output dùng để phát âm thanh.
  - **input**: giá trị này bằng **true** thì đây là stream input dùng để thu âm thanh.

- Khi cần phát âm thanh, ta ghi dữ liệu âm thanh lên output stream bằng hàm **pyaudio.Stream.write()**. Khi cần thu âm thanh, ta đọc dữ liệu âm thanh thu được từ input stream thông qua hàm **pyaudio.Stream.read()** (3). Đoạn code minh họa đang chạy ở chế độ blocking, nên các hàm **pyaudio.Stream.write()** và **pyaudio.Stream.read()** sẽ block chương trình đến khi chúng thực hiện việc đọc/ghi xong.

- Để tạm dừng đọc/ghi âm thanh ta dùng hàm **pyaudio.Stream.stop_stream()**. Để kết thúc stream ta dùng hàm **pyaudio.Stream.close()** (4).

- Cuối cùng để giải phóng PortAudio ta dùng hàm **pyaudio.PyAudio.terminate()** (5).

2.4.5 Các ưu, khuyết điểm

- **Ưu điểm**: thư viện cài đặt đơn giản, hỗ trợ chạy trên nhiều hệ điều hành.

- **Nhược điểm**: thư viện hỗ trợ ít tính năng. Không hỗ trợ thu âm đồng thời từ nhiều micro.

2.4.6 Ứng dụng

Trong luận văn này, thư viên PyAudio đảm nhận vai trò thu âm giọng nói của người dùng. Cung cấp thông tin cho các thành phần khác của hệ thống xử lý.

Chương 3

Speech to Text

Nội dung chương 3 sẽ giới thiệu tổng quan về bài toán Speech to Text, mô hình hoạt động, các ứng dụng của Speech to Text, các vấn đề cần giải quyết của module Speech to Text trong một hệ thống trợ lý ảo và cách giải quyết các vấn đề đó. Chương 3 cũng sẽ giới thiệu về chức năng, cách cài đặt, cách sử dụng cũng như các ưu nhược điểm của các thư viện pocketsphinx và Google Speech to Text.

3.1 Tổng quan

Speech to Text, hay Speech Recognition là một lĩnh vực trong khoa học máy tính, trong đó nghiên cứu và phát triển các phương pháp và công nghệ để máy tính có thể nhận biết và chuyển đổi ngôn ngữ nói sang dạng văn bản. Speech recognition là một bài toán khó dành cho các nhà khoa học, vì tiếng nói luôn thay đổi theo thời gian và có sự khác biệt giữa tiếng nói của những người khác nhau, tốc độ nói, ngữ cảnh và môi trường khác nhau.

Những hệ thống speech recognition đầu tiên trên thế giới có khả năng nhận biết rất hạn chế: số lượng từ vựng mà chúng có thể nhận biết chỉ ở mức vài chục, và chỉ có thể nhận biết chính xác nếu người dùng nói một cách rất rõ ràng. Ngày nay, với sự phát triển bùng nổ của deep learning và big data, các công nghệ speech recognition cũng đã phát triển rất nhanh về số lượng từ vựng và độ chính xác. Các hệ thống speech recognition hiện đại nhất có thể nhận biết được hàng chục nghìn, thậm chí là hàng trăm nghìn từ khác nhau, và độ lỗi khi nhận biết đã giảm đến gần mức nhận biết của con người. Ngày càng nhiều công ty công nghệ đã tham gia vào cuộc đua về speech

recognition, có thể kể đến Google, Microsoft, IBM, Baidu, Apple, Amazon,...

3.2 Mô hình hoạt động

Hiện nay có rất nhiều kỹ thuật khác nhau để phát triển speech recognition. Tuy nhiên có thể nhận thấy một điểm chung của đa phần các kỹ thuật này là các mô hình thống kê, trong đó nổi bật là Hidden Markov Model (HMM).

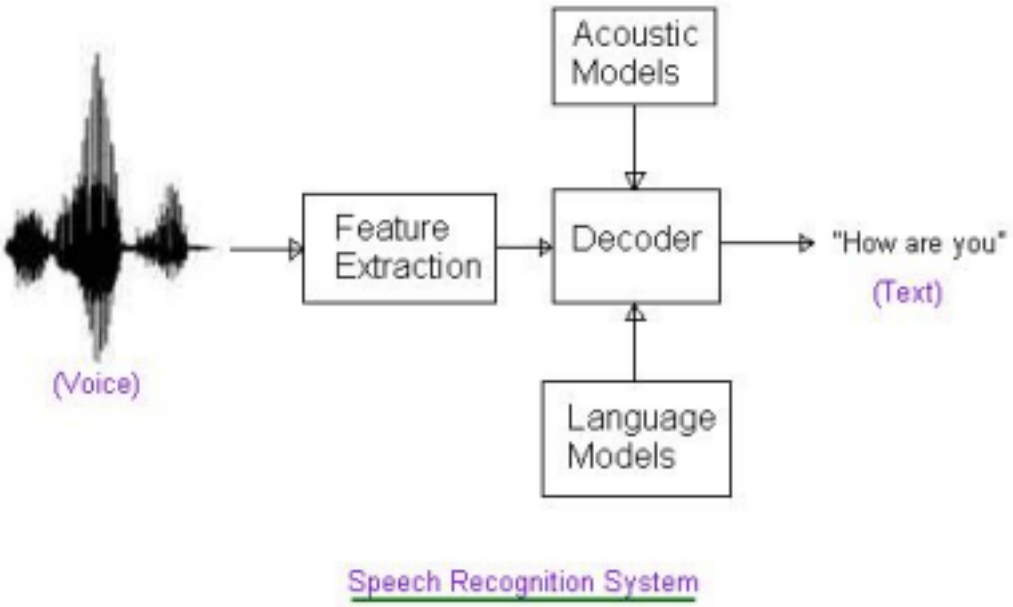
Khi huấn luyện, các câu nói trong tập dữ liệu sẽ thường được chia thành các thành phần âm (phonemes), tín hiệu âm thanh đầu vào thường sẽ được chia thành những đoạn ngắn, đưa qua các bước tiền xử lý như FFT hoặc MFCC để trích xuất đặc trưng. Hệ thống speech recognition sẽ sử dụng HMM để tìm sự liên hệ giữa các phonemes và các đặc trưng của tín hiệu âm thanh. Khi đó, khi một đoạn âm thanh mới được đưa vào, HMM sẽ trả về những chuỗi âm có thể tương ứng với đoạn âm thanh đó. Mô hình này được gọi là acoustic model.

Bên cạnh acoustic model, người ta cũng sẽ tạo ra các language model. Language model thường sẽ gồm:

- Danh sách các từ (gọi là từ điển). Hệ thống sẽ nhận biết được những từ nằm trong từ điển này.
- Cấu trúc âm của các từ trong từ điển, ví dụ như từ "stuff" gồm 3 âm: "st", "uh" và "ff".
- Mối quan hệ giữa các từ trong từ điển (từ nào thường đứng cạnh từ nào).

Language model sẽ nhận các chuỗi âm output từ acoustic model, tìm cách ghép các chuỗi âm này thành các chuỗi từ có trong từ điển, và tìm cách ghép các chuỗi từ này thành câu.

Ngoài HMM, nhiều mô hình khác cũng đã được các nhà phát triển speech recognition sử dụng để thay thế hoặc hỗ trợ cho HMM như Gaussian Mixture Model, Deep Neural Networks, Recurrent Neural Networks, Finite-State Transducers...



Hình 3.1: Cấu trúc một hệ thống speech recognition đơn giản

3.3 Ứng dụng

- Chuyển hướng cuộc gọi tự động, quay số bằng giọng nói, tìm kiếm bằng giọng nói.
- Dùng trong việc ra lệnh bằng giọng nói trên các máy bay quân sự.
- Dùng trong các ứng dụng dạy học ngoại ngữ.
- Phụ đề tự động trong các video.
- Điều khiển robot bằng giọng nói.
- Hỗ trợ người khuyết tật.
- Phiên dịch tự động.
- ...

3.4 Các vấn đề cần giải quyết

Speech to Text luôn đóng vai trò là một trong những phần quan trọng nhất trong một ứng dụng trợ lý ảo. Trong một hệ thống trợ lý ảo điển hình, sẽ có hai vấn đề lớn về speech recognition: dò tìm keyword và chuyển đổi lệnh của người dùng thành văn bản.

3.4.1 Dò tìm keyword

Mỗi hệ thống trợ lý ảo sẽ được xác định trước một keyword, keyword này sẽ được dùng để kích hoạt trợ lý ảo. Mỗi khi người dùng gọi keyword này, ứng dụng sẽ chuyển sang trạng thái thu âm để ghi nhận lệnh từ phía người dùng. Keyword này thường là một từ hoặc cụm từ ngắn, dễ phát âm và dễ nhận biết, và thường chứa tên của ứng dụng, ví dụ như của Google Assistant là "OK Google", của Siri là "Hey Siri",...

Yêu cầu

Thứ nhất, chức năng dò tìm keyword của hệ thống phải hoạt động liên tục trong suốt quá trình ứng dụng chạy, do hệ thống phải có phản ứng ngay lập tức khi người dùng gọi keyword. Nếu chức năng này bị gián đoạn, hệ thống sẽ dễ bỏ lỡ mất keyword. Do đó, phần nhận biết tiếng nói trong chức năng này nên hoạt động offline, để tránh những gián đoạn trên đường mạng.

Thứ hai, tốc độ nhận biết keyword phải rất nhanh. Hệ thống sẽ liên tục thu âm từ microphone thành các frame âm thanh để xử lý liên tục trong thời gian thực, và phải liên tục kiểm tra các đoạn âm thanh liên tiếp nhau xem có chứa keyword hay không. Khi phát hiện ra một đoạn âm thanh thu được có chứa keyword, hệ thống phải ngay lập tức phản hồi lại cho người dùng và chuyển sang trạng thái thu âm lệnh của người dùng. Nếu tốc độ nhận biết keyword chậm, việc chuyển trạng thái sang thu âm sẽ bị trễ, và lệnh của người dùng khi thu vào có thể sẽ không đầy đủ, dẫn tới phản hồi sai.

Thứ ba, độ chính xác của việc nhận biết keyword phải đạt mức tương đối cao. Do từ khóa được chọn là một từ dễ đọc và dễ nhận biết nên độ chính xác của chức năng này không cần phải quá cao. Ngoài ra việc đặt ngưỡng nhận biết ở mức không quá cao sẽ làm hạn chế tối đa số lượng false negatives (trường hợp người dùng gọi keyword

nhưng ứng dụng không nhận ra). Tuy nhiên điều đó sẽ làm gia tăng số lượng false positives (trường hợp người dùng không gọi keyword nhưng ứng dụng lại nhận ra), do đó cần phải tìm giải pháp để giảm số lượng false positives này.

Giải pháp 

Trong hệ thống này, chúng tôi sẽ sử dụng thư viện pocketsphinx để cài đặt chức năng phát hiện keyword. Đây là một thư viện speech recognition có khả năng hoạt động offline, tốc độ xử lý nhanh, tuy nhiên độ lỗi của thư viện này là lớn hơn so với một vài thư viện khác.

Ngoài ra, ngay khi pocketsphinx nhận ra được keyword trong một đoạn âm thanh, hệ thống sẽ gửi đoạn âm thanh đó vào thư viện Google Speech to Text để kiểm tra lại một lần nữa. Hai bước kiểm tra qua hai model khác nhau sẽ giúp hệ thống hạn chế được số lượng false positives.

3.4.2 Chuyển đổi lệnh người dùng thành văn bản

Sau khi phát hiện ra keyword, hệ thống sẽ được kích hoạt và bắt đầu thu âm lệnh của người dùng. Sau khi người dùng nói xong, hệ thống sẽ phải chuyển lệnh của người dùng sang dạng văn bản để có thể xử lý và hiểu được lệnh đó.

Yêu cầu

Phần chức năng chuyển đổi câu lệnh của người dùng thành văn bản chỉ bắt đầu hoạt động khi người dùng muốn ra lệnh, tức là sau khi người dùng gọi keyword, còn những khoảng thời gian khác, chức năng này sẽ nằm ở trạng thái chờ. Việc thu âm lệnh người dùng chỉ ngừng lại khi người dùng ngừng ra lệnh. Nếu hệ thống ngừng thu âm quá sớm, lệnh của người dùng thu vào sẽ bị thiếu và không chính xác. Nếu hệ thống ngừng thu âm quá muộn sẽ tạo ra độ trễ lớn từ lúc người dùng ra lệnh đến lúc ứng dụng phản hồi, và có thể tạp âm sau khi người dùng nói xong sẽ lọt vào phần speech recognition khiến kết quả nhận biết bị sai lệch.

Ngoài ra, độ chính xác của việc chuyển lệnh của người dùng về văn bản phải đạt mức rất cao, để đảm bảo hệ thống có thể hiểu đúng được ý định của người dùng.

Giải pháp

Trong hệ thống trợ lý ảo này, module chuyển đổi lệnh người dùng thành văn bản được cài đặt sử dụng thư viện Google Speech to Text. Đây là công cụ nhận biết tiếng nói của Google được đánh giá rất cao về độ chính xác, nhờ đó hệ thống sẽ đảm bảo hiểu đúng được hầu hết các lệnh của người dùng.

Để việc thu âm có thể ngừng đúng lúc, trong hệ thống sẽ có một giá trị thời gian T và một ngưỡng cường độ D. Khi âm thanh thu được từ microphone có cường độ nằm dưới mức D trong một khoảng thời gian T liên tục thì hệ thống sẽ xem như người dùng đã kết thúc việc ra lệnh và dừng trạng thái thu âm để bắt đầu xử lý đoạn âm thanh đã thu được.

3.5 Thư viện pocketsphinx

3.5.1 Tổng quan

Pocketsphinx là một thư viện speech recognition mã nguồn mở được phát triển bởi Viện Công nghệ Ngôn Ngữ, thuộc trường Đại học Carnegie Mellon (Mỹ). Pocketsphinx là phiên bản tối ưu của CMU Sphinx-II, cũng là một hệ thống nhận dạng tiếng nói khá nổi tiếng của ĐH Carnegie Mellon. Nhóm nghiên cứu này đã tối ưu hóa pocketsphinx rất nhiều về bộ nhớ và thuật toán, nhằm có thể tạo ra một thư viện nhận dạng tiếng nói liên tục có khả năng đưa vào các thiết bị cầm tay.

Hệ thống của pocketsphinx chủ yếu dựa trên những mô hình phổ biến của speech recognition như Gaussian Mixture Model và thuật toán Viterbi trên Hidden Markov Model, kết hợp với nhiều thuật toán hỗ trợ khác[3].

Pocketsphinx được đánh giá cao nhờ vào khả năng hoạt động offline và tốc độ xử lý khá nhanh. Ngoài ra, pocketsphinx có khả năng xử lý real-time liên tục trên stream âm thanh (thay vì phải hoàn tất việc thu âm và xử lý trên file âm thanh thu được).

3.5.2 Cách cài đặt

- Windows, Mac OS, Linux: `pip install pocketsphinx`

3.5.3 Cách sử dụng

- Trước hết, để khởi tạo thư viện Pocketsphinx ta cần thực hiện 3 bước:
  - Bước 1: Khởi tạo đối tượng config.
  - Bước 2: Khởi tạo các tham số cho đối tượng config. Trong đó:
    - `-hmm`: đường dẫn đến thư mục chứa Hidden Markov model của thư viện. Đối với ngôn ngữ Anh, thư mục này có tên là "en-us" và nằm tại thư mục mà thư viện được cài.
    - `-dict`: đường dẫn đến tập tin từ điển của thư viện. Tập tin này thường có tên là "cmudict-en-us.dict" đối với ngôn ngữ Anh.
    - `-keyphrase`: là keyword mà bạn muốn pocketsphinx theo dõi.
    - `-kws_threshold`: là ngưỡng dùng để xác định độ nhạy của pocketsphinx. Ngưỡng này càng thấp thì pocketsphinx càng nhạy, nghĩa là khả năng False Positive khi dò tìm keyword cũng sẽ tăng. Ngược lại, ngưỡng này càng cao thì pocketsphinx càng kém nhạy, khả năng bỏ lỡ keyword (False Negative) sẽ tăng lên.
  - Bước 3: Khởi tạo pocketsphinx bằng đối tượng config.

Đoạn mã mô phỏng cách khởi tạo thư viện Pocketsphinx

```
from pocketsphinx import *

#step 1
config = Decoder.default_config()

#step 2
config.set_string('-hmm', "path to hmm model")
config.set_string('-dict', "path to dict")
config.set_string('-keyphrase', "keyword")
config.set_float('-kws_threshold', threshold)

#step 3
self.decoder = Decoder(config)
```

- Sau khi khởi tạo, ta sẽ bắt đầu dò keyword bằng cách gọi hàm `start_utt()`. Lần lượt đọc dữ liệu âm thanh input và xử lý bằng hàm `process_raw(data)`. Ta kiểm tra kết quả trả về của hàm `hyp()`, nếu kết quả khác `None` nghĩa là đã dò thấy keyword.

  Đoạn mã mô phỏng cách sử dụng pocketsphinx để dò keyword

```
self.decoder.start_utt()
while True:
   data = readInputData()
   self.decoder.process_raw(data, False, False)
         if self.decoder.hyp() != None:
            # Keyword detected
            self.decoder.end_utt()
            self.decoder.start_utt()
```

3.5.4 Các ưu, nhược điểm

- **Ưu điểm:** Hỗ trợ xử lý offline, xử lý real-time liên tục, hỗ trợ chức năng dò tìm keyword.
- **Nhược điểm:** Độ chính xác khá kém. Độ lỗi word error rate khi thử nghiệm của pocketsphinx lên đến 13.95%[3].

3.5.5 Ứng dụng

Trong hệ thống trợ lý ảo của khóa luận này, pocketsphinx sẽ được sử dụng trong module WakeUp. Pocketsphinx sẽ giúp phát hiện ra khi người dùng gọi wake-up word, qua đó kích hoạt hệ thống.

3.6 Thư viện Google Speech To Text

3.6.1 Tổng quan

Google Speech Recognition là hệ thống nhận biết tiếng nói do Google nghiên cứu và phát triển. Hệ thống này đã được Google tích hợp vào trợ lý ảo Google Assistant

trên các thiết bị Android và loa thông minh Google Home, và cũng đã được đưa vào ứng dụng tìm kiếm Google Search. Google cũng đã đưa ra Google Speech API cho phép các lập trình viên sử dụng hệ thống speech recognition này vào các ứng dụng của họ. Google Speech to Text là một thư viện trên Python có khả năng nhận biết tiếng nói nhờ vào việc gọi đến Google Speech API.

Trong quá trình phát triển, các nhà nghiên cứu của Google đã sử dụng nhiều kỹ thuật khác nhau cho Google Speech Recognition, từ những kỹ thuật cổ điển như Gaussian Mixture Model, đến những kỹ thuật hiện đại hơn như Deep Neural Networks hay Long Short-term Memory Recurrent Neural Networks (LSTM RNNs)[2].

Nhờ sự thay đổi và tiến bộ liên tục, Google Speech Recognition được đánh giá rất cao về độ chính xác của mình. Tại Google I/O 2017, Google đã công bố độ chính xác của công nghệ nhận biết tiếng nói này đã đạt độ lỗi word error rate là 4.9%[6], tức là độ chính xác lên đến hơn 95%.

3.6.2 Cách cài đặt

- Windows, Mac OS, Linux: `pip install SpeechRecognition`

3.6.3 Cách sử dụng

- Thư viện Google Speech to Text sử dụng hết sức đơn giản gồm 2 bước
  - Bước 1: Import module `speech_recognition` và lớp `AudioData`.
  - Bước 2: Gọi hàm `sr.Recognizer().recognize_google` để thu được văn bản kết quả. Chú ý hàm này không thể xử lý trực tiếp với dữ liệu âm thanh thô mà phải được bao bởi lớp AudioData với các tham số:
    - `audio`: dữ liêu âm thanh thô
    - `sample_rate`: tần số lấy mẫu của âm thanh.
    - `sample_width`: số byte dùng để biểu diễn mỗi mẫu của âm thanh.

Đoạn mã mô phỏng cách sử dụng thư viện Google Speech to Text

```
#step 1
import speech_recognition as sr
```

```
from speech_recognition import AudioData

#step 2
text = sr.Recognizer().recognize_google(AudioData(audio, sample_rate,
    sample_width))
```

3.6.4 Các ưu, nhược điểm

- **Ưu điểm:** Độ chính xác rất cao.
- **Nhược điểm:** Yêu cầu internet để hoạt động, không xử lý real-time trên stream âm thanh được mà phải gửi toàn bộ file âm thanh.

3.6.5 Ứng dụng

Trong hệ thống này, thư viện Google Speech to Text được dùng trong module SpeechToText. Module này sẽ đóng vai trò chuyển đổi những câu lệnh của người dùng từ dạng âm thanh sang dạng văn bản.

Chương 4

Text To Speech

Nội dung chương 4 sẽ giới thiệu tổng quan về bài toán Text to Speech, mô hình hoạt động, các ứng dụng của Text to Speech, các vấn đề cần giải quyết của module Text to Speech trong một hệ thống trợ lý ảo và cách giải quyết các vấn đề đó. Chương 4 cũng sẽ giới thiệu về chức năng, cách cài đặt, cách sử dụng cũng như các ưu nhược điểm của các thư viện iSpeech và Google Text to Speech.

4.1 Tổng quan

Text to Speech (TTS), hay còn gọi là hệ thống tổng hợp giọng nói, là một lĩnh vực trong khoa học máy tính. Text to Speech nghiên cứu phương pháp tạo ra giọng nói nhân tạo từ văn bản. Giọng nói nhân tạo này được đánh giá dựa trên hai tiêu chí: mức độ tự nhiên và mức độ dễ nghe. Mức độ tự nhiên chỉ sự tương đồng về ngữ điệu của giọng nói tổng hợp với giọng nói con người. Mức độ dễ nghe đánh giá khả năng phát âm rõ ràng, và khả năng nghe hiểu của con người với giọng nói tổng hợp. Việc có quá nhiều ngôn ngữ trên thế giới, cộng thêm việc mỗi ngôn ngữ lại có nhiều ngữ điệu khác nhau tùy vùng miền đã đặt ra những thách thức không hề đơn giản cho các nhà khoa học.

Text to Speech đã được bắt đầu phát triển từ rất lâu trước đây và đã trải qua một quá trình cải tiến lâu dài. Có thể nói khởi nguồn của nó là mô hình bắt chước giọng nói người với năm nguyên âm (u, e, o , a, i), được phát triển vào năm 1779 bởi nhà khoa học người Đan Mạch Christian Kratzenstein tại viện hàn lâm khoa học Nga. Từ đó đến nay, sau nhiều năm phát triển cải tiến, hệ thống tổng hợp giọng nói đã có nhiều

bước phát triển vượt bật, giọng nói tạo ra ngày càng giống với ngữ điệu người và hỗ trợ nhiều loại ngôn ngữ trên thế giới. Hiện nay, có rất nhiều công ty tham gia vào phát triển hệ thống Text to Speech, trong đó nổi bật là các công ty: Google, Microsoft, iSpeech, Amazon,...

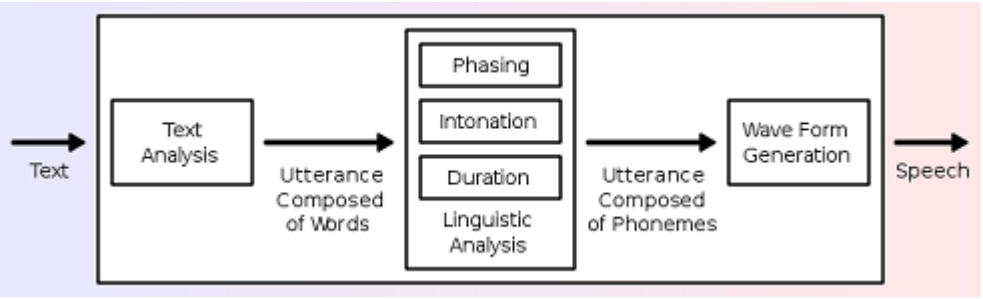
4.2 Mô hình hoạt động

Một hệ thống Text to Speech thông thường bao gồm hai thành phần chính: front-end và back-end.

- Front-end, hay bộ phận tiền xử lý (Pre-processor), thực hiện ba nhiệm vụ chính.
  - Phân đoạn (Tokenization): là quá trình phân tích cấu trúc của đoạn văn. Quá trình này phân phia và đánh dấu văn bản thành từng từ, nhóm từ, mệnh đề, câu văn và đoạn văn.
  - Chuẩn hóa (Normalization): là quá trình chuyển đổi các ký tự số, ký tự đặc biệt, viết tắt trong văn bản về dạng viết đầy đủ. Ví dụ "Dr" sẽ được chuyển đổi thành "doctor".
  - Phân tích ngôn ngữ (Linguistic analysis): bao gồm phân tích hình thái học (Morphological analysis) để tìm ra cách phát âm tương ứng của từng từ và phân tích cú pháp (Syntactic analysis) nhằm hiểu cách diễn đạt hay nhấn mạnh của từ.
- Back-end, còn gọi là bộ phận tổng hợp âm thanh (Speech synthesizer). Nhiệm vụ của phần này là tổng hợp các thông tin từ front-end thành giọng nói ở dạng sóng âm thanh.

Có nhiều kỹ thuật dùng trong tổng hợp âm thanh. Tùy thuộc vào đặc tả của hệ thống, thiên về mức độ dễ nghe, thiên về mức độ tự nhiên, hay đồng thời cả hai tính chất trên mà sẽ lựa chọn kỹ thuật thích hợp. Có hai kỹ thuật chính thường được dùng là tổng hợp ghép nối và tổng hợp cộng hưởng tần số, ngoài ra còn có một số kỹ thuật khác.

- **Tổng hợp ghép nối**: Tổng hợp ghép nối là kỹ thuật tạo tiếng nói từ việc nối các đoạn âm thanh đã được ghi âm trước. Kỹ thuật này cho ra giọng nói khá tự



Hình 4.1: Minh họa mô hình hoạt động của hệ thống Text to Speech

nhiên. Tuy nhiên, do các lần ghi âm có chất lượng không đồng đều, nên sẽ tạo rao những âm thanh cọ xát khó chịu tại các điểm ghép nối. Có ba kiểu ghép nối tổng hợp:

- **Tổng hợp chọn đơn vị**: Tổng hợp chọn đơn vị dùng một cơ sở dữ liệu lớn các giọng nói ghi âm (thông thường dài hơn 1 giờ đồng hồ ghi âm). Trong lúc ghi âm, mỗi câu phát biểu được tách ra thành các đơn vị khác như: các âm tỏ lời đơn lẻ, âm tiết, hình vị, từ, nhóm từ, và câu văn. Thông thường, việc tách ra như vậy cần một máy nhận dạng tiếng nói được đặt ở chế độ khớp với văn bản viết tương ứng với đoạn ghi âm, và dùng đến hiển thị sóng âm và phổ âm thanh. Một bảng tra các đơn vị được lập ra dựa trên các phần đã tách và các thông số âm học như tần số cơ bản, thời lượng, vị trí của âm tiết, và âm tỏ lời gần đó. Khi chạy, các câu phát biểu được tạo ra bằng cách xác định chuỗi đơn vị phù hợp nhất từ cơ sở dữ liệu. Quá trình này được gọi là chọn đơn vị, và thường cần dùng đến cây quyết định để thực hiện.

  Kỹ thuật chọn đơn vị tạo ra độ tự nhiên cao do không áp dụng các kỹ thuật xử lý tín hiệu số lên các đoạn giọng nói đã ghi âm, tuy rằng một số hệ thống có thể áp dụng xử lý tín hiệu tại các đoạn nối giữa các đơn vị để làm liền mạch kết quả sau khi ghép nối. Thực tế, các hệ thống chọn đơn vị có thể tạo ra giọng nói không thể phân biệt được với người thật. Tuy nhiên, để đạt độ tự nhiên cao, thường cần một cơ sở dữ liệu lớn chứa các đơn vị để lựa chọn; có thể lên tới vài gigabyte, tương đương với hàng chục giờ ghi âm.

- **Tổng hợp âm kép**: Tổng hợp âm kép dùng một cơ sở dữ liệu giọng nói

nhỏ chứa tất cả các âm kép (chuyển tiếp âm thanh) xuất hiện trong ngôn ngữ đang xét. Số lượng âm kép phụ thuộc vào đặc tính ghép âm học của ngôn ngữ: tiếng Tây Ban Nha có 800 âm kép, tiếng Đức có 2500. Trong tổng hợp âm kép, chỉ có một ví dụ của âm kép được chứa trong cơ sở dữ liệu. Khi chạy, lời văn được chồng lên các đơn vị này bằng kỹ thuật xử lý tín hiệu số như mã tiên đoán tuyến tính, PSOLA hay MBROLA.

Chất lượng của âm thanh tổng hợp theo cách này thường không cao bằng phương pháp chọn đơn vị nhưng tự nhiên hơn tổng hợp cộng hưởng tần số. Tổng hợp âm kép tạo ra các tiếng cọ xát ở phần ghép nối và đôi khi giọng nói kiểu robot do các kỹ thuật xử lý tín hiệu số gây ra. Lợi thế của phương pháp này là kích thước cơ sở dữ liệu nhỏ. Các ứng dụng thương mại của phương pháp này đang ít dần, tuy nhiên có nhiều hệ thống như thế này được phân phát tự do, và phục vụ cho nghiên cứu.

- **Tổng hợp chuyên ngành**: Tổng hợp chuyên biệt ghép nối các từ và đoạn văn đã được ghi âm để tạo ra lời phát biểu. Nó được dùng trong các ứng dụng có các văn bản chuyên biệt cho một chuyên ngành, sử dụng lượng từ vựng hạn chế, như các thông báo chuyến bay hay dự báo thời tiết.

  Công nghệ này rất đơn giản, và đã được thương mại hóa từ lâu, đã đi vào các đồ vật như đồng hồ biết nói hay máy tính bỏ túi biết nói. Mức độ tự nhiên của các hệ thống này có thể rất cao vì số lượng các câu nói không nhiều và khớp với lời văn và âm điệu của giọng nói ghi âm. Tuy nhiên các hệ thống này bị hạn chế bởi cơ sở dữ liệu chuyên ngành, không phục vụ mọi mục đích mà chỉ hoạt động với các câu nói mà chúng đã được lập trình sẵn.

- **Tổng hợp cộng hưởng tần số**: Tổng hợp cộng hưởng tần số không sử dụng bất cứ mẫu giọng thật nào khi chạy. Thay vào đó, tín hiệu âm thanh cho ra dựa trên một mô hình âm thanh. Các thông số như tần số cơ bản, sự phát âm, và mức độ tiếng ồn được thay đổi theo thời gian để tạo ra dạng sóng cho giọng nói nhân tạo. Phương pháp này đôi khi còn được gọi là tổng hợp dựa trên quy tắc, dù cho nhiều hệ thống ghép nối mẫu âm thanh thật cũng có dùng các thành phần dựa trên quy tắc.

Nhiều hệ thống dựa trên tổng hợp cộng hưởng tần số tạo ra giọng nói nhân tạo, như giọng rôbốt, không tự nhiên, và phân biệt rõ ràng với giọng người thật. Tuy nhiên độ tự nhiên cao không phải lúc nào cũng là mục đích của hệ thống và hệ thống này cũng có các ưu điểm riêng của nó.

Hệ thống này nói khá dễ nghe, ngay cả ở tốc độ cao, không có tiếng cọ xát do ghép âm tạo ra. các hệ thống này hoạt động ở tốc độ cao, có thể hướng dẫn người khiếm thị nhanh chóng dò dẫm trên máy tính, bằng cách đọc to những gì hiện ra trên màn hình. Các hệ thống này cũng nhỏ gọn hơn các hệ thống ghép nối âm, vì không phải chứa cơ sở dữ liệu mẫu âm thanh lớn. Nó có thể dùng trong các hệ thống nhúng khi bộ nhớ và tốc độ xử lý có hạn. Hệ thống này cũng có khả năng điều khiển mọi khía cạnh của tín hiệu âm thanh đi ra, no cho ra một dải rộng các lời văn và ngữ điệu, và không chỉ thể hiện được câu nói thường hay câu hỏi, mà cả các trạng thái tình cảm thông qua âm điệu của giọng nói.

Các ví dụ về các hệ thống cho ra ngữ điệu chính xác (nhưng không cho ra ngay lập tức sau khi nhận đầu vào) là các công trình cuối những năm 1970 của đồ chơi Speak & Spell của Texas Instruments, và các trò chơi video của SEGA đầu những năm 1980 như: Astro Blaster, Zektor, Space Fury, và Star Trek. Hiện vẫn chưa có hệ thống cho ra intonation chính xác ngay sau khi nhận văn bản đầu vào.

- **Tổng hợp mô phỏng phát âm**: Tổng hợp mô phỏng phát âm là các kỹ thuật tổng hợp giọng nói dựa trên mô hình máy tính của cơ quan phát âm của người và quá trình phát âm xảy ra tại đó. Hệ thống tổng hợp mô phỏng phát âm đầu tiên là ASY, thường được dùng cho các thí nghiệm trong nghiên cứu, được phát triển ở phòng thí nghiệm Haskins vào giữa những năm 1970 bởi Philip Rubin, Tom Baer, và Paul Mermelstein. ASY dựa trên mô hình cơ quan phát âm đã được tạo ra bởi phòng thí nghiệm Bell vào những năm 1960 và 1970 bởi Paul Mermelstein, Cecil Coker, và các đồng nghiệp khác. Tổng hợp mô phỏng phát âm đã từng chỉ là hệ thống dành cho nghiên cứu khoa học cho mãi đến những năm gần đây. Lý do là rất ít mô hình tạo ra âm thanh chất lượng đủ cao hoặc có thể chạy hiệu quả trên các ứng dụng thương mại. Một ngoại lệ là hệ thống dựa trên NeXT; vốn được phát triển và thương mại hóa bởi Trillium Sound Research

Inc, ở Calgary, Alberta, Canada. Đây là một công ty tách ra từ Đại học Calgary nơi các nghiên cứu ban đầu đã được thực hiện. Theo sau các vụ chuyển nhượng các từng phần của NeXT (bắt đầu từ Steve Jobs vào cuối những năm 1980 và việc hợp nhất với Apple năm 1997), phần mềm của Trillium được phân phát với giấy phéo tự do GPL. Dự án gnuspeech, một dự án của GNU, tiếp tục phát triển phần mềm này. Phần mềm gốc NeXT và các chuyển đổi sang cho Mac OS/X và GNUstep trong GNU/Linux có thể tìm thấy tại trang GNU savannah; chúng đều kèm theo tài liệu hướng dẫn trực tuyến và các bài viết liên quan đến lý thuyết nền tảng của công trình. Hệ thống, vốn được thương mại hóa lần đầu vào năm 1994, tạo ra một máy tổng hợp giọng nói dựa trên mô phỏng phát âm hoàn chỉnh, dựa trên mô hình ống dẫn sóng tương đương với cơ quan phát âm của người. Nó được điều khiển bởi Mô hình Phần Riêng biệt của Carré; bản thân mô hình này lại dựa trên công trình của Gunnar Fant và các người khác ở Phòng thí nghiệm Công nghệ Giọng nói Stockholm thuộc Viện Cộng nghệ Hoàng gia Thụy Điển về tổng hợp giọng nói cộng hưởng tần số. Công trình này cho thấy các cộng hưởng tần số trong ống cộng hưởng có thể được điều khiển bằng cách thay đổi tám tham số tương đồng với các cách phát âm tự nhiên của cơ quan phát âm của người. Hệ thống bao gồm một từ điển phát âm cùng với các quy tắc phát âm tùy thuộc ngữ cảnh để giúp ghép nối âm điệu và tạo ra các tham số phát âm; mô phỏng theo nhịp điệu và ngữ điệu thu được từ các kết quả nghiên cứu ngữ âm học.

- **Tổng hợp lai**: Các hệ thống tổng hợp lai kết hợp các yếu tố của tổng hợp cộng hưởng tần số với tổng hợp ghép nối để giảm thiểu các tiếng cọ xát khi ghép nối các đoạn âm thanh.

- **Tổng hợp dựa trên HMM** Tổng hợp dựa trên HMM là một phương pháp dựa vào mô hình Markov ẩn (HMM, viết tắt cho thuật ngữ tiếng Anh Hidden Markov model). Trong hệ thống này, phổ tần số của giọng nói, tần số cơ bản, và thời lượng đều được mô phỏng cùng lúc bởi HMM. Dạng sóng của giọng nói được tạo từ mô hình Markov ẩn dựa trên tiêu chí khả thực cực đại.

4.3 Ứng dụng

Hệ thống Text to Speech đang được nghiên cứu và ứng dụng trong nhiều lĩnh vực:

- Máy đọc văn bản dùng cho những người mù chũ, có thị lực kém hoặc khiếm thị.
- Dùng trong ngành công nghiệp robot.
- Là công cụ giao tiếp của các trợ lý ảo.
- Trong luận văn này, hệ thống Text to Speech được sử dụng làm phương tiện để ứng dụng phản hồi lại lệnh của người dùng.

4.4 Thư viện Google Text To Speech (gTTS)

4.4.1 Tổng quan

gTTS là thư viện mã nguồn mở, được viết bằng ngôn ngữ python và hỗ trợ hoạt động trên đa nền tảng. Thư viện này được phát triển bởi Pierre Nick Durette một lập trình viên người Canada. gTTS là một thư viện bao phủ trên nền Google's Text to Speech API cung cấp các tính năng giúp người dùng tương tác đơn giản hơn với hệ thống Text to Speech API của Google.

4.4.2 Chức năng

Thư viện gTTS có chứ năng chuyển một đoạn văn bản thanh giọng nói dưới dạng tập tin mp3. Thư viện này không giới hạn số từ của văn bản bằng cách chia đoạn văn bản ra thành các câu ngắn hơn tại các vị trí mà khi nói giọng người sẽ tạm ngừng một cách tự nhiên.

4.4.3 Cách cài đặt

- Windows, Mac OS, Linux: `pip install gTTS`

4.4.4 Cách sử dụng

Thư viện gTTS có thể được sử dụng như là một python module hoặc chạy trên command line

- Python Module

```
# Import gTTS
from gtts import gTTS

# Create an instance
 tts = gTTS(text='Hello', lang='en', slow=True)
#Parameters:
#text - String - Text to be spoken.
#lang - String - ISO 639-1 language code (supported by the Google
    Text to Speech API) to speak in.
#slow - Boolean - Speak slowly. Default False (Note: only two speeds
    are provided by the API).

#Write to a file
#To disk using save(file_name)
tts.save("hello.mp3")

#To a file pointer using write_to_fp(file_object)
f = TemporaryFile()
tts.write_to_fp(f)
# <Do something with f>
f.close()
```

- Command line

```
gtts-cli.py [-h] (["text to speak"] | -f FILE) [-l LANG] [--slow] [--
    debug] [-o destination_file]
$ # Example:
$ # Read the string 'Hello' in English to hello.mp3
$ gtts-cli "Hello" -l 'en' -o hello.mp3

$ # Read the string 'Hello' in English (slow speed) to hello.mp3
```

```
$ gtts-cli "Hello" -l 'en' -o hello.mp3 --slow

$ # Read the contents of file 'hello.txt' in Czech to hello.mp3:
$ gtts-cli -f hello.txt -l 'cs' -o hello.mp3

$ # Read the string 'Hello' from stdin in English to hello.mp3
$ echo "Hello" | gtts-cli -l 'en' -o hello.mp3 -
```

4.4.5 Ưu điểm và nhược điểm

- **Ưu điểm**:
  - Thư viện nhẹ, dễ sử dụng.
  - Google's Text to Speech API hỗ nhiều nhiều ngôn ngữ, nhiều giọng đọc và cho ra giọng nói khá hay.
  - Không giới hạn số từ của văn bản.
- **Khuyết điểm**
  - Chỉ có thể xuất ra tập tin mp3
  - Không hỗ trợ stream âm thanh.

4.5 iSpeech

4.5.1 Tổng quan

iSpeech là một dịch vụ chuyển văn bản thành tiếng nói dưới dạng giao thức GET API. iSpeech được phát triển công ty iSpeech, một công ty được thành lập năm 2007 chuyên cung cấp các giải pháp về nhận diện tiếng nói và chuyển đổi văn bản thành tiếng nói. iSpeech là một dịch vụ có thu phí, tuy nhiên công ty có cung cấp phiên bản demo miễn phí nhưng có nhiều giới hạn.

4.5.2 Chức năng

Chuyển đổi văn bản đầu vào thành tiếng nói dưới dạng tập tin mp3. Phiên bản demo của iSpeech giới giạn số từ tối đa trong văn bản là 36 từ.

4.5.3 Cách cài đặt

iSpeech là dịch vụ chạy hoàn toàn trên nền Web nên không cần bất cứ thao tác cài đặt nào.

4.5.4 Cách sử dụng

iSpeech được sử dụng bằng cách gửi request đến server bằng liên kết có dạng
https://www.ispeech.org/p/generic/getaudio?action=convert&voice=usenglishfemale&speed=0&text=good+morning

Trong đó:

- **voice**: tham số xác định giọng người đọc. iSpeech hỗ trợ hơn 30 giọng đọc với nhiều ngôn ngữ khác nhau. Xem thêm các giọng đọc khác mà iSpeech hỗ trợ http://www.ispeech.org/api/#voices-standard.
- **speed**: tốc độ đọc giá trị này nằm trong khoảng từ -20 đến 20.
- **text**: là văn bản cần chuyển thành tiếng nói. Các từ tỏng văn bản được nối với nhau bởi ký từ '+'.
- Xem thêm các tham số mà iSpeech hỗ trợ http://www.ispeech.org/api/#request-parameters

4.5.5 Ưu điểm, nhược điểm

- **Ưu điểm**:
  - Giọng đọc rất giống với con người.
  - Hỗ trợ stream giọng đọc trực tiếp từ server.
  - Không cần cài đặt, dễ dàng sử dụng

- **Nhược điểm**:
  - Giới hạn 36 từ mỗi lần đọc.
  - Giá của dịch vụ quá cao.

Chương 5

Intent Classification và Entity Extraction

Nội dung chương 5 sẽ giới thiệu về hai bài toán Intent Classification và Entity Extraction, mô hình hoạt động và ứng dụng của chúng. Chương 5 cũng sẽ giới thiệu về chức năng, cách cài đặt, cách sử dụng của thư viện Rasa NLU, một thư viện được dùng để giải quyết hai bài toán Intent Classification và Entity Extraction.

5.1 Tổng quan

Intent classification, hay intent recognition, là bài toán xác định intent (ý định) của một câu nói. Bài toán này bắt nguồn từ các hệ thống chuyển hướng cuộc gọi. Một hệ thống intent classification khi nhận vào một câu nói sẽ trả về kết quả là một lớp intent đã được định nghĩa trong hệ thống. Ví dụ: Câu nói "Good morning." có thể cho kết quả intent là "greetings". (kết quả có thể khác phụ thuộc vào việc định nghĩa các lớp intent trong hệ thống).

Entity extraction, hay named-entity recognition (NER), là bài toán xác định các entity (thực thể) trong một câu nói và phân lớp chúng về các loại entity đã được định nghĩa sẵn, ví dụ như tên người, tên tổ chức, địa điểm, thời gian,... Các lớp entity này cũng sẽ được định nghĩa sẵn trong hệ thống. Đa phần các hệ thống entity extraction sẽ chia câu nói thành các cụm ký tự rời nhau (cách nhau bởi khoảng trắng hoặc dấu câu), và các entity trong câu sẽ là một hoặc nhiều cụm liên tiếp nhau. Ví dụ: Câu "Jim bought 300 shares of Acme Corp. in 2006." sau khi được xử lý bởi một hệ thống

entity extraction có thể sẽ cho ra kết quả như sau:

$[\text{Jim}]_{\text{Person}}$ bought 300 shares of $[\text{Acme Corp.}]_{\text{Organization}}$ in $[2006]_{\text{Time}}$.

Trong ví dụ trên ta thấy có 3 entity được xác định trong câu nói:

- "Jim" gồm 1 cụm ký tự, thuộc lớp Person
- "Acme Corp." gồm 2 cụm ký tự, thuộc lớp Organiztion
- "2006" gồm 1 cụm ký tự, thuộc lớp Time

Intent classification và entity extraction là hai bài toán đặc trưng của natural language understanding (hiểu ngôn ngữ tự nhiên). Intent classification và entity extraction thường đi cùng nhau, giúp máy tính có thể "hiểu" được những gì người dùng muốn làm thông qua câu nói, thông qua việc biến đổi ngôn ngữ nói thành dạng dữ liệu có cấu trục. Việc một câu nói dài được rút ngắn lại thành một intent và một vài entity sẽ giúp việc xử lý và phản hồi của máy diễn ra hiệu quả hơn.

5.2 Mô hình hoạt động

Hai bài toán intent classification và entity extraction thường được xử lý một cách riêng biệt. Rất nhiều công trình đã được công bố trên hai bài toán này, với những cách tiếp cận rất đa dạng và phong phú.

Intent classification thường được giải quyết bằng những cách tiếp cận giống với những bài toán phân lớp cổ điển như support vector machines (SVM) và deep neural networks (DNN).

Trong khi đó, entity extraction thường được đưa về bài toán gán nhãn tuần tự (sequence labeling). Có rất nhiều loại mô hình đã được áp dụng cho bài toán này như hidden Markov models (HMM), maximum entropy Markov models (MEMM), conditional random fields (CRF), hay recurrent neural networks (RNN).

5.3 Ứng dụng

Một số ứng dụng của hai bài toán intent classification và entity extraction:

- Dùng trong các hệ thống trợ lý ảo, giúp trợ lý ảo hiểu được lệnh của người dùng.

- Dùng để phát triển các ứng dụng trả lời câu hỏi tự động.
- Dùng để phân tích hành vi người dùng thông qua các truy vấn tìm kiếm.

5.4 Thư viện Rasa NLU

5.4.1 Tổng quan

Rasa NLU là một công cụ mã nguồn mở được phát triển trên ngôn ngữ Python bởi Rasa, một công ty của Đức chuyên làm các sản phẩm về trí tuệ nhân tạo. Rasa NLU là một thư viện hoàn toàn miễn phí, cho phép các nhà phát triển đưa hai tác vụ intent classification và entity extraction vào phần mềm của mình. Rasa NLU đã được hàng nghìn lập trình viên trên thế giới sử dụng trên các ứng dụng chatbot hoặc trợ lý ảo.

Rasa NLU được xây dựng dựa trên các công cụ có sẵn khác như MITIE, spaCy và sklearn, tạo ra một API cấp cao đa nền tảng đơn giản và dễ sử dụng cho các nhà phát triển. Mặc dù được phát triển trên Python, các nhà phát triển có thể sử dụng Rasa NLU cho các dự án phần mềm trên tất cả các nền tảng khác nhau, nhờ vào việc cung cấp 2 hình thức tương tác: gọi hàm trong Python hoặc tự tạo một HTTP server chạy trên máy. Rasa NLU được đánh giá cao ở việc không thu phí và cho phép các nhà phát triển điều chỉnh cấu hình sao cho phù hợp nhất với dự án.

5.4.2 Cách cài đặt

Cài đặt Rasa NLU

- **Windows, Mac OS, Linux:** `pip install rasa_nlu`
- **Build từ source:**

```
git clone git@github.com:golastmile/rasa_nlu.git
cd rasa_nlu
pip install -r requirements.txt
python setup.py install
```

Cài đặt backend

Cần phải cài đặt MITIE, spaCy hoặc sklearn để làm backend cho Rasa NLU.

- **MITIE:** `pip install git+https://github.com/mit-nlp/MITIE.git`

  Sau khi cài đặt MITIE, cần tải về MITIE models: https://github.com/mit-nlp/MITIE/releases/download/v0.4/MITIE-models-v0.2.tar.bz2. Tìm file `total_word_feature_extractor.dat` lưu vào đâu đó và thêm dòng sau vào file `config.json` của Rasa NLU:

  `'mitie_file': '/path/to/total_word_feature_extractor.dat'`

- **Kết hợp spaCy và sklearn:**

  Cài đặt spaCy: `pip install -U spacy && python -m spacy download en`

  Cài đặt sklearn:

  - Cài Anaconda: https://www.continuum.io/downloads
  - `conda install scikit-learn`

- **Kết hợp MITIE và sklearn:** Cài đặt MITIE và sklearn theo hướng dẫn ở những phần trước.

5.4.3 Cách sử dụng

Cấu hình Rasa NLU

Một file cấu hình đơn giản của Rasa NLU:

config.json

```
{
  "pipeline": "mitie_sklearn",
  "mitie_file": "/path/to/total_word_feature_extractor.dat",
  "path": "./models",
  "data": "data.json"
}
```

- **"pipeline":** Loại backend muốn sử dụng, ví dụ ở đây sử dụng kết hợp MITIE và sklearn thì dùng `"mitie_sklearn"`.
- **"mitie_file":** Đường dẫn đến file `total_word_feature_extractor.dat` của MITIE models, chỉ cần dùng khi backend có sử dụng MITIE.
- **"path":** Đường dẫn đến thư mục để lưu model sau khi huấn luyện.
- **"data":** Đường dẫn đến file dữ liệu huấn luyện.

Xem thêm các trường khác có thể có trong file cấu hình tại: https://rasa-nlu.readthedocs.io/en/latest/config.html.

Huấn luyện model

Chạy lệnh sau để huấn luyện model:

```
python -m rasa_nlu.train -c config.json
```

Trong đó `config.json` là tên file cấu hình Rasa NLU.

Rasa NLU sẽ hoàn tất huấn luyện sau vài phút. Sau đó, trong thư mục trong trường `"path"` của file cấu hình sẽ xuất hiện thư mục mới có tên `model_YYYYMMDD-HHMMSS` trong đó `YYYYMMDD-HHMMSS` là thời điểm việc huấn luyện kết thúc.

Sử dụng model

Rasa NLU cho phép ta sử dụng bằng cách chạy một HTTP server hoặc thông qua các hàm trên Python.

Chạy HTTP server bằng cách chạy lệnh:

```
python -m rasa_nlu.server -c config.json --server_model_dirs=./model_YYYYMMDD-HHMMSS
```

Đường dẫn model là đường dẫn tương đối đối với đường dẫn trong trường `"path"` của file cấu hình. Khi server đã chạy, ta thực hiện truy vấn bằng cách gửi một GET request đến địa chỉ `http://localhost:5000/parse`. Ví dụ để truy vấn kết quả cho câu "what time is it in new york", ta gửi GET request đến `http://localhost:5000/parse?q=what+time+is+it+in+new+york`.

Để truy vấn bằng Python, ta cần tạo một đối tượng Interpreter:

```
from rasa_nlu.model import Metadata, Interpreter
from rasa_nlu.config import RasaNLUConfig

metadata = Metadata.load("model_YYYYMMDD-HHMMSS")
interpreter = Interpreter.load(metadata, RasaNLUConfig("config.json"))
```

Sau đó, mỗi lần truy vấn ta gọi hàm `interpreter.parse`, ví dụ như `interpreter.parse("what time is it in new york")`.

Kết quả của truy vấn sẽ là một chuỗi JSon có cấu trúc như sau:

```
{
  "text": "what time is it in new york",
  "entities": [
    {
      "start": 19,
      "end": 27,
      "entity": "location",
      "extractor": "ner_mitie",
      "value": "new york"
    }
  ],
  "intent": {
    "confidence": 0.77642937607205087,
    "name": "time.get"
  },
  "intent_ranking": [
    {
      "confidence": 0.77642937607205087,
      "name": "time.get"
    },
    {
      "confidence": 0.0265279318597689,
      "name": "general_question"
    }
  ]
}
```

- **"text":** Chuỗi câu nói input.
- **"entities":** Mảng chứa danh sách các entity được xác định trong câu.

  Thông tin của mỗi entity sẽ gồm `"start"` và `"end"` thể hiện vị trí của entity đó trong câu, `"entity"` là tên lớp entity, `"value"` là giá trị của entity.
- **"intent":** Thông tin về intent kết quả có độ tin cậy cao nhất, trong đó `"name"` là tên lớp intent và `"confidence"` là độ tin cậy.
- **"intent_ranking":** Mảng chứa danh sách các intent có confidence cao nhất, cùng với confidence tương ứng.

5.4.4 Chuẩn bị dữ liệu

Tập dữ liệu huấn luyện là một file JSon có cấu trúc như sau:

```
{
    "rasa_nlu_data": {
        "common_examples": []
    }
}
```

Trong đó mảng `"common_examples"` sẽ chứa tất cả các mẫu dữ liệu. Mỗi mẫu huấn luyện sẽ gồm 3 trường:

- **"text":** Chuỗi chứa câu nói input.
- **"intent":** Chuỗi chứa tên lớp intent tương ứng của input.
- **"entities":** Mảng chứa danh sách các entity có trong input.

  Mỗi entity sẽ được xác định bằng các giá trị `start` và `end` thể hiện vị trí bắt đầu và kết thúc của entity trong chuỗi, các giá trị này được tính theo kiểu của Python. Ví dụ với chuỗi `"what time is it in new york"` thì entity `"new york"` sẽ có vị trí là `start=19` và `end=27`.

Ví dụ về một mẫu huấn luyện:

```
{
  "text": "what time is it in new york",
  "intent": "time.get",
  "entities": [
    {
      "start": 19,
      "end": 27,
      "value": "new york",
      "entity": "location"
    }
  ]
}
```

Khi chuẩn bị dữ liệu huấn luyện, ta cần xác định tất cả các intent cần thiết cho hệ thống, sau đó xác định các loại entity có thể xuất hiện trong mỗi loại intent, sau đó tìm tất cả các dạng câu có thể có của intent đó. Với mỗi loại entity ta cần tạo ra nhiều mẫu input với các giá trị khác nhau cho entity để khả năng xác định entity của model sẽ chính xác hơn.

5.4.5 Đánh giá model 

Ta có thể đánh giá model bằng một tập dữ liệu test có cấu trúc tương tự tập dữ liệu huấn luyện, hoặc dùng chính tập dữ liệu huấn luyện để đánh giá. Dùng đoạn code sau để in ra độ chính xác của model trên tập dữ liệu test:

```
from rasa_nlu.model import Metadata, Interpreter
from rasa_nlu.config import RasaNLUConfig
import json

model_dir = "./models/model_YYYYMMDD-HHMMSS"
config_dir = "config.json"
test_data_dir = "data.json"

metadata = Metadata.load(model_dir)
interpreter = Interpreter.load(metadata, RasaNLUConfig(config_dir))
```

```
data = {}
with open(test_data_dir, "r") as f:
    data = json.load(f)

true = 0
false = 0

for dt in data['rasa_nlu_data']['common_examples']:
    parse = interpreter.parse(dt['text'])
    if parse['intent']['name'] == dt['intent']:
        true += 1
    else:
        false += 1

print true * 1.0 / (true + false)
```

5.4.6 Ứng dụng

Trong hệ thống trợ lý ảo của khóa luận, Rasa NLU sẽ được sử dụng trong module IntentDetector. Module này sẽ nhận vào các câu nói của người dùng dưới dạng văn bản, và đưa ra kết quả là intent và các entity của câu nói đó. Kết quả của IntentDetector sẽ được đưa sang module IntentProcessor để xử lý và đưa ra phản hồi phù hợp.

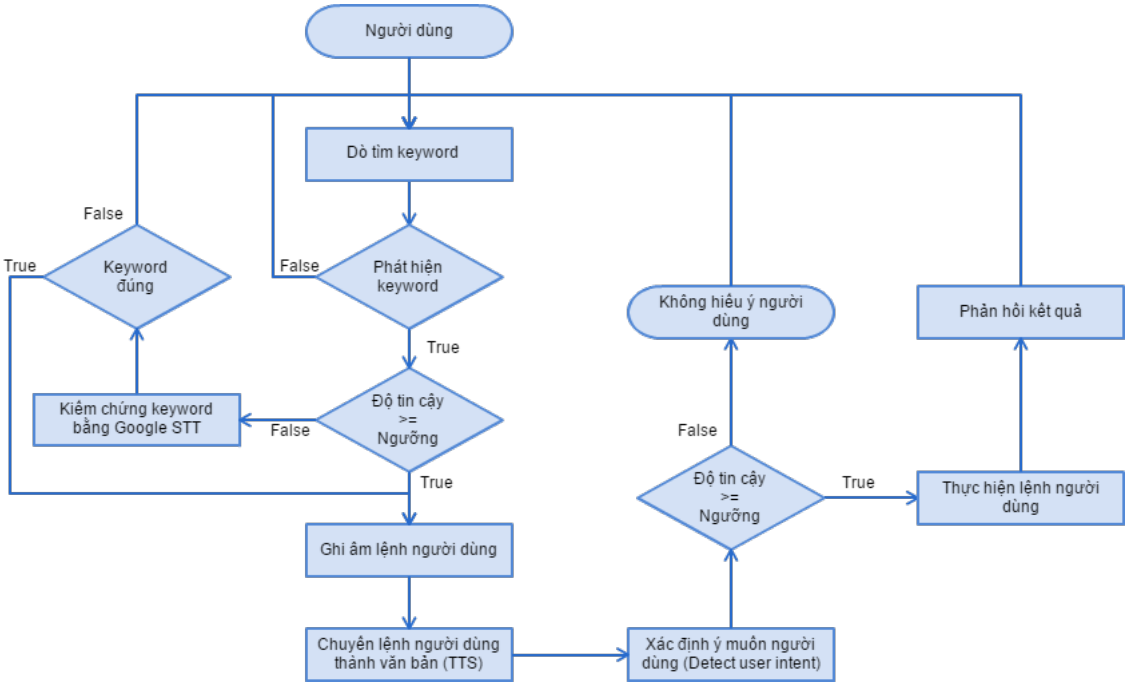
Chương 6

Ứng dụng Alexa

Nội dung chương 6 sẽ giới thiệu về ứng dụng Alexa, mô hình hoạt động và các thành phần của hệ thống. Chương 6 cũng sẽ giới thiệu về các chức năng của ứng dụng

6.1 Tổng quan

6.2 Mô hình hoạt động



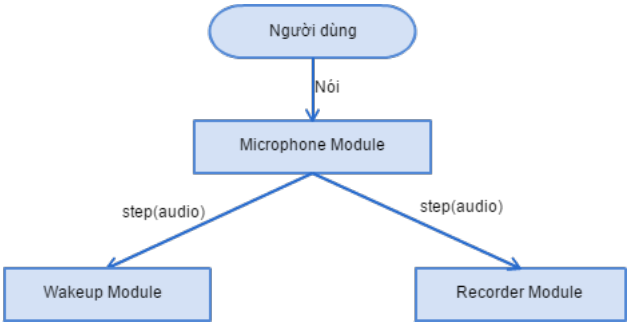
Hình 6.1: Mô hình hoạt động của ứng dụng Alexa

6.2.1 Các module chính

Microphone

- **Chức năng**: Microphone đảm nhiệm chức năng thu âm thanh cho toàn bộ ứng dụng. Module này cung cấp dữ liệu âm thanh cho hai module Recorder và Wakeup xử lý.
- **Cài đặt**: Module Microphone sử dụng thư viện PyAudio để cài đặt và chạy trên một thread riêng biệt với các module khác của ứng dụng.Khi hoạt động module này sẽ đọc dữ liệu audio từ microphone của máy tính và truyền lại cho các module khác sử lý thông qua hàm `step(audio)`.

  Đoạn code xử lý chính của module Microphone:



Hình 6.2: Mô hình hoạt động của module Microphone

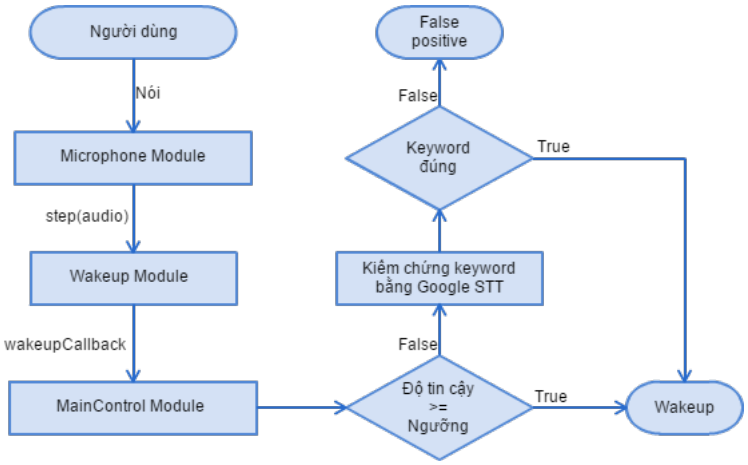
```
def run(self):
        if not self.mainControl.isRunning():
            return

        try:
            p = pyaudio.PyAudio()
            self.inputStream = p.open(format=pyaudio.paInt16, channels
                =cf.paChanels, rate=cf.paRate, input=True,
                frames_per_buffer=cf.paChunk)
            self.inputStream.start_stream()
        except:
            self.mainControl.onThread(self.mainControl.handleException
                , self, Microphone.ERR_INPUTSTREAM)
            return

        while True:
            if self.mainControl.isRunning() and self.stopFlag == False
                :
                buffer = self.inputStream.read(cf.paChunk)
                for listener in self.listeners:
                    listener.onThread(listener.step, buffer)
            else:
                self.inputStream.stop_stream()
                p.terminate()
                print "Microphone stop"
                break
        self.inputStream.stop_stream()
```

```
        self.inputStream.close()
```

- **API**: Các hàm tương tác mà module Microphone cung cấp
  - `addListener(listener)`: thêm đối tượng `listener` vào danh sách các đối tượng sẽ nhận được dữ liệu từ microphone.
  - `delListener(listener)`: xóa đối tượng `listener` khỏi danh sách các đối tượng sẽ nhận được dữ liệu từ microphone.
  - `stopThread()`: dừng thread Microphone.

Wakeup

- **Chức năng**: Module Wakeup có chức năng dò tìm keyword (trong ứng dụng này keyword là "Alexa"). Mỗi khi người dùng gọi keyword, module Wakeup sẽ phát hiện và chuyển ứng dụng sang trạng thái thu âm để ghi nhận lệnh từ phía người dùng.



Hình 6.3: Mô hình hoạt động của module Wakeup

- **Cài đặt**: Module Wakeup sử dụng thư viện Pocketsphinx để cài đặt. Module Wakeup sau khi nhận dữ liệu từ module Microphone sẽ gọi hàm `process_raw(audio)` của Pocketsphinx để xử lý. Khi Pocketsphinx xác định được keyword, module Wakeup sẽ gửi tín hiệu cho Module MainControl tiếp tục xử lý.

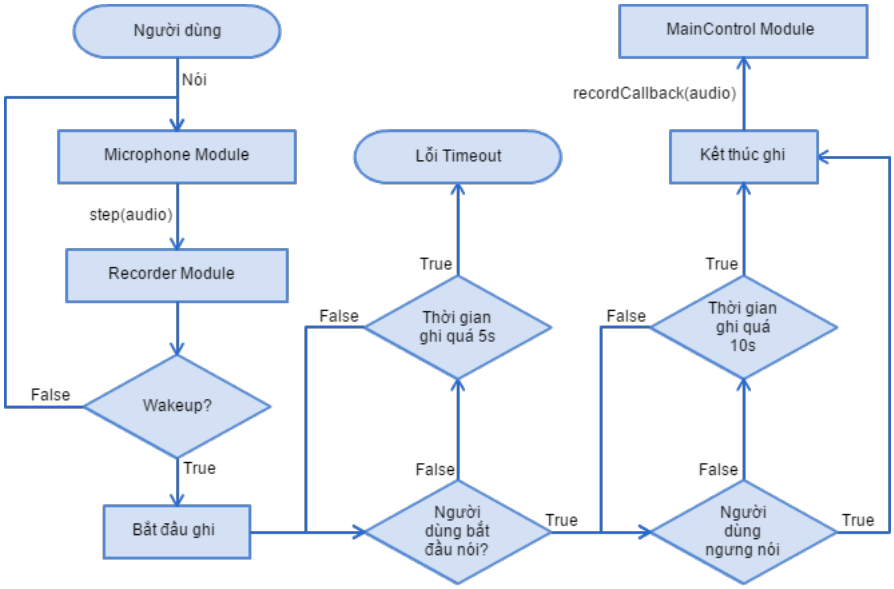
  Đoạn code xử lý chính của module Wakeup:

```
def step(self, data):
        self.caches.append(data)

        if self.verify > 0:
            self.verify -= 1

        if self.isPause or rs.isRecording:
            return

        if self.verify % 3 == 1:
            caches_data = b"".join(list(self.caches))
            self.mainControl.onThread(self.mainControl.wakeupVerify,
                caches_data)

        self.decoder.process_raw(data, False, False)
        if self.decoder.hyp() != None:
            for seg in self.decoder.seg():
                if seg.prob >= cf.cmuProbThreshole:
                    self.mainControl.onThread(self.mainControl.
                        wakeupCallback, seg.word, seg.prob)
                else:
                    self.verify = 5
                    caches_data = b"".join(list(self.caches))
                    self.mainControl.onThread(self.mainControl.
                        wakeupVerify, caches_data)
            self.decoder.end_utt()
            self.decoder.start_utt()
```

- **API**: Các hàm tương tác mà module Wakeup cung cấp
  - `step(audio)`: nhận dữ liệu audio và xử lý.
  - `pause()`: tạm dừng hoạt động.
  - `resume()`: tiếp tục hoạt động.
  - `stopThread()`: dừng thread Wakeup.

Recorder

- **Chức năng**: Module Recorder có chức năng ghi âm lệnh người dùng. Sau khi người dùng gọi keyword, module Recorder sẽ bắt đầu ghi âm và tự động dừng ghi ăm khi người dùng kết thúc lệnh.



Hình 6.4: Mô hình hoạt động của module Recorder

- **Cài đặt**: Module Recorder được cài đặt dựa trên mã nguồn của thư viện Google Speech to Text. Module Recorder sau khi nhận dữ liệu âm thanh từ module Microphone sẽ tính giá trị năng lượng của âm thanh (giá trị rms). Dựa vào giá trị năng lượng ta sẽ xác định được thời điểm người dùng bắt đầu nói và ngừng nói. Sau khi kết thúc ghi âm, đoạn âm thanh thu được sẽ được chuyển cho Module MainControl để tiếp tục xử lý.

  Đoạn code xử lý chính của module Recoder:

```
def step(self, buffer):
        if self.adjustAmbientNoise and not rs.isRecording:
            energy = audioop.rms(buffer, 2) # energy of the audio
                signal
            # dynamically adjust the energy threshold using
                assymmetric weighted average
            damping = self.dynamicEnergyAdjustmentDamping ** self.
                secondsPerBuffer # account for different chunk sizes
```

```
            and rates
        target_energy = energy * self.dynamicEnergyRatio
        self.energyThreshold = self.energyThreshold * damping +
            target_energy * (1 - damping)
        return

    # store audio input until the phrase starts
    if not self.isSpeaking:
        self.elapsedTime += self.secondsPerBuffer
        if self.elapsedTime >= cf.rcTimeout: # handle timeout if
            specified
            self.mainControl.onThread(self.mainControl.
                handleException, self, Recorder.ERR_TIMEOUT)
            rs.isRecording = False
            return

        self.frames.append(buffer)
        if len(self.frames) > self.nonSpeakingBufferCount: #
            ensure we only keep the needed amount of non-speaking
            buffers
            self.frames.popleft()

        # detect whether speaking has started on audio input
        energy = audioop.rms(buffer, 2) # energy of the audio
            signal
        if energy > self.energyThreshold:
            self.isSpeaking = True
            return

        # dynamically adjust the energy threshold using
            assymmetric weighted average
        if self.dynamicEnergyThreshold:
            damping = self.dynamicEnergyAdjustmentDamping ** self.
                secondsPerBuffer # account for different chunk
                sizes and rates
            target_energy = energy * self.dynamicEnergyRatio
            self.energyThreshold = self.energyThreshold * damping
```

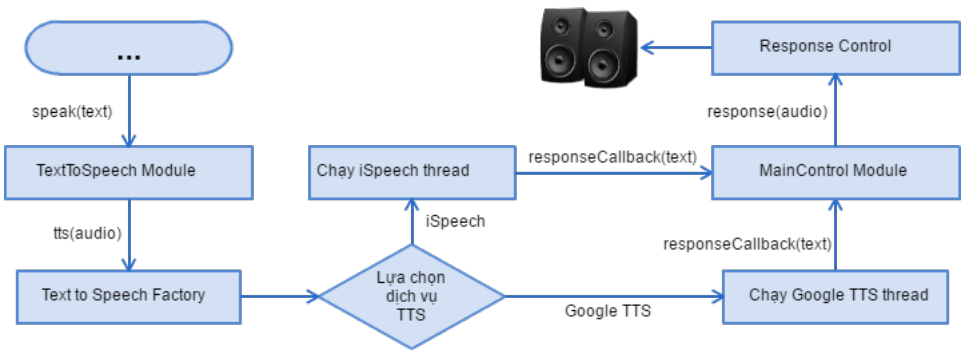
```
                + target_energy * (1 - damping)

        else:
            self.elapsedTime += self.secondsPerBuffer

            self.frames.append(buffer)
            self.phraseCount += 1

            if self.elapsedTime >= cf.rcMaxDuration:
                self.finishRecord()
                return

            # check if speaking has stopped for longer than the pause
                threshold on the audio input
            energy = audioop.rms(buffer, 2) # energy of the audio
                signal
            # print "begin speak: " + str(energy) + " - " + str(self.
                energyThreshold)
            if energy > self.energyThreshold:
                self.pauseCount = 0
            else:
                self.pauseCount += 1

            if self.pauseCount > self.pauseBufferCount: # end of the
                phrase
                # check how long the detected phrase is, and retry
                    listening if the phrase is too short
                self.phraseCount -= self.pauseCount
                if self.phraseCount < self.phraseBufferCount:
                    # self.mainControl.onThread(self.mainControl.
                        handleException, self, Recorder.
                        ERR_PHRASE_TOO_SHORT)
                    return

                for i in range(self.pauseCount - self.
                    nonSpeakingBufferCount):
                     self.frames.pop() # remove extra non-speaking
```

```
                    frames at the end
                self.finishRecord()
```

- **API**: Các hàm tương tác mà module Recorder cung cấp
  - `step(audio)`: nhận dữ liệu audio và xử lý.
  - `record()`: bắt đầu ghi âm.
  - `finishRecord()`: dừng việc ghi âm và gửi đoạn âm thanh thu được tới MainControl.
  - `stopThread()`: dừng thread Recorder.

Text To Speech

- **Chức năng**: Module Text to Speech có chức năng chuyển đổi văn bản thành giọng nói để phản hồi cho người dùng.



Hình 6.5: Mô hình hoạt động của module Text to Speech

- **Cài đặt**: Module Text to Speech (TTS) được cài đặt theo mẫu thiết kế Singleton và Factory. Nhờ mẫu Singleton module TTS có thể được sử dụng từ bất cứ module nào của hệ thống mà không làm phung phí tài nguyên khi phải tạo ra nhiều instance. Mẫu Factory cung cấp quyền lựa chọn dịch vụ TTS, hiện module TTS hỗ trợ hai dịch vụ đó là iSpeech và Google Text to Speech. Sau khi thực hiện xong công việc, tập tin âm thanh thu được sẽ được chuyển cho module MainControl để tiếp tục xử lý.

  Đoạn code xử lý chính của module Text to Speech:

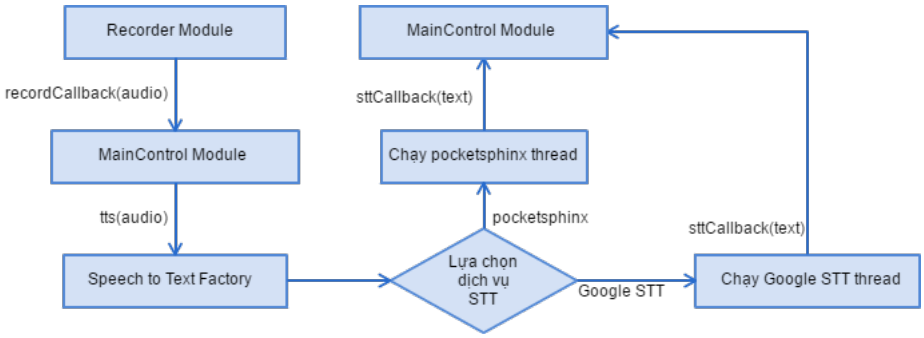
```
class TextToSpeech:
    instance = None
    @staticmethod
    def getInstance(callback = None):
        if TextToSpeech.instance is None:
            TextToSpeech.instance = TextToSpeech(callback)
        return TextToSpeech.instance

    def __init__(self, callback):
        self.callback = callback
        self.ttsFactory = TTSFacroty.getInstance()

    def speak(self, text):
        if rs.isMute:
            return
        ttsSource = self.ttsFactory.getTTSSource(cf.ttsSource)
        ttsSource.textToSpeech(text, self.callback, self.callback.
            responseCallback)
        ttsSource.start()

    def play(self, filename):
        if rs.isMute:
            return
        info = {}
        info['url'] = filename
        info['type'] = "tts"
        info['tts'] = "file"
        if self.callback is not None and self.callback.
            responseCallback is not None:
            self.callback.onThread(self.callback.responseCallback,
                info)
```

- **API**: Các hàm tương tác mà module Text to Speech cung cấp
  - `speak(text)`: chuyển đổi văn bản `text` thành tiếng nói.

– `play(filename)`: phát tập tin âm thanh có đường dẫn `filename`.

Speech to Text

- **Chức năng**: Module Speech to Text (STT) có chức năng chuyển dữ liệu âm thanh tiếng nói về dạng văn bản. Cụ thể, module này đảm nhiệm hai chức năng chính: chuyển mệnh lệnh của người dùng về văn bản và kiểm chứng lại keyword khi kích hoạt hệ thống.



Hình 6.6: Mô hình hoạt động của module Speech to Text

- **Cài đặt**: Module Speech to text (STT) được cài đặt theo mẫu thiết kế Singleton và Factory. Nhờ mẫu Singleton module STT có thể được sử dụng từ bất cứ module nào của hệ thống mà không làm phung phí tài nguyên khi phải tạo ra nhiều instance. Mẫu Factory cung cấp quyền lựa chọn dịch vụ STT, hiện module STT hỗ trợ hai dịch vụ đó là pocketsphinx và Google Speech to Text. Sau khi thực hiện xong công việc, đoạn văn bản thu được sẽ được chuyển cho module MainControl để tiếp tục xử lý.

  Đoạn code xử lý chính của module Speech to Text:

```
class STTFactory:
    instance = None
    def __init__(self):
        self.map = {}
        self.addSTTSource("googlestt", GoogleSTT())
        self.addSTTSource("sphinxstt", SphinxSTT())

    @staticmethod
    def getInstance():
```

```
        if STTFactory.instance is None:
            STTFactory.instance = STTFactory()
        return STTFactory.instance

    def getText(self, audio, callback, callbackFuncion):
        stt = self.getSTTSource(cf.sttSource)
        if stt is not None:
            stt.setAudio(audio, callback, callbackFuncion)
            stt.start()

    def addSTTSource(self, key, ttsObject):
        self.map[key] = ttsObject

    def getSTTSource(self, key):
        if key in self.map:
            return self.map[key].clone()
        return None
```

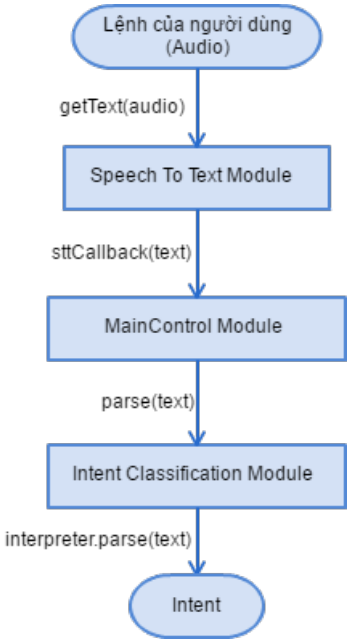
- **API**: Các hàm tương tác mà module Speech to Text cung cấp
  - `setAudio(audio)`: nhận dữ liệu audio và chuyển thành văn bản.

Intent Detector

- **Chức năng**: Module Intent Detector có chức năng xác định mệnh lệnh của người dùng.
- **Cài đặt**: Module Intent Detector sử dụng thư viện Rasa_NLU để cài đặt. Module Intent Detector sau khi nhận đoạn văn bản của lệnh người dụng từ module MainControl sẽ gọi hàm `pasrse(text)` để xử lý. Khi xác định được mệnh lệnh của người dùng, mệnh lệnh này sẽ được chuyển qua cho Module Intent Processor để xử lý.

  Đoạn code xử lý chính của module Intent Detector:

```
class IntentDetector:
    instance = None
```



Hình 6.7: Mô hình hoạt động của module Intent Detector

```
@staticmethod
def getInstance():
    if IntentDetector.instance is None:
        IntentDetector.instance = IntentDetector()
    return IntentDetector.instance

def __init__(self):
    print "Init Intent Detector"
    metadata = Metadata.load(cf.idModelDir)
    self.interpreter = Interpreter.load(metadata, RasaNLUConfig(
        cf.idConfigDir))

def parse(self, text):
    return self.interpreter.parse(text)
```

- **API**: Các hàm tương tác mà module Intent Detector cung cấp
  - `parse(text)`: xác định mệnh lệnh người dùng từ đoạn văn bản `text`.
- **Tập dữ liệu huấn luyện**: Tập dữ liệu huấn luyện cho model Rasa NLU dùng

trong hệ thống này gồm 650 mẫu dữ liệu thuộc 31 lớp intent. Có 5 lớp entity khác nhau trong tập dữ liệu.

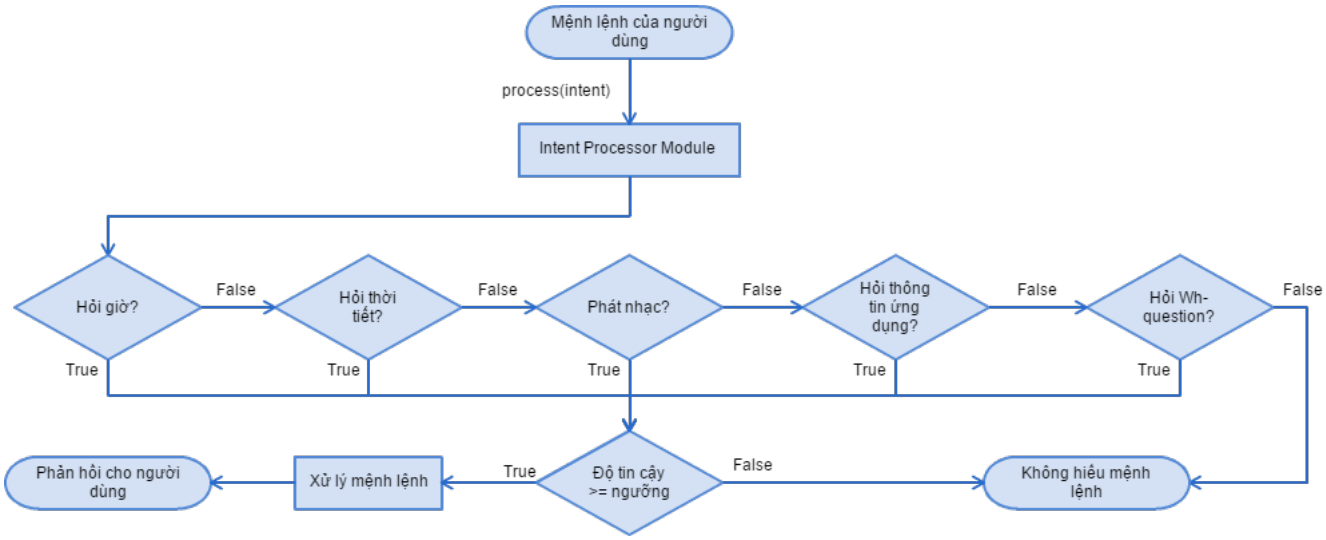
Bảng 6.1: Danh sách các intent trong tập dữ liệu huấn luyện

Nhóm intent	**Intent**	**Các entity**	**Số mẫu**
general_question	general_question	Không có	25
music	music.play	song, artist	17
	music.stop	Không có	5
	music.pause	Không có	4
	music.resume	Không có	7
	music.forward	Không có	7
	music.backward	Không có	7
smalltalk	smalltalk.agent.acquaintance	Không có	18
	smalltalk.agent.age	Không có	8
	smalltalk.agent.birth_date	Không có	7
	smalltalk.agent.boss	Không có	7
	smalltalk.agent.hobby	Không có	7
	smalltalk.agent.name	Không có	6
	smalltalk.agent.residence	Không có	23
	smalltalk.appraisal.good	Không có	119
	smalltalk.appraisal.thank_you	Không có	32
	smalltalk.appraisal.well_done	Không có	9
	smalltalk.greetings.bye	Không có	34
	smalltalk.greetings.goodevening	Không có	6
	smalltalk.greetings.goodmorning	Không có	13
	smalltalk.greetings.goodnight	Không có	19
	smalltalk.greetings.hello	Không có	18
	smalltalk.user.will_be_back	Không có	5
time	time.get	location	91
volume	volume.check	Không có	7
	volume.set	number	10
	volume.up	number	12

	volume.down	number	17
	volume.mute	Không có	9
	volume.unmute	Không có	7
weather	weather.get	Không có	94

Intent Processor

- **Chức năng**: Module Intent Processor có chức năng xử lý và trả về kết quả của mệnh lệnh từ người dùng.



Hình 6.8: Mô hình hoạt động của module Intent Processor

- **Cài đặt**: Module Intent Processor được cài đặt theo mẫu Singleton để có thể được sử dụng từ bất kỳ module nào mà không cần khởi tạo nhiều instance. Trong module Intent Processor sẽ chứa instance của nhiều skill khác nhau. Mỗi skill đảm nhận giải quyết một nhiệm vụ. Khi nhận được yêu cầu người dùng từ module Intent Detector, module Intent Processor sẽ chọn ra skill phù hợp và giao cho nó xử lý yêu cầu người dùng.

  Đoạn code xử lý chính của module Intent Processor:

```
class IntentProcessor:
    instance = None

    @staticmethod
```

```
    def getInstance(callback = None):
        if IntentProcessor.instance is None:
            IntentProcessor.instance = IntentProcessor(callback)
        return IntentProcessor.instance

    def __init__(self, callback):
        print "Init Intent Processor"
        self.tts = TextToSpeech.getInstance()

        listSkills = [TimeSkill, WeatherSkill, MusicSkill,
           GeneralQuestionSkill, VolumeSkill, SmalltalkSkill]
        self.skills = {}

        for skill in listSkills:
            self.skills[skill.intent] = skill.getInstance()

        self.callback = callback

    def process(self, info):
        try:
            print info['intent']['name'] + ": " + str(info['intent']['
                confidence'])
            key = info['intent']['name'].split('.')[0]
            if key in self.skills:
                self.skills[key].process(info, self.callback)
            else:
                print "Comming soon"
        except:
            self.tts.play("./Audio/do-not-understand-intent.mp3")

    def getSkill(self, intent):
        if intent in self.skills:
            return self.skills[intent]
        return None

    def addSkill(self, skill):
        self.skills[skill.intent] = skill.getInstance()
```

Đoạn code minh họa cách xử lý của một skill:

```
class SmalltalkSkill:
    intent = 'smalltalk'
    name = '...'
    description = "..."

    instance = None
    @staticmethod
    def getInstance():
        if SmalltalkSkill.instance is None:
            SmalltalkSkill.instance = SmalltalkSkill()
        return SmalltalkSkill.instance

    def __init__(self):
        self.tts = TextToSpeech.getInstance()
        self.responses_file = "./Response/response-rasa-smalltalk.
            json"
        self.responses = {}
        self.responseControl = ResponseControl.getInstance()

        with open(self.responses_file, 'r') as f:
            self.responses = json.load(f)

    def process(self, intent, callback):
        if intent['intent']['confidence'] < 0.1:
            self.tts.play("./Audio/do-not-understand-intent.mp3")
            return

        if intent['intent']['name'] in self.responses:
            responses_text = self.responses[intent['intent']['name']]
            text = responses_text[random.randint(0, len(responses_text
                ) - 1)]

            if intent['intent']['name'] == "smalltalk.agent.name":
```

```
            callback.wakeupModule.onThread(callback.wakeupModule.
                pause)
            rs.ttsEndCallback = callback.wakeupModule
            rs.ttsEndFunction = callback.wakeupModule.resume
        self.response(text)
    else:
        self.response("uh")

def response(self, text):
    self.responseControl.textResponse(text)
    self.tts.speak(text)
```

- **API**: Các hàm tương tác mà module Intent Processor cung cấp
  - `process(intent)`: nhận mệnh lệnh người dùng và xử lý.
  - `getSkills(intent)`: chọn skill phù hợp để xử lý `intent`.
  - `addSkills(skill)`: thêm skill vào module.

6.3 Các chức năng chính

6.3.1 Thông báo giờ

Chức năng chi tiết:

Trợ lý ảo có thể cho người dùng biết thời gian tại địa điểm hiện tại của người dùng, hoặc tại một địa điểm bất kỳ nào đó.

Cách hoạt động:

- Có thể request Google Maps Time Zone API để tìm chênh lệch giờ của một địa điểm so với múi giờ UTC. Tuy nhiên, Time Zone API chỉ nhận địa điểm bằng tọa độ chứ không nhận tên địa điểm. Do đó, hệ thống sẽ phải tìm tọa độ của địa điểm đó.

- Nếu trong câu hỏi của người dùng không có địa điểm thì hệ thống sẽ request đến IPInfoDB để tìm tọa độ gần đúng của máy.
- Nếu trong câu hỏi có tên địa điểm thì sẽ dùng Google Maps Geocode API để tìm tọa độ của địa điểm đó.
- Sau khi tìm được chênh lệch giờ so với UTC, lấy chênh lệch đó cộng với timestamp hiện tại, rồi dùng lớp `datetime` của Python để tạo câu trả lời theo format giờ (ví dụ "It's 4:15 PM").

6.3.2 Thông báo thời tiết

Chức năng chi tiết:

Trợ lý ảo có thể cho người dùng biết thời tiết tại địa điểm hiện tại của người dùng, hoặc tại một địa điểm bất kỳ nào đó.

Cách hoạt động:

- Nếu trong câu hỏi của người dùng không có địa điểm thì hệ thống sẽ request đến IPInfoDB để tìm địa điểm hiện tại của máy.
- Sau khi có tên địa điểm, request đến Apixu API để lấy thông tin thời tiết, sau đó in ra các thông tin đó theo format phù hợp.

6.3.3 Phát nhạc

Chức năng chi tiết:

Trợ lý ảo có khả năng phát nhạc theo yêu cầu của người dùng (tên bài, tên ca sĩ). Trong lúc phát nhạc có thể tạm dừng, dừng hẳn hoặc sang bài tiếp theo.

Cách hoạt động:

Sử dụng Zing MP3 API để tìm bài hát, lưu lại danh sách các bài hát tìm được và stream nhạc từ Zing MP3 về khi biết được ID bài hát.

6.3.4 Giao tiếp cơ bản

Chức năng chi tiết:

Trợ lý ảo có thể trả lời một số câu giao tiếp đơn giản như chào, chào buổi sáng, chúc ngủ ngon, hỏi tên, tuổi,...

Cách hoạt động:

Trong hệ thống sẽ có danh sách một số câu trả lời cho mỗi intent, khi câu nói của người dùng được phân lớp ra intent nào thì sẽ lấy một câu trả lời ngẫu nhiên trong danh sách câu trả lời của intent đó.

6.3.5 Trả lời câu hỏi Wh-question

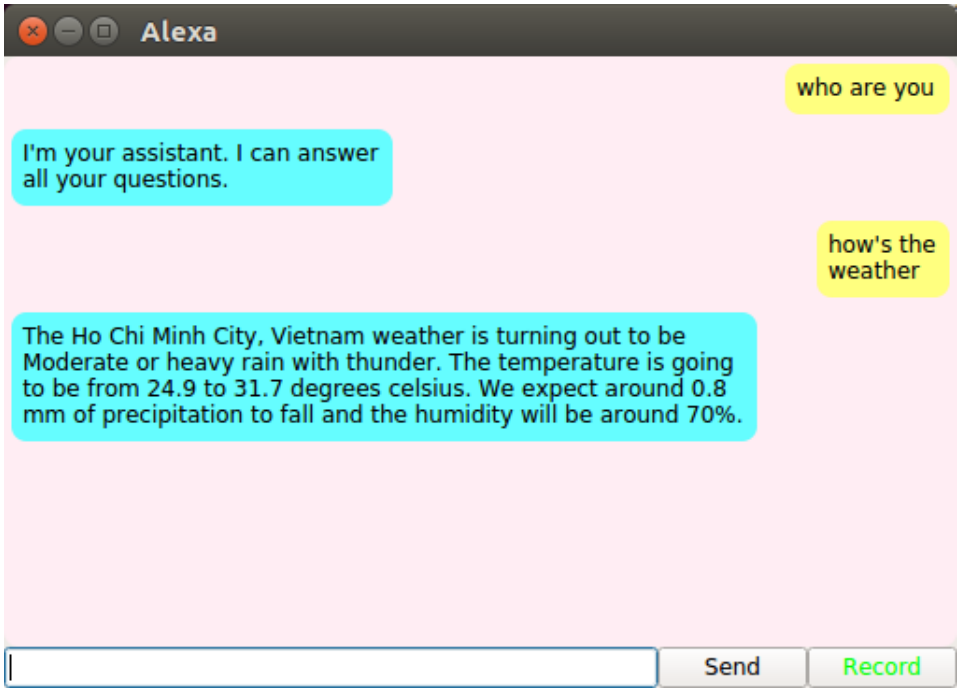
Chức năng chi tiết:

Trợ lý ảo có khả năng trả lời một số câu hỏi WH-question đơn giản (what, when, who, where, which, how).

Cách hoạt động:

Khi nhận được câu hỏi, hệ thống sẽ dùng câu hỏi đó để request đến WolframAlpha Spoken Result API để lấy câu trả lời.

6.4 Giao diện hoạt động của ứng dụng



Hình 6.9: Giao diện hoạt động của ứng dụng Alexa

Ứng dụng cho phép người dùng ra lệnh bằng một trong hai cách: gõ lệnh vào khung input hoặc ra lệnh bằng lời nói (gọi tên "Alexa" trước khi ra lệnh). Ứng dụng sẽ phản hồi cho người dùng dưới dạng lời nói và văn bản.

Chương 7

Kết Luận và Hướng Phát Triển

Nội dung chương 6 sẽ tổng hợp lại các kết quả đạt được về mặt lý thuyết và thực nghiệm của khóa luận, đồng thời nêu ra những điểm còn hạn chế và hướng phát triển của đề tài.

7.1 Kết quả đạt được

7.1.1 Về mặt lý thuyết

- Tìm hiểu được cấu trúc cơ bản của một ứng dụng trợ lý ảo.
- Tìm hiểu về âm thanh và giọng nói trên máy tính, cách sử dụng thư viện PyAudio để xử lý các tác vụ về âm thanh.
- Tìm hiểu cách sử dụng và ưu nhược điểm của các thư viện nhận biết tiếng nói là pocketsphinx và Google Speech to text.
- Tìm hiểu các API Text to speech như gTTS và iSpeech.
- Tìm hiểu về công cụ Rasa NLU dùng trong bài toán natural language understanding.
- Tìm hiểu cách sử dụng các API như Google Maps API, Apixu, Zing MP3 API, WolframAlpha API,...

7.1.2 Về mặt thực nghiệm

- Xây dựng thành công ứng dụng trợ lý ảo Alexa với các chức năng cơ bản:
  - Hỏi đáp, tra cứu thông tin
  - Hỏi giờ, thời tiết
  - Phát nhạc
  - Trò chuyện đơn giản
  - Thay đổi âm lượng của máy.
- Ứng dụng có thể chạy trên nhiều nền tảng hệ điều hành khác nhau như Windows, Mac OS, Linux.
- Ứng dụng có khả năng tự kích hoạt khi người dùng gọi keyword.
- Ứng dụng có khả năng nhận biết giọng nói đạt mức chính xác rất cao.
- Ứng dụng có giọng nói tương đối tự nhiên, trả lời chính xác các truy vấn của người dùng.

7.2 Hướng phát triển

Mặc dù đã căn bản thực hiện được những mục tiêu đã đề ra nhưng ứng dụng vẫn còn nhiều điểm hạn chế so với các trợ lý ảo khác. Do đó, một số hướng phát triển của đề tài sắp tới sẽ là:

- Xây dựng khả năng phản hồi của ứng dụng thành một cuộc "nói chuyện" thực sự, những câu sau sẽ có sự liên kết đến những câu trước.
- Tăng số lượng chức năng.
- Hỗ trợ các ngôn ngữ khác tiếng Anh.
- Thêm khả năng tương tác bằng hình ảnh.

TÀI LIỆU THAM KHẢO

[1] "Rasa nlu docs." [Online]. Available: https://rasa-nlu.readthedocs.io/

[2] F. Beaufays, "Google research blog: The neural networks behind google voice transcription," https://research.googleblog.com/2015/08/the-neural-networks-behind-google-voice.html, Aug. 2015.

[3] D. Huggins-Daines, M. Kumar, A. Chan, A. W. Black, M. Ravishankar, and A. I. Rudnicky, "Pocketsphinx: A free, real-time continuous speech recognition system for hand-held devices," in *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, vol. 1. IEEE, 2006, pp. I–I.

[4] S. R. Mache, M. R. Baheti, and C. N. Mahender, "Review on text-to-speech synthesizer," *IJARCCE*, vol. 4, 2015.

[5] A. McCarthy, "How do people use virtual assistants on their smartphones?" https://www.emarketer.com/Article/How-Do-People-Use-Virtual-Assistants-on-Their-Smartphones/1015251, Feb. 2017.

[6] E. Protalinski, "Google's speech recognition technology now has a 4.9% word error rate," https://venturebeat.com/2017/05/17/googles-speech-recognition-technology-now-has-a-4-9-word-error-rate/, May 2017.