**Journal of Industrial Information Technology and Application** *ISSN(Online): 2586-0852*
Vol. No. ()

# BEACON - Voice Assistant for Impaired People on Windows PC

Truong Quoc Thang[1,*)], Pham Quoc Bao[1)], Nguyen Van Vi[1)], Than Thi Thao[1)], Nguyen Trieu Tien[1)] and Tran Kim Sanh[1)]
[1)] International School, Duy Tan University, Da Nang City, Vietnam

**Abstract.** This project presents the development and implementation of a specialized voice assistant tailored to support visually impaired individuals operating Windows PC systems called Beacon. The voice assistant is designed to enhance accessibility by facilitating seamless interaction with integrated functions, including music playback, news consumption, receiving usage instructions, and volume control, catering to the unique needs of visually impaired users. Notably, the assistant is equipped with support for the Vietnamese language and employs a user intention recognition model based on PhoBERT, and FastAPI, contributing to the accurate interpretation of user commands. Evaluation through testing demonstrates an impressive accuracy rate of up to 90% for the provided test data, showcasing the efficacy of the voice assistant in empowering visually impaired users to navigate and utilize Windows PC functionalities independently and efficiently.

**Keywords;** Voice Assistant, Visually Impaired, Windows PC Accessibility, User Intention Recognition, PhoBERT-based Interaction

## 1. Introduction

According to statistics, Vietnam currently has about 2 million blind and visually impaired people. Among them, one-third are impoverished and cannot afford treatment to regain their sight.

Additionally, here are some other pieces of information about the visually impaired in Vietnam:

[*] Corresponding author: windev.thang@gmail.com
Received: Feb. 20, 2024; Accepted: May. 19, 2024; Published: Jun. 30, 2024

- Over 7% of the population aged 2 and older (about 6.2 million people) have visual impairments.
- 13% of the population (almost 12 million people) live in families with visually impaired members.
- This rate is expected to increase with the aging population trend.
- Households with disabled members, including the visually impaired, tend to be poorer, and visually impaired children are at a higher risk of not attending school compared to their peers.
- Employment opportunities for the visually impaired are also lower compared to those without visual impairments.
- The most common disabilities in children are related to social and psychological aspects, posing significant barriers to the social integration of visually impaired children.

Especially in today's adolescent generation, the increased use of electronic devices (smartphones, laptops, etc.) has led to a decline in vision. Is there any solution to help visually impaired individuals reduce their dependence on staring at screens?

To address these issues, we are researching and developing a voice assistant for Windows. With this voice assistant, visually impaired individuals won't need to rely too much on the screen for basic tasks such as searching for information online or reading the news. This helps visually impaired individuals integrate more with the world of technology and reduces the risk of serious eye conditions.

In this project, we have conceptualized the research and development of a voice assistant, analyzing, and implementing actions on a Windows PC to assist visually impaired individuals in easily using basic functions such as searching for information, listening to music, and reading news. The assistant is named Beacon, suggesting its role as a guide or signal, capable of helping users navigate around the computer interface. The assistant performs key functions, including: Executing actions according to the user's requests (reading news, listening to music, etc.), Collecting user conversations - ensuring necessary security measures - to enhance the accuracy of responses to users, Predicting behavior and providing recommendations to users based on their chat history with the assistant.

## 2. Related Work

In the era of artificial intelligence, many models have been developed for various purposes such as Multimodal, Computer Vision, Natural Language Processing, Audio, Tabular, and Reinforcement Learning.

PhoBERT [1] is specifically designed for Vietnamese text and is based on the RoBERTa architecture. It was trained on a large corpus of 20GB of Wikipedia and news text, achieving state-of-the-art results on several natural language processing tasks for Vietnamese. These tasks include part-of-speech tagging, parsing, named entity recognition, and natural language inference. PhoBERT comes in two versions: the "base" version with 135 million parameters and the "large" version with 370 million parameters. Researchers and practitioners can utilize PhoBERT in libraries such as FAIRSeq or Transformers for tasks like text classification, question answering, sentiment analysis, and more. Its availability allows us to leverage existing knowledge and build upon it, rather than starting from scratch.

This system serves as a prime example. By using PhoBERT as its base model, we can efficiently train a new model on our specific data – in this case, for the purpose of intent classification. Think of it like building a specialized tool on top of a powerful engine. PhoBERT provides the core understanding of Vietnamese language, while our data shapes it to expertly handle the nuances of our unique intent classification problem. This not only saves time and resources but also ensures the new model inherits the solid foundation of PhoBERT's knowledge.

Table I. COMPARE BEACON WITH OTHER VOICE ASSISTANT

| Features | BEACON | CORTANA | PIGO |
|---|---|---|---|
| News aggregation | | | X |
| Date, Weather, Chate | | X | X |
| Play Music | X | X | |
| Interesting commands | X | X | |
| Timer | | X | |
| Control browser by speech | X | X | |
| Commands for setting up on the computer. | X | X | |
| Technical guidance | X | X | |
| Read newspapers | X | X | X |
| Vietnamese language support | X | | X |
| Longer Support | X | | X |

Similar to advancements in the development of artificial intelligence, the field of voice assistants on Windows has witnessed significant growth and diversification. Let's delve into some notable contributions:

**Pioneering Efforts:** The emergence of Cortana led to the initial efforts, marking Microsoft's first foray into personal assistants. Cortana seamlessly integrated with Windows, providing basic functions like setting reminders, making calls, and web searches. However, limitations in its feature set and language support hindered widespread adoption.

**Rise of Third-Party Solutions:** Recognizing the increasing demand for voice-controlled experiences, third-party solutions stepped in. Popular options like Alexa and Google Assistant became accessible through dedicated apps, offering users expanded capabilities such as smart home control, streaming music, and personalized news sources. Despite providing greater flexibility, these assistants lacked deep integration with the Windows ecosystem.

**Emerging Trends and Future Prospects:** Several compelling trends are shaping the future of voice assistants on Windows.

**Skill Development:** Open platforms like Open Assistant SDK enable developers to create custom skills, allowing for tailored voice experiences to align with specific workflows and industries.

Despite the abundance of voice assistants in the current market, there is still a lack of assistants with the capability for personalization and optimal support for the visually impaired in general, and specifically for Vietnamese.

*To address this issue, we have built and developed Beacon, a voice assistant on Windows PC that not only personalizes user behavior but also aims to provide maximum support for the visually impaired.*

## 3. Proposed Solutions

A high-level overview of the process by which we built a user intent classification system using machine learning:

*1) Collect Data: Gather a diverse dataset containing examples of user queries or sentences along with their corresponding intent labels. Ensure that the data covers various scenarios and user intents you want to classify (e.g., play music, read news, customer support)*

*2) Data Preprocessing: Clean and normalize text data by removing irrelevant characters, converting to lowercase, and standardizing punctuation. Segment text into appropriate units (words, sentences). Utilize Underthesea for Vietnamese-specific preprocessing tasks like tokenization, normalization, and named entity recognition.*

*3) Model Selection: Consider both traditional (rule-based) and machine learning approaches based on data size and complexity. For machine learning, choose a*

*Transformer-based architecture like PhoBERT for Vietnamese text. Explore pre-trained models like PhoBERT-base or PhoBERT-large based on computational resources and desired accuracy.*

*4) Model Training: Split data into training, validation, and test sets. Fine-tune the pre-trained PhoBERT model on your labeled dataset using an appropriate optimizer like Adam. Utilize transfer learning to leverage PhoBERT's knowledge while adapting to specific intent categories.*

*5) Evaluate the Model: Calculate metrics like accuracy, precision, recall, and F1-score for each intent on the validation set. Employ visualization techniques like confusion matrices to analyze model performance and identify areas for improvement.*

*6) Deploy the Model: Deloy User Intent Classification Model with Hugging Face as an API service.*

*7) Continue to Improve: Collect user feedback and analyze model performance in production. Retrain the model with new data or refine hyperparameters regularly. Explore advanced techniques like active learning to gather more informative data and improve model accuracy over time.*

We are building an application on Windows PC called Beacon with the feature of converting speech into commands, thereby helping users perform tasks they want through voice, such as reading news, listening to music, and adjusting the volume. The assistant has a speech output, starting from receiving the user's speech using Azure's speech-to-text service. In the next step, the system uses the converted text to perform analysis using a pre-trained model and then returns the corresponding action to be executed. Finally, the assistant uses the returned result to process and perform the requested action.

The solution will assist visually impaired individuals in using a computer through voice without the need for direct manipulation. The proposed solution is to build a system called "Beacon - Voice Assistant for Visually Impaired on Windows PC," as illustrated in Figure 1.

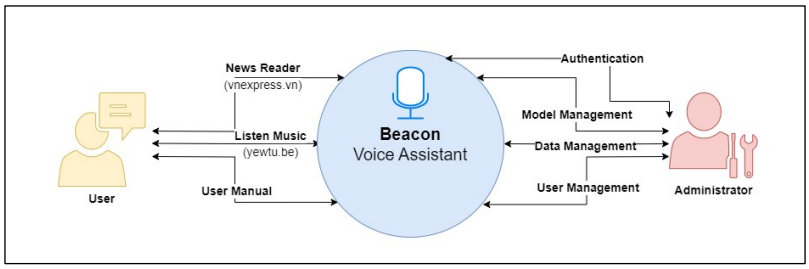


Figure 1. System context diagram

The system context diagram of Beacon comprises two entities: Visually impaired users and Administrators. Among them, visually impaired users can utilize the application through voice with functions such as reading news (vnexpress.vn), listening

to music (yewtu.be - an ad-free YouTube clone), and requesting system usage instructions (to assist users in easily using the system without requiring technological knowledge). Administrators are responsible for managing data, including user data, application versions, and corresponding application usage guides.

The overall architecture of our proposed solution, along with associated components and connector, is illustrated in Figure 2.

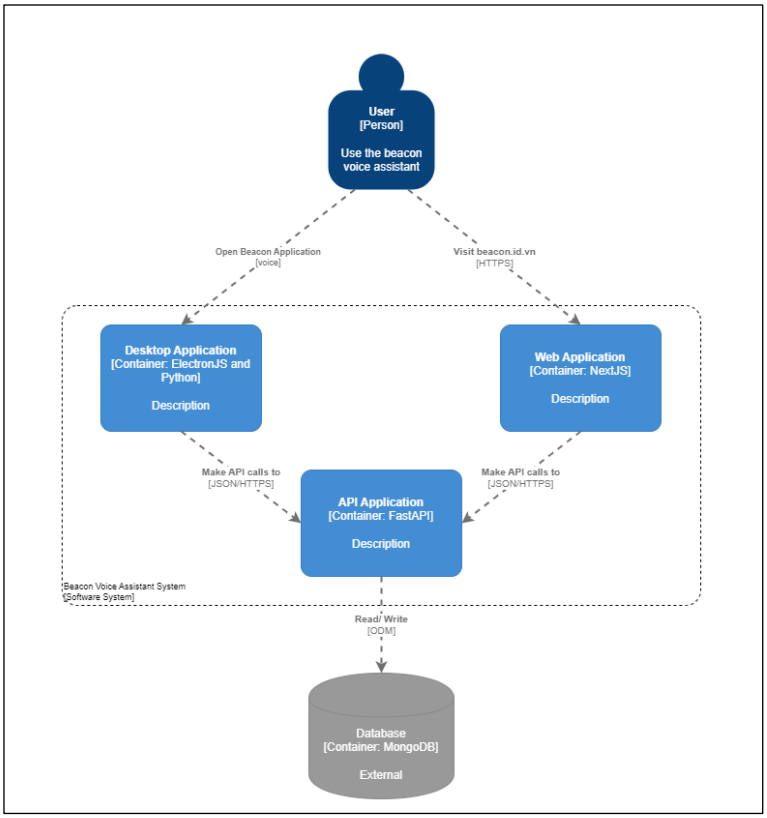


Figure 2. Component and connector of Beacon

Users can interact on two platforms: web application and desktop application. Users access the web to download the app link and then can use the app on the computer. Web application calls API application to get the latest version of the app and desktop application calls API application to use the services. API application interacts with the database and manipulates data in the database. Our project uses MongoDB to easily map the actual object to the entity in the DB and make it easy to manipulate the data in the database.

Table II. COMPONENT AND CONNECTOR ROLES AND RESPONSIBILITIES

| Role | Responsibilities |
|---|---|
| User | Access to the website call beacon.id.vn and download app then use app on the desktop. |
| Web Application | Contains information about us, contact section, app download link and call API to get the latest version. |
| Desktop Application | Communicate directly with the user and call the API application to use the corresponding service that the API application provides. |
| API Application | Contains middleware to check the valid authentication and provide the corresponding service from user's request, contains all the services of the application, interacts with database to manipulate data |
| Database | Store all the data of the application |

The main components are described in more detail below.

## *B. C&C view based on Web Application*

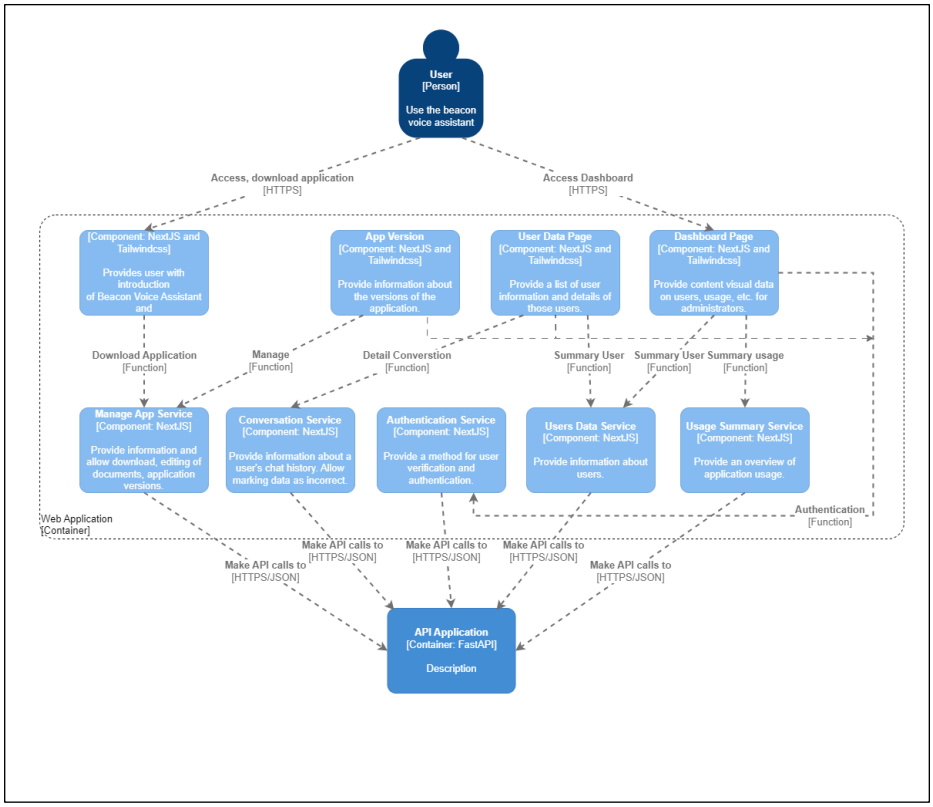


Figure 3. Component and connector of Beacon – Web Application

When users access the website via beacon.id.vn, they will see an interface landing page including sections about us, contact, and download. When the user clicks download,

the web application calls the API to the server to get the version along with the latest download link for the application.

When users access the admin page with admin rights, they can use the functions of app version management, user information statistics, labeling, and user conversation management.

Table III. COMPONENT AND CONNECTOR ROLES AND RESPONSIBILITIES

| Role | Responsibilities |
|---|---|
| Home Page | Displays information describing Beacon Voice Assistant clearly, providing an easy-to-understand and easy-to-use download method |
| Manage App Service | Show details about app versions, including information like version number, description, new features, and ensure that only authorized people can download or edit information. |
| App Version | Contains information about the application's table sessions |
| User Data Page | Provide a list of user information and details of those users |
| Dashboard Page | Provide content visual data on users, usage for administrators |
| Conversation Service | Provide information about a user's chat history |
| Authentication Service | Provide a method for user verification and authentication |
| User Data Service | Provide information about users |
| Usage Summary Service | Provide an overview of application usage |
| API Application | An application to provide an application programming interface and use of functions and services, helping to connect and interact |

*C. C&C view based on Desktop Application*

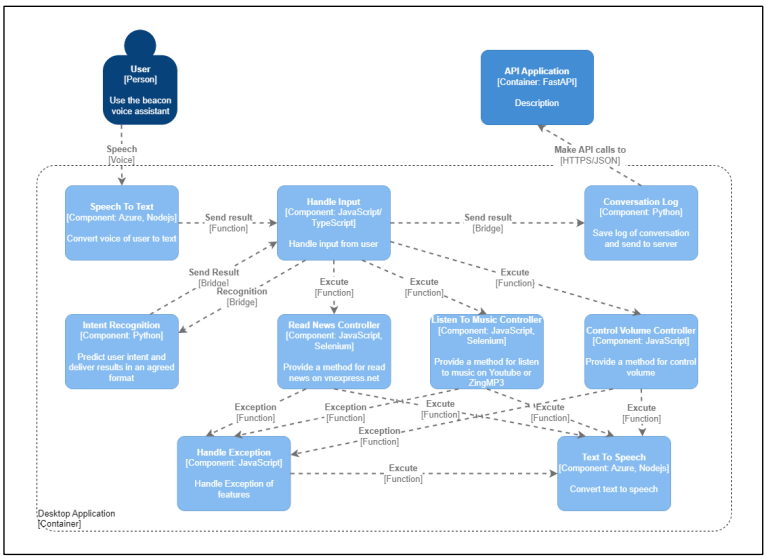


Figure 4. Component and connector of Beacon – Web Application

When the user starts the laptop, the beacon application will automatically connect to the laptop at the same time and the user can use voice to communicate directly with the application. When the user wants to communicate, the user will say the request they want the application to perform. When the user speaks, the application will receive the voice and convert that voice into text for the application to process through the function called "Speech to text". Users will choose 1 of 2 services: listening to music or reading news. Once the user has selected and the request is sent, the application will process it through the "Handle input from speech" function and then return the results from the text that the user wants through the pc speakers - "Text to speech".

Table IV. COMPONENT AND CONNECTOR ROLES AND RESPONSIBILITIES

| Role | Responsibilities |
|---|---|
| User | Articulate their request, using Interactive Voice Assistant. |
| API Application | Contain All APIs of application, Processes User's request and forward to the corresponding service. |
| Speech To Text | Convert Speech from user into Text for application to process. |
| Handle Input | Process Input from user, initiates the request and forward request to the corresponding controller to process after receiving processing from Intent Recognition. |
| Conversation Log | Contains a log of previous uses of the application |
| Intent Recognition | Intent Processing, Command Execution / Functionality, send result to Handle Input |
| Read News Controller | Contain methods about Read News to processing request from user when the request is sent here |
| Listen To Music Controller | Contain methods about Listen to Music to processing request from user when the request is sent here |
| Control Volume Controller | Adjust the volume for the application |
| Handle Exception | Provides a method for control Exception when execution flow is misaligned |
| Text To Speech | Convert Text into Speech |

## 4. Experiment Results

Beacons are deployed according to the business requirements and system architecture described in Figure 5.

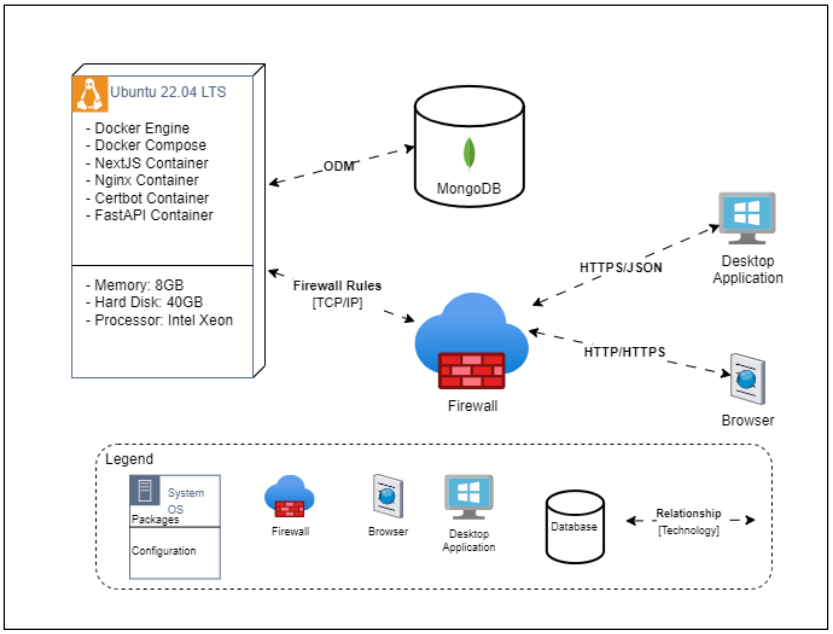


Figure 5. Deployment Architecture of Beacon

The software system consists of several components that interact with each other to provide various services to the users. The system is hosted on an Ubuntu 22.04 LTS server, which has 8GB of memory, 40GB of hard disk space, and an Intel Xeon processor. The server also has Docker Engine and Docker Compose installed, which allow the system to run multiple containers in a coordinated manner.

The containers include NextJS, Nginx, Certbot, and FastAPI, each with a specific role and responsibility. NextJS is a React framework that handles the front-end development and rendering of the web pages. Nginx is a web server that serves static files and proxies requests to other containers. Certbot is a tool that automates the process of obtaining and renewing SSL certificates for secure communication. FastAPI is a Python framework that handles the back-end logic and API endpoints of the system.

The system also uses MongoDB as the database, which stores data in a structured format that can be accessed by other components. MongoDB is connected to the Ubuntu server via ODM, which is an object-document mapper that simplifies the interaction between Python objects and MongoDB documents.

The system is protected by a firewall, which implements TCP/IP rules to safeguard against unauthorized access while allowing necessary communications. The firewall is positioned centrally in the architecture, connecting all major components. The system communicates with external applications or browsers using HTTP/S and JSON protocols.

The desktop applications and the browser are the interfaces that users interact with to access the services provided by the system.

Table V. COMPONENT AND CONNECTOR ROLES AND RESPONSIBILITIES

| Role | Responsibilities |
|---|---|
| User Ubuntu 22.04 LTS Server | Hosts the core system packages and configurations necessary for running containers. |
| Containers (Docker Engine, Docker Compose, NextJS, Nginx, Certbot, FastAPI) | Each container has a specific role in ensuring that applications are deployed efficiently and securely |
| MongoDB | Stores data in a structured format accessible by other components of the system |
| Firewall | Implements TCP/IP rules to safeguard against unauthorized access while allowing necessary communications |
| Desktop Application/Browser | Interfaces that users interact with to access services provided by the server |

The implementation of intent recognition and classification based on the PhoBERT model is carried out through the following six steps during the experiment phase.

**Step 1: Data Preparation (Phase: Data Exploration & Preprocessing)**

- Imports libraries and dependencies.
- Reads training and test data from CSV files.
- Defines labels and converts them to IDs.
- Tokenizes sentences using UnderTheSea library.
- Creates a datasets.DatasetDict for easier handling.

```
train_df = pd.read_csv("pat_to_train_data")
test_df = pd.read_csv("path_to_valid_data")
labels = train_df["label"].unique().tolist()
label2id = {label: i for i, label in enumerate(labels)}
id2label = {i: label for i, label in enumerate(labels)}
def tokenize(sentence):
    return word_tokenize(sentence, format="text")
train_df["text"] = train_df["text"].apply(tokenize)
test_df["text"] = test_df["text"].apply(tokenize)
train_df["label"] = train_df["label"].map(label2id)
```

```
test_df["label"] = test_df["label"].map(label2id)
imdb = datasets.DatasetDict(
  {
    "train": datasets.Dataset.from_pandas(train_df),
    "test": datasets.Dataset.from_pandas(test_df),
  }
)
```

**Step 2: Classification Function (Phase: Feature Engineering & Modeling)**

- Defines a classification function that uses UnderTheSea to classify a sentence.
- This could be considered a simple classification phase, but its interaction with the next step blurs the line.

```
def classification(sentence):
 result = {
   "text": sentence,
   "predcict": classify(sentence)
 }
     return result
```

**Step 3: Preprocessing & Tokenization (Phase: Feature Engineering & Preprocessing)**

- Defines a preprocess_function that uses a provided tokenizer (assumed to be defined elsewhere) to preprocess text data.
- This transforms data into model-specific features.

```
def preprocess_function(content):
  return tokenizer(content["text"], truncation=True)
tokenized_imdb = imdb.map(preprocess_function, batched=True)
```

**Step 4: Data Collation & Tokenization (Phase: Model Training & Evaluation Preparation)**

- Defines a data_collator using the same tokenizer for padding and batching data.
- This prepares data for efficient model training and evaluation.

```
data_collator = DataCollatorWithPadding(tokenizer=tokenizer)
```

**Step 5: Evaluation Metric Definition (Phase: Model Evaluation Planning)**

- Defines an accuracy metric using the evaluate library.
- This outlines how model performance will be measured.

```
accuracy = evaluate.load("accuracy")

def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = np.argmax(logits, axis=1)
    return accuracy.compute(predictions=predictions, references=labels)
```

**Step 6: Model Training & Evaluation (Phase: Model Training & Evaluation)**

- Loads a pre-trained model vinai/phobert-base-v2 for intent classification.
- Defines training arguments with:

*1) learning_rate (float): This determines the step size the optimizer takes when updating model parameters during training. Lower values make training more stable but slower, while higher values can be faster but risk instability. Here, it's set to 5e-5 (which is 0.00005).*
*2) per_device_train_batch_size (int): This defines the number of data samples fed to the model in each training step on each device (GPU or TPU). Setting it to 16 means processing 16 samples per step on each device.*
*3) per_device_eval_batch_size (int): Similar to per_device_train_batch_size, but determines the batch size for evaluation (e.g., validation). Here, it's also set to 16.*
*4) num_train_epochs (int): This specifies the number of times the entire training dataset will be passed through the model during training. In this case, it's set to 10 epochs.*
*5) weight_decay (float): This is a regularization technique that penalizes large model weights during training, discouraging overfitting. Higher values encourage smaller weights and potentially reduce overfitting, but lower values can improve performance. Here, it's set to 0.01.*
*6) evaluation_strategy (str): This controls how often the model is evaluated during training. "epoch" means the model is evaluated after each training epoch, while other options like "steps" or "wall_time" trigger evaluation based on the number of training steps or elapsed time.*
*7) save_strategy (str): This defines when to save the model during training. "epoch" saves the model after each epoch, while other options like "no" avoid saving during training but might save at the end.*

*8) load_best_model_at_end (bool): If True, this loads the model with the best performance on the evaluation metric during training, even if it was saved during an earlier epoch. This helps select the best performing model.*

- Creates a Trainer object for training and evaluation:

*1) model: it's the pre-trained vinai/phobert-base-v2 model for sequence classification.*
*2) args: This is a TrainingArguments object containing hyperparameters and training settings like learning rate, epochs, etc.*
*3) train_dataset: This defines the training dataset that will be used to train the model. Here, it's the "train" part of tokenized_imdb dataset obtained after preprocessing.*
*4) eval_dataset: This defines the evaluation dataset that will be used to assess model performance during training. It's the "test" part of tokenized_imdb dataset.*
*5) data_collator: This specifies a function that prepares batches of data for the model during training and evaluation. Defined this object earlier as data_collator.*
*6) tokenizer: This specifies the tokenizer used to pre-process the text data into numerical tokens. Might have defined this object elsewhere and are passing it here.*
*7) compute_metrics: This is a function that calculates evaluation metrics (e.g., accuracy) on the model predictions. Defined this function earlier as compute_metrics.*

- Trains the model on the prepared data and evaluates it on the test set.
- Saves the best performing model.

```
model = AutoModelForSequenceClassification.from_pretrained(
    "vinai/phobert-base-v2", cache_dir='cache', num_labels=22, id2label=id2label, label2id=label2id
).to("cuda")
training_args = TrainingArguments(
    output_dir="model",
    learning_rate=5e-5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=10,
    weight_decay=0.01,
    evaluation_strategy="epoch",
    save_strategy="epoch",
```

```
    load_best_model_at_end=True,
    # push_to_hub=True,
    overwrite_output_dir=True
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_imdb["train"],
    eval_dataset=tokenized_imdb["test"],
    data_collator=data_collator,
    tokenizer=tokenizer,
    compute_metrics=compute_metrics
)

trainer.train()
trainer.evaluate()
trainer.save_model("model")
```

| Epoch | Training Loss | Validation Loss | Accuracy |
|---|---|---|---|
| 1 | No log | 1.097832 | 0.820106 |
| 2 | No log | 0.525013 | 0.939153 |
| 3 | No log | 0.307612 | 0.941799 |
| 4 | No log | 0.214880 | 0.947090 |
| 5 | No log | 0.223745 | 0.949735 |
| 6 | 0.682300 | 0.190405 | 0.962963 |
| 7 | 0.682300 | 0.171637 | 0.965608 |
| 8 | 0.682300 | 0.137264 | 0.970899 |
| 9 | 0.682300 | 0.140251 | 0.968254 |
| 10 | 0.682300 | 0.137373 | 0.970899 |

Figure 6. Train model

*A. Results*

The test results are based on the model publicly available at: https://huggingface.co/yensubldg/model

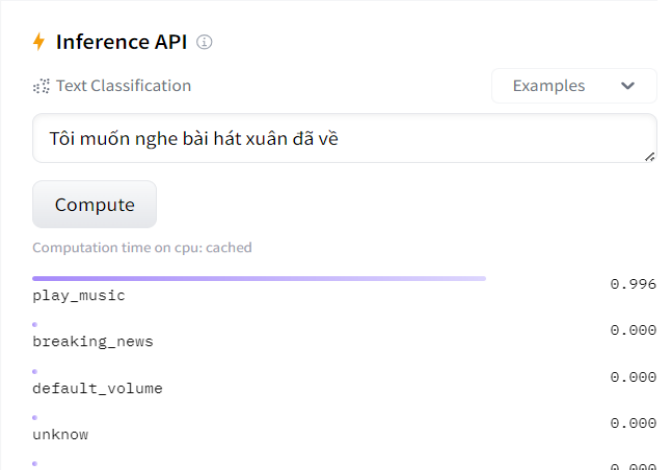


Figure 7. Test with play_music label

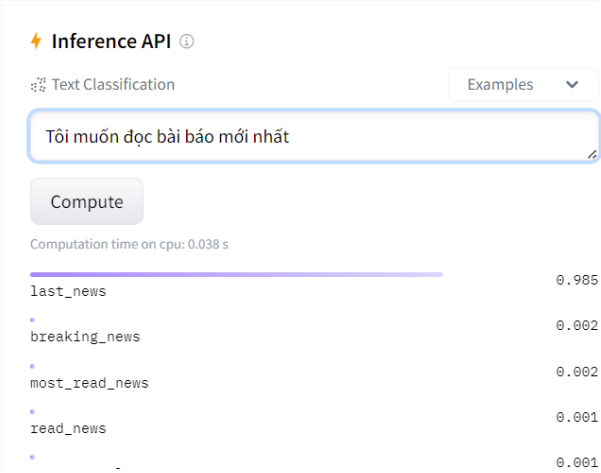


Figure 8. Test with last_news label

Table VI. THE EXPERIMENT RESULTS

| Title | Result |
|---|---|
| Average time per response at-tempt | Approximately 0.3s per request |
| Average accuracy | 90% overall |

## 5. Conclusion

The application successfully converts and executes user voice commands related to functions such as listening to music, reading the news, and providing usage instructions. It accurately identifies common commands from users with 22 researched and labeled categories, such as play_music, read_news, last_news, breaking_news, etc. The achieved accuracy is over 90% based on the validation dataset.

## References

[1] Nguyen, Dat Quoc, and Anh Tuan Nguyen, Findings of the Association for Computational Linguistics: EMNLP 2020. Viet Nam, pp. 1037–1042, 2020.

[2] S. M. Felix, S. Kumar and A. Veeramuthu, A Smart Personal AI Assistant for Visually Impaired People, 2018 2nd International Conference on Trends in Electronics and Informatics (ICOEI), Tirunelveli, India, pp. 1245-1250, 2018.