Bil-DOS: A Bi-lingual Dialogue Ordering System (for Subway)

Zirong Chen
Georgetown University
Department of Computer Science
`zc157@georgetown.edu`

Haotian Xue
Georgetown University
Department of Computer Science
`hx82@georgetown.edu`

Abstract

Due to the unfamiliarity to particular words(or proper nouns) for ingredients, non-native English speakers can be extremely confused about the ordering process in restaurants like Subway. Thus, We developed a dialogue system, which supports Chinese(Mandarin)[1] and English[2] at the same time. In other words, users can switch arbitrarily between Chinese(Mandarin) and English as the conversation is being conducted. This system is specifically designed for Subway ordering[3]. In Bil-DOS, we designed a `Discriminator` module to tell the language is being used in inputted user utterance, a `Translator` module to translate used language into English if it is not English, and a `Dialogue Manager` module to detect the intention within inputted user utterances, handle 'outlier' inputs by throwing clarification requests, map detected *Intention* and detailed *Keyword*[4] into a particular intention class, locate the current ordering process, continue to give queries to finish the order, conclude the order details once the order is completed, activate the evaluation process when the conversation is done.

1 Introduction

From personal previous experiences, non-native English speakers can find Subway ordering greatly difficult at the first time due to the unfamiliarity to particular words(or proper nouns) for ingredients. Although most of the system we encountered and implemented so far are in English[5], we still would like to take a step further to develop a bi-lingual dialogue system, in order to make events like ordering more convenient for non-native English speakers.

In Bil-DOS, users can switch between two languages as the conversation goes on. And at the same time, the system will be able to react to each user utterance using the same language that was most recently used by the user. And after the conversation ends, the system will report order details. Here is happy case[6] that the system is supposed to handle:

[1] ***User***: Hi there!

[2] ***System***: Hi, welcome to our Bi-Lingual Ordering System! What can I do for you? Any bread you prefer?

[3] ***User***: Italian bread please!

[4] ***System***: Nice choice, you ordered Italian bread as bread, anything else for cheese?

[5] ***User***: 羊奶奶酪。(Feta cheese.)

[6] ***System***: 您刚刚点了羊奶奶酪作为奶酪，还有什么想要的蔬菜吗？(You just ordered feta cheese as cheese, what do you want for vegetable?)

[7] ***User***: 牛油果。(Avocado.)

[8] ***System***: 您刚刚点了牛油果作为蔬菜，还有喜欢的的酱料吗？(You just ordered avocado as vegetable, any preferred sauce?)

[9] ***User***: Barbecue sauce.

[10] ***System***: Nice choice, you just ordered barbecue source as sauce, is there any extra you want?

[11] ***User***: No, thanks!

[12] ***System***: Okay, you just ordered nothing for extra. I think that is all for your order!

[13] ***System***: Fantastic, your order is: one Italian bread sandwich with feta cheese as cheese, avocado as vegetable, barbecue sauce as sauce and with extra Nothing! Thanks for visiting!

---

[1] In this paper, term *Chinese* refers to *Chinese(Mandarin)*.

[2] All codes are available online at our GitHub Repo.

[3] See menu at Subway website.

[4] In our project settings, we assume there have to be an *Intention* and a *Keyword* within every user utterance, like 'Bread' as *Intention* and '9-Grain Wheat' as *Keyword*.

[5] Some of them support other languages as well, but we did not have the chance to develop our own version

[6] Unhappy cases of user utterances include: typo, unseen *Intention*s, unseen *Keyword*s.

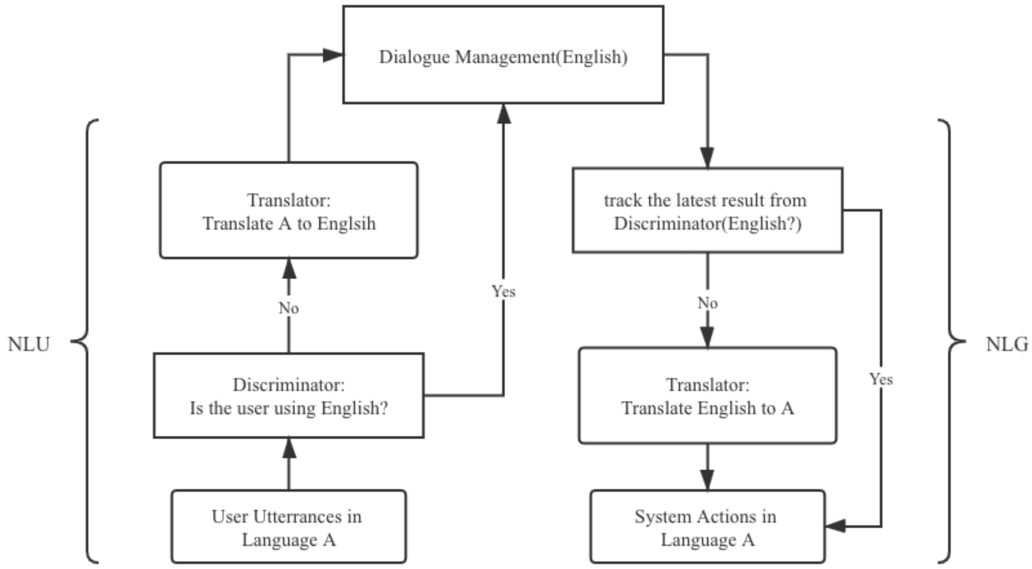


Figure 1: Overall structure of Bi-Lingual Dialogue System

....Evaluation Process Begins Afterwards...

2 Related Work

We model Bil-DOS as a task-oriented dialogue system (Gravano et al., 2011; Jurafsky, 2000; Gao et al., 2018). Bil-DOS follows the generally-used *Natural Language Understanding*(NLU)-*Dialogue Management*(DM)-*Natural Language Generation*(NLG) structure. Furthermore, a `Discriminator` module, a `Translator` module and a `Dialogue Manager` module are employed to coordinate with each other and to better execute the 'mission' given by the user. The overall structure of our dialogue system is shown in Figure 1.

2.1 Natural Language Understanding

An *Natural Language Understanding*(NLU) module is usually built to let the `Dialogue Manager` better 'understand' human language. In our case, the `Dialogue Manager` module is designed and implemented for English only, so, as shown in the figure 1, our NLU module integrates a `Discriminator` module to discriminate the language being used and a `Translator` module to translate non-English language to English. Only by doing this, can the `Dialogue Manager` understand non-English language, which is Chinese(Mandarin) in this case, and handle user requests in both languages.

2.2 Dialogue Management

Inspired by Jurafsky's (Jurafsky, 2000) and Gao's (Gao et al., 2018) book, our Dialogue Management(DM) module is drafted and implemented in a slot-filling fashion[7]. Typically, for Subway ordering, there are *slots*, like bread, cheese and sauce, need to be filled as the conversation goes on. By DM *request*ing slot values and the user *inform*ing slot values, the order details will be further supplemented. Once the order details are fulfilled, the conversation session will be terminated and a conclusion will be drawn.

2.3 Natural Language Generation

The generation of human language from computer language can be viewed as a general definition of NLG. And to simplify the task, inspired by AIML (Bush, 2001; AbuShawar and Atwell, 2015), Bil-DOS applies pre-written templates to emit actions, in other words, Bil-DOS is a template-based dialogue system.

2.4 Case Analysis

Take turn[2] and turn[3] in the conversation sample as the case here. The detailed co-operations between NLU, DM and NLG modules are shown in Figure 2.

---

[7] In Bil-DOS, the slot is *Intention* and the slot value is *Keyword*.

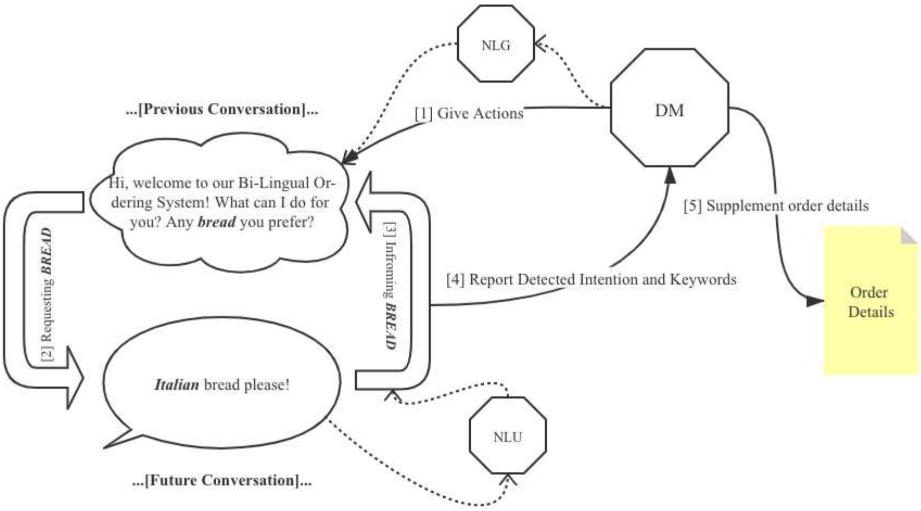

NLG
DM
...[Previous Conversation]...
[1] Give Actions
Hi, welcome to our Bi-Lingual Ordering System! What can I do for you? Any *bread* you prefer?
[2] Requesting *BREAD*
[3] Infroming *BREAD*
[5] Supplement order details
[4] Report Detected Intention and Keywords
Order Details
Italian bread please!
NLU
...[Future Conversation]...

Figure 2: Co-operations between modules in Bil-DOS

3 Approach and System Design

To realize the the *Natural Language Understanding*(NLU)-*Dialogue Management*(DM)-*Natural Language Generation*(NLG) structure, Bil-DOS employs a `Discriminator` module, a `Translator` module and a `Dialogue Manager` module. In this section, the detailed design of each module are shown in the UML class diagram in Figure 3 and will be discussed.

3.1 The Discriminator

This module is designed for turn-level language detection. Current methods include rule based detection (Buryak and Swartz, 2015), Bayes Theorem approximation (Jimenez-Feltström, 2006), cluster-based inference (Buryak et al., 2015), neural networks (Kim et al., 2014). However, most of these methods are dealing with languages that at least share some part of their alphabets. In our case, English and Chinese have totally different alphabets, then character-level Unicode (Allen et al., 2012) comparison[8] will be sufficient to tell the differences.

The `Discriminator` module has `language`, `utterance` as its attributes[9], and `load(.)`, `tell()` as its public functions.

The `language` attribute tells the supported languages and currently it is '*en_zh*' by default, which means the `Discriminator` supports English('en') and Chinese('zh'). The `utterance` attribute is used to store current user utterance and will be set to *None* upon another time of use.

The `load(.)` function takes user utterance once a time and store it for further use, once one turn is over, the storage will be evicted. The `tell()` is able to tell what the language is being used from the most recent user utterance.

3.2 The Translator

The `Translator` module is wielded to translate non-English language, which is Chinese(Mandarin) in our case, to English. We first explored the possibility to apply BERT (Devlin et al., 2018) language models using Hugging Face package (Wolf et al., 2020). However, after several attempts to implement a pipeline for language translation, this method turned out to be far more time-consuming than we expected. For our task, a more efficient approach is needed for conversation fluency[10]. Thus, another solution for translation tasks is found: (packages or API of)Online translators. The package used is named as *translators* (UlionTse, 2020). This package is a free, online and toy-level library that helps users to send translation requests to some famous online translators, like Google, Yandex,

---

[8] In Unicode comparison, once a Chinese character shows up, the user utterance will be marked as '*zh*'.

[9] N.B. All the attributes are predefined with the class initialization function

[10] In the process, we do not want any palpable lagging between the order system and the user.

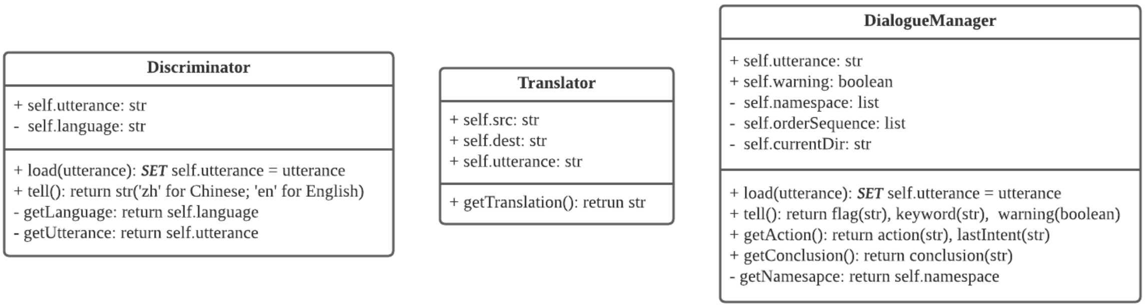

Discriminator
+ self.utterance: str
- self.language: str
+ load(utterance): SET self.utterance = utterance
+ tell(): return str('zh' for Chinese; 'en' for English)
- getLanguage: return self.language
- getUtterance: return self.utterance
Translator
+ self.src: str
+ self.dest: str
+ self.utterance: str
+ getTranslation(): retrun str
DialogueManager
+ self.utterance: str
+ self.warning: boolean
- self.namespace: list
- self.orderSequence: list
- self.currentDir: str
+ load(utterance): SET self.utterance = utterance
+ tell(): return flag(str), keyword(str), warning(boolean)
+ getAction(): return action(str), lastIntent(str)
+ getConclusion(): return conclusion(str)
- getNamesapce: return self.namespace

Figure 3: UML class diagram for modules

Microsoft(Bing)and Baidu, and then fetch the translation results from those translators. The `Translator` module in Bil-DOS is built on top of the *translators* package[11] .

The `Translator` module has `src`, `dest`, `utterance` as its attributes, and `getTranslation()` as its public function.

The `src` and `dest` attributes are used to describe the *source* and *destination* languages that need to be clarified in the translation process. And again, the translation process is turn-level, thus it needs `utterance` attribute to store the user utterance for further translation.

The `getTranslation()` function returns the translated English sentence(s) from the store user utterance if the utterance is not in English.

3.3 The Dialogue Manager

The Dialogue Management module serves as the 'heart' of a dialogue system. In Bil-DOS, the DM takes user utterances as input and gives system actions according to the user requests as output. Furthermore, Bil-DOS's Dialogue Manager is able to detect user intentions, map and extract keywords, fulfill order details, and handle error inputs.

The `Dialogue Manager` module wields `utterance`, `namespace`, `orderSequence`, `currentDir`, `warning` as its attributes, and `tell()`, `getAction()`, `getConclusion()` as its public functions.

The `currentDir` is directory where the order details are placed and `namespace` is a list of the names of all the currently stored user intentions stored in the directory '*/IntentDetails*'. The `orderSequence` is the default ordering logic, which is '*bread*', '*cheese*', '*vegetable*', '*sauce*' and '*extra*'. Although is is named as sequence, user can still break the order sequence as he/she wants in some cases[12]. The `warning` is a boolean variable to indicate whether there are any 'error' inputs detected. And again, the dialogue management process is turn-level, thus it needs `utterance` attribute to store the user utterance for further analysis.

The `load(.)` serves for the same purpose as the one in the `Discriminator` module. The `tell()` function returns the matched intention in the user utterance, the keyword detected under that intention and a boolean value to indicate whether there is a unseen intention involved. The `getAction()` returns system actions and the last system request. The `getConclusion()` reads the order details stored under `currentDir` and returns the order summary.

Intention and Keywords Detection in Bil-DOS is to detect the intention within the user utterance, in order to interpret what the user is trying to inform by mapping and extracting keywords. There are lots of methods to detect intention or keywords, like neural networks (Hussein and Granat, 2002), TF-iDF based evaluations (Rose et al., 2010), SVM based methods (Zhang et al., 2006) or Attention Mechanism (Vaswani et al., 2017), in Bil-DOS, the intention and keywords detection needs to be much easier to ensure the conversation fluency. Since Bil-DOS is designed for ordering, a menu should be presented at the same when Bil-DOS is being used. Thus, for this kind of consideration, *intention* and its list of *keyword*s are pre-loaded manually into the di-

---

[11] After experiments, this package fails to handle extremely frequent requests and there exists a time limit(approximately 10 minutes) for each session. As stated by the author, this package is only for individual use. For more stable translation service, please refer to official APIs.

[12] The sequence was broken into pairs of two intention, as long as the intention is not '*extra*', the conversation will not end and will start at the intention just entered.

rectory '*/IntentDetails*' according to the information on the menu, like '*bread*' and ['*Italian*', '*9-Grain Wheat*', etc]. As long as the user utterance is loaded by the `load(.)` function, the `tell()` function is going to turn the utterance to lower cases to increase the match rate, and iterate through all the *intention*s in the `namespace`, then read the *keyword*s stored under its specific files, and return the matched *intention* and *keyword*.

None and 'Invalid'[13] **Inputs** are very common and difficult issues for template-based dialogue systems. The ability to handle all kinds of inputs is required to boost the user experiences. In Bil-DOS, the conservation is assumed in a particular order, which is '*bread*', '*cheese*', '*vegetable*', '*sauce*' and '*extra*', and that means, for **None-type inputs**, if a decline intention, which can be '*No*' or '*Nope*' is encountered, Bil-DOS needs to backtrack the last requested slot in the last system action and then fill the word '*Nothing*' into it. And that is why the `getAction()` function needs to return the intention in the last system request. For **'invalid' inputs**, Bil-DOS will start a human-in-the-loop annotation process to ask the user to annotate the inputted utterance. And the generated data will be stored in the '*/IntentDetails*' directory for further use, so theoretically, if there is not mislabelling or too many duplicates, Bil-DOS will be more and more robust to all kinds of user utterance as the time of use increases. The robustness of Bil-DOS is shown in Figure 4 and 5. First, let's try not to be confused about what the 'Japanese bread' really is. As shown in Figure 4, the currently stored intentions and keywords failed to handle the input 'Japanese bread', so Bil-DOS threw two annotation questions and the annotation results were stored in '*/IntentDetails/bread.txt*' for further use. In Figure 5, for another time, the 'Japanese bread' showed up. However, since there were relevant annotations about 'Japanese bread' in previous uses, Bil-DOS was able to detect 'Japanese bread' this time. In conclusion, Bil-DOS is able to take **ALL** kinds of user utterances, as long as the user wants to annotate, the intention directory is well maintained, and the conversation is allowed to continue.

3.4 The Evaluator

The `Evaluator` module is built for the evaluation after each conversation. It supports multiple users and stores user scores for each user in his/her file locally. The evaluation process in Bil-DOS is inspired by Gao's book (Gao et al., 2018). In general, we need the evaluation process to capture the system performance in solving user tasks with least resource, which means we need Bil-DOS to complete the ordering process with as few turns as possible, or in other words, Bil-DOS needs to make every turn most helpful for the task completion. So the key standards are 'how well the system helped the user to complete the task' and 'how many turns the system consumed'. Thus, in Bil-DOS built-in evolution module, we prepared parameters like *num_of_turns*, *task_reward*, *turn_penalty*, *user_experience* and *score_factor*. The *task_reward* is a fixed value, 20 by default, by the end of each conversation, if the user's goal is reached, which is to order a sandwich, then the system will get +*task_reward*, otherwise it will get -*task_reward* as penalty. The *turn_penalty* is set to -1 by default. Once a turn is consumed, the system will get one point off as penalty. The intuition behind is to make the conversation shorter and more efficient. The *user_experience* is for user subjective evaluation with the scale from 0 to 10. The *score_factor* is used to balance the weights between user subjective evaluation and Bil-DOS built-in self evaluation, and Sigmoid is used to smooth the results. The reason why Sigmoid function was applied here is that we want both scores taken into consideration without losing the balance. Sigmoid squeezes all numbers between 0 and 1, and it is able to handle extremely large *score_factor* without losing the consideration of the score from built-in self evaluation. The real evaluation process is shown in Figure 6. The evaluation function[14] is:

$$Sc = uSc \times scFac + (tuSc + taSc) \times (1 - scFac)$$

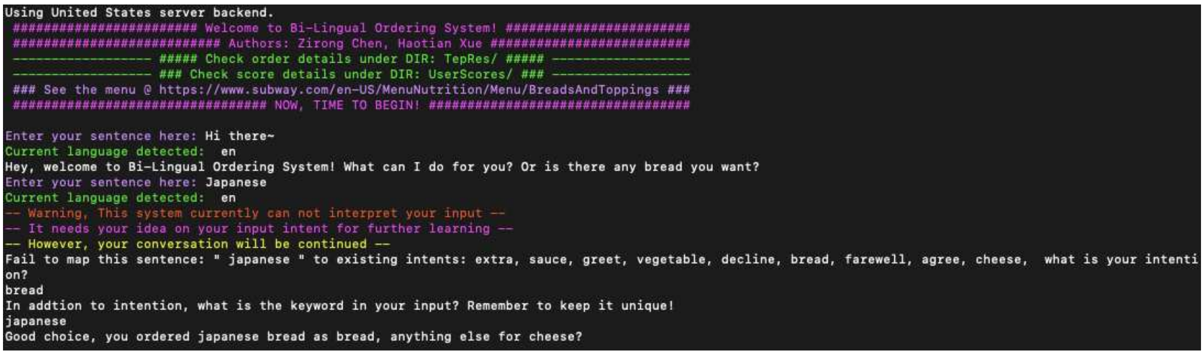
3.5 The Main Session

The main session for Bil-DOS is similar to Figure 1. However, there are several crucial parameters for conversation restraints and future evaluation. First, *num_of_turns*. Since Bil-DOS is a task-oriented dialogue system, there needs to be a

---

[13] The word 'invalid' is not telling the inputs themselves are unacceptable, we use this word for the conclusion of all unseen and undetectable inputs for Bil-DOS so far.

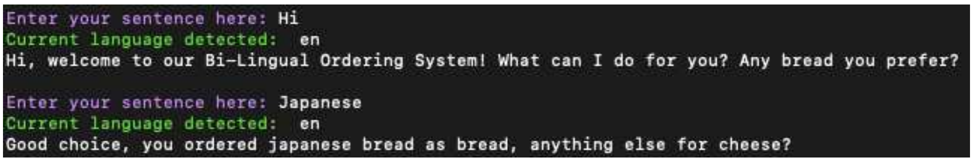
[14] *Sc* stands for Score, *uSc* stands for user_score, *scFac* stands for score_factor, *tuSc* stands for turn_score, *Sc*taSc stands for task_score.



```
Using United States server backend.
 ######################## Welcome to Bi-Lingual Ordering System! ########################
 ########################### Authors: Zirong Chen, Haotian Xue ##########################
 ------------------ ##### Check order details under DIR: TepRes/ ##### ------------------
 ------------------ ### Check score details under DIR: UserScores/ ### ------------------
 ### See the menu @ https://www.subway.com/en-US/MenuNutrition/Menu/BreadsAndToppings ###
 ################################# NOW, TIME TO BEGIN! ##################################

Enter your sentence here: Hi there~
Current language detected:  en
Hey, welcome to Bi-Lingual Ordering System! What can I do for you? Or is there any bread you want?
Enter your sentence here: Japanese
Current language detected:  en
-- Warning, This system currently can not interpret your input --
-- It needs your idea on your input intent for further learning --
-- However, your conversation will be continued --
Fail to map this sentence: " japanese " to existing intents: extra, sauce, greet, vegetable, decline, bread, farewell, agree, cheese,  what is your intenti
on?
bread
In addtion to intention, what is the keyword in your input? Remember to keep it unique!
japanese
Good choice, you ordered japanese bread as bread, anything else for cheese?
```

Figure 4: The unhappy case in Bil-DOS



```
Enter your sentence here: Hi
Current language detected:  en
Hi, welcome to our Bi-Lingual Ordering System! What can I do for you? Any bread you prefer?

Enter your sentence here: Japanese
Current language detected:  en
Good choice, you ordered japanese bread as bread, anything else for cheese?
```

Figure 5: The same unhappy case after ‘learning’

specific restraint of the number of turns. By doing so, Bil-DOS will terminate the conversation if the *num_of_turns* is consumed. Second, *task_reward*, *turn_penalty* and *score_factor* need to be clarified for the Evaluator module. The last one is *translator*, which is designed in the latest beta version(has not been released yet) and is able to give user options to select different translator backends. The detailed main session is shown in the UML sequence diagram[15] in Figure 7.

3.6 User Interfaces

Bil-DOS has a poly-chromatic (Joeld, 2008) UI by adding additional statements in Python3 terminal outputs. Because we think a poly-chromatic can grab more attention from the users, express the ‘feelings’ behind system messages, like blue as welcoming, green as ok-ish, red as warning, ect, and even make the system more vivid. The experiment results do show the boost of user scores after applying poly-chromatic UI.

4 Experiments and Improvements

In this section, we organized several controlled variable experiments about different system designs. All the participants are Chinese speakers and non-native English speakers, and they are target users that Bil-DOS aims to serve.

4.1 Different Translator Backends

The translators package supports bountiful backends, including Google, Yandex, Microsoft(Bing), Baidu, Alibaba, Tencent, NetEase(Youdao), Sogou, Deepl, etc(UlionTse, 2020). In our experiment, we tested user experiences(0-10) scores from different users after using Bil-DOS with Google, Baidu and Bing backends[16]. The results are shown beneath.

Backend	AvgUsrExp
Google	7.86
Baidu	7.51
Bing	7.54

Surprisingly, although there are lags while using Baidu backend, Baidu backend turns out to be almost as good as Bing backend. Baidu translation can be viewed as a translation service specifically designed with Chinese as its source language. So the lag problem can be fixed, Baidu’s score would be much higher.

4.2 Poly-chromatic vs. Mono-chromatic

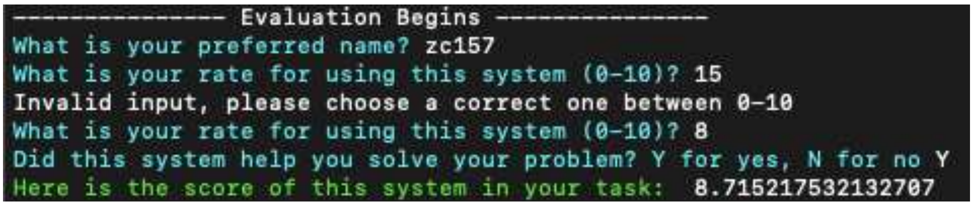
Similar to the experiments about translator backends, the comparison of user experiences between different color settings were conducted as well(with Google as translator backend).

Color settings	AvgUsrExp
Poly-chromatic	7.86
Mono-chromatic	7.22

---

[15] The diagram is not a rigorous UML sequence diagram. To better illustrate the cooperation between modules as a sequence, the main session was also placed into this diagram.

[16] Google and Baidu servers can be available due to the high volume of current user requests.



```
--------------- Evaluation Begins ---------------
What is your preferred name? zc157
What is your rate for using this system (0-10)? 15
Invalid input, please choose a correct one between 0-10
What is your rate for using this system (0-10)? 8
Did this system help you solve your problem? Y for yes, N for no Y
Here is the score of this system in your task:  8.715217532132707
```

Figure 6: Evaluation process in Bil-DOS

As can be found from the results, the polychromatic UI can boost the user experience scores by almost 10%.

4.3 Word-level vs. Phrase-level

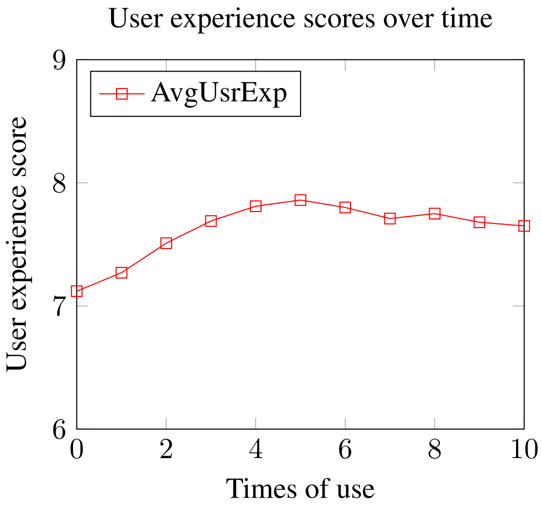
While implementing, we designed two different methods to tokenize utterances and map keywords. The first one is word-level, another one is phrase-level. In word-level, Bil-DOS breaks all user utterances into single words, then maps the keywords from the currently stored resource pool. And in Phrase-level, Bil-DOS keeps the initial user utterance and directly maps the keywords. We defined failure rate as the ratio of the number of task incompletion out of all times of use(3 out of 47 times for word-level, 2 out of 51 times for phrase-level) with a restrained number of turns. In this experiment, we tested Bil-DOS failure rate under these two strategies. Here is the result:

Tokenization&Mapping	FailureRate
Word-level	6.38%
Phrase-level	3.92%

The phrase-level tokenization and mapping strategy yields only half of the failure rate compared with the word-level one. However, Bil-DOS needs to solve task in-completion with other more sophisticated strategies.

4.4 Robustness and Correctness

Bil-DOS is designed with an increasing robustness as the times of use increments. We designed another experiment to record the average user experience scores as the times of use increase.



After experiments, the human-in-the-loop annotation did help the conversation process at the beginning, however, as the times of use increase, the robustness does not always mean correctness and user satisfaction. This phenomenon is quite similar to the relation between recall, precision and F-1 score. More robustness here means higher recall, however, higher recall will not bring a higher F-1 score every time.

5 Error Analysis and Conclusion

After experiments, although Bil-DOS got fairly good evaluation results, it still was not able to handle more complex situations. In this section, we will discuss some unhappy cases that Bil-DOS is not able to handle in the current version.

5.1 Imbalanced Attention

Bil-DOS tends to give more attention to the currently stored intentions and keywords. Because, in the mapping process, Bil-DOS always uses the stored keywords, keeps them fixed, and iterates the inputted sentences to seek the matched patterns. So no matter what kind of tokenization or mapping strategy is being used, the stored keywords will receive more attention than the words in user utterance. One unhappy case we encountered was every-time when the word '*Chinese*' is

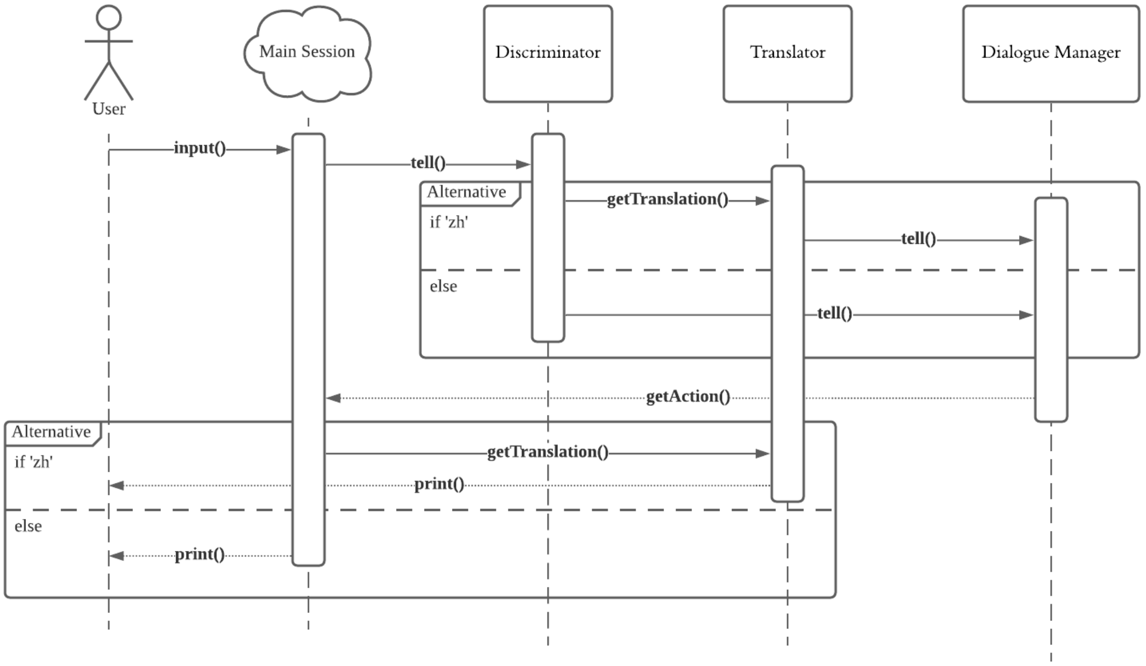


Figure 7: UML sequence diagram with main session

entered, the keyword detected will be ’*hi*’ and the matched intention will be ‘*greet*’. We think a good solution to it might be introducing some statistical learning classification methods like Maximum Likelihood Estimation (McCallum et al., 1998b; Xu et al., 2003), Naive Bayes (McCallum et al., 1998a; Kim et al., 2006), Support Vector Machine (Sun et al., 2009; Wang et al., 2006) or even Neural Nets (Lai et al., 2015; Kim, 2014).

5.2 Co-reference, Polysemy

In Linguistics, the term polysemy refers to the phenomenon where one word carries multiple meanings. In Bil-DOS, polysemy means a little bit different. The easiest example, in the menu, the word ‘*Italian*’ can refer to the keyword ‘*Italian* bread’ with the intention bread, or the keyword ‘*Italian* cheese’ with the intention cheese. However, neither word-level nor phrase level strategy is able to handle this situation. In the mapping process in both strategies, Bil-DOS maintains a hash map to track and store relations, which can mean complex situations like co-reference or polysemy will confuse Bil-DOS and lead to task in-completion.

5.3 Synonym

Due to the lack of ground knowledge, every time when Bil-DOS meets a synonym of any stored keyword, it will emit the human-in-the-loop annotation process. Because, Bil-DOS only analyzes morphology instead of semantics of each word. And it might lead to redundant annotation. Solutions to the lack of ground knowledge in Machine Learning can be rule based methods (Liu and Mei, 2020), pre-trained word embeddings (Pennington et al., 2014; Miller, 1995) or even applying BERT (Devlin et al., 2018) as prior knowledge.

Some translator backends also show no consistency when translating synonyms, which might make the human-in-the-loop annotation process more often to take place. For example, the word ‘*Avocado*’ can be translated to ‘牛油果(can be translated as ‘ao fruit’ by some translators)’ or ‘鰐梨’. If the user has already re-annotated the one of the word and enters its synonym, due to the translation inconsistency, the user has to go through the re-annotation process again.

6 Future Work

We have finished some basic functions in Bil-DOS, and it is capable of conducting conversations with users to help them with their orders. However, Bil-DOS still needs refinements to completely meet user requirements. Here we list several unimplemented ideas to make Bil-DOS better.

6.1 Break Turn-level Analysis

As stated before, currently, all the modules in Bil-DOS support only turn-level analysis, which means it only memorizes the information within each turn and throws the stored information once the turn is finished. So it fails to analyze the relations between different turns. More context information between turns should be considered in the further versions to make Bil-DOS better understand what the user is trying to convey. However, when there are more than one turns that need to be analyzed, the current naive keyword-based search might not work well. Thus, a better or more mathematically sophisticated approach, like LSTM (Hochreiter and Schmidhuber, 1997) or RCNN (Lai et al., 2015), is needed to analyze the semantics, map with certain keywords, and classify into certain intentions.

6.2 All in Two

All targeted users of Bil-DOS are supposed to speak English AND Chinese. Thus, the human-in-the-loop annotation process is currently not in Chinese, which means the process needs the user to be able to use English. In order to serve as many users as possible, Bil-DOS needs to support two different languages in any process at any time. In future versions, we would like to make the system completely bi-lingual, which means all processes in the system are able to take two different languages. By doing this, users will really be able to arbitrarily switch between languages in any process at any time. However, this idea might make more requirements about the translation process and make the execution sequence even more complex.

6.3 Plot More Stories

Although Bil-DOS is able to take all kinds of user utterances with all kinds of keywords and all kinds of intentions, the stored ordering process still remains a fixed story, which is '*bread*', '*cheese*', '*vegetable*', '*sauce*' and '*extra*'. When an irrelevant intention is involved in the user utterance, the intention will be detected and stored locally, and will never be able to participate in the ordering process. So the human-in-the-loop process can be useless for ordering while handling irrelevant intentions. However, we would like Bil-DOS to be capable of handling more stories other than Subway ordering only. In order to do that, Bil-DOS might need another module called `StroyWeaver` to customize the stories for all different situations.

6.4 Teamwork Allocation

Job Done	By Whom
Preliminary Design	Zirong Chen; Haotian Xue
Translator	Haotian Xue
Discriminator	Zirong Chen
Dialogue Manager	Zirong Chen; Haotian Xue
Main Session	Zirong Chen; Haotian Xue
Experiments	Haotian Xue
GitHub Maintenance	Zirong Chen

Acknowledgments

Our work is designed for the final project for the course Dialogue System(COSC 483, Fall 2020) at Georgetown University. We would especially thank Ziyao Ding from Georgetown University, Yuansheng Xie from Dartmouth College, Hanyue Gu from UIUC for their participation in the experiments and the improvements for Bil-DOS. We would also like to thank Professor Matthew Marge for his support and assistance.

References

Bayan AbuShawar and Eric Atwell. 2015. Alice chatbot: trials and outputs. *Computación y Sistemas*, 19(4):625–632.

Julie D Allen, Deborah Anderson, Joe Becker, Richard Cook, Mark Davis, Peter Edberg, Michael Everson, Asmus Freytag, Laurentiu Iancu, Richard Ishida, et al. 2012. The unicode standard. *Mountain view, CA*.

Kirill Buryak and Luke Hiro Swartz. 2015. Rules-based language detection. US Patent App. 13/781,426.

Kirill Buryak, Andrew Swerdlow, Clément Roux, Luke Hiro Swartz, and Cibu Johny. 2015. Cluster-based language detection. US Patent 9,104,744.

Noel Bush. 2001. Artificial lntelligence markup language (aiml) version 1.0. 1: Allce al foundation working drafi. *dalam http://Alicebot.org/TR/2001AffD-aiml*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Jianfeng Gao, Michel Galley, and Lihong Li. 2018. Neural approaches to conversational ai. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 1371–1374.

Agustín Gravano, Julia Hirschberg, and Štefan Beňuš. 2011. Affirmative cue words in task-oriented dialogue. *Computational Linguistics*, 38(1):1–39.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Sherif E Hussein and Malcolm H Granat. 2002. Intention detection using a neuro-fuzzy emg classifier. *IEEE Engineering in Medicine and Biology Magazine*, 21(6):123–129.

Alberto Jimenez-Feltström. 2006. Text language detection. US Patent 7,035,801.

Joeld. 2008. How to print colored text in python? Stack Overflow. URL: https://stackoverflow.com/a/287944 (version: 2020-10-14).

Dan Jurafsky. 2000. *Speech & language processing*, chapter 26 Dialogue Systems and Chatbots. Pearson Education India.

Sang-Bum Kim, Kyoung-Soo Han, Hae-Chang Rim, and Sung Hyon Myaeng. 2006. Some effective techniques for naive bayes text classification. *IEEE transactions on knowledge and data engineering*, 18(11):1457–1466.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

Yoon Kim, Yi-I Chiu, Kentaro Hanaki, Darshan Hegde, and Slav Petrov. 2014. Temporal analysis of language through neural language models. *arXiv preprint arXiv:1405.3515*.

Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. 2015. Recurrent convolutional neural networks for text classification. In *Twenty-ninth AAAI conference on artificial intelligence*.

Bing Liu and Chuhe Mei. 2020. Lifelong knowledge learning in rule-based dialogue systems. *arXiv preprint arXiv:2011.09811*.

Andrew McCallum, Kamal Nigam, et al. 1998a. A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, volume 752, pages 41–48. Citeseer.

Andrew McCallum, Ronald Rosenfeld, Tom M Mitchell, and Andrew Y Ng. 1998b. Improving text classification by shrinkage in a hierarchy of classes. In *ICML*, volume 98, pages 359–367.

George A Miller. 1995. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Stuart Rose, Dave Engel, Nick Cramer, and Wendy Cowley. 2010. Automatic keyword extraction from individual documents. *Text mining: applications and theory*, 1:1–20.

Aixin Sun, Ee-Peng Lim, and Ying Liu. 2009. On strategies for imbalanced text classification using svm: A comparative study. *Decision Support Systems*, 48(1):191–201.

UlionTse. 2020. translators. `https://https://github.com/UlionTse/translators`.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30:5998–6008.

Zi-Qiang Wang, Xia Sun, De-Xian Zhang, and Xin Li. 2006. An optimal svm-based text classification algorithm. In *2006 International Conference on Machine Learning and Cybernetics*, pages 1378–1381. IEEE.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Zhao Xu, Kai Yu, Volker Tresp, Xiaowei Xu, and Jizhi Wang. 2003. Representative sampling for text classification using support vector machines. In *European conference on information retrieval*, pages 393–407. Springer.

Kuo Zhang, Hui Xu, Jie Tang, and Juanzi Li. 2006. Keyword extraction using support vector machine. In *international conference on web-age information management*, pages 85–96. Springer.