Recognizing Multi-Intent Commands of the Virtual Assistant with Low-Resource Languages

Van-Vinh Nguyen[1], Ha Nguyen-Tien[2]*, Anh-Quan Nguyen-Duc[3], Trung-Kien Vu[4], Cong Pham-Chi[5], Minh-Hieu Pham[6]
Department of Computer Science, VNU-University of Engineering and Technology, Cau Giay, Hanoi 11300[1]
Faculty of Engineering Technology, Hung Vuong University, Viet Tri, Phu Tho 291930[2]
VNU-University of Engineering and Technology, Cau Giay, Hanoi 11300[3,4]
Vietnam Research Institute of Electronics, Informatics and Automation, Ba Dinh, Hanoi 100000[5]
Research Institute of Electronics, Informatics and Automation, Ba Dinh, Hanoi 100000[6]

***Abstract*—Virtual Assistants (VAs) are widely used in many fields. Recently, VAs have been effectively applied in technical drawing tasks, such as in Photoshop and Microsoft Word. Understanding multi-intent commands in VAs poses a significant challenge, especially when the language in query is low-resource, like Vietnamese (no training dataset available for technical drawing domain), which features complex grammar and a limited domain of usage. In this work, we proposing a three-step process to develop a voice assistant capable of understanding multi-intent commands in VAs for low-resource languages, particularly in responding to the SCADA Framework (SF) for performing drawing tasks: (1) for the training dataset, we developed a semi-automatic method for building a labeled command corpus; applying this method to Vietnamese, we built a corpus that includes 3,240 labeled commands; (2) for the multi-intent command processing phase, we introduced a method for splitting multi-intent commands into single-intent commands to enable VAs to perform them more efficiently. By experimenting with the proposed method in Vietnamese, we developed a VA that supports drawing on SF with an accuracy of over 96%. With the results of this study, we can completely apply them to SCADA system products to support the automatic control of techinical drawing operations in them as VAs.**

Keywords*—Vietnamese command corpus; chatbot; virtual assistants; multi-intent command; artificial intelligence; technical drawing; SCADA framework; build semi-automatic data; low-resource languages

I. Introduction

Nowadays, virtual assistants (VAs) are widely used in daily life, and the number of people using VAs has significantly increased [11] for a few decades. This is because of its ability to support organizations and individuals in doing their tasks in various aspects and scopes, such as Google Assistant, Apple's Siri, Samsung's Bixby and Microsoft's Cortana. They can operate on mobile phones and use sensors of the device to better react to specific contextual information. Besides, VAs such as Amazon Echo and Google Home can operate in smart homes and help users fulfill various daily tasks [14].

Thanks to the rapid development of AI and information technology areas, VAs will soon become increasingly intelligent [12]. VAs have come a long way. In the past, VA only focused on helping functions in textual form, but personal assistants on mobile phones can now process natural language and react in a human-like way. One challenge of the VA problem is to develop their drawing capabilities, which means that the user interacts with the VA to draw pictures in natural language. This problem is difficult when implemented on rich-resource languages because: (1) technical drawing field usually has no training data and how to build a dataset that fully covers common commands in the technical drawing domain; (2) command utterances are often abbreviated and have incorrect grammar, spoken. It is even more difficult when implemented on languages with low resources and complex grammar, such as Vietnamese.

In this work, we focus on solving the VA problem in two aspects: the first studying how to build a high-quality training data. The second is to efficiently handle multi-intent commands in low-resource languages.

There are two approaches for solving the VA problem: using Large Language Modles (LLMs) and the traditional method (such as the Rasa* platform). We chose Rasa platform for our proposed method because: (1) it could be controlling and explaining when using the Rasa framework, this is an important feature in technical drawing; (2) RASA has the advantage of execution speed, so it is suitable for limited hardware platforms, such as deploying on CPUs (but LLMs need run on GPUs).

Our contribution is as follows:

1) Proposing a three-step process to develop a voice assistant capable of understanding multi-intent commands in the field of technical drawing.
2) Proposing the method of semi-automatic data building and building the labeled Vietnamese command corpus, which includes 3,240 commands. This corpus is shared with the community.†
3) Proposing the method for splitting a multi-intent command into single-intent commands so that the VA can perform them more efficiently.

To the best of our knowledge, this is the first comprehensive study of understanding intent-command

*Corresponding authors.

*https://rasa.com/

†https://github.com/HaHVU/VAVietnam

for the drawing virtual assistant with Vietnamese language.

The remainder of this paper is structured as follows: Section II reviews related works on voice assistant architectures and their advancements. Section III presents our proposed method for addressing challenges in low-resource language VAs. Section IV details our experimental setup, datasets, and results, highlighting the effectiveness of our approach. Finally, Section V concludes the paper and outlines future research directions.

II. Related Works

VAs have become increasingly popular and have been widely integrated into our daily lives. They respond to voice commands and offer various functionalities, like scheduling appointments, controlling smart home devices, and performing web searches. Most VAs follow a common architecture. Firstly, Speech Recognition module converts spoken commands to text. Natural Language Understanding (NLU) module extracts necessary information from the text. After that, Dialog management (DM) module determines the response action based on the the information obtained from previous steps. Finally, Natural Language Generation (NLG) module formulates a response [9]. In recent years, the design of VAs has attracted the attention of many researchers.

Matthew B. et al. [10] have introduced to readers the concept of voice assistants and their growing presence in daily life. It aims to provide a basic understanding of these virtual helpers, including:

- What they are: Software agents that can understand spoken language and respond through synthesized voices.
- How they work: Briefly explain the technology behind speech recognition and response generation.
- What they can do: Common functionalities like setting alarms, playing music, controlling smart home devices, and answering questions.

By introducing these key points, the work ideally empowers readers with foundational knowledge of voice assistants and their potential applications.

Timo Strohmann et al. [26] provided guidelines for designing in-vehicle VAs that offer a clear and structured overview of what designers have to consider when designing an in-vehicle VA for a convincing user experience. They designed guidelines based on the existing literature on the requirements of assistant systems and on the interviewing results of experts. In order to demonstrate the applicability of the guidelines, they developed a virtual reality prototype that considered the design guidelines. In a user experience test with 19 participants, they found that the prototype was easy to use, allowed good interaction, and increased the users' overall comfort.

Anxo Pérez et al. [18] presented the design of an assistant that is developed with open-source and widely used components. They proposed an end-to-end process, from information gathering and processing to visual and speech-based interaction.

Marco Brambilla et al. [5] proposed a VA that allows model building using voice commands. They describe three alternative strategies that apply voice-based assistance at three levels: a fully guided strategy; a template-based strategy; and an element-based strategy to demonstrate the generality of the approach. They describe their implementation experience with developing a design assistant that incorporates the three strategies described above for OMG's IFML (Interaction Flow Modeling Language) in the context of user interaction design, including integration with the Amazon and Alexa VA.

Sanju Ahuja et al. [1] made arguments in which designers and policymakers need to be aware of the ethical side of the future of VAs and systematic frameworks are required to aid their moral imagination. They proposed a framework that helps designers imagine potential ethical concerns pertaining to users' autonomy. They demonstrated the usefulness of the framework that they proposed by showing how existing ethical concerns can be situated within the framework. They also used the framework to imagine ethical concerns with emerging VA technologies. This framework can aid in the systematic identification of autonomy-related ethical concerns within human-computer interactions.

Piñeiro-Martín et al. [17] presented an extension of their previous work. They analyze the current regulatory framework for AI-based VAs in Europe and delve into ethical issues, examining the potential benefits and drawbacks of integrating large language models (LLMs) into VAs. Based on the analysis, their paper argues that the development and use of VAs powered by LLMs should be guided by a set of ethical principles that prioritize transparency, fairness, and harm prevention. It presents specific guidelines for the ethical use and development of this technology, including recommendations for data privacy, bias mitigation, and user control. By implementing these guidelines, the potential benefits of visual assistants powered by LLMs can be fully explored while minimizing the risks of harm and ensuring that ethical considerations are at the forefront of the development process.

For the technical drawing field: Dries Van Daele et al. [29] presented a software tool that is able to interpret different parts of a drawing and translate this information to allow automated reasoning and machine learning on a huge database of technical drawings. To achieve that, the proposed the method that automatically learns a parser capable of interpreting technical drawings with Using limited interaction from the expert. Their method uses both neural networking and symbolic methods. Neural network methods to interpret visual images and recognize parts of two-dimensional drawings. Symbolic methods to process relational structures and understand data encapsulated in complex tables contained in technical drawings.

Rodrigo Pereira et al. [15] systematically reviewed the applications of VAs in the context of Industry I4.0, discussed the design principles of technical assistants, and identified the characteristics, services, and limitations associated with the use of VAs in production environments.They found that Virtual Assistants offer Physical and Virtual Assistance. Virtual Assistance provides real-time contextualized information mainly for support, while Physical Assistance is oriented toward task execution. In terms of services, applications include integration with legacy systems and static information processing. Limitations of the application include concerns about information security and adaptation to noisy and unstable environments. They argue that the future should focus on expanding the scope of research to provide more significant conclusions and research capabilities with new AI models and services

Published works have focused on reviewing and addressing various aspects of the VA problem to make VAs more user-friendly and functional. However, these approaches face significant limitations when applied to low-resource languages due to the lack of training data. Additionally, little attention has been given to handling multi-intent commands, and studies on VAs in the field of technical drawing are particularly scarce. This paper seeks to address these gaps by tackling the challenges of limited training data, enabling VAs to perform technical drawing tasks, and facilitating interaction with users in low-resource languages. Notably, this is the first study to explore multi-intent command recognition for technical drawing in Vietnamese.

III.Our Proposed Method

The challenge of the VA problem in low-resource languages is the lack of training data and an efficient solution in handling multi-intent commands. To overcome these challenges, we propose a three-step process, as follows:

- Phase 1: **Build semi-automatic data** for training and testing.
- Phase 2: **Choose the good model for training VA**. This phase will make a VA that takes single-intent commands from the user in the input and outputs the JSON file that contains information about their requirements.
- Phase 3: **Generate the JSON file**. In this phase, the multi-intent command from the user is split into single-intent commands before being fed into the VA to get a JSON file and send it to SF to do the drawing task.

Our proposed method is illustrated in Fig. 1.

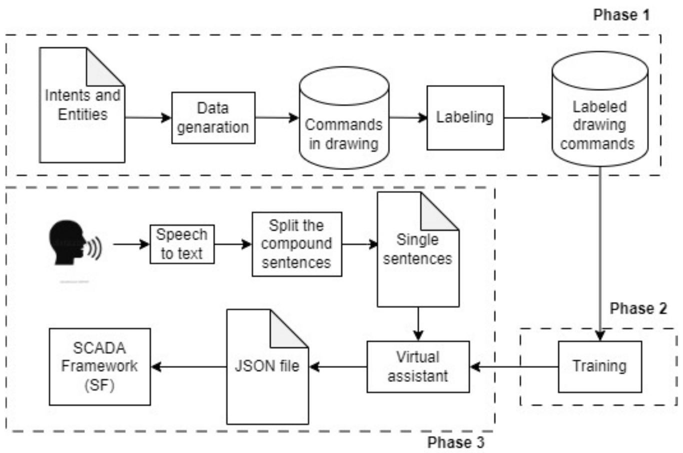


Fig. 1. Our proposed method

A. Build Semi-automatic Data

Being built specifically for providing SF users with technical support, the VA is designed to have two main functions. The first function, called automatic answering, is the ability of a VA to answer the questions given by the user. The questions that are within the domain of this function are related to SF, particularly about the instruction manual, the main functions of the software, and frequent bugs and difficulties that users may encounter when coding in the script and interacting with the working interface of the software. The second main function of the VA is user-task assistance, in which the assistant does some tasks directly in the software working interface when asked by the user. The supported tasks are tasks that most users usually do at the interface of the software.

To have a dataset for training this VA. The first thing we do is design and define intent and entity labels that are presented in Section III-A1. then generate types of commands, and finally use the built label set to label them. The last two processes are presented in Section III-A2.

1) Designing intent and entity labels

Firstly, we have to define the intent and entity label set that match the domain of the SF. The intents are used to define the overall requirement in a user command. The dataset is split into 3 groups. The first intent group contains intents in which the user wants to choose objects in the software interface. The second intent group is involved in changing attributes of objects, and the last one is about drawing objects directly on the SF interface. The intent and entity label list for Vietnamese is shown in Tables X and XI in Appendix.

2) Semi-automatic data construction

a) Manual Data Construction

Initially, raw data generation and annotation are carried out manually by humans. Particularly, data builders role-play as platform users and give orders indicating the content of a specific intent. Each intent requires one or several specific pieces of information to appear in the command, and these pieces of information are dedicated in various ways and appear in different orders in different commands.The semi-automatic data construction method requires a few data to initialize, so we will build them manually. The more and better the quality of the initial data, the higher the efficiency of the method achieved. For Vietnamese, we have built 540 commands in total manually.

After it is annotated following the format of Rasa chatbot framework [20]. Rasa is an open-sourced framework used to build conversational chatbots. One of its advantages over other chatbot frameworks is its various number of integrated machine learning and deep learning models used to solve intent classification (IC) and entity extraction (EE) which are the two main tasks of a VA. That makes building an efficient VA much faster than building from scratch.

In the Rasa framework, the data is stored in a .yml file and the commands are grouped by intent. In a command, each labeled phrase is included in a couple of square brackets, and the entity label used to label that phrase is included in the couple of braces that are next to the square bracket couple.

Semi-automatic data construction methods require some data to train the model, so we have to build them manually. The more and better quality data, the higher the efficiency of the method.

b) Semi-automatic data construction

In this process, we propose to use an NLU model trained on the manual-built data as a core component of the semi-automatic data construction process. Our proposed method is illustrated in the Fig. 2.

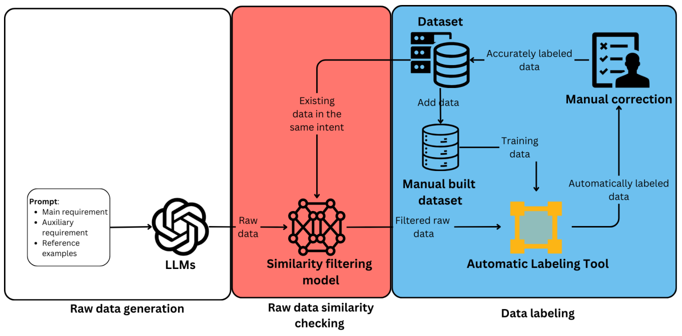


Fig. 2. Data building pipeline.

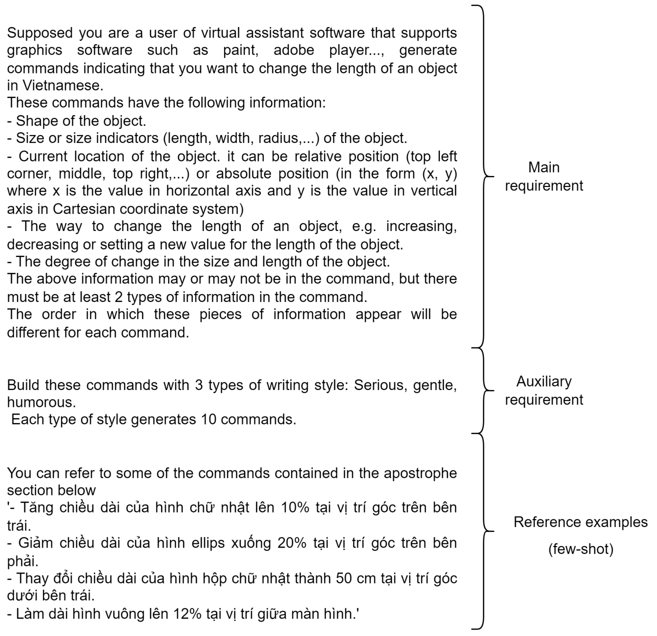
The idea of our proposed method is as follows: Firstly, we design prompts that are put into LLMs to generate the raw commands. Then, in these commands, we remove ones that have cosine similarity to existing commands in the dataset greater than the α threshold. Next, the remaining commands from the previous step are put into the NLU model (which is trained on manual building data) to get the labeled data. Finally, we use instructed annotators to manually check and correct the automatically labeled data. The dataset obtained after the last step is the expected high-quality dataset. It is also combined with existing data to retrain the NLU model in order to make that model better at labeling raw data.

Raw data generation: In this stage, the data samples built in the manual data construction process will be used as reference examples to guide Large Language Models (LLMs) to generate high-quality raw commands for a specific intent in larger quantity and faster speed compared to manual data construction. The guidance includes three main parts:

- Description of the main requirements of the task indicated by intent.
- Description of auxiliary requirements.
- Reference examples (few-shot).

Fig. 3 shows an example of guidance for raw data generation sent to LLM.

In the example guidance in Fig. 3, the description of the main requirement includes the content of the intent in which the user wants to do and the types of information that should appear in the user's command. Furthermore, the last two sentences in this section also require LLMs to generate more diverse and realistic commands because, in reality, users of popular VAs barely provide the assistant with all of the information needed for completing the task. The description of auxiliary requirements in the guidance is mainly about the quantity and writing style of data, which makes the generated data more diverse and the generating process faster. Lastly, the reference examples contain commands created manually in the previous process. These commands are supposed to teach LLMs such things as:


Supposed you are a user of virtual assistant software that supports graphics software such as paint, adobe player..., generate commands indicating that you want to change the length of an object in Vietnamese.
These commands have the following information:
- Shape of the object.
- Size or size indicators (length, width, radius,...) of the object.
- Current location of the object. it can be relative position (top left corner, middle, top right,...) or absolute position (in the form (x, y) where x is the value in horizontal axis and y is the value in vertical axis in Cartesian coordinate system)
- The way to change the length of an object, e.g. increasing, decreasing or setting a new value for the length of the object.
- The degree of change in the size and length of the object.
The above information may or may not be in the command, but there must be at least 2 types of information in the command.
The order in which these pieces of information appear will be different for each command.

Main requirement

Build these commands with 3 types of writing style: Serious, gentle, humorous.
 Each type of style generates 10 commands.

Auxiliary requirement

You can refer to some of the commands contained in the apostrophe section below
'- Tăng chiều dài của hình chữ nhật lên 10% tại vị trí góc trên bên trái.
- Giảm chiều dài của hình ellips xuống 20% tại vị trí góc trên bên phải.
- Thay đổi chiều dài của hình hộp chữ nhật thành 50 cm tại vị trí góc dưới bên trái.
- Làm dài hình vuông lên 12% tại vị trí giữa màn hình.'

Reference examples
(few-shot)

Fig. 3. Example of raw data generation for intent "change_length" indicating that user wants to change the length of an object.

- The way information appears in the command.
- The order in which information appears in the command. This is necessary because if only 1 or a few commands are given, LLMs have the tendency to generate commands with the same order of information, similar to fixed words or phrases filled in an available pattern.

Raw data similarity checking: As the dataset grows, new raw data generated by LLMs is likely to have great similarity with the available one in the dataset. If this kind of raw data is added to the dataset, it will make the dataset reduce diversity and coverage, thus making the training process less effective. To prevent this phenomenon, each new command (called new raw command set) generated by LLMs for one specific intent is compared with commands that already exist in the dataset of the same intent (called old raw command set) in terms of consine similarity [6]. Particularly, after the new raw command set and the old raw command set are encoded into vectors using pre-trained models specialized in sentence encoding, these two vector sets are used to compute the cosine similarity matrix by Eq. (1) shown below.

$$\text{cosine\_sim}_{ij} = \frac{M_i \cdot N_j}{\|M_i\| \|N_j\|} \tag{1}$$

Where cosine_sim_{ij} is the cosine similarity value between i^{th} command in the new raw command set and j^{th} command in the old raw command set. M_i and N_j are vectors obtained by encoding the i^{th} and j^{th} commands in the new and old raw command sets, respectively.

Finally, the commands in the new set will be removed if their similarity with any commands in the old set is over a threshold.

Data labeling: The NLU model trained on manual-built data is integrated into Label-Studio, which is an automated data labeling tool, and its predicted results for raw commands are displayed and corrected in this tool. This is a robust and powerful open-sourced labeling tool that supports both manual and automatic labeling processes. Built on the Web-UI platform, this tool provides flexibility and convenience for users, allowing them to customize and personalize their experience.

However, due to the nature of the dataset, which is used to train a model to solve two tasks, which are IC and EE simultaneously, while the labeling tool only supports labeling for datasets used for one task, the raw data needs to be preprocessed before being fed into the integrated NLU model of the automatic labeling tool. Particularly, all raw commands are normalized by Underthesea[‡] and added a special token "¡i¿" at the beginning. This token is used to store the intent of the command and is labeled with a special entity label that has the following format: "intent—¡intent_name¿". Other normal entity labels have the following format: "¡group name¿—¡entity name¿" and are used to label phases and words indicating the information used to complete the task in the user's command.

After being labeled automatically by the NLU model, the data is checked and corrected manually. The accurately checked data is then stored in the overall dataset. This dataset, after a specific time of building, is used to retrain the NLU model to make it better at labeling data.

Applying this method to Vietnamese, we have built a dataset of 2700 samples in total. This dataset is combined with the manually built dataset to create the overall dataset that is used to train and evaluate the VA.

B. Train the VA

In our proposed method, the mission of a VA is to extract valuable information to complete the required task given the user command. To obtain this information, our VA is modeled to solve two tasks simultaneously. The first task is IC which is used to identify the task required by the user. The second task is EE which is used to extract valuable information to complete the required task in the user's command. The existing state-of-the-art (SOTA) models of the two tasks are built from variants of Transformers [27] which are pretrained on a unified language dataset (BERT, GPT, ...). Although making SOTA results in a variety of NLU datasets, most of them have some disadvantages. Firstly, they need high training and computation costs to produce good results. Furthermore, due to their large size and long inference time, it is very difficult to deploy them on various platforms. Therefore, choosing a suitable model for each language is important.

For Vietnamese, we suggest to use the DIETClassifier [4] with components pretrained on Vietnamese datasets in training VA. In the paper publishing this model, it is shown to overcome SOTA models on varius NLU datasets, including NLU-Benchmark [13], ATIS [2] and SNIP [24] while being about six times faster to train. We also have used phoBERT [16] which is a language model pretrained on Vietnamese as the dense featurizer of DIETClassifier. The reason we choose this model is because it is the first public large-scaled monolingual language model pretrained for Vietnamese and has archived SOTA results in many Vietnamese-specific NLP tasks. Its performance when integrated into the architecture of the DIETClassifier model on our test data is shown in Tables IV and V in the Section IV-C. In the training process, the weight of the dense featurizer is frozen, which makes it much faster to train the model.

[‡]https://pypi.org/project/underthesea/

C. Generate the JSON File for Foundation Software

In this phase, the commands of the user are recorded and saved as an audio file. Then, it is converted to a text file. Next, the command in text that is a multi-intent command is split into single-intent commands based on our proposed method. Next, we feed single-intent commands into the VA to get information about the required task, which is then post-processed to store in a JSON file. Finally, the JSON file is sent to the SF to do the drawing requests of the user.

1) Convert speech to text

To convert the voice input of the user into text, we suggest using Whisper [30]. This is a speech-to-text multilingual model. In practical usage, it has proven to be suitable for the functional requirements of the VA. Particularly, it can catch the voice input fractions that represent numbers and convert them into Arabic numbers instead of plain text, which makes the transcribed text more similar to the normal text inputs of the user.

2) Split multi-intent commands

In practical usage, users often give commands that contain multiple tasks (compound command), also known as multi-intent or complex commands, to VAs in order that the whole work process is completed quickly. For users, giving commands for single tasks and waiting for them to finish are inconvenient and time-consuming. Attempting to build VAs capable of handling complex commands by directly training the assistant on labeled complex commands is a simple and straightforward approach. The author in [21] proposed the first work to handle multi-intent commands, which has hierarchical structures to identify multiple intents in the user's command. In 2020, [19] proposed a model named AGIF which uses an adaptive graph attention network to model joint intent-model interaction. However, these existing works have some drawbacks. Firstly, the data collection cost for complex commands is very high because of the uncontrollable variety of this kind of command, especially in the case of a large number of single intents. Particularly, with a dataset of n single intents, the number of bi-intents (a mutli-intent containing 2 single intents) is $\frac{n*(n-1)}{2}$ and the number of tri-intents (a multi-intent containing three single intents) is also very big, which is the polynomial value of n, making collecting data for all combinations of intents nearly impossible. Furthermore, the labeling process for multi-intent commands is also very challenging. To effectively annotate entities of a multi-intent command, the entity labels have to indicate both the role of labeled phrases and which intent in the command that phrases belong to, instead of just indicating the role of the phrases like in normal single-intent commands. For example, in the multi-intent command "Draw a yellow rectangle at the center of the screen and move the black square to the left" which has two intents named draw_square and move_left, the word "yellow" has to be labeled by a label indicating the color of an object of the first intent. Lastly, this approach potentially reduces the effectiveness of modules of VAs when they have to extract too much information (multiple intents and entities aligned to each intent) in a single command.

Another approach for handling multi-intent commands used in recent studies is to build a module specified for detecting and splitting a multi-intent command into a list of single-intent commands for a VA and feed these commands into the remaining modules. This approach overcomes the limitations of the previous approach when it does not need to label multi-intent commands. The author in [28] have proposed DialogUSR, a module built as a sequence-to-sequence model so that from a multi-intent command, it can generate a list of single-intent commands in the form of a text string with each single-intent commands separated by a special token "¡SEP¿". One year later, [25] proposed a module built as a NER model with entities used to bound the single-intent commands in a multi-intent command. However, the two studies above have their own limits. Firstly, the module proposed by [28] heavily relies on training data, thus poorly performs when confronted with inputs that are highly different from training data, i.e. it rarely can split a triple-intent command accurately when being trained only with single and bi-intent commands. On the other hand, the SPM module of [25] is likely to split commands with more intents than ones in the training data accurately because of its building strategy to model multi-intent command splitting as a NER task to bound single-intent commands. Nonetheless, this modeling approach absolutely cannot split a multi-intent command whose information of each intent is interleaved due to its modeling strategy. For example, the commands "Draw a square and a circle whose sizes are 50x50 pixels and 60x60 pixels, respectively, in random positions" has two intents named "draw_square" and "draw_cirle" but the information of each intent is mentioned separately throughout the command instead of standing next to each other.
To overcome these limitations, we proposed to build a module specified for splitting these multi-intent commands into sequences of single-intent commands based on in-context learning using LLMs with a context database of multi-intent commands built from single-intent commands.

Our proposed approach has three main processes:

- Context database construction: This process involves using LLMs to merge single-intent commands chosen from the previously built training data into a multi-intent command.
- Context retrieval model construction: In this process, we build a model that is able to choose contexts that are semantically similar to the user's input from the context database.
- LLMs input construction: This process involves building a complete input from the user's input and contexts chosen by the context retrieval model.

The overall architecture of the multi-intent command splitting module is illustrated in Fig. 4.

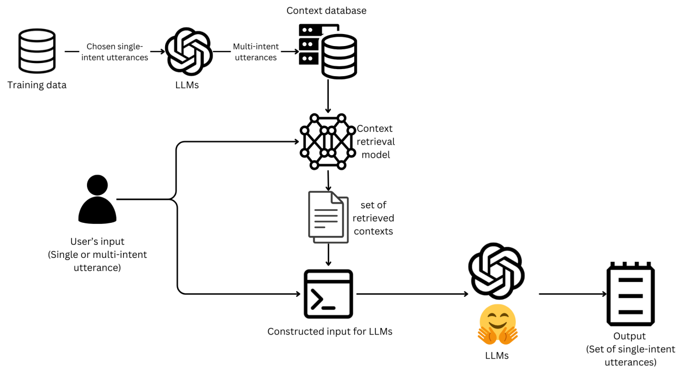
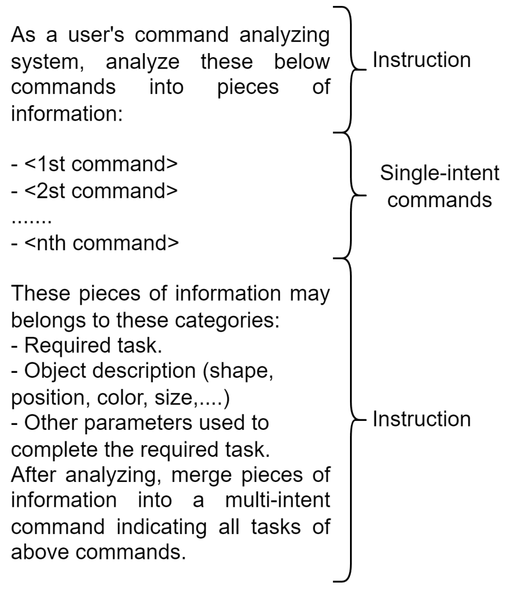

Context database
Training data
Chosen single-intent utterances
LLMs
Multi-intent utterances
Context retrieval model
set of retrieved contexts
User's input (Single or multi-intent utterance)
Constructed input for LLMs
LLMs
Output (Set of single-intent utterances)

Fig. 4. Overall architecture of the multi-intent utterance splitting module.

a) Context database construction

To build the context dataset, two single-intent commands with different intents are chosen from the VA training dataset and then fed into LLMs to construct a multi-intent command. Along with single-intent commands, a merging instruction snippet is also added into the input of the LLMs to make good multi-intent commands. The example input built for LLMs to create multi-intent commands is shown in Fig. 5.



As a user's command analyzing system, analyze these below commands into pieces of information: — Instruction

- <1st command>
- <2st command>
....
- <nth command>
— Single-intent commands

These pieces of information may belongs to these categories:
- Required task.
- Object description (shape, position, color, size,....)
- Other parameters used to complete the required task.
After analyzing, merge pieces of information into a multi-intent command indicating all tasks of above commands. — Instruction

Fig. 5. Example guidance of input for LLMs to create multi-intent commands.

As Fig. 5 shows, it can be seen that the LLMs are required to merge pieces of information extracted from single-intent commands to create multi-intent commands instead of just concatenating single-intent commands by linking words or punctuation. That can make resulting multi-intent commands more diverse and complicated to segment with pieces of information being interleaved, or even become tricky. For example, from the above input and two single-intent commands "Choose all objects on the screen" with intent "select_all_objects" and "Give up choosing all cirles" with intent "exit_select", the LLMs can generate a multi-intent command "Choose all objects on the screen, except for circles". At first glance, this multi-intent command has only one intent, but to complete the required task, the visual assistant has to select all objects, then give up selecting all circles. However, this kind of input can make LLMs potentially align information incorrectly, or even get rid of some of the information of single-intent utterances. Furthermore, due to the lack of reference examples of inputs and expected outputs, the output in reality of LLMs is usually unstable, making post-processing challenging.

To overcome these drawbacks, some reference examples are added to the input for LLM to stabilize its inference process. Two types of reference examples we used are basic examples (normal few-shot) and advanced examples (Chain-of-thought few-shot, also known as CoT few-shot) which are listed in Fig. 6 and 7, respectively.

Input:
- Draw a black circle at coordinates (20, 40).
- Paint the red for its background
Output: Draw a black circle at coordinates (20, 40) then paint the red for its background

Input:
- Draw a square whose size is 40 x 40 pixel square at the center of the screen.
- Draw a circle whose size is 40 x 40 pixel square at the top left corner of the screen.
Output: Draw a square and a circle with the same size of 40 x 40 pixel square at the center and the top left corner of screen, respectively.

Input:
- Draw an arrow whose length and width are 70 and 30 pixel respectively the the left side of the screen.
- Make a copy
- Paste the copy in the center of the screen.
Output: Draw an arrow whose length and width are 70 and 30 pixel respectively the the left side of the screen, make a copy then paste it in the center of the screen.

Fig. 6. Basic examples added into input of LLM as references to make its inference process more stable.

Input:
- Draw a black circle at coordinates (20, 40).
- Paint the red for the background of the black circle at coordinates (20, 40).
Output:
Analysis:
- Sentence 1: Task: Draw object, object shape: circle, object color: black, object position: (20, 40).
- Sentence 2: Task: Color background, object shape: circle, object color: black, object position: (20, 40).
Merge and create new command: Draw a black circle at coordinates (20, 40) then paint the red for its background.

Input:
- Draw a square whose size is 40 x 40 pixels at the center of the screen.
- Draw a circle whose size is 40 x 40 pixels square at the top left corner of the screen.
Output:
Analysis:
- Sentence 1: Task: Draw object, object shape: square, object size: 40 x 40, object position: center of the screen.
- Sentence 2: Task: Draw object, object shape: circle, object size: 40 x 40, object position: top left corner of the screen.
Merge and create new command: Draw a square and a circle with the same size of 40 x 40 pixels at the center and the top left corner of screen, respectively.

Input:
- Draw an arrow whose length and width are 70 and 30 pixel respectively on the the left side of the screen.
- Make a copy
- Paste the copy in the center of the screen.
Output:
Analysis:
- Sentence 1: Task: Draw object, object shape: arrow, object size: 70 pixel long and 30 pixel wide, object position: left side of the screen.
- Sentence 2: Task: Copy object.
- Sentence 3: Task: Paste, position: center of the screen.
Merge and create new command: Draw an arrow whose length and width are 70 and 30 pixel respectively on the left side of the screen, make a copy then paste it in the center of the screen.

Fig. 7. Advanced examples added into input of LLM as references to make its inference process more stable.

While basic examples just contain input (a list of single-intent utterances) and corresponding output (multi-intent utterance), advanced examples contain interpretation for the output, apart from input and output. With the two types of examples, the resulting multi-intent commands generated by LLMs are more stable and can contain all the information of single-intent commands, but seem to be monotonic in term of information presentation. As a result, we have used all three types of inputs to build context database in the experiment. Particularly, the basic input with no example is called "zero-shot merging", the input with simple examples is called "few-shot merging" and the input with analyzing examples is called "few-shot CoT merging".

In the work, we build multi-intent commands by combining two single-intent commands with different intents. Furthermore, five single-intent commands of each intent are also randomly chosen from training data in order that module can detect single-intent command.

Context retrieval model construction

The target of context retrieval is to provide LLMs with informative examples to refer to, thus allowing LLMs to process users's input accurately and give the correct output for other modules of VAs. As a result, retrieved contexts need to be greatly similar to the user's input. To be able to retrieve these contexts, the model is designed to solve the Semantic Textual Similarity (STS) task. The input of this task contains two text sequences, and the output is the similarity score between the two sequences. The model instruction process consists of two parts: STS dataset building and model training on the STS dataset.

STS dataset building: To build a STS dataset, the two inputs of each sample in this dataset are chosen arbitrarily from the previously built context dataset, and the output (similarity score of two input sentences) is identified by the intents included in each input sentence. The equation used to compute the similarity score is below:

$$\text{sim\_score} = \begin{cases} 1.0 & \text{if} \quad a_1 = b_1 \text{ and } a_2 = b_2 \\ 0.75 & \text{if} \quad a_1 = b_2 \text{ and } a_2 = b_1 \\ 0.5 & \text{if} \quad \begin{matrix} a_1 = b_1 \text{ and } a_2 \neq b_2 \text{ or} \\ a_1 \neq b_1 \text{ and } a_2 = b_2 \end{matrix} \\ 0.25 & \text{if} \quad \begin{matrix} a_1 = b_2 \text{ and } a_2 \neq b_1 \text{ or} \\ a_1 \neq b_2 \text{ and } a_2 = b_1 \end{matrix} \\ 0.0 & \text{otherwise} \end{cases} \tag{2}$$

Where a_1, a_2 are the first and the second intent of the first multi-intent command and and b_1 and b_2 are the first and the second intent of the second multi-intent command. This equation is applied in the case that both input sentences are bi-intent commands, which make up the majority of the context dataset. In the case that both input sentences are single-intent commands, the assigned similarity value is 1.0 if the intents of both sentences are the same and 0.0 in the other case. If one of the input sentences is single-intent and the other is bi-intent, the assigned value is 0.25 if the intent of the single-intent command is the same as one of the intents of the bi-intent command and 0.0 in other cases. The reason why the maximum similarity value between two sentences with different numbers of intents (1 intent compared to 2 intents) is just 0.25 is that in some cases, multi-intent commands and single-intent commands can be greatly similar to each other. e.g. "Let's move the red square to the left for 10 pixels, then flip it horizontally" and "Let's move the red square to the left for 10 pixels." If the user's input is the same as the example multi-intent command and the result of the retrieval model is the same as the example single-intent sentence, the module is likely to segment the user's input incorrectly. As a result, by setting a low similarity value for couples of sentences with different numbers of intents, especially between single-intent commands and multi-intent commands, being trained on that dataset can allow the retrieval model to be capable of clearly discriminating between single-intent and multi-intent commands.

Context retrieval model building and training: The module trained on the previously built STS dataset is a variant of the sentence-BERT [23]. In this model, we use MEAN$_{pooling}$ pooling layer to get the sentence vector from the hidden state matrix encoded from the input sentence by BERT model. The loss function used to optimize the retrieval model is Mean Square Error (MSE) loss.

LLMs input construction: To build a complete input for LLMs, contexts with great similarity with the user's input need to be retrieved. To get these contexts, all contexts in the context dataset are encoded by the context retrieval model, which has been trained in the previous stage to get their semantic representative vectors. This collection of vectors, along with raw contexts, is then stored in a database. The vector obtained by encoding the user's input is used to compute the similarity value between it and other contexts to get the most suitable context.

3) *Generate the JSON file for SF*

After the multi-intent command handling module separates the user's command into a list of single-intent commands, that list of commands is fed into the VA one by one. After each command is processed, the intent and entity values extracted from that command are obtained and postprocessed. After post-processing, intent and entity values are filled into a pre-defined form whose keys are intent and entity names. For entity names with no corresponding entity values extracted from the user's utterance, their value is assigned $NULL$ value. Finally, this form is stored in a JSON file and sent to the SF by calling its API. The Fig. 8 shows the content of the JSON file obtained from the Vietnamese user's input, which is a multi-intent command, and how the required tasks are done directly on the user interface of the SF.

IV. EXPERIMENTS

In this section, we conduct the experiment of our proposed method on Vietnamese, which is a low-resource language. The first describes the training and testing datasets for VA, then presents the configuration settings and evaluation metrics. Next, we show the archived results, and finally, we present some discussions related to the results of the experiment.

A. *Experimental Datasets*

1) *Experiment datasets for evaluating the quality of semi-automatically built data and the performance of VA*

Our dataset includes 3240 labeled commands (or 3240 samples). It is made up of a manually built dataset of 540 samples and a semi-automatically built dataset of 2700 samples. We split this dataset into two subsets: the training dataset and the testing dataset.

a) *Testing dataset*

It is denoted **NLU**$_{test}$ including 360 samples that are randomly taken from the manually built dataset, such that 10 samples of each intent type.

b) *Training dataset*

It includes 2880 remaining samples (denote **NLU**$_{train}$) divided into six different training datasets. Dataset are denoted as **NLU**$_n$ means we take the n first samples of each intent type in **NLU**$_{train}$ for making it. Statistics of these datasets are shown in Table I.

TABLE I. STATISTICS OF THE COMMAND NUMBERS IN EACH DATASET

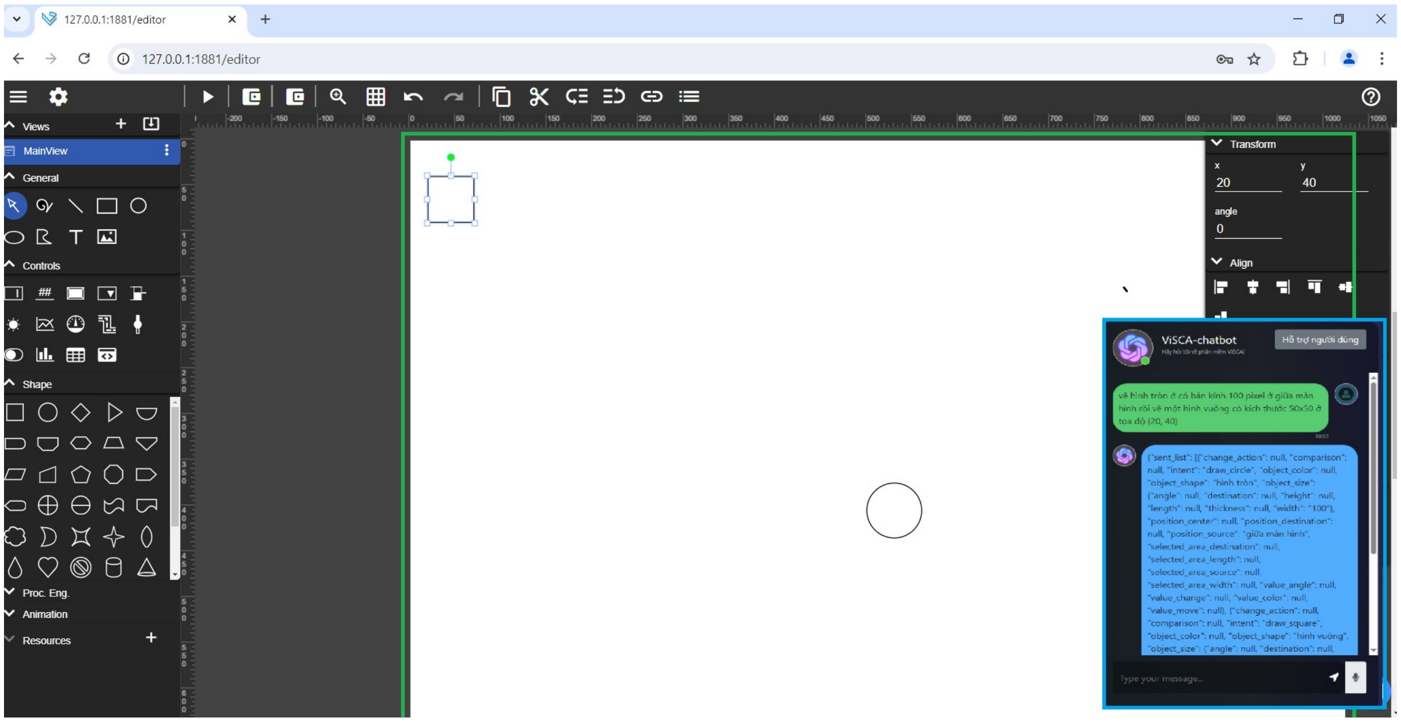
Datasets	**Number of samples**
NLU$_{test}$	360
NLU$_{40}$	1440
NLU$_{50}$	1800
NLU$_{60}$	2160
NLU$_{70}$	2520
NLU$_{train}$	2880

2) *Experiment datasets for evaluating the performance of multi-intent handling module*

The multiple context datasets are made by choosing data from **NLU**$_{train}$ dataset by different context-building methods. Particularly, the context datasets made by Zero-Shot Merging, Few-Shot Merging and Few-Shot CoT Merging methods are called **Context**$_{zero-shot}$, **Context**$_{few-shot}$ and **Context**$_{few-shot\,CoT}$ respectively. Each of these context datasets contains 2696 samples, including 180 single-intent commands and 2516 multi-intent commands.

a) *Testing dataset*

Our context testing dataset, called **Context**$_{test}$, contains 1754 samples, including 1662 multi-intent commands and 92 single-intent commands. It is built based on all three merging methods with single-intent commands taken from **NLU**$_{test}$ dataset. After that, all incorrect and duplicated samples are removed.



vẽ hình tròn ở có bán kính 100 pixel ở giữa màn hình rồi vẽ một hình vuông có kích thước 50x50 ở tọa độ (20, 40)

10:53

{"sent_list": [{"change_action": null, "comparison": null, "intent": "draw_circle", "object_color": null, "object_shape": "hình tròn", "object_size": {"angle": null, "destination": null, "height": null, "length": null, "thickness": null, "width": "100"}, "position_center": null, "position_destination": null, "position_source": "giữa màn hình", "selected_area_destination": null, "selected_area_length": null, "selected_area_source": null, "selected_area_width": null, "value_angle": null, "value_change": null, "value_color": null, "value_move": null}, {"change_action": null, "comparison": null, "intent": "draw_square", "object_color": null, "object_shape": "hình vuông", "object_size": {"angle": null, "destination": null,

Fig. 8. The user's command which has the English translation text : "draw a circle with radius of 100 pixels at the center of the screen, then draw a square with size of 50 x 50 pixels at the coordinates (20, 40)" and the content of the obtained JSON file after VA handles the command are shown in the box 1 (surrounded by the thick blue border). The result after sending the JSON file to the SF by calling its API are shown in the box 2 (surrounded by the thin green line).

b) *Training dataset*

The STS datasets used to train context retrieval models is built by choosing two samples randomly from the corresponding context datasets. However, if all combinations of two samples are chosen, the STS training datasets will become very large and imbalanced in term of similarity value. Particularly, each of STS training datasets will have over 3,632,860 samples in total, including over three million samples with Intent Label Similarity Value of 0 (ILSV0). As a result, only a specific number of samples with ILSV0 chosen randomly. Particularly, each of the three training datasets contains 770,926 samples in total. Table II shows the statistics of each STS training dataset in terms of Intent Label Similarity Value.

TABLE II. Statistics of STS Training Dataset

Similarity value	Number of samples
1	1618
0.75	2516
0.5	170816
0.25	195976
0.0	400000

Based on three context datasets, namely $\textbf{Context}_{zero-shot}$, $\textbf{Context}_{few-shot}$ and $\textbf{Context}_{zero-shot\,CoT}$, we have built three STS training datasets, namely $\textbf{STS}_{zero-shot}$, $\textbf{STS}_{few-shot}$ and $\textbf{STS}_{few-shot\,CoT}$. Similarly, Sentence-BERT model trained on these datasets is called $\textbf{SBERT}_{zero-shot}$, $\textbf{SBERT}_{few-shot}$ and $\textbf{SBERT}_{few-shot\,CoT}$.

c) *STS Validation dataset*

To evaluate the training process of each Sentence-BERT model, a STS validation dataset, called $\textbf{STS}_{valid}$, is also created based on $\textbf{Context}_{test}$ dataset. Its samples are created by all three merging methods, then filtered manually to get rid of incorrect and duplicated ones. This dataset contains 31,763 samples in total. The below Table III shows the statistics of the $\textbf{STS}_{valid}$ in terms of Intent Label Similarity Value.

TABLE III. Statistics of the STS Validation Dataset

Similarity values	Number of samples
1	638
0.75	1125
0.5	5000
0.25	5000
0.0	20000

B. *Experimental Setup*

1) *Configuration setup*

a) *Hardware configuration*

Both the NLU module and multi-intent handling module are trained on GTX 3090 GPUs (24GB VRAM). The training processes of the two models can take up to 20 GB of VRAM.

b) *Model configuration*

In the VA, we have used DIETClassifier model [4] with some customization on its components. Particularly, we use

Bag-Of-Word (BOW) method with vocabulary built by n-grams whose length is from 1 to 5 characters as the main sparse featurizer for the model.

For dense featurizer, we use two methods. First, we use phoBERT pretrained model [16] to get dense features from input tokens and its vector output from the special token "¡s¿" acts as the sentence feature of the model. The DIETClassifier model using this dense featurizer is called **DIET**$_{pB+BOW}$. The second method, inspired by an existing work of [7], uses the Fasttext model [8] pretrained on the Vietnamese dataset[§] as the main dense featurizer and the sentence feature is computed by averaging all dense features obtained from input tokens. The DIETClassifier model with that dense featurizer is called **DIET**$_{Ft+BOW}$. In the last model configuration, we do not use any dense featurizer, thus is called **DIET**$_{BOW}$. The number of Transformer layers and the number of attention heads in each Transformers layer used in all configurations are two and four, respectively.

Furthemore, in order to evaluate the effectiveness of using Vietnamese pretrained language models as dense featurizers, we have also used a DIETClassifier model with no dense featurizer and compared its performance with the above two DIETClassifier models.

c) Training configuration

In experiments evaluating VA, we have trained DIETClassifier model on training dataset for 100 epochs with batch size of 16 and the learning rate of 0.001. The loss function we used to optimize the model is Cross Entropy and the optimizer used to adjust the learning process is AdamW optimizer.

In experiments evaluating the multi-intent handling module, we have trained Sentence-BERT model on training dataset for 20 epochs with batch size of 16 and the learning rate of 0.25×10^{-4}. The loss function we used to optimize the model is Mean Square Error and the optimizer used to adjust the learning process is AdamW optimizer. Furthermore, to evaluate the contribution of the context retrieval model, we have also used a LLM (called Baseline) with static contexts, containing three single-intent commands and four multi-intent commands, and compared it with LLMs using the above context retrieval model.

2) Evaluation metrics

a) VA evaluation metrics

The two tasks used to evaluate performance of NLU module are IC and EE. Two main metric used in two tasks are Accuracy and F1.

b) Multi-intent handling evaluation metrics

Firstly, to evaluate the training process of context retrieval model when being trained on different STS training datasets, we have used Spearman's rank correlation coefficient. Secondly, in the experiment evaluating the overall performance of multi-intent handling module, we have used various metrics.

Secondly, to evaluate the output single-intent commands of the module, we have concatenated the two lists of output single-intent commands and ground-truth single-intent commands with the special token "¡SEP¿" into two single strings and used BLEU [3] and ROUGE [22]. Furthermore, we have also used 2 metrics named Split Accuracy (SACC) and Exact Match (EM) proposed by [28]. In the original paper, SACC is used to measure the ratio of correct command splitting and computed the following Eq. (3):

$$\text{SACC} = \frac{1}{n} \sum_{1 \le i \le n} \mathbb{I}_{(\text{len}(Q^{(i)}_{pred}) = \text{len}(Q^{(i)}_{ref}))} \tag{3}$$

Where n is the number of samples, $Q^{(i)}_{pred}$ and $Q^{(i)}_{ref}$ are the i^{th} predicted and references single-intent command list, respectively. As for EM, [28] considered the correct result if the predicted command is exactly the same as the reference one:

$$\text{EM} = \frac{\sum_i \sum_j F(Q^{(ij)}_{pred}, Q^{(ij)}_{ref})}{\sum_{1 \le i \le n} len(Q^{(i)}_{ref})} \tag{4}$$

Where $Q^{(ij)}_{pred}$ and $Q^{(ij)}_{ref}$ are the j^{th} predicted single-intent command and ground-truth single-intent command in the i^{th} sample in the evaluation dataset. The function $F(Q^{(ij)}_{pred}, Q^{(ij)}_{ref})$ in the above equation is the indicator function:

$$F(Q^{(ij)}_{pred}, Q^{(ij)}_{ref}) = \mathbb{I}_{(Q^{(ij)}_{pred} = Q^{(ij)}_{ref})} \tag{5}$$

However, applying the metric EM directly in our experiment is not suitable. We have seen that our merging methods only retain the essential information of the merged single-intent commands (considered as ground-truth commands in an evaluation sample) and get rid of their writing style, e.g. formal, informal, or even humorous, in the resulting multi-intent command. As a result, the output single-intent commands that are predicted by the multi-intent handling module are mostly different from the ground-truth, which means the metric EM cannot make an accurate evaluation. As a result, apart from EM metric, we proposed a new metric called proportional match (PM), which modifies the F function in EM. This new metric is used to measure the accuracy in the essential information of predicted single-intent commands. The F function in the metric is computed by Eq. (6):

$$F(Q^{(ij)}_{pred}, Q^{(ij)}_{ref}) = \begin{cases} & \exists k \in D^{(ij)}_{ref} \text{ and} \\ 0 \quad \text{if} & D^{(ij)}_{ref_k} \neq NULL \text{ and} \\ & D^{(ij)}_{ref_k} \neq D^{(ij)}_{pred_k} \\ 1 \quad \text{otherwise} & \end{cases} \tag{6}$$

Where $D^{(ij)}_{pred}$ and $D^{(ij)}_{ref}$ are the dictionaries containing the intents and entity values extracted from $Q^{(ij)}_{pred}$ and $Q^{(ij)}_{ref}$ respectively by the VA (trained on the overall dataset). k is the key in $D^{(ij)}_{pred}$ and D^{ij}_{ref}. The condition that $D^{(ij)}_{ref} \neq NULL$ is set to get rid of abundant information, which appears very common in almost every sample of the evaluation dataset.

[§]https://huggingface.co/facebook/fasttext-vi-vectors

C. Experimental Results

1) VA experimental result

We conducted an evaluation of all three DIETClassifier models that are trained on the same **NLU**$_{40}$ training dataset on the **NLU**$_{test}$ testing dataset. The **DIET**$_{pB+BOW}$ is chosen for our proposed method (called **Our model**). The results are shown in Table IV.

TABLE IV. Result of DIETClassifier Models on **NLU**$_{test}$

Model	IC		EE	
	F1	Acc	F1	Acc
Our model	**0.93**	**0.93**	**0.96**	**0.98**
DIET$_{Ft+BOW}$	0.92	0.92	0.96	0.98
DIET$_{BOW}$	0.92	0.92	0.95	0.97

Table IV shows our model achieved the best performance for both IC and EE tasks on the NLU$_{test}$.

In order to know the quality of the our semi-automatically built datasets, we used **Our model** to train on **NLU** datasets. The results are shown in Table V.

TABLE V. Results of **DIET**$_{pB+BOW}$ Model on **NLU**$_{test}$

Training dataset	IC		EE	
	F1	Acc	F1	Acc
NLU$_{40}$	0.93	0.93	0.96	0.98
NLU$_{50}$	0.94	0.93	0.96	0.98
NLU$_{60}$	0.94	0.93	0.97	0.98
NLU$_{70}$	0.95	0.95	0.97	0.98
NLU$_{train}$	**0.96**	**0.96**	**0.97**	**0.99**

2) Multi-intent handling experimental result

Fig. 9 shows the result of three sentence-BERT on **STS**$_{valid}$ throughout their training processes.

Tables VI and VII show the BLEU, ROUGE SACC, EM and PM score of the multi-intent handling module using static contexts and different context retrieval models. The **SBERT**$_{few-shot\,COT}$ is chosen in our proposed method.

D. Discussion

1) Our semi-automatically data built method and VA

Table V shows that the model is trained on the training dataset **NLU**$_{40}$ includes 180 samples made manually and 1260 samples made by our proposed method that achieved a 0.93 F1 and Acc score in the IC task and a 0.96 F1 and 0.98 Acc score in the EE task. This demonstrates that our data is of high quality, and the VA we made works so well.

The dataset **NLU**$_{train}$ is made by adding 1440 samples is built by our proposed method to **NLU**$_{40}$. The model is trained on **NLU**$_{train}$ achieved F1 and Acc score are higher than the model is trained on demonstrates our proposed semi-automatically data building method is a good method. Especially its ability to update itself with new data to become better.

2) Multi-intent handling experiment

In the first experiment evaluating the performance of sentence-BERT models on **STS**$_{valid}$ throughout their training process shown in Fig. 9, it can be clearly seen that the best performance that all three models can achieve converges at a result of 0.82. Their convergence speeds, however, differ greatly from each other. The model **SBERT**$_{zero-shot}$ trained on **STS**$_{zero-shot}$ dataset converges much slower than the other two sentence-BERT models when needing about 100 training steps to converge compared to just about 30 to 50 training steps of **SBERT**$_{few-shot}$ and **SBERT**$_{few-shot\,COT}$. The reason for its slow convergence speed is because its training dataset **STS**$_{zero-shot}$ is built from the context dataset **Context**$_{zero-shot}$ which has plenty of false multi-intent commands. Fig. 10 shows some examples of these kinds of commands in the **Context**$_{zero-shot}$ dataset.

The false multi-intent commands shown in the "merged multi-intent commands" column in the Table X are actually single-intent commands. When this kind of command is matched with true multi-intent commands in the dataset, it will make the false sample in the STS dataset used to train the context retrieval dataset, thus making the training process of the model divergent.

In the second experiment shown in Tables VI and VII which evaluates the performance of multi-intent handling module when using trained context retrieval models compared to using static contexts with various metrics, it can be clearly seen that the performance of multi-intent handling module using trained context retrieval models is relatively higher than one of multi-intent handling module using static context in every metric, emphasizing the great contribution of context retrieval models trained on STS training datasets to the performance of multi-intent handling module. In Table VI, the difference in ROUGE in BLEU metric between the worst-performing model (Baseline model) and the best-performing model (**SBERT**$_{few-shot\,COT}$) is not really notable, ranging around 0.02 and 0.04. We hypothesize that we have concatenated the output single-intent command list into a single string and evaluated on this concatenated string so that the evaluated strings of different models may be similar to each other in some substrings. The results in SACC, EM and PM metrics shown in Table VII, on the other hand, see a significant gap between the worst-performing and the best-performing model, ranging from 0.04-0.05. Furthermore, what we can clearly see in that table is that the gap between the results of multi-intent handling module using a trained context retrieval model is pretty trivial, indicating that with contexts that are relatively similar to the user's input, the LLMs can accurately split the input into single-intent commands in terms of intent and assign the correct information to each of the output commands.

To investigate the capability of multi-intent handling in single-intent and multi-intent command detection, we have conducted two additional experiments for the module with different configurations on single-intent commands and multi-intent commands separately. The Tables VIII and IX show the results in SACC metric of the two experiments.

In Table VIII, the multi-intent handling module using **SBERT**$_{zero-shot}$ as context retrieval model archives the worst result, even much worse than the Baseline module. This

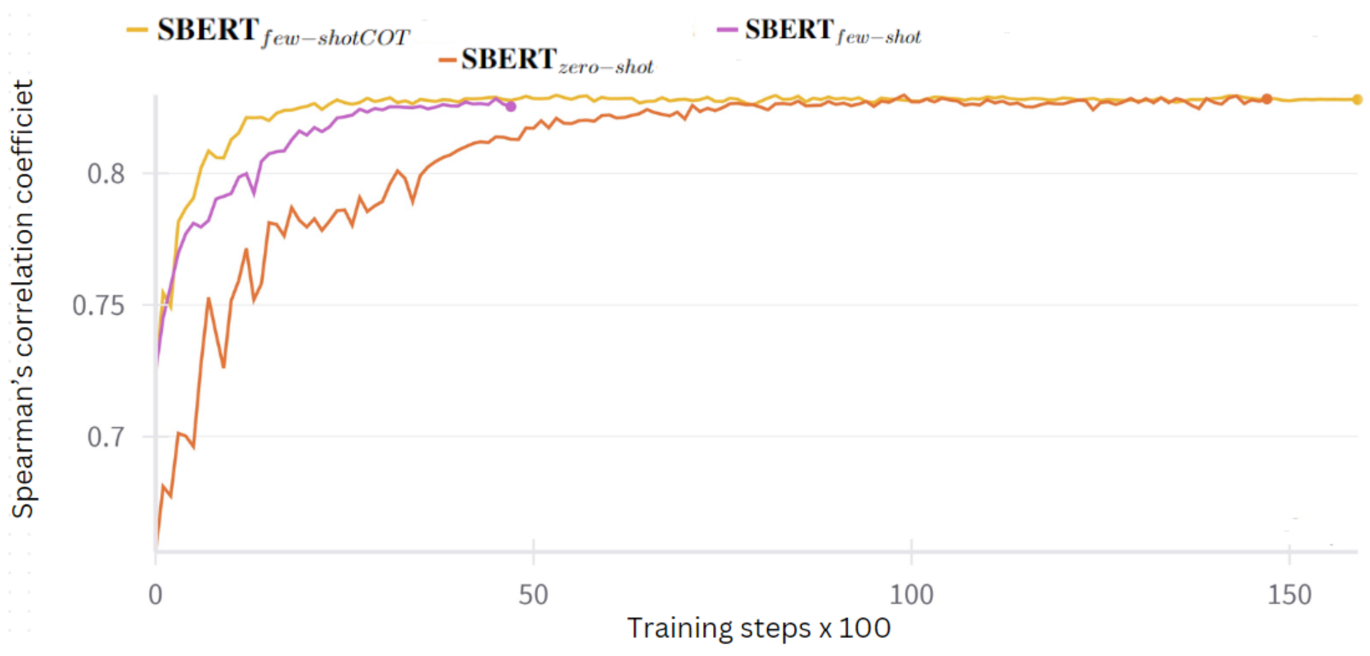


Fig. 9. Spearman's correlation coefficient of three sentence-BERT models trained on three different STS datasets

TABLE VI. ROUGE and BLEU Score of Multi-intent Handling Module using Static Contexts and Different Context Retrieval Models

Model	ROUGE-1	ROUGE-2	ROUGE-L	ROUGE-S	BLEU
Baseline	0.87	0.81	0.84	0.84	0.7
$\textbf{SBERT}_{zero-shot}$	0.89	0.83	0.85	0.85	0.74
$\textbf{SBERT}_{few-shot}$	0.89	0.83	0.86	0.86	0.74
$\textbf{SBERT}_{few-shot\,COT}$	**0.9**	**0.84**	**0.86**	**0.86**	**0.75**

TABLE VII. SACC, EM and PM of Multi-intent Handling Module using Static Contexts and Different Context Retrieval Models

Model	SACC	EM	PM
Baseline	0.87	0.75	0.83
$\textbf{SBERT}_{zero-shot}$	**0.93**	**0.79**	**0.87**
$\textbf{SBERT}_{few-shot}$	0.90	0.79	0.86
$\textbf{SBERT}_{few-shot\,COT}$	0.92	0.79	0.87

TABLE VIII. The Result in SACC Metric on Single-intent Commands of the Evaluation Dataset

Module	SACC Score
Baseline	0.93
$\textbf{SBERT}_{zero-shot}$	0.85
$\textbf{SBERT}_{few-shot}$	0.94
$\textbf{SBERT}_{few-shot\,COT}$	**0.96**

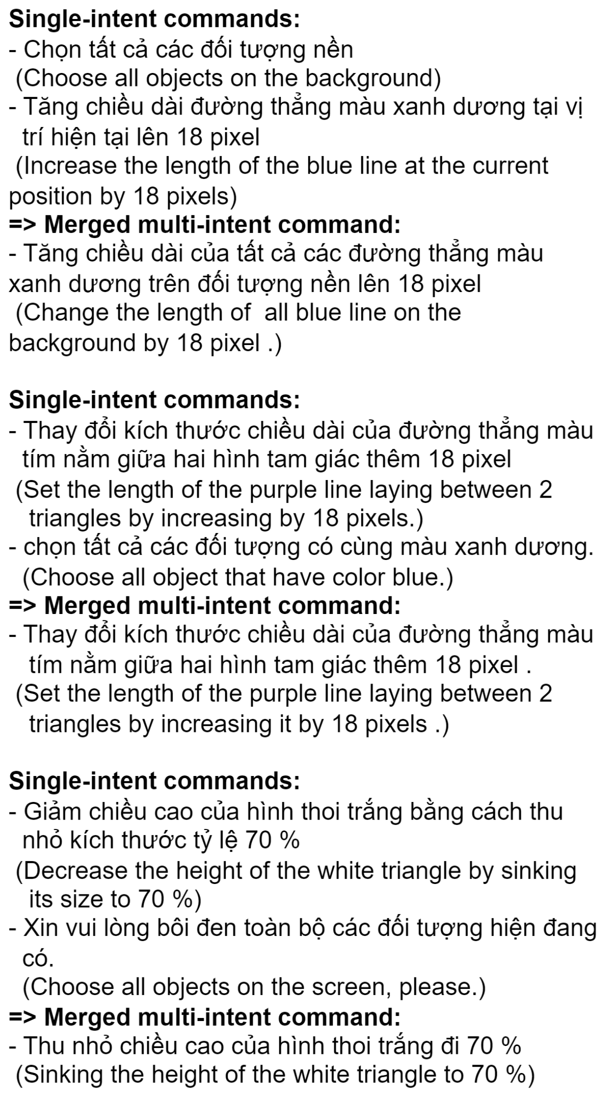
TABLE IX. The Result in SACC Metric on Multi-intent Commands of the Evaluation Dataset

Module	SACC Score
Baseline	0.87
$\textbf{SBERT}_{zero-shot}$	**0.94**
$\textbf{SBERT}_{few-shot}$	0.91
$\textbf{SBERT}_{few-shotCOT}$	0.92

is due to the false multi-intent commands in its context database, as mentioned in Fig. 10. That makes the model likely to retrieve true multi-intent commands when receiving the single-intent command of the user, leading to the wrong segmentation. In contrast, the multi-intent handling module using $\textbf{SBERT}_{zero-shot}$ as context retrieval model achieves the best result, and the gap between the modules using trained context retrieval models is relatively trivial. This is due to the majority of multi-intent commands in the evaluation dataset being simple commands that have single-intent commands linked by linking words and punctuation, which appear very common in all three context datasets. The remaining multi-intent commands in the evaluation dataset, however, have information interleaved or even tricky to assign to an intent. The $\textbf{Context}_{zero-shot}$ context dataset can have many complex multi-intent commands due to the lack of reference examples in the Zero-shot merging method, which is used to generate multi-intent commands in that dataset, thus making LLMs likely to generate out-of-the-box commands with no limit.

V. Conclusion

In this work, we have proposed the effective method for recognizing multi-intent commands in low-resource languages and respond to SF in doing the drawing task. In the phase of data building, we proposed a semi-automatic data building



Single-intent commands:
- Chọn tất cả các đối tượng nền
(Choose all objects on the background)
- Tăng chiều dài đường thẳng màu xanh dương tại vị trí hiện tại lên 18 pixel
(Increase the length of the blue line at the current position by 18 pixels)

=> Merged multi-intent command:
- Tăng chiều dài của tất cả các đường thẳng màu xanh dương trên đối tượng nền lên 18 pixel
(Change the length of all blue line on the background by 18 pixel .)

Single-intent commands:
- Thay đổi kích thước chiều dài của đường thẳng màu tím nằm giữa hai hình tam giác thêm 18 pixel
(Set the length of the purple line laying between 2 triangles by increasing by 18 pixels.)
- chọn tất cả các đối tượng có cùng màu xanh dương.
(Choose all object that have color blue.)

=> Merged multi-intent command:
- Thay đổi kích thước chiều dài của đường thẳng màu tím nằm giữa hai hình tam giác thêm 18 pixel .
(Set the length of the purple line laying between 2 triangles by increasing it by 18 pixels .)

Single-intent commands:
- Giảm chiều cao của hình thoi trắng bằng cách thu nhỏ kích thước tỷ lệ 70 %
(Decrease the height of the white triangle by sinking its size to 70 %)
- Xin vui lòng bôi đen toàn bộ các đối tượng hiện đang có.
(Choose all objects on the screen, please.)

=> Merged multi-intent command:
- Thu nhỏ chiều cao của hình thoi trắng đi 70 %
(Sinking the height of the white triangle to 70 %)

Fig. 10. False multi-intent commands in the $\mathbf{Context}_{zero-shot}$ dataset.

method. Applying it to the Vietnamese language, we built an intent list including 36 intents, an entity list including 31 entities, and the labeled Vietnamese command corpus including 3240 commands. In the phase of generating the JSON file, we proposed the method that separates the multi-intent command from the user's command into single-intent commands to be better understood by the VA, from which it supports more effectively for SF. Our labeled command corpus and the open source code of the splitting tool are shared with the research community. In the future, we will continue to research and improve our data construction method and extend it to some low-resource languages. Furthermore, we will conduct research to improve the capabilities and accuracy of the VA problem.

Acknowledgment

This work has been supported by the Vietnamese government under Program KC-4.0-30/19-25.

References

[1] Ahuja, Sanju, and Jyoti Kumar. Assistant or Master: Envisioning the User Autonomy Implications of VAs. In Proceedings of the 4th Conference on Conversational User Interfaces, pp. 1-5. 2022. DOI: 10.1145/3543829.3544514.

[2] Charles Hemphill, John Godfrey, and George Doddington. The ATIS Spoken Language Systems Pilot Corpus. Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27,1990. New York, NY, USA. 1990.

[3] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. Proceedings of the 40th annual meeting of the Association for Computational Linguistics, pp. 311–318. New York, NY, USA. 2002.

[4] Tanja Bunk, Daksh Varshneya, Vladimir Vlasov, and Alan Nichol. DIET: Lightweight Language Understanding for Dialogue Systems. arXiv, pp. arXiv:2004.09936. New York, NY, USA. 2020.

[5] Brambilla, M., Molinelli, D. (2021). Voice-Based Virtual Assistants for User Interaction Modeling. In: Brambilla, M., Chbeir, R., Frasincar, F., Manolescu, I. (eds) Web Engineering. ICWE 2021. Lecture Notes in Computer Science(), vol 12706. Springer, Cham.

[6] Jiawei Han, Micheline Kamber, and Jian Pei. Data mining concepts and techniques third edition. University of Illinois at Urbana-Champaign Micheline Kamber Jian Pei Simon Fraser University. New York, NY, USA. 2012.

[7] Trang Mai and Shcherbakov Maxim. Enhancing Rasa NLU model for Vietnamese chatbot. International Journal of Open Information Technologies, vol. 9, no. 1, pp. 31–36. New York, NY, USA. 2021.

[8] Edouard Grave, Piotr Bojanowski, Prakhar Gupta, Armand Joulin, and Tomas Mikolov. Learning word vectors for 157 languages. arXiv, pp. arXiv:1802.06893. New York, NY, USA. 2018.

[9] S. Cobos-Guzman, S. Nuere, L. Miguel, and C. König. Design of a VA to Improve Interaction Between the Audience and the Presenter. International Journal of Interactive Multimedia and Artificial Intelligence, ch. 7, sec. 2, pp. 232-240. New York, NY, USA. 2021. DOI: https://dx.doi.org/10.9781/ijimai.2021.08.017

[10] Hoy, Matthew B. Alexa, Siri, Cortana, and More: An Introduction to Voice Assistants. *Medical reference services quarterly*, vol. 37, ch. 1, pp. 81-88. 2018. DOI: 10.1080/02763869.2018.1404391.

[11] Michael McTear, Zoraida Callejas, and David Griol, Talking to Smart Devices. The Conversational Interface. New York, NY, USA. 2016. Springer Publishing Company, Incorporated. https://link.springer.com/book/10.1007/978-3-319-32967-3

[12] Morana, Stefan; Friemel, Celina; Gnewuch, Ulrich; Maedche, Alexander; Pfeiffer, Jella, Interaktion mit smarten Systemen — Aktueller Stand und zukünftige Entwicklungen im Bereich der Nutzerassistenz. Wirtschaftsinformatik & Management, Springer. PISSN: 1867-5913. vol. 9, no. 5, pp. 42-51.

[13] Xingkun Liu, Arash Eshghi, Pawel Swietojanski, and Verena Rieser. Benchmarking natural language understanding services for building conversational agents. Increasing naturalness and flexibility in spoken dialogue interaction: 10th international workshop on spoken dialogue systems, pp. 165–183. New York, NY, USA. 2021.

[14] C. Pearl, Principles of Conversational Experiences. Designing Voice User Interfaces. New York, NY, USA. 2016. O'Reilly Media. https://www.oreilly.com/library/view/designing-voice-user/9781491955406

[15] Pereira R, Lima C, Pinto T, Reis A. Virtual Assistants in Industry 4.0: A Systematic Literature Review. Electronics. 2023; 12(19):4096.

[16] Dat Nguyen and Anh Nguyen. PhoBERT: Pre-trained language models for Vietnamese. arXiv, pp. arXiv:2003.00744. New York, NY, USA. 2020.

[17] Andrés Piñeiro-Martín, Carmen García-Mateo, Laura Docío-Fernández, and María López-Pérez. Ethical Challenges in the Development of VAs Powered by Large Language Models. Electronics, vol. 12, no. 14. New York, NY, USA. 2023. DOI: 10.3390/electronics12143170.

[18] Pérez, Anxo & Lopez-Otero, Paula & Parapar, Javier. Designing an Open Source VA. Proceedings, vol. 54, no. 1, pp. 30. 2020. DOI: 10.3390/proceedings2020054030.

[19] Libo Qin, Xiao Xu, Wanxiang Che, and Ting Liu. AGIF: An Adaptive Graph-Interactive Framework for Joint Multiple Intent Detection and Slot Filling. Findings of the Association for Computational Linguistics: EMNLP 2020, pp. 1807–1816. New York, NY, USA. 2020. Association for Computational Linguistics. DOI:10.18653/v1/2020.findings-emnlp.163.

[20] Tom Bocklisch, Joey Faulkner, Nick Pawlowski, and Alan Nichol. Rasa: Open source language understanding and dialogue management. arXiv, pp. arXiv:1712.05181. New York, NY, USA. 2017.

[21] Barbara Rychalska, Helena Glabska, and Anna Wroblewska. Multi-intent hierarchical natural language understanding for chatbots. 2018 Fifth international conference on social networks analysis, management and security (SNAMS), pp. 256–259. New York, NY, USA. 2018.

[22] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. Text summarization branches out, pp. 74–81. New York, NY, USA. 2004.

[23] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. arXiv, pp. arXiv:1908.10084. New York, NY, USA. 2019.

[24] Alice Coucke, Alaa Saade, Adrien Ball, Théodore Bluche, Alexandre Caulier, David Leroy, Clément Doumouro, Thibault Gisselbrecht, Francesco Caltagirone, Thibaut Lavril, and others. Snips voice platform: an embedded spoken language understanding system for private-by-design voice interfaces. arXiv, pp. arXiv:1805.10190. New York, NY, USA. 2018.

[25] Sheng Jiang, Su Zhu, Ruisheng Cao, Qingliang Miao, and Kai Yu. 2023. SPM: A Split-Parsing Method for Joint Multi-Intent Detection and Slot Filling. In Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 5: Industry Track), pp. 668–675, Toronto, Canada. Association for Computational Linguistics.

[26] Strohmann, T., Siemon, D., & Robra-Bissantz, S. . Designing Virtual In-vehicle Assistants: Design Guidelines for Creating a Convincing User Experience. AIS Transactions on Human-Computer Interaction, ch 11, sec. 2, pp. 54-78. 2019. DOI: 10.17705/1thci.00113.

[27] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. Advances in neural information processing systems, vol. 30. New York, NY, USA. 2017.

[28] Haoran Meng, Zheng Xin, Tianyu Liu, Zizhen Wang, He Feng, Binghuai Lin, Xuemin Zhao, Yunbo Cao, and Zhifang Sui. Dialogusr: Complex dialogue utterance splitting and reformulation for multiple intent detection. arXiv, pp. arXiv:2210.11279. New York, NY, USA. 2022.

[29] VAN DAELE, D.; DECLEYRE, N.; DUBOIS, H.; MEERT, W. An Automated Engineering Assistant: Learning Parsers for Technical Drawings. Proceedings of the AAAI Conference on Artificial Intelligence, [S. l.], v. 35, n. 17, p. 15195-15203, 2021. DOI: 10.1609/aaai.v35i17.17783.

[30] Alec Radford, Jong Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision. International conference on machine learning, pp. 28492–28518. New York, NY, USA. 2023.

Appendix

Table X shows all intent labels we have used in our dataset and their meanings.

TABLE X. The Intent Labels and their Meaning

Intent	Meaning
select_all_object	The user wants to select all object (with optionally the same properties) on the interface of the SF
exit_select	The user wants to give up selecting some (or all) selected objects (with optionally the same properties) on the interface of the SF
delete_selected_objects	The user wants to delete some (or all) selected objects (with optionally the same properties) on the interface of the SF
selected_area	The user wants to select all objects (with optionally the same properties) in an area defined by a pair of coordinates on the interface of the SF
rotate_left	The user wants to rotate a specific object on the interface of the SF to the left side
rotate_right	The user wants to rotate a specific object on the interface of the SF to the right side
horizontal_flip	The user wants to flip a specific object on the interface of the SF horizontally
vertical_flip	The user wants to flip a specific object on the interface of the SF vertically
move_left	The user wants to move a specific object on the interface of the SF to the left side
move_right	The user wants to move a specific object on the interface of the SF to the right side
move_up	The user wants to move a specific object on the interface of the SF upward.
move_down	The user wants to move a specific object on the interface of the SF downwards.
color_background	The user wants to paint the background of a specific object on the interface of the SF by a specific color.
color_foreground	The user wants to paint the foreground of a specific object on the interface of the SF by a specific color.
change_width	The user wants to change the width of a specific object on the interface of the SF.
change_height	The user wants to change the height of a specific object on the interface of the SF.
change_length	The user wants to change the length of a specific object on the interface of the SF.
change_radius	The user wants to change the radius of a specific object on the interface of the SF.
change_top	The user wants to move a specific object on the interface of the SF in the vertical direction so that its new position is a certain distance from the top border of the interface of the SF.
change_left	The user wants to move a specific object on the interface of the SF in the horizontal direction so that its new position is a certain distance from the left border of the interface of the SF.
change_right	The user wants to move a specific object on the interface of the SF in the horizontal direction so that its new position is a certain distance from the right border of the interface of the SF.
change_bottom	The user wants to move a specific object on the interface of the SF in the vertical direction so that its new position is a certain distance from the bottom border of the interface of the SF.
draw_line	The user wants to draw a line on the interface of the SF. The information about attributes of the object is optionally provided. Any attributes with no provided information will be set to its default value.
draw_circle	The user wants to draw a circle on the interface of the SF. The information about attributes of the object is optionally provided. Any attributes with no provided information will be set to its default value.
draw_ellipse	The user wants to draw a ellipse on the interface of the SF. The information about attributes of the object is optionally provided. Any attributes with no provided information will be set to its default value.
draw_rectangle	The user wants to draw a rectangle on the interface of the SF. The information about attributes of the object is optionally provided. Any attributes with no provided information will be set to its default value.
draw_square	The user wants to draw a square on the interface of the SF. The information about attributes of the object is optionally provided. Any attributes with no provided information will be set to its default value.

draw_rhombus	The user wants to draw a rhombus on the interface of the SF. The information about attributes of the object is optionally provided. Any attributes with no provided information will be set to its default value.
draw_parallelogram	The user wants to draw a parallelogram on the interface of the SF. The information about attributes of the object is optionally provided. Any attributes with no provided information will be set to its default value.
draw_trapezoid	The user wants to draw a trapezoid on the interface of the SF. The information about attributes of the object is optionally provided. Any attributes with no provided information will be set to its default value.
draw_arrow	The user wants to draw an arrow on the interface of the SF. The information about attributes of the object is optionally provided. Any attributes with no provided information will be set to its default value.
copy_selected_objects	The user wants to copy all selected objects (with optionally the same properties) to the clipboard.
cut_selected_objects	The user wants to cut all selected objects (with optionally the same properties) and save them into the clipboard.
paste	The user wants to paste the object saved in the clipboard to a certain position on the interface of the SF.
undo	The user wants to undo the task.
redo	The user wants3 to redo the task.

Table XI shows all entity labels which are grouped into entity groups with their meanings.

TABLE XI. ENTITY LABELS AND THEIR MEANINGS

Entity group	Entity	Meaning
Object (used to label phrases describing attributes of the object)	object—shape	Indicates the shape of the object
	object—width	Indicates dimensions like the upper base of a trapezoid, radius of a circle, width of a rectangle, etc.
	object—height	Indicates the height of various shapes such as triangles and parallelograms
	object—length	Indicates the length of various shapes like rectangles, arrows, etc.
	object—color	Indicates the color of the object
	object—thickness	Indicates the thickness of the object's border
	object—destination	Indicates the end point of a line (e.g. center of the screen or top left corner)
	object—angle	Indicates the value of the angle at the top left corner of the object
	object—destination_x	Indicates the x-axis of the end point of the line
	object destination_y	Indicates the y-axis of the end point of the line
Value (used to label phrases indicating essential parameters for specific tasks)	value—color	Indicates the color for tasks like ”color_background” and ”color_foreground”
	value—change	Indicates changes to the size of an object (e.g. increase, decrease, set new value)
	value—move	Indicates the distance between the new and old positions of the object
	value—angle	Indicates the angle for rotating the object (e.g. “rotate_left”, “rotate_right”)
Position (used to label phrases indicating the position of the object)	position—source	Indicates the current position of the object, like “top left corner” or “center of the screen”
	position—source_x	Indicates the x-axis of the current position of the object

	position—source_y	Indicates the y-axis of the current position of the object
	position—destination	Indicates the target position for tasks like “paste” or selecting an area
	position—destination_x	Indicates the x-axis of the target position
	position—destination_y	Indicates the y-axis of the target position
	position—center	Indicates the center of the object
	position—center_x	Indicates the x-axis of the object’s center
	position—center_y	Indicates the y-axis of the object’s center
Selected_area (used for “select_area” tasks)	selected_area—height	Indicates the height of the selected area
	selected_area—width	Indicates the width of the selected area
	selected_area—length	Indicates the length of the selected area
Change_action (used for resizing objects)	change_action—increase	Indicates that the user wants to increase the size of the object
	change_action—decrease	Indicates that the user wants to decrease the size of the object
	change_action—set	Indicates that the user wants to set a new size for the object
Aspect	aspect	Used to compare an object with others based on features like size or length
Comparison	comparison	Used to label phrases that compare objects (e.g. “bigger than”, “equal to”)