TRƯỜNG ĐẠI HỌC THỦY LỢI

BÁO CÁO BÀI TẬP LỚN

HỌC PHẦN: Học sâu và ứng dụng

Nhận dạng ý định tiếng Việt cho trợ lý thú cưng ảo sử dụng PhoBERT kết hợp Rule-Based

Sinh viên thực hiện: *Hà Nhật Khánh Duy*

MSV: *2251262596*

Lớp**:** 64TTNT2

Giảng viên hướng dẫn: <PGS.TS. Lê Văn Hưng>

Hà Nội, tháng 12 năm 2025

MỤC LỤC

Contents

I. Giới thiệu đề tài

Sự phát triển mạnh mẽ của trí tuệ nhân tạo, đặc biệt là trong lĩnh vực học sâu (Deep learning), đã mở ra nhiều hướng tiếp cận mới trong việc xây dựng các hệ thống tương tác giữa người và máy. Trong đó trợ lý ảo (Virtual Assistant) ngày càng trở nên nổi tiếng và đóng góp vai trò quan trọng trong việc hỗ trợ người dùng sử dụng công nghệ, thao tác các thiết bị, tìm kiếm thông tin hay tự động hóa các thiết bị hàng ngày. Ta có thể nói đến các trợ lý ảo nổi tiếng như Siri của Apple và Alexa của Amazon được ứng dụng trên nhiều ngôn ngữ, nói chuyện hiểu lời người một cách rất tự nhiên và ứng dụng vào nhiều việc như nhà thông minh, tìm nhạc, ghi chú, gọi điện hay nhắn tin. Lấy ý tưởng từ đó em muốn làm một trợ lý ảo sử dụng trên máy tính, nó sẽ giúp người sử dụng mới sử dụng máy tính hay những người muốn tối ưu hóa thời gian chỉ cần nói để thực hiện các tác vụ cơ bản như Siri và Alexa làm với các thiết bị của họ. Không chỉ thế đối với một người yêu động vật nhưng không thể nuôi được do vấn đề cá nhân, em muốn nhân hóa trợ lý ảo như một thú cưng trên máy tính vừa hỗ trợ vừa chơi đùa như thú cưng thật. Đây là một dự án sử dụng cho đồ án sẽ sử dụng nhiều môn như xử lý âm thanh và tiếng nói với Whisper cho ra các văn bản lệnh, học sâu sẽ đóng vai trò chủ chốt để xây dựng hệ thống giúp trợ lý ảo có thể nghe-hiểu-hành động chính xác bằng Tiếng Việt. Để giúp trợ lý ảo hiểu sâu về các lệnh đưa ra, trong báo cáo này em sẽ tập chung phần học sâu sử dụng mô hình PhoBert + Rule chú trọng vào ngôn ngữ Tiếng Việt với các câu lệnh ngắn, không chuẩn ngữ pháp hoặc mang tính khẩu ngữ. Xây dựng hệ thống phân loại ý định Intern kết hợp học sâu và luật thủ công (Rule – based) nâng cao độ chính xác, tập chung vào việc hiểu và thực hiện các lệnh điều khiển máy tính bằng tiếng Việt, đồng thời mang lại trải ngiệm vui vẻ, gần gũi.

II. Cơ sở lý thuyết

2.1. Giới thiệu học sâu

Học sâu (Deep Learning) là một nhánh của học máy (Machine Learning) dựa trên các mạng nơ-ron nhân tạo nhiều tầng (deep neural networks). Trong lĩnh vực xử lý ngôn ngữ tự nhiên (NLP), học sâu đã thay thế gần như hoàn toàn các phương pháp truyền thống dựa trên đặc trưng thủ công và mô hình thống kê nông nhờ khả năng học biểu diễn ngữ nghĩa tự động từ dữ liệu lớn.

Các mô hình học sâu nổi bật như Mạng nơ-ron hồi quy (RNN), Convolution Neural Networks(CNN), Transformer đặc nền tảng cho mô hình NLP hiện đại như BERT, trong bài này là PhoBert.

2.2. Bài toán Intent Clasification

Nhận dạng ý định (Intent Clasification) là một bài toán quan trọng trong các hệ thống trợ lý ảo và chatbot. Mục tiêu của bài toán là xác định mục đích của người dùng thông qua câu lệnh văn bản đầu vào, từ đó hệ thống có thể thực hiện các hành động phù hợp. Đối với trợ lý ảo trên máy tính, việc nhận dạng chính

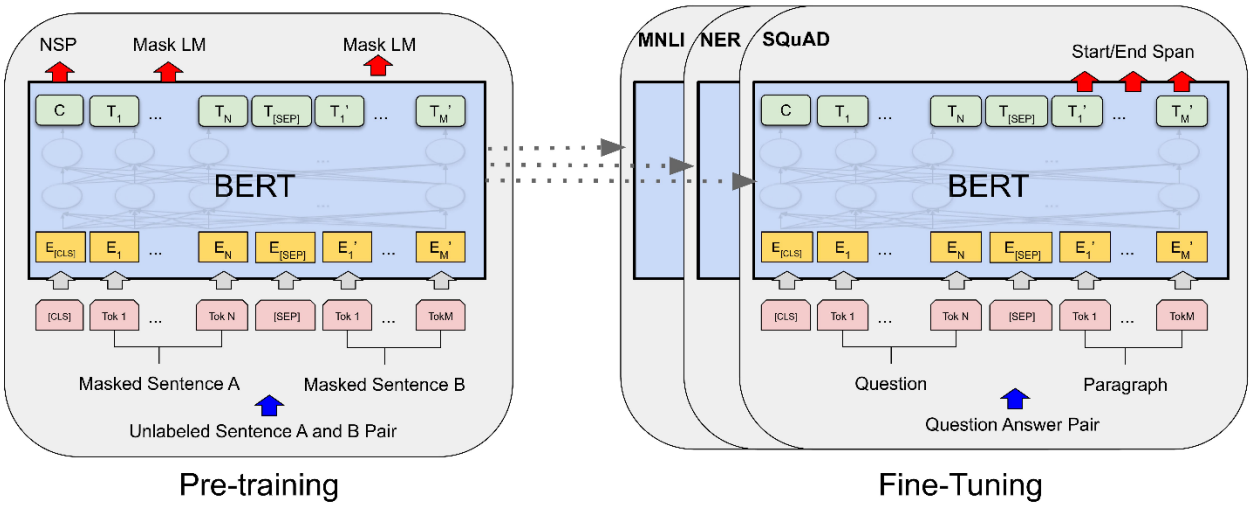
xác các ý định giúp hệ thống hiểu đúng yêu cầu của người dùng và đảm bảo thao tác điều khiển được thực hiện chính xác.

2.3. Mô hình BERT

2.3.1. Giới thiệu mô hình BERT

Mô hình BERT (Bidirectional Encoder Representations from Transformers) là một mô hình ngôn ngữ mạnh mẽ được phát triển bởi Google dựa trên kiến trúc Transformer. BERT đã tạo ra một bước đột phá lớn trong lĩnh vực xử lý ngôn ngữ tự nhiên nhờ khả năng hiểu sâu ngữ cảnh và ý nghĩa văn bản. Khác với mô hình ngữ cảnh truyền thống chỉ đọc câu theo một chiều (trái sang phải hay phải sang trái), BERT có khả năng xem xét đồng thời cả ngữ cảnh phía trước và sau của mỗi từ đó hiểu ngữ nghĩa chính xác hơn.

2.3.2. Kiến trúc BERT



Mô hình BERT bao gồm:

- Embedding layer: gồm token embedding, segment embedding và position embedding. Giúp mô hình phân biệt được vị trí từ trong câu cũng như ranh giới giữa các câu khác nhau.
- N lớp Transformer Encoder: mỗi lớp gồm Multi-Head Self-Attention và Feed Forward Network là cơ chế tự chú ý đa đầu.
- Output layer: được fine-tune tùy theo từng bài toán cụ thể.

Mỗi câu đầu vào được thêm token đặc biệt:

[CLS]: được thêm vào đầu câu, đại diện cho toàn bộ câu, dùng cho các bài toán phân loại.

[SEP]: phân tách giữa các câu hoặc đánh dấu kết thúc câu.

Thông qua cơ chế self-attention, mỗi từ trong câu có thể chú ý đến tất cả các từ còn lại, giúp mô hình học được mối quan hệ phức tạp trong văn bản.

2.3.3. Quá trình huấn luyện mô hình BERT

2.3.3.1. Tiền huấn luyện (Pre-trainning)

Trong giai đoạn huấn luyện trước (pre-training), BERT được huấn luyện trên tập dữ liệu văn bản lớn không gán nhãn thông qua hai nhiệm vụ chính:

- Masked Language Modeling (MLM)

Một số từ trong câu được che đi (mask) và nhiệm vụ của mô hình là dự đoán từ bị che dựa trên ngữ cảnh xung quanh.

Ví dụ: Mở hộ tôi [MASK] chrome với -> Dự đoán từ: trình duyệt.

Nhiệm vụ này sẽ giúp BERT học được ngữ cảnh hai chiều.

- Next Sentence Prediction (NSP)

Mô hình được cung cấp hai câu A và B, nhiệm vụ là xác định xem câu B có phải là câu tiếp theo của câu A trong văn bản gốc hay không. Nhiệm vụ giúp BERT học mối quan hệ giữa các câu, rất quan trọng cho các bài toán hiểu ngữ cảnh đoạn văn.

2.3.3.2. Giai đoạn tinh chỉnh (Fine-tuning)

Sau khi pre-training, BERT được fine-tune cho các bài toán cụ thể như: phân loại văn bản, hỏi đáp, nhận dạng thực thể. Trong giai đoạn fine-tuning, toàn bộ trọng số của Bert được cập nhật dựa trên dữ liệu có nhãn của bài toán mục tiêu.

Trong bài toán phân loại, vector biểu diễn của token [CLS] ở tầng cuối cùng của BERT được sử dụng làm đặc trưng đại diện cho toàn bộ câu. Vector này sau đó được đưa qua một hoặc nhiều tầng fully connected để dự đoán nhãn đầu ra tương ứng.

Quá trình fine-tuning cho phép mô hình thích nghi với dữ liệu và nhiệm vụ cụ thể, đồng thời tận dụng được tri thức ngôn ngữ đã học trong giai đoạn tiền huấn luyện.

2.3.4. Hạn chế đối với Tiếng Việt

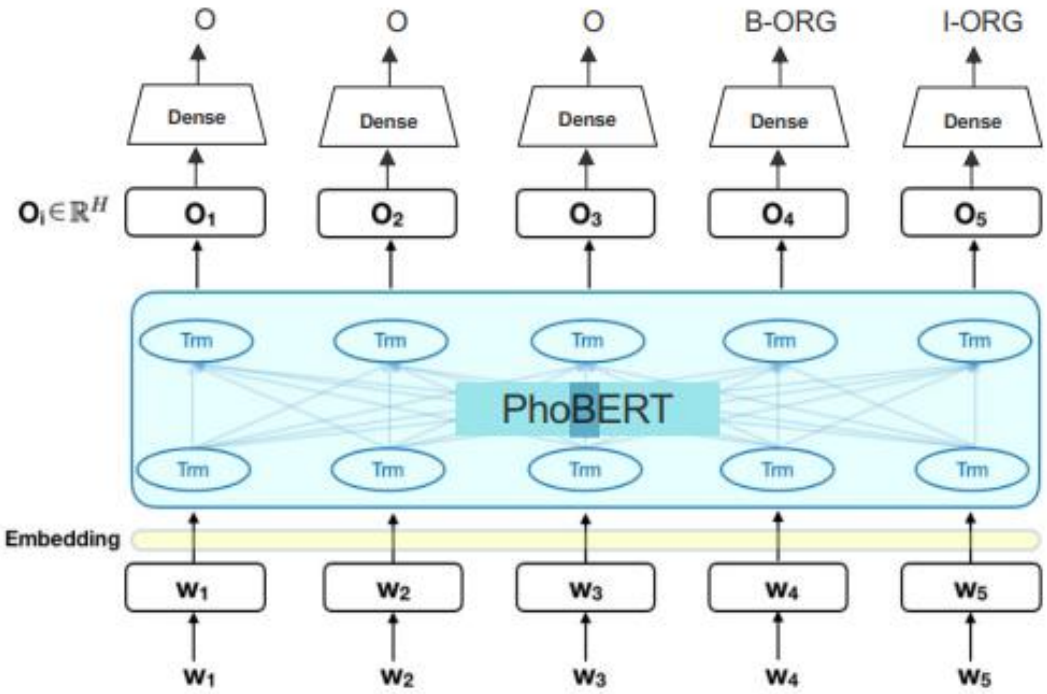
Mặc dù BERT cho thấy hiệu quả cao trong nhiều bài toán xử lý ngôn ngữ tự nhiên mô hình BERT gốc chủ yếu được huấn luyện trên dữ liệu tiếng Anh và một số ngôn ngữ phổ biến khác. Do đó, khi áp dụng trực tiếp tiếng Việt, mô hình có thể gặp hạn chế trong việc biểu diễn ngữ nghĩa,đặc biệt là với các đặc trưng ngôn ngữ như từ ghép, dấu câu và cách tách từ. Bên cạnh đó, tiếng Việt là ngôn ngữ đơn lập, có cấu trúc khác biệt so với tiếng Anh dẫn đến việc BERT gốc không khai thác tối ưu các đặc trưng ngữ nghĩa của tiếng Việt. Vì vậy, việc sử dụng một mô hình ngôn ngữ được thiết kế và huấn luyện riêng cho tiếng Việt là cần thiết nhằm nâng cao hiệu quả xử lý.

2.4. Mô hình PhoBERT

2.4.1. Giới thiệu mô hình PhoBERT

PhoBERT là mô hình ngôn ngữ tiền huấn luyện cho tiếng Việt, được phát triển bởi VinAI Research. PhoBERT kế thừa kiến trúc BERT nhưng được huấn luyện trên tập dữ liệu tiếng Việt lớn với cách tiền xử lý phù hợp với đặc trưng ngôn ngữ tiếng Việt. Được thiết kế nhằm khắc phục những hạn chế của BERT gốc khi áp dụng cho tiếng Việt.

2.4.2. Kiến trúc mô hình PhoBERT



Hình 2.1: Kiến trúc PhoBERT

PhoBERT sử dụng kiến trúc Transformer Encoder tương tự BERT, gồm:

- PhoBERT-base: 12 lớp Transformer.
- PhoBERT-large: 24 lớp Transformer.

Về mặt kiến trúc, PhoBERT kế thừa hoàn toàn cấu trúc Transformer Encoder của BERT. Mô hình bao gồm nhiều tầng encoder xếp chồng, sử dụng cơ chế self-attention để học biểu diễn ngữ cảnh hai chiều cho từng câu. PhoBERT sử dụng các token đặc biệt như [CLS] và [SEP] tương tự BERT. Trong đó, vector biểu diễn của token [CLS] ở tầng cuối được sử dụng cho các tác vụ phân loại văn bản.

2.4.3. Ứng dụng PhoBERT trong bài toán nhận dạng ý định

Trong đề tài này, mô hình PhoBERT được sử dụng để giải quyết bài toán nhận dạng ý định từ văn bản lệnh tiếng Việt cho trợ lý ảo trên máy tính máy tính. Văn bản sau khi được tiền xử lý sẽ được đưa vào mô hình PhoBERT để trích xuất đặc trưng ngữ nghĩa

Vector biểu diễn của token [CLS] được sử dụng làm đầu vào cho tầng phân loại nhằm dự đoán nhãn ý định tương ứng với câu lệnh. Kết quả dự đoán này sau đó được sử dụng để xác định hành động mà trợ lý ảo cần thực hiện. Việc này sẽ giúp hệ thống hiểu tốt hơn các câu lệnh tiếng Việt đa dạng về cách diễn đạt và từ đó nâng cao độ chính xác và tính ổn định của trợ lý ảo.

Trên cơ sở mô hình PhoBERT, hệ thống được xây dựng để nhận dạng ý định từ văn bản lệnh của người dùng, từ đó kết hợp với các luật (rule-based) nhằm thực hiện các hành động tương ứng cho trợ lý ảo trên máy tính.

2.5. Tiền xử lý dữ liệu văn bản

2.5.1. Vai trò của tiền xử lý trong bài toán nhận dạng ý định

Tiền xử lý dữ liệu là một bước quan trọng trong nhiều bài toán, cả trong xử lý ngôn ngữ tự nhiên. Ở đây dữ liệu đầu vào là các câu lệnh ngắn để biểu thị lệnh cho trợ lý ảo thực hiện, các câu lệnh sẽ rất đa dạng và việc tiền xử lý giúp làm sạch dữ liệu, giảm nhiễu đảm bảo đầu vào phù hợp với mô hình học sâu.

Trong bài toán nhận dạng ý định này, dữ liệu thường sẽ chứa các biến thể ngôn ngữ như từ đồng nghĩa, cách viết không thống nhất hoặc dấu câu không cần thiết. Nếu không được tiền xử lý phù hợp, mô hình có thể học các đặc trưng không quan trọng làm giảm hiệu quả dự đoán.

2.5.2. Các bước tiền xử lý văn bản

Để có thể đưa dataset dữ liệu ở đây là các câu lệnh vào huấn luyện, ta sẽ xử lý qua các văn bản tránh các lỗi không đáng có. Quy trình tiền xử lý dữ liệu văn bản bao gồm các bước chính sau:

- Chuẩn hóa chữ viết: chuyển toàn bộ văn bản về chữ thường nhằm giảm sự phân biệt không cần thiết giữa các từ.

- Loại bỏ ký tự đặc biệt: loại bỏ các ký tự không mang ý nghĩa như dấu câu dư thừa.

- Chuẩn hóa khoảng trắng: loại bỏ khoảng trắng thừa trong câu lệnh.

- Tokenization: văn bản được tách thành các token phù hợp với tokenizer của PhoBERT.

PhoBERT sử dụng Tokenizer được huấn luyện riêng cho tiếng Việt, do đó ta sẽ không cần phải tách từ một cách thủ công như những mô hình truyền thống nữa.

2.5.3. Chuẩn hóa dữ liệu nhãn

Mỗi câu lệnh trong tập dữ liệu sẽ được gán một nhãn ý định (intent). Các nhãn này được chuẩn hóa theo dạng định danh thống nhất (ví dụ: open_browers,

volum_up, pet_sleep) nhằm đảm bảo tính nhất quán trong quá trình huấn luyện và triển khai hệ thống.

2.6. Phương pháp Rule-based

2.6.1. Giới thiệu phương pháp Rule-based

Phương pháp Rule-based là phương pháp xây dựng hệ thống bằng cách định nghĩa trước các luật (rules) dưới dạng điều kiện – hành động (if-then). Mỗi luật được thiết kế để xử lý một trường hợp cụ thể dựa trên tri thức của con người. Trong lĩnh vực xử lý ngôn ngữ tự nhiên, Rule-based thường được sử dụng để ánh xạ kết quả phân loại sang hành động cụ thể, xử lý các trường hợp đặc biệt, giảm lỗi của mô hình học máy trong những tình huống dễ dự đoán.

Ưu điểm của phương pháp này là dễ hiểu, dễ triển khai hoạt động ổn định không phụ thuộc dữ liệu huấn luyện và phù hợp với các hệ thống điều khiển và trợ lý ảo. Nhưng vẫn có những hạn chế như khó mở rộng khi số lượng luật lớn, không linh hoạt với các câu lệnh mới và phụ thuộc vào thiết kế thủ công. Do đó Rule-based thường không được sử dụng độc lập mà được kết hợp với các mô hình học sâu, giúp giảm sự phụ thuộc hoàn toàn vào mô hình học sâu.

2.6.2. Vai trò của Rule-based trong hệ thống trợ lý ảo

Phương pháp Rule-based được sử dụng với các vai trò chính sau:

- Thực thi hành động dựa trên kết quả nhận dạng ý định.

- Xử lý các trường hợp đặc biệt hoặc các lệnh có độ chắc chắn cao.

- Đảm bảo an toàn khi thực hiện các hành động trên hệ thống máy tính.

Ví dụ khi mô hình dự đoán ý định là open_browser, hệ thống sẽ áp dụng luật tương ứng để gọi lệnh mở trình duyệt.

2.7. Lớp ánh xạ hành động Mapping

2.7.1. Giới thiệu lớp ánh xạ hành động Mapping

Lớp ánh xạ hành động (mapping layer) là một thành phần trung gian trong hệ thống xử lý ngôn ngữ, có nhiệm vụ chuyển đổi kết quả dự đoán từ mô hình học máy sang các mã hành động cụ thể của hệ thống. Trong bài toán trợ lý ảo, mô hình học sâu thường chỉ đảm nhiệm việc nhận dạng ý định (intent) từ câu lệnh của người dùng. Tuy nhiên, các nhãn ý định này không thể được sử dụng trực

tiếp để thực thi hành động trên hệ thống. Do đó, cần có một lớp ánh xạ để liên kết nhãn ý định với các hành động tương ứng.

2.7.2. Vai trò của Mapping trong hệ thống trợ lý ảo

Lớp ánh xạ hành động đóng vai trò quan trọng trong hệ thống trợ lý ảo với các chức năng chính:

- Chuyển đổi nhãn ý định sang hành động tương ứng.
- Giảm sự phụ thuộc giữa mô hình học sâu và logic thực thi.
- Dễ dàng mở rộng thêm hành động mới hay chỉnh sửa mà không cần huấn luyện lại mô hình.

Việc sử dụng mapping giúp hệ thống trở nên linh hoạt và có tính kỹ thuật cao hơn so với việc gọi hành động trực tiếp từ mô hình học sâu.

Ví dụ: pet_sleep (intent) | PET_MINIMIZE (Action). Nhãn ý định được ánh xạ sang hành động tương ứng thu nhỏ nhân vật trợ lý xuống góc màn hình.

2.8. Phương pháp kết hợp PhoBERT, Mapping và Rule-based

2.8.1. Tổng quan phương pháp kết hợp

Để đạt được độ chính xác cao, ổn định và khả năng triển khai thực tế trong hệ thống trợ lý ảo, áp dụng phương pháp kết hợp ba thành phần chính: mô hình học sâu PhoBERT, lớp ánh xạ hành động Mapping và phương pháp dựa trên luật (Rule-based). Phương pháp này sẽ tận dụng ưu điểm của từng thành phần. Chúng ta sẽ kết hợp theo thứ tự ưu tiên Rule-based =>PhoBERT =>Mapping giúp giảm thiểu sai sót của mô hình học sâu trong các tình huống thực tế, đồng thời giữ được tính linh hoạt khi gặp câu lệnh mới.

2.8.2. Quy trình xử lý tổng thể của hệ thống

1.Nhận đầu vào: câu lệnh văn bản hoặc giọng nói từ người dùng.

2.Tiền xử lý: chuẩn hóa văn bản (chữ thường, loại bỏ ký tự đặc biệt, chuẩn hóa khoảng trắng).

3.Rule-based kiểm tra ưu tiên: nếu câu lệnh khớp với một hoặc nhiều luật được định nghĩa thì trả về intent tương ứng ngay lập tức.

4.PhoBERT dự đoán: nếu rule không khớp đưa văn bản vào mô hình PhoBERT fine-tune, lấy intent có xác suất cao nhất từ vector [CLS].

5.Mapping intent sang hành động: sử dụng bảng ánh xạ để chuyển intent thành mã hành động cụ thể, đồng thời cập nhật trạng thái giao diện thú cưng.

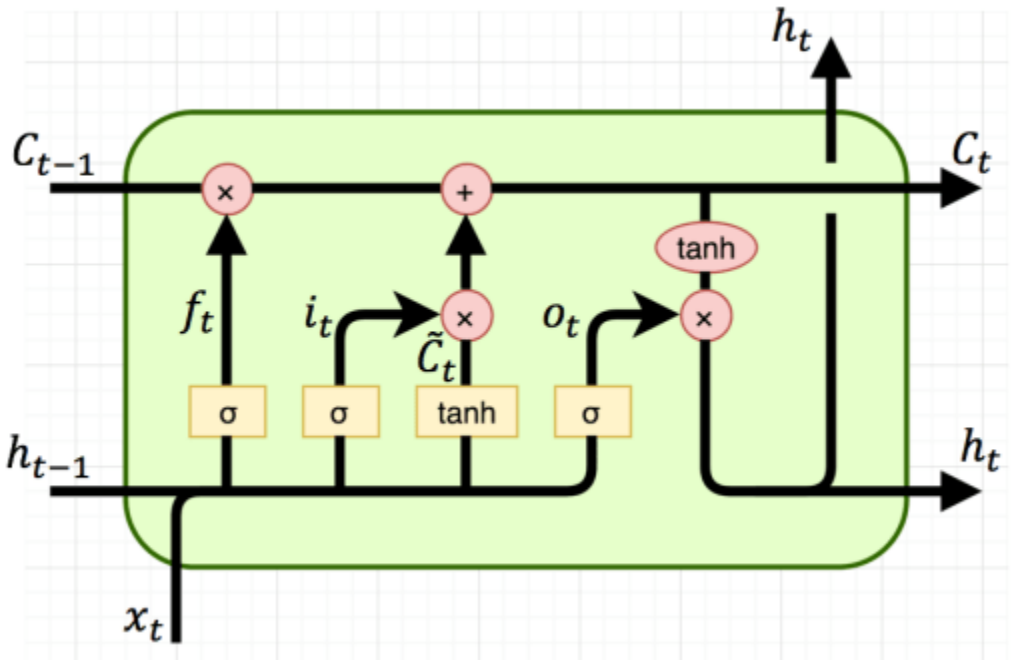
6.Thực thi hành động: gọi hàm tương ứng để thực thi, hiển thị cho người dùng qua text và giao diện pixel art.

2.8.3. Ưu điểm, hạn chế và phương pháp kết hợp

Rule-based sẽ xử lý hoàn hảo các trường hợp phổ biến, PhoBERT bù đắp cho các câu lệnh phức tạp, sẽ giúp tổng thể đạt gần 100% trên tập test. Tốc độ nhanh hơn khi Rule-based ưu tiên giảm số lần gọi vào PhoBERT quá mức nếu các câu lệnh đơn giản không quá dài. Nó cũng ổn định và an toàn tránh thực hiện các hành động sai trong trường hợp mô hình dự đoán nhầm như open/close browser. Không chỉ vậy còn dễ mở rộng khi cần thêm intent thì bổ sung rule hoặc train lại dữ liệu, thêm hành động thì chỉ cần sửa mapping thôi. Cuối cùng giảm overfit tốt nhờ Rule-based.

Nhưng vẫn còn nhiều hạn chế như phụ thuộc vào thiết kế rule viết thủ công, khó bao quát mọi cách diễn đạt. Độ trễ PhoBERT cao nếu dữ liệu tăng hay phát triển lên, có thể khiến ứng dụng chạy chậm lâu.

2.9. Mô hình truyền thống BiLSTM + Embedding



Hình 2.1: Mô hình LSTM

f_t, i_t, o_t tương ứng với **f**orget gate, **i**nput gate và **o**utput gate.

- Forget gate: $f_t = \sigma(U_f * x_t + W_f * h_{t-1} + b_f)$
- Input gate: $i_t = \sigma(U_i * x_t + W_i * h_{t-1} + b_i)$
- Output gate: $o_t = \sigma(U_o * x_t + W_o * h_{t-1} + b_o)$

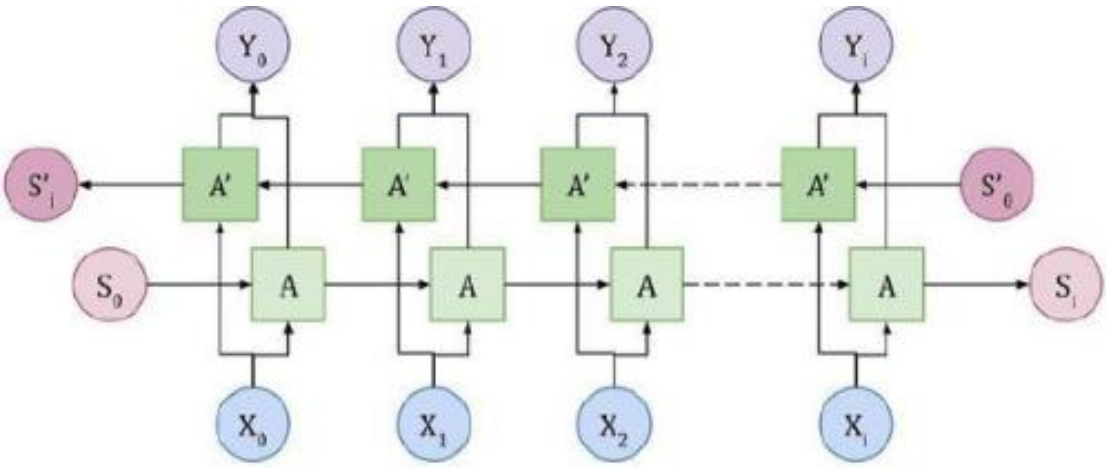
2.9.1. Giới thiệu BiLSTM

Bi-directional Long Short-Term Memory (BiLSTM) là một biến thể của mạng nơ-ron hồi quy (Recurrent Neural Network - RNN) được đề xuất bởi Hochreiter và Schmidhuber (1997) với cải tiến của Graves et al. (2005) cho hướng hai chiều. BiLSTM được sử dụng rộng rãi trong các bài toán xử lý chuỗi, đặc biệt là xử lý ngôn ngữ tự nhiên trước khi các mô hình Transformer trở nên thống trị.

BiLSTM giải quyết hai hạn chế lớn của RNN truyền thống là vanishing/exploding gradient - lstm sử dụng các cổng (forget gate, input gate, output gate) để kiểm soát dòng thông tin, giúp học được phụ thuộc dài hạn. Chỉ xem ngữ cảnh một chiều – BiLSTM xử lý chuỗi theo cả hai hướng từ trái sang phải và từ phải sang trái sau đó kết hợp vector ẩn ở mỗi thời điểm để tạo biểu diễn ngữ cảnh hai chiều.

Trong đó bài toán phân loại văn bản hoặc nhận dạng ý định, BiLSTM thường được kết hợp với lớp embedding để chuyển đổi từ thành vector số trước khi đưa vào mạng.

2.9.2. Kiến trúc BiLSTM



Hình 2.2: Kiến trúc BiLSTM

Trong đó:

- X_i là các đầu vào của mạng
- Y_i là các đầu ra của mạng
- A là lớp LSTM thuận và A' là lớp LSTM nghịch của mạng
- Y_i đầu ra cuối cùng là sự kết hợp giữa các nút A và A'

Kiến trúc cốt lõi của BiLSTM bao gồm hai lớp LSTM độc lập hoạt động song song trên cùng một chuỗi đầu vào. Lớp thứ nhất, LSTM thuận (forward LSTM), xử lý chuỗi theo thứ tự thông thường từ đầu đến cuối (từ t=1 đến T). Tại mỗi bước thời gian t, nó tạo ra một vector trạng thái ẩn tóm tắt thông tin quá khứ. Lớp thứ hai, LSTM nghịch (backward LSTM), xử lý chuỗi theo thứ tự ngược lại (từ t = T đến 1), tạo ra vector trạng thái ẩn tóm tắt thông tin tương lai (tính từ cuối chuỗi ngược về) đến thời điểm t. Đầu ra cuối cùng tại mỗi bước thời gian t, yt, thường được hình thành bằng cách kết hợp 2 vector trạng thái này, phổ biến nhất là thông qua phép ghép nối. Kết quả là một biểu diễn tại t, chứa đựng thông tin ngữ cảnh phong phú từ cả hai chiều của chuỗi được tính như sau:

$$y_t = y_{t\_f} + y_{t\_b}$$

Trong đó:

- y_t: vector đầu ra của mạng tại thời điểm t;
- $y_{t,f}$: vector trạng thái từ lớp thuận của mạng;
- $y_{t,b}$: vector trạng thái từ lớp nghịch của mạng.

Trong đề tài, vector ẩn cuối cùng của BiLSTM kết hợp hai chiều được đưa qua một lớp fully connected để dự đoán lớp intent, kiến trúc cụ thể:

- Lớp embedding: chuyển mỗi token thành vector 300 chiều (random iniotialized).

- Lớp BiLSTM: một lớp LSTM hai chiều với hidden size 256 (tổng 512 chiều khi kết hợp).

- Dropout: 0.5 để giảm overfit.

- Lớp Linear: đầu ra 512 chiều => số lớp intent (9 lớp).

Nhờ cơ chế này, BiLSTM có khả năng nắm bắt phụ thuộc ngữ nghĩa từ cả ngữ cảnh trước và sau từ hiện tại, rất hiệu quả cho các bài toán phân loại chuỗi ngắn như nhận dạng ý định câu lệnh.

2.10. Pygame cho giao diện pixel art

2.10.1. Giới thiệu về Pygame

Pygame là một thư viện mã nguồn mở viết bằng ngôn ngữ Python, được thiết kế chuyên biệt để phát triển trò chơi 2D và ứng dụng đa phương tiện. Thư viện được xây dựng dựa trên SDL (Simple DirectMedia Layer) – một thư viện thấp cấp cho phép truy cập trực tiếp vào âm thanh, bàn phím, chuột, joystick và card đồ họa.

Pygame cung cấp các chức năng cơ bản cần thiết để xây dựng giao diện đồ họa:

- Tạo cửa sổ hiển thị.
- Vẽ hình ảnh, hình chữ nhật, đường thẳng, văn bản.
- Xử lý sự kiện người dùng (chuột, bàn phím).
- Hỗ trợ animation thông qua vòng lặp game loop (phần cần trong đề tài).
- Quản lý thời gian (frame rate).

Với đặc thù nhẹ, dễ sử dụng và hoàn toàn miễn phí, Pygame là lựa chọn phù hợp cho các ứng dụng desktop nhỏ, trò chơi 2D đơn giản và đặc biệt là các dự án học thuật, đồ án sinh viên.

2.10.2. Pixel art và ứng dụng trong đề tài

Pixel art là phong cách thiết kế đồ họa sử dụng các pixel (điểm ảnh) có kích thước lớn để tạo hình ảnh, thường mang tính hoài cổ và dễ thương. Phù hợp với nhân vật "thú cưng ảo" nhờ tính đơn giản, dễ nhận diện và khả năng biểu đạt cảm xúc qua các frame animation nhỏ.

Trong đề tài này, quyết định tạo nhân vật pixel thú cưng trợ lý ảo hiển thị liên tục trên màn hình desktop, thể hiện các trạng thái cảm xúc và hành động khác nhau. Tăng tính tương tác và trải nghiệm người dùng, biến trợ lý ảo thành một bạn đồng hành dễ thương thay vì công cụ khô khan.

2.11. Các chỉ số đánh giá

2.11.1. Độ chính xác (Accuracy)

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

Tỷ lệ dữ đoán đúng trên tổng số mẫu trong tập kiểm tra (Số mẫu dự đoán đúng/tổng số mẫu).

2.11.2. Precision, Recall, F1-score, AVG time inference

Do bài toán có nhiều lớp intent, các chỉ số được tính theo hai cách:

- Macro average: tính trung bình không trọng số của chỉ số trên từng lớp => đánh giá hiệu suất đồng đều giữa các lớp.

- Weighted average: tính trung bình có trọng số theo số lượng mẫu của từng lớp => phản ánh hiệu suất trên toàn bộ dữ liệu.

Độ chính xác dự đoán (Precision):

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Độ bao phủ (Recall):

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

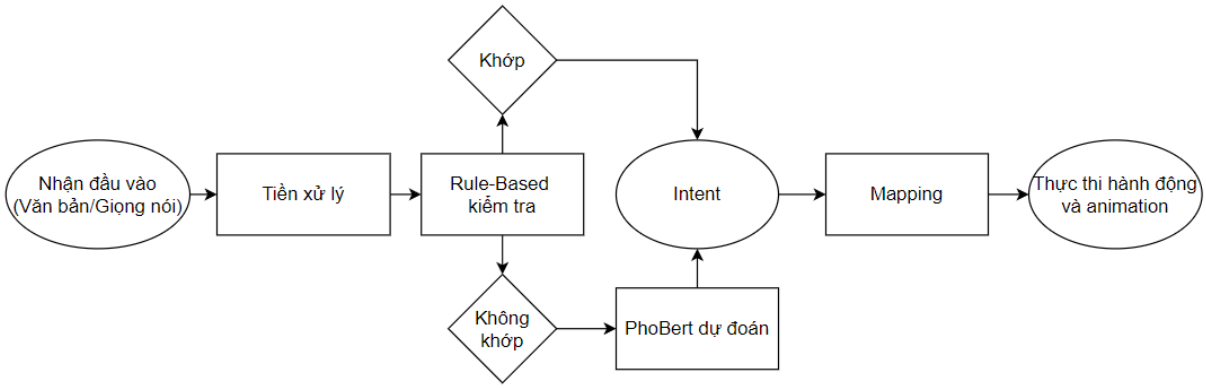
Điểm F1 (F1-Score):

$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Thời gian inference trung bình: thời gian inference là thời gian trung bình để mô hình dự đoán intent cho một câu lệnh, đo bằng cách tính thời gian xử lý một batch và quy đổi về mỗi câu.

III. Phương pháp thực hiện

3.1. Tổng quan qua kiến trúc hệ thống



Hình 3.1: Cấu trúc hoạt động hệ thống

Module nhận lệnh: hiện tại nhận văn bản qua giao diện console, có thể mở rộng tích hợp STT chuyển giọng nói thành văn bản lệnh trong những bài sau.

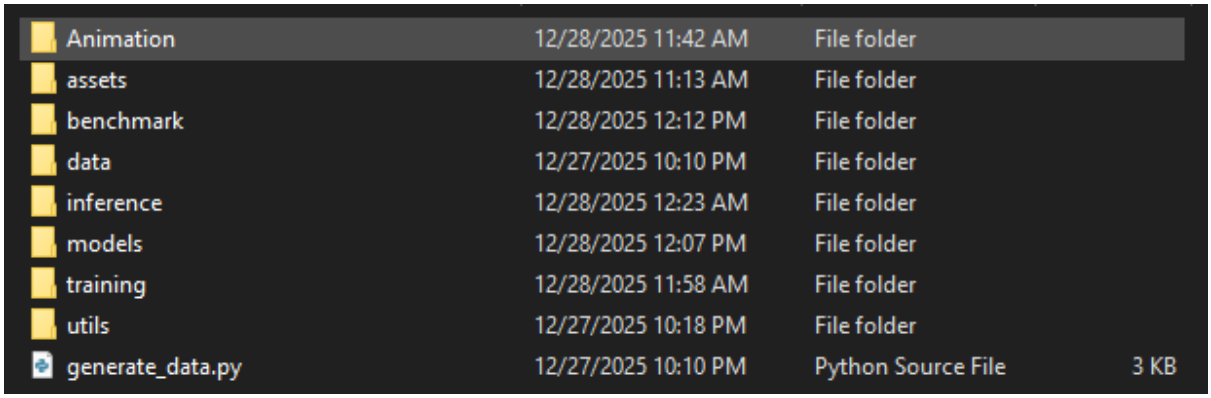
Module tiền xử lý văn bản: chuẩn hóa văn bản tiếng Việt, xử lý qua trước khi đưa vào hệ thống tránh rủi ro.

Module nhận dạng ý định: sử dụng Rule-based kiểm tra, nếu không khớp chuyển sang sử dụng mô hình học sâu.

Module ánh xạ hành động (mapping): chuyển intent thành hành động cụ thể và trạng thái animation của thú cưng.

Module thực thi hành động: thực hiện lệnh trên hệ thống (mở ứng dụng, điều chỉnh âm lượng, …)

Module giao diện: hiển thị nhân vật pixel với animation chuyển động với Pygame.



Hình 3.2: Cấu trúc thư mục

3.2. Thu thập và sinh dữ liệu

Do chưa có bộ dữ liệu lệnh điều khiển máy tính tiếng Việt nào công khai phù hợp với ước muốn và ý định sử dụng của bản thân, quyết định sử dụng code sinh dữ liệu tự động với đa dạng cách nói và cân bằng. Số lượng dữ liệu là 900 câu lệnh (9 intent với 100 câu/intent). Các intent open/close_browser, open_youtube, greet_pet, praise_pet, volume_up/down, pet_sleep, pet_minimize. Tạo cơ bản với các intent có thể sử dụng hàng ngày, có thể phát triển thêm các ứng dụng mới trong tương lai. Phương pháp sinh sẽ bao gồm động từ + đối tượng + từ bổ nghĩa + chủ ngữ + thời gian. (generate_data.py)

Kết quả cho ra là một bộ dữ liệu sạch, bao quát nhiều cách diễn đạt khẩu ngữ tiếng Việt, phù hợp để huấn luyện mô hình tuy nhiên vẫn chưa đầy đủ cách diễn đạt do tiếng Việt rất phong phú vẫn sẽ phải cập nhật thêm.

3.3. Tiền xử lý văn bản

Văn bản được tiền xử lý qua hàm normalize_text trong utils/preprocess.py:

- Chuyển về chữ thường.
- Loại bỏ ký tự đặc biệt thừa.
- Chuẩn hóa khoảng trắng.
- Xử lý tiếng Việt không dấu và viết tắt phổ biến.

```python
def normalize_text(text: str) -> str:
    if not isinstance(text, str):
        return ""
    # Chuẩn hóa unicode (rất quan trọng cho tiếng Việt)
    text = unicodedata.normalize("NFC", text)
    # Chuyển về chữ thường
    text = text.lower()
    # Loại bỏ ký tự đặc biệt, giữ chữ cái, số, khoảng trắng
    text = re.sub(r"[^\w\s]", " ", text)
    # Chuẩn hóa khoảng trắng
    text = re.sub(r"\s+", " ", text).strip()
    return text
```

Bước này giúp giảm nhiễu và tăng tính nhất quán cho cả Rule-based và mô hình học sâu PhoBERT. Đây sẽ là bước kiểm tra lại sau khi ngẫu nhiên dữ liệu và nếu sau này mở rộng ra, sẽ tránh được nhiều lỗi có thể xảy ra.

3.4. Nhận dạng ý định

3.4.1. Phương pháp Rule-based

Triển khai trong file inference/rule_engine.py, ta ưu tiên kiểm tra để xử lý nhanh các lệnh phổ biến, kiểm tra từ khóa động từ + đối tượng và có hỗ trợ không dấu.

```python
# 1. MỞ TRÌNH DUYỆT / YOUTUBE (ưu tiên cao nhất vì hay bị nhầm)
open_verbs = ["mở", "mo", "bật", "bat", "vào", "vao", "khởi động", "khoi dong", "cho xem", "xem"]
if any(verb in t for verb in open_verbs):
    if any(obj in t for obj in ["chrome", "trình duyệt", "trinh duyet", "browser", "web", "cốc cốc", "coc coc", "google"]):
        return "open_browser"
    if any(obj in t for obj in ["youtube", "yt", "you tube"]):
        return "open_youtube"
# 2. ĐÓNG TRÌNH DUYỆT
close_verbs = ["đóng", "dong", "tắt", "tat", "thoát", "thoat", "close", "đóng lại", "dong lai"]
if any(verb in t for verb in close_verbs):
    if any(obj in t for obj in ["chrome", "trình duyệt", "trinh duyet", "browser", "tab", "cửa sổ", "cua so"]):
        return "close_browser"
```

Điều này sẽ giúp tiết kiệm thời gian xử lý các lệnh giúp ứng dụng thực hiện nhanh các câu lệnh tăng độ trải nghiệm, mượt mà.

3.4.2. Mô hình PhoBERT

Trước khi huấn luyện chuẩn hóa nhãn intent rồi lưu lại trong models với định dạng pkl.

Khi Rule-based không khớp, hệ thống chuyển qua PhoBERT fine-tune: tokenizer cảu PhoBERT chuyển văn bản thành input_ids và attention_mask. Đưa input vào mô hình, 900 câu lệnh với mỗi 100 câu/intent, lấy logit từ layer classifier để chọn lớp có logit cao nhất =>intent

Lưu đủ các code chỉnh cần thiết trước huấn luyện mô hình trong thư mục utils, dataset.py và preprocess.py. Huấn luyện mô hình bắt đầu chia train/test 80/20 mở cmd

nhập F: rồi cd vào vị trí thư mục chính chạy python –m training.train_phobert.py sau khi chạy trên vòng lặp 10 vòng thì lưu xuống dưới dạng pth vào thư mục models để sử dụng demo sau.

3.4.3. Triển khai Mapping hành động

```
def get_action(intent: str):
    mapping = {
        "open_browser": "Mở trình duyệt Chrome",
        "close_browser": "Đóng trình duyệt",
        "open_youtube": "Mở YouTube",
        "greet_pet": "Thú cưng vẫy đuôi chào lại",
        "praise_pet": "Thú cưng vui mừng, nhảy nhót",
        "volume_up": "Tăng âm lượng hệ thống",
        "volume_down": "Giảm âm lượng hệ thống",
        "pet_sleep": "Thú cưng đi ngủ (ẩn tạm)",
        "pet_minimize": "Thu nhỏ thú cưng xuống góc màn hình"
    }
    return mapping.get(intent, "Không hiểu lệnh")
```

Mapping lại các hành động theo đúng ý định câu lệnh thực hiện.

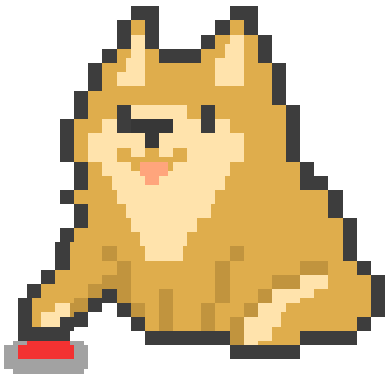
3.5. Giao diện trợ lý ảo/ thú cưng

Chuẩn bị các hình ảnh sống động về một nhận vật hay động vật, em tự vẽ nhân vật trên trang piskel, mỗi hành động vẽ 12 frame ảnh định dạng 64x64 khi lưu về dưới dạng đường thẳng spritesheet 12 cột 1 dòng 2304x192 px với độ phân giải mỗi hình là 192x192 đã được scale tăng tích cỡ. Mỗi hành động sẽ được lưu dưới định dạng PNG trong file assets.

Idle:

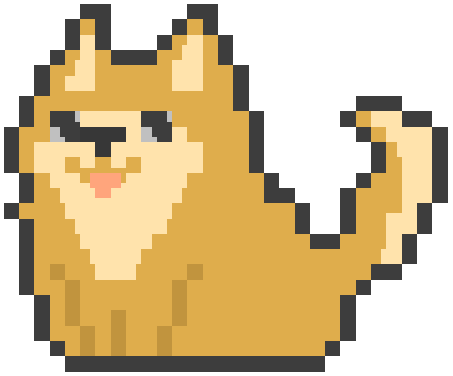
Sử dụng mặc định khi mới mở chương trình và luôn hiển thị cho đến khi có lệnh mới nhập vào, sau khi thực hiện xong lệnh, quay lại idle mặc định.

Push:



Hành động này sẽ được thực hiện khi thực hiện các lệnh chức năng ở đấy là open/close_browser – mở tắt trình duyệt, open_youtube – mở youtube lên, volume_up/down – tăng giảm âm lượng.

Happy:



Để tăng trải nghiệm, sự thú vị và vui vẻ khi sử dụng thêm các tác động chân thực ở đây là biểu thị sự vui vẻ trước các câu nói khen ngợi hay chào hỏi greet_pet – chào thú cưng trợ lý, praise_pet – khen ngợi thú cưng trợ lý.

Sleep:

Cuối cùng sẽ là hành động cho trợ lý đi nghỉ khi chúng ta không cần sử dụng nữa hay chỉ là mình muốn nó nghỉ nghơi thôi pet_sleep và pet_minimize.

Đây mới chỉ là phần nhỏ vẫn có thể thêm nhiều thứ khác nữa giúp ứng dụng trở nên càng sinh động hơn như một chú chó thật đang vui đùa cùng mình vậy

Tất cả sẽ được hiển thị trong bằng cửa sổ mở bằng inference/pet_window.py và gán hành động thực tế trên máy tính với các hành động bằng inference/action_executor.py.

3.6. Điều khiển máy tính thật

Sau khi nhận dạng được intent từ câu lệnh người dùng, hệ thống cần chuyển đổi intent thành các hành động thực tế trên máy tính. Triển khai action_executor.py với mục tiêu là thực hiện chính xác và an toàn các lệnh phổ biến mở/đóng ứng dụng, điều chỉnh âm lượng. Đồng bộ với giao diện thú cưng animation phản hồi, nó khá tương thích với windows 10/11. Các hành động được thực hiên dựa trên thư viên chuẩn của Python và một số thư viện hỗ trợ: subprocess, os.system dùng để mở và đóng tiến trình và keyboard giả lập phím vật lý để điều chỉnh âm lượng.

```
# Import hàm chung từ pet_window
from inference.pet_window import set_pet_state

# MỞ / ĐÓNG TRÌNH DUYỆT & YOUTUBE
def open_browser():
    set_pet_state("push")
    subprocess.Popen(["start", "chrome"], shell=True)
    print(" Đang mở Google Chrome...")

def close_browser():
    set_pet_state("push")
    os.system("taskkill /f /im chrome.exe >nul 2>&1")
    print(" Đã đóng tất cả cửa sổ Chrome!")

def open_youtube():
    set_pet_state("push")
    subprocess.Popen(['start', 'chrome', 'https://www.youtube.com'], shell=True)
    print(" Đang mở YouTube...")

# ÂM LƯỢNG
def volume_up():
    set_pet_state("push")
    keyboard.press_and_release('volume up')
    print(" Âm lượng đã tăng!")

def volume_down():
    set_pet_state("push")
    keyboard.press_and_release('volume down')
    print(" Âm lượng đã giảm!")
```

Đây chính là module quan trọng biến từ lý thuyết sang ứng dụng thực tế, mang lại giá trị sử dụng cho người sử dụng.

3.7. Mô hình so sánh BiLSTM + Embedding

Để đánh giá khách quan hiệu suất của mô hình học sâu hiện đại PhoBERT, đề tài triển khai thêm một mô hình truyền thống dựa trên BiLSTM kết hợp với lớp

embedding tự huấn luyện. Mô hình này sẽ đại diện cho hướng tiếp cận học sâu chuỗi phổ biến trước khi các mô hình Transformer thống trị lĩnh vực xử lý ngôn ngữ tự nhiên.

Mô hình BiLSTM được xây dựng với cấu trúc đơn giản nhưng hiệu quả cho bài toán phân loại chuỗi ngắn như nhận dạng ý định, đã nói qua trên phần cơ sở lý thuyết, giờ sẽ nói chi tiết hơn:

Lớp Embedding:

- Chuyển mỗi token trong câu thành vector có chiều 300.
- Embedding được khởi tạo ngẫu nhiên và huấn luyện cùng mô hình (random initialized embedding).
- Padding token được gán index 0 để xử lý câu có độ dài khác nhau.

Lớp BiLSTM:

- Sử dụng một lớp LSTM hai chiều (bidirectional).
- Hidden size: 256 cho mỗi chiều (tổng 512 khi kết hợp).
- Dropout 0.5 được áp dụng giữa các lớp để giảm overfit.

Lớp phân loại:

- Vector ẩn cuối cùng từ cả hai chiều LSTM được ghép nối (concatenate).
- Đưa qua lớp fully connected (Linear) để dự đoán lớp intent (9 lớp).

Quy trình forward của mô hình:

1. Tokenize câu thành chuỗi index dựa trên vocab tự xây.
2. Pad câu về độ dài cố định (max_len=32).
3. Embedding => BiLSTM => Ghép nối hidden state cuối => Dropout => Linear => Output logits.

Quá trình huấn luyện BiLSTM

- Xây dựng vocab: từ điển được tạo từ toàn bộ dữ liệu (khoảng 76 từ duy nhất do dữ liệu sinh từ template có cấu trúc).
- Dataset: Custom PyTorch Dataset với padding và encoding theo vocab.
- Huấn luyện: Optimizer Adam (learning rate 0.001), loss CrossEntropyLoss, Batch size 32 và Epochs 30 early stopping dựa trên accuracy.

Kết quả huấn luyện cũng sẽ được lưu xuống định dạng pth để so sánh với mô hình học sâu PhoBERT.

IV. Kết quả và thảo luận

Chỉ số	PhoBERT	BiLSTM
Accuracy	1.0	1.0
Precision (macro)	1.0	1.0
Recall (macro)	1.0	1.0
F1-score (macro)	1.0	1.0
F1-score (weighted)	1.0	1.0
Inference time (ms/câu)	18.14	0.3
Model size (MB)	515.08	4.47

Dựa trên kết quả benchmark trên tập kiểm tra (180 câu lệnh), cả hai mô hình PhoBERT và BiLSTM đều đạt hiệu suất tuyệt đối với Accuracy, Precision, Recall và F1-score đều bằng 1.0 (100%). Lý do dẫn đến chỉ số đánh giá tuyệt đối do dữ liệu không quá lớn, chỉ 900 câu lệnh với 9 intent và do được tạo ngẫu nhiên một cách thủ công dữ liệu được cân bằng tốt mỗi intent là 100 câu. Các câu nói cũng không quá dài mà chỉ là các câu lệnh ngắn để thực hiện sử dụng máy tính với tốc độ nhanh và ứng dụng cao. Vậy nên trong tương lai việc tăng số lượng dữ liệu, thêm các chức năng hành động mới có thể giảm chỉ số xuống.

Tuy nhiên, ta vẫn sẽ có sự khác biệt ở hai chỉ số inference time và model size. Về tốc độ inference BiLSTM nhanh hơn PhoBERT đáng kể với chỉ 0.3ms/câu, điều này phù hợp với lý thuyết BiLSTM có cấu trúc đơn giản, ít tham số hơn, phù hợp cho inference trên CPU hoặc thiết bị hạn chế tài nguyên. PhoBERT dù được tối ưu nhưng vẫn nặng hơn do kiến trúc Transformer nhiều tầng. Về kích thước mô hình, BiLSTM chỉ chiếm 4.47 MB, trong khi PhoBERT lên đến 515.08 MB gấp nhiều lần, BiLSTM rõ ràng vượt trội về khả năng triển khai trên thiết bị nhúng, ứng dụng di động hoặc hệ thống có bộ nhớ hạn chế.

Ưu nhược điểm tổng hợp, PhoBERT có thể hiểu ngữ nghĩa sâu, dễ mở rộng cho dữ liệu phức tạp hơn (khẩu ngữ, câu dài, tiếng địa phương). Nhưng kích thước lớn, inference chậm hơn trên CPU, tiêu tốn tài nguyên. BiLSTM lại nhẹ, nhanh, dễ huấn luyện trên dữ liệu nhỏ nhưng khó xử lý phụ thuộc dài hạn, cần embedding chất lượng cao nếu dữ liệu đa dạng hơn.

Kết luận lại hệ thống sử dụng PhoBERT kết hợp Rule-based là lựa chọn tối ưu nếu muốn phát triển hơn đề tài với nhiều chức năng mới về sau. Rule sẽ hoạt động nhanh và chính xác tốt với các lệnh phổ biến, PhoBERT sẽ chỉ được gọi khi cần, tận dụng sức mạnh ngữ nghĩa mà vẫn giữ tốc độ tổng thể cao. Và với

hướng phát triển là sử dụng áp dụng thêm giọng nói để ra lệnh thì nó là tốt nhất. BiLSTM có thể dùng thay thế đối với các thiết bị quá yếu.

Kết quả đã cho ta thấy dữ liệu lệnh tiếng Việt được thiết kế khá ổn, cả hai hướng tiếp cận truyền thống và hiện đại đều đạt kết quả cao.

V.Demo và ứng dụng thực tế

Hệ thống được triển khai dưới dạng ứng dụng desktop chạy trên windows, không yêu cầu kết nối internet hoặc GPU. Demo chính thức được thực hiện qua file inference/demo_text.py với giao diện console đơn giản và cửa sổ thú cưng pixel art hiển thị đồng thời.

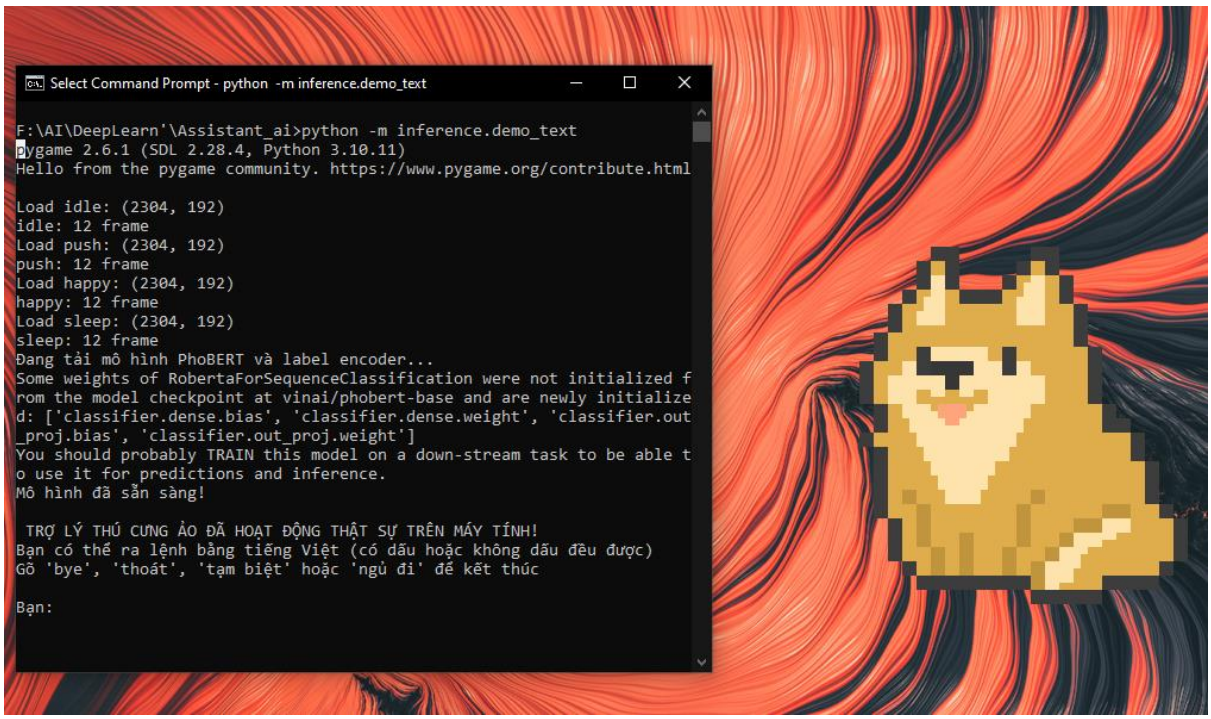
Đầu tiên ta sẽ khởi động demo như sau:

```
C:\Users\hi>F:

F:\>cd F:\AI\DeepLearn'\Assistant_ai

F:\AI\DeepLearn'\Assistant_ai>python -m inference.demo_text
```

Chạy trong file lưu trữ tổng và chờ nó load.

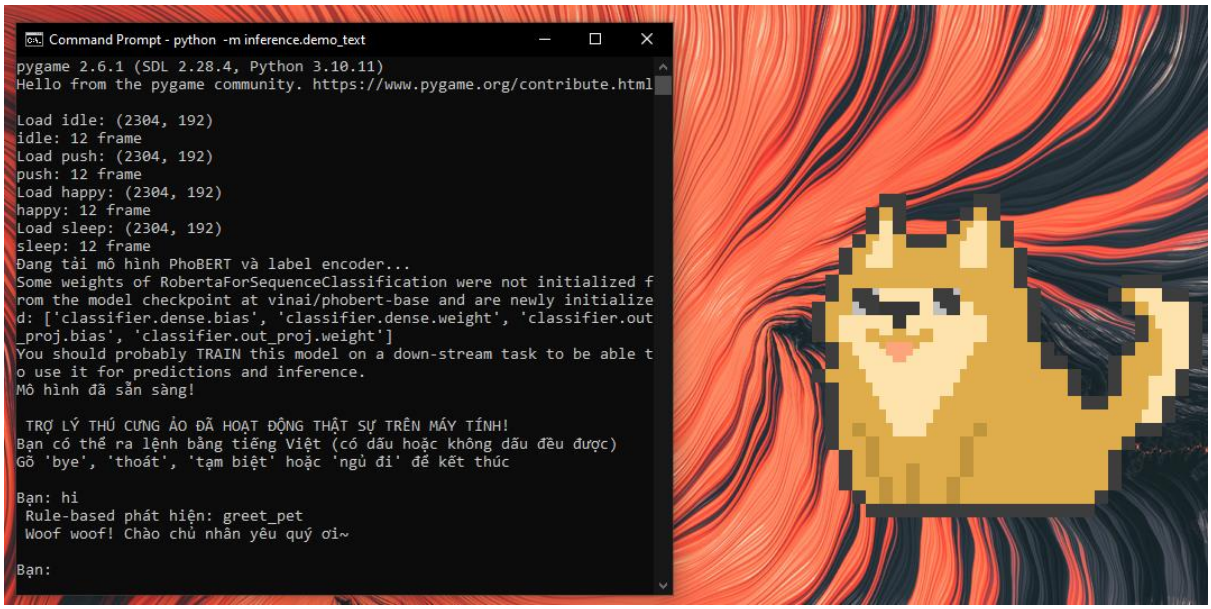
```
F:\AI\DeepLearn'\Assistant_ai>python -m inference.demo_text
pygame 2.6.1 (SDL 2.28.4, Python 3.10.11)
Hello from the pygame community. https://www.pygame.org/contribute.html
Load idle: (2304, 192)
idle: 12 frame
Load push: (2304, 192)
push: 12 frame
Load happy: (2304, 192)
happy: 12 frame
Load sleep: (2304, 192)
sleep: 12 frame
Đang tải mô hình PhoBERT và label encoder...
Some weights of RobertaForSequenceClassification were not initialized fr
d are newly initialized: ['classifier.dense.bias', 'classifier.dense.wei
t_proj.weight']
You should probably TRAIN this model on a down-stream task to be able to
Mô hình đã sẵn sàng!

 TRỢ LÝ THÚ CƯNG ẢO ĐÃ HOẠT ĐỘNG THẬT SỰ TRÊN MÁY TÍNH!
Bạn có thể ra lệnh bằng tiếng Việt (có dấu hoặc không dấu đều được)
Gõ 'bye', 'thoát', 'tạm biệt' hoặc 'ngủ đi' để kết thúc

Bạn:
```

Sau khi load xong nó sẽ hiển thị chỗ nhập lệnh như sau và thêm ảnh động nhân vật trợ lý ảo thú cưng.

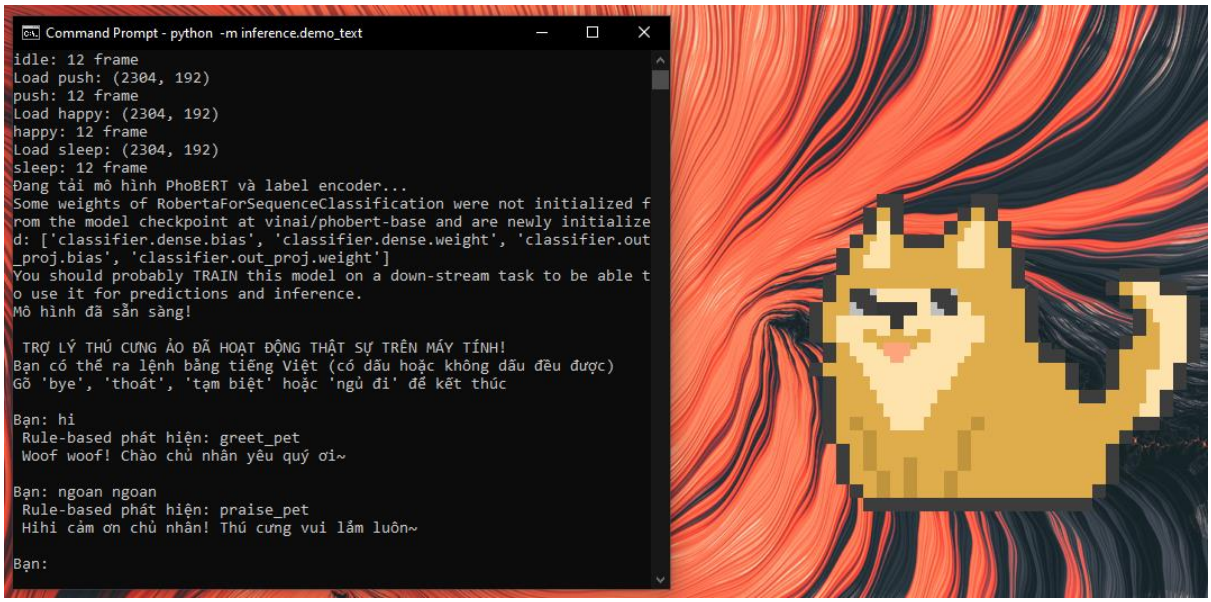


Được hiển thị ở giữa màn hình desktop, giờ tôi sẽ thực hiện các tác vụ để xem sự thay đổi hoạt ảnh và cách trả lời nhận dạng đúng ý định câu lệnh của nhân vật.

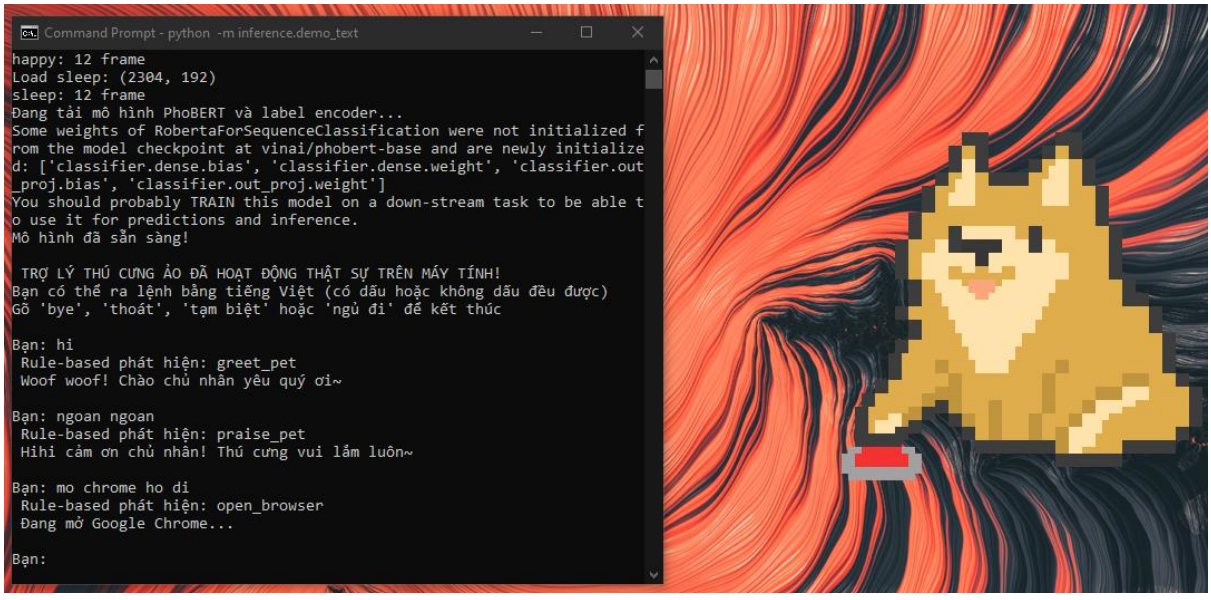
Chào hỏi:

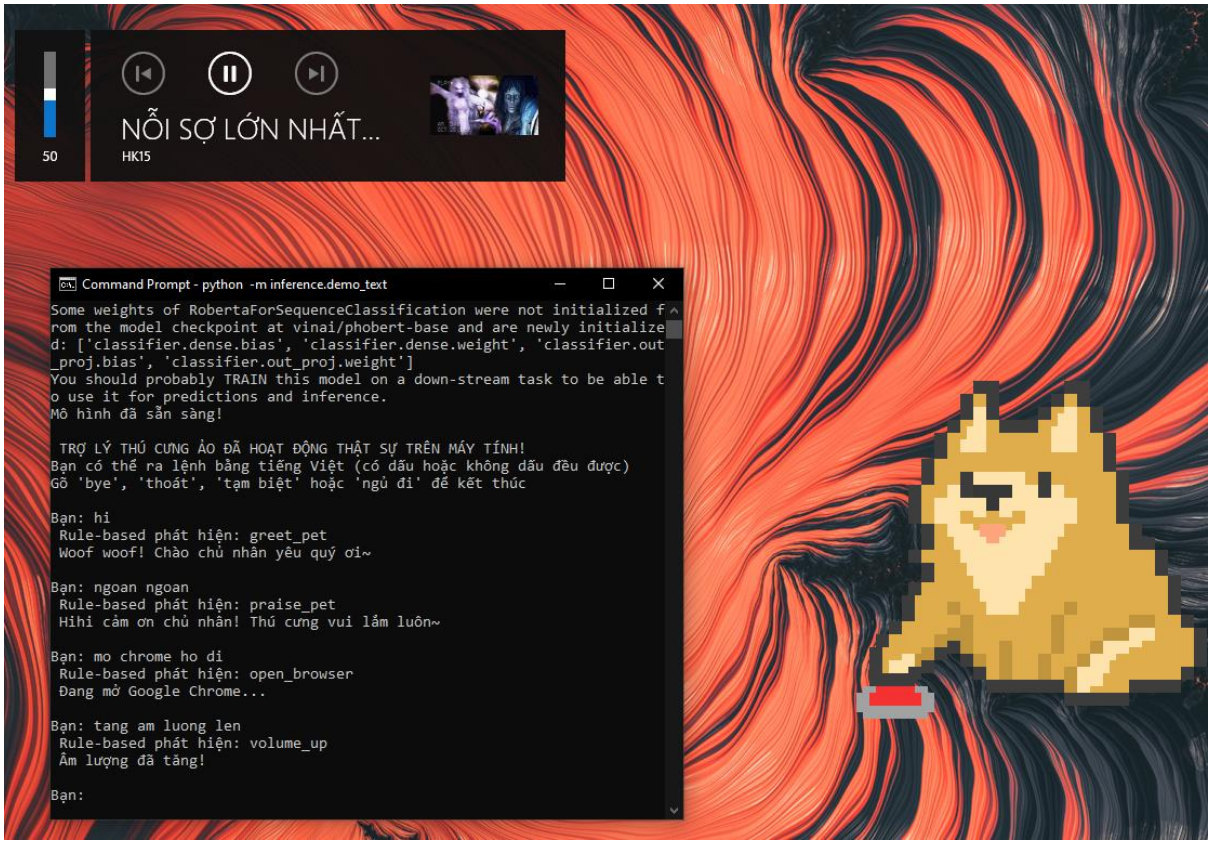


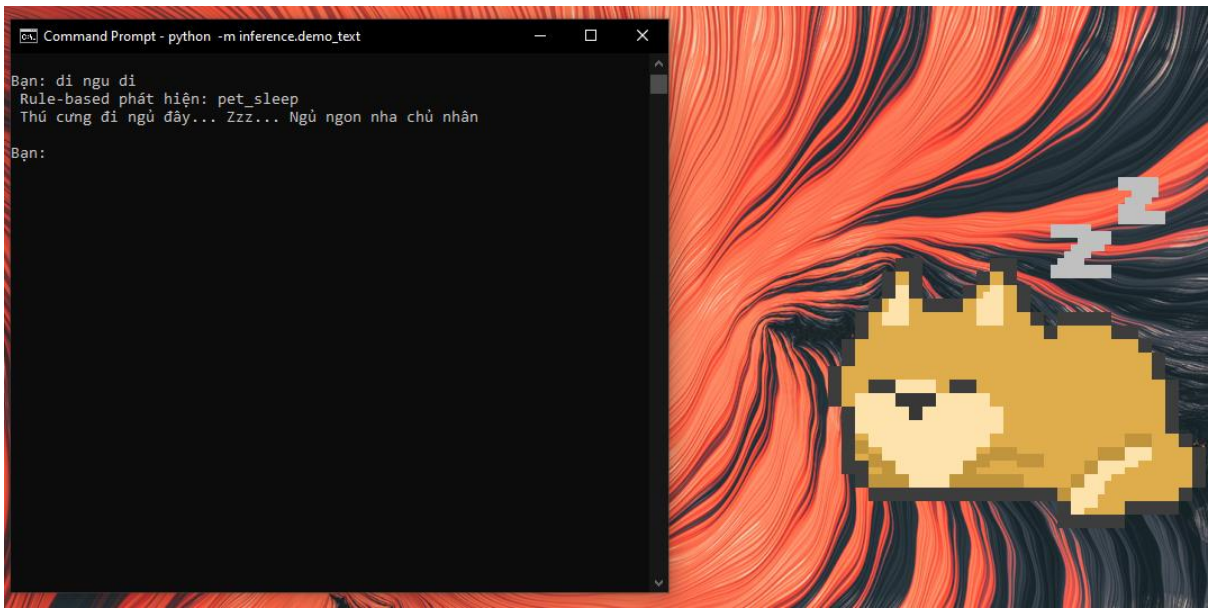
Khen ngợi:



Yêu cầu:







Việc các chức năng có hoạt động chuẩn và áp dụng được hay không sẽ được quay lại video là lưu lại trong file tổng.

Tổng kết thì các chức năng bản thân muốn đã hoạt động rất tốt và ổn định, sẽ phát triển thêm nhiều chức năng khác trong tương lai và có thể thêm nhiều cơ chế vui mang lại sự sống động như vuốt ve nhân vật, cho ăn hay chơi đùa với nhau, mục tiêu sắp tới là xử lý âm thanh để ra lệnh cho nhân vật bằng lời nói.

Phụ lục

[1] Nguyễn Quang Vinh et al., "PhoBERT: Pre-trained language models for Vietnamese," in Findings of the Association for Computational Linguistics: EMNLP 2020, 2020, pp. 1037–1042. (DOI: https://doi.org/10.18653/1/2020.findings-emnlp.93)

[2] Hugging Face, "PhoBERT: Pre-trained language models for Vietnamese," Hugging Face Model Hub, 2020. [Online]. Available: https://huggingface.co/vinai/phobert-base. (Accessed: Dec. 2025).

[3] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I., "Attention is all you need," in Advances in Neural Information Processing Systems 30 (NIPS 2017), 2017, pp. 5998–6008.

[4] Hochreiter, S., & Schmidhuber, J., "Long short-term memory," Neural Computation, vol. 9, no. 8, pp. 1735–1780, 1997.

[5] Graves, A., Fernández, S., & Schmidhuber, J., "Bidirectional LSTM networks for improved phoneme classification and recognition," in Artificial Neural Networks: Formal Models and Their Applications – ICANN 2005, 2005, pp. 799–804.

[6] Pygame Community, "Pygame documentation," 2025. [Online]. Available: https://www.pygame.org/docs/. (Accessed: Dec. 2025).

[7] Python Software Foundation, "Python documentation," 2025. [Online]. Available: https://docs.python.org/3/. (Accessed: Dec. 2025).

[8] Hugging Face, "Transformers documentation," 2025. [Online]. Available: https://huggingface.co/docs/transformers/. (Accessed: Dec. 2025).

[9] scikit-learn developers, "scikit-learn: Machine Learning in Python," 2025. [Online]. Available: https://scikit-learn.org/stable/. (Accessed: Dec. 2025).