FRAMES: A CORPUS FOR ADDING MEMORY TO GOAL-ORIENTED DIALOGUE SYSTEMS

Layla El Asri* **Hannes Schulz*** **Shikhar Sharma*** **Jeremie Zumer***
Justin Harris **Emery Fine** **Rahul Mehrotra** **Kaheer Suleman**

Maluuba Research
Montreal, Canada
first.last@maluuba.com
***these authors contributed equally**

ABSTRACT

This paper presents the *Frames* dataset, a corpus of 1369 human-human dialogues with an average of 15 turns per dialogue. We developed this dataset to study the role of memory in goal-oriented dialogue systems. Based on Frames, we introduce a task called *frame tracking*, which generalizes state tracking to a setting where several states are tracked simultaneously. We propose a baseline model for this task. We show that Frames can also be used to study memory in dialogue management and information presentation through natural language generation.

1 INTRODUCTION

Goal-oriented, information-retrieving dialogue systems have traditionally been designed to help users find items in a database given a certain set of constraints (Henderson et al., 2014; Lemon et al., 2006; Raux et al., 2003; Singh et al., 2002). For instance, the *LET'S GO* dialogue system finds a bus schedule given a bus number and a location (Raux et al., 2003).

These systems model dialogue as a sequential process: the system asks for constraints until it can query the database and return a few results to the user. Then, the user can ask for more information about a given result or ask for other possibilities. If the user wants to know about database items corresponding to a different set of constraints (*e.g.*, another bus line), then these constraints simply overwrite the previous ones. As a consequence, users can neither compare results corresponding to different constraints, nor go back-and-forth between results.

We can assume users in the bus domain know exactly what they want. In contrast, user studies in e-commerce have shown that several information-seeking behaviours are encountered: users may come with a very well defined item in mind, but they may also visit an e-commerce website with the intent to compare items and explore different possibilities (Moe and Fader, 2001). Supporting this kind of decision-making process in conversational systems implies adding *memory*. Memory is necessary to track different items or preferences set by the user during the dialogue. For instance, consider product comparisons. If a user wants to compare different items using a dialogue system, then this system should be able to separately recall properties pertaining to each item.

This paper presents the *Frames* dataset, which comprises dialogues that require this type of memory. Frames is a corpus of 1369 human-human dialogues collected in a Wizard-of-Oz (WOz) setting – *i.e.*, users were connected to humans, whom we refer to as *the wizards*, who were assuming the role of the dialogue system. The wizards had access to a database of vacation packages containing round-trip flights and a hotel. The users were asked to find packages based on a few constraints such as a destination and a budget.

In order to test the memory capabilities of conversational agents, we formalize a new task called *frame tracking*. In frame tracking, the agent must simultaneously track multiple semantic frames throughout the dialogue. For example, two frames would be constructed and recalled while comparing two products – each containing the properties of a specific item. Frame tracking is a general-

ization of the state tracking task (Williams and Zweig, 2016). In state tracking, all the information summarizing a dialogue history is compressed into one semantic frame. In contrast, several frames are kept in memory during frame tracking, with each frame corresponding to a particular context, *e.g.*, one or more vacation packages.

Another important property of human dialogue that we want to study with the Frames dataset is how to provide the user with information on the database. When a set of user constraints leads to no results, users would benefit from knowing that relaxing a given constraint (*e.g.*, increasing the budget by a reasonable amount) would lead to results instead of navigating the database blindly. This recommendation behaviour is in accordance with Grice's cooperative principle (Grice, 1989): "Make your contribution as informative as is required (for the current purposes of the exchange)". We study this by including suggestions when the database returns no results for a given user query.

This paper describes the Frames dataset in detail, formally defines the frame tracking task, and provides a baseline model for frame tracking. The next section discusses motivation for the Frames dataset. Section 3 explains the data collection process and Section 4 describes the dataset in detail. We describe the annotation scheme in Section 5. In Section 6, we identify the main research topics of the corpus and formalize the frame tracking task. The dialogue data format is described in Section 7. Section 8 proposes a baseline model for frame tracking. Finally, we conclude in Section 9 and suggest directions for future work.

2 MOTIVATION

Much work has focused on spoken dialogue (Lemon and Pietquin, 2007; Walker et al., 1998; Williams and Zweig, 2016), since spoken dialogue systems are useful in many settings, including in hands-free environments such as cars (Lemon et al., 2006). A generation of voice assistants – such as SIRI, Cortana, and Google Voice – have popularized spoken dialogue systems. More recently, users have become familiar with *chatbots*. Many platforms for deploying chatbots are now available, such as Facebook Messenger, Slack, or Kik. Text offers advantages over voice such as privacy and the ability to avoid bad speech recognition in noisy environments. Chatbots provide a welcome alternative to downloading and installing applications, and make a lot of sense for everyday services such as ordering a cab or knowing the weather. Chatbots have been proposed for tasks that one would traditionally perform through Graphical User Interfaces (GUIs). For instance, many chatbots for booking a flight are now entering the market.

In most cases, as with current voice-based assistants, the conversation with a chatbot is very limited: asking for the weather and ordering a cab are accomplished with simple, sequential slot-filling. These tasks have in common the fact that in both cases the user knows exactly what she wants, *i.e.*, the destination for the cab or the city for the weather. Booking a flight is a bit different. Flight booking requires specifying many parameters, and these are usually determined during the search process[1]. Technically, finding a weather forecast is only about reading a database: the task is to form a complete database query and then to verbalise the result to the user. The user might start with very few constraints and then refine her query given the database results. In the case of booking a flight, there is a decision-making process requiring comparison and backtracking.

GUIs are not optimal on many levels when it comes to helping users through this decision-making process. A first point of friction is the limited visual space. Consider the example of e-commerce websites. A user is very likely to compare different options before picking an item to buy. This often results in a large number of open browser tabs among which the user must navigate. In order to avoid this situation, some websites provide a comparator that can be used to display several items on a single page. However, this option is not optimal for hierarchical objects such as vacation packages because these objects have global properties (dates of the trips) but are also composed of different modules (flights and hotel) which have different properties (*e.g.*, seat class and hotel category). Optimally, the user should be able to define the properties for the different modules while being able to compare items corresponding to each set of properties. A text interface could complement a GUI by offering this flexibility while remembering the properties mentioned by the user and displaying comparisons when asked.

[1] One can easily imagine a user changing from economy to business class if the price difference is small.

We propose the Frames dataset to support work on text-based conversational agents which help a user make a decision. The decision-making process is tightly coupled with the notion of memory. Indeed, if a user intends to compare different options in the course of defining the options, the system should follow the user's path and remember every option. In this paper, we formalize this aspect of conversation in the *frame tracking* task. This task is the main challenge of the Frames corpus, and Section 6 describes it in detail.

3 DATA COLLECTION

We collected data over a period of 20 days and with 12 participants. To increase variation in the dialogues, 8 participants were hired for only one week each, and one participant was hired for one day. The three remaining participants participated in the entire data collection. The participants were paired up for each dialogue and interacted through a chat interface.

3.1 WIZARD-OF-OZ DATA COLLECTION

Data collection for goal-oriented dialogue is challenging. To control the data such that specific aspects of the problem can be studied, it is common to collect dialogues using an automated system. This requires, *e.g.*, a natural language understanding module that already performs well on the task, which implies possession of in-domain data or the ability to generate it (Henderson et al., 2014; Raux et al., 2003). Another possibility, which permits even greater control, is to generate dialogues using a rule-based system (Bordes and Weston, 2016). These approaches are useful for studying specific modules and analysing the behaviour of different architectures. However, it is costly to generate new dialogues for each experiment and skills acquired on artificial data are not directly usable in real settings because of natural language understanding noise (Bordes and Weston, 2016).

The Wizard-of-Oz (WOz) approach offers a powerful alternative (Kelley, 1984; Rieser et al., 2005; Wen et al., 2016). In WOz data collection, one participant (the wizard) plays the role of the dialogue system. The wizard has access to a search interface connected to the database. She receives the user's input in text form and decides what to say next. This does not require preexisting dialogue system components, except potentially automatic speech recognition for transcribing the user's inputs. Dialogues collected in WOz settings can be used for studying and developing every part of a dialogue system, from language understanding to language generation. They are also essential for offline training of end-to-end dialogue systems (Bordes and Weston, 2016; Wen et al., 2016) on different domains, which may reduce costs from handcrafting new systems for each domain.

WOz dialogues also have the considerable advantage of exhibiting realistic behaviours that cannot be supported by current (end-to-end or not) architectures. Since there is no dialogue system that incorporates the type of memory that we want to study with this dataset, we need to work directly on human-human dialogues. Our setting is a bit different from the usual WOz setting because, in our case, the users did not think they were interacting with a dialogue system but instead knew that they were talking to a human-being. We made the choice not to give templated answers to the wizards because, apart from studying memory, we also want to study information presentation and dialogue management. We have chosen to work on text-based dialogues because this allows a more controlled wizard behaviour, obviates handling time-sensitive turn-taking and speech recognition noise, and allows studying more complex dialogue flows.

3.2 TASK TEMPLATES AND INSTRUCTIONS

Dialogues were performed on Slack[2]. We deployed a Slack bot named *wozbot* to pair up participants and record conversations. The participants in the user role indicated when they were available for a new dialogue through this bot. They were then assigned to an available wizard and received a new task. The tasks were built from templates such as the following:

> "Find a vacation between `[START_DATE]` and `[END_DATE]` for `[NUM_ADULTS]` adults and `[NUM_CHILDREN]` kids. You leave from

[2] www.slack.com

[ORIGIN_CITY]. You are travelling on a budget and you would like to spend at most $[BUDGET]."

Each template had a probability of success. The tasks were generated by drawing values (*e.g.*, BUDGET) from the database. The generated tasks were then added to a pool. The values were drawn in order to comply with the template's probability of success. For example, if 20 tasks were generated at probability 0.5, about 10 tasks would be generated with successful database queries and the other 10 would be generated so the database returned no results for the constraints. This mechanism allowed us to emulate cases when a user would not find anything meeting her constraints. If a task was unsuccessful, the user either ended the dialogue or got an alternate task such as:

"If nothing matches your constraints, try increasing your budget by $200."

We wrote 38 templates. 14 templates were generic such as the one presented above and the other 24 were written to encourage more role-playing from users. One example is:

"Pokemon are now a worldwide currency. You are the best Pokemon hunter in the world. You have caught them all except for one Pokemon worth a fortune: Mewtwo. You heard it was spotted somewhere in [DESTINATION_CITY] and [DESTINATION_CITY]. You want to visit one of these cities, leaving from [ORIGIN_CITY] and starting on or after [START_DATE]. You are leaving with your hunting team and you will be a total of [NUM_ADULTS] adults. You have a budget of [PRICE_MAX]. You want to compare the packages between the different cities and book one, the one that will take you to your destiny."

These templates were meant to add variety to the dialogues. The generic templates were also important for the users to create their own character and personality. We found that the combination of the two types of templates prevented the task from becoming too repetitive. Notably, we distributed the role-playing templates throughout the data collection process to bring some novelty and surprise. We also asked the participants to write templates (13 of them) to keep them engaged in the task.

To control data collection, we gave a set of instructions to the participants. The users received the following instructions:

- Do not use uncommon slang terms, but feel free to use colloquialisms.
- Make up personalities.
- Feel free to end the conversation at any time.
- Try to spell things correctly.
- You do not necessarily have to choose an option.
- Try to determine what you can get for your money.

These instructions were meant to encourage a variety of behaviours from the users. As for the wizards, they were asked to only talk about the database results and the task-at-hand. This is necessary if one wants to build a dialogue system that emulates the wizards' behaviour in this corpus. The wizard instructions were as follows:

- Be polite, and do not jump in on the role play of the users.
- Vary the way you answer the user, sometimes you can say something that would be right at another point in a dialogue.
- Your knowledge of the world is limited by your database.
- Try to spell things correctly.

At the end of each dialogue, the users were asked to rate the cooperativity of the wizards they had talked to on a scale of 1 to 5. The second point in the wizard instruction list was introduced in order to increase variability in these scores. Indeed, it is interesting from a dialogue management point of view to have examples of bad behaviour and of how it impacts user satisfaction.

3.3 Database Search Interface

Wizards received a link to a search interface every time a user was connected with them. The search interface was a simple GUI with all the searchable fields in the database (see Appendix A). For every search in the database, up to 10 results were displayed. These results were sorted by increasing price.

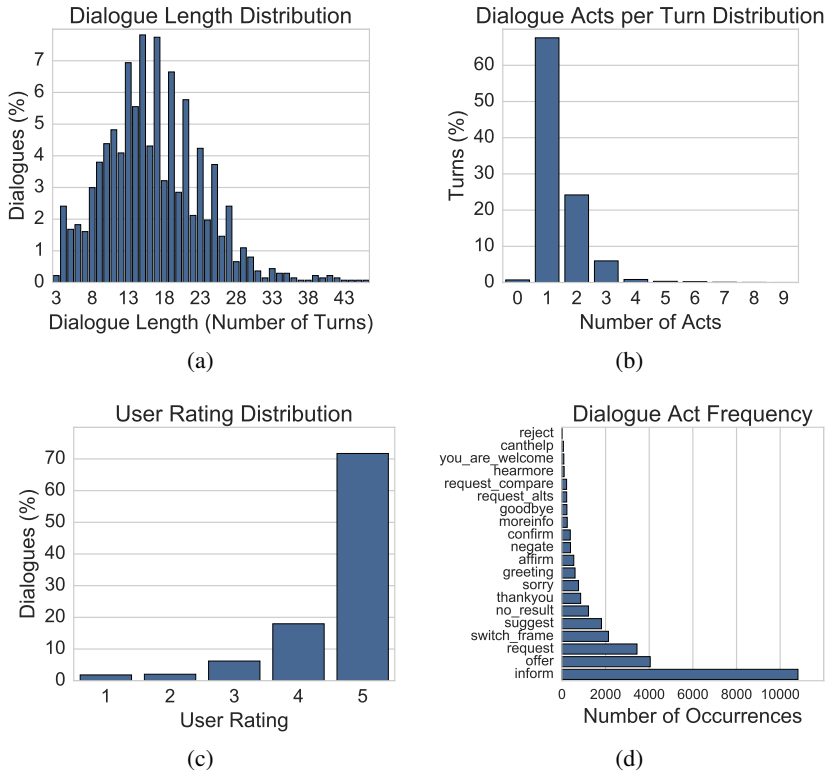

Figure 1: Overview of the Frames corpus

3.4 SUGGESTIONS

When a database query returned no results, suggestions were sometimes displayed to the wizards. Suggestions were packages obtained by relaxing one or more constraints. It was up to the wizard to decide whether or not to use suggestions. In the dataset, the suggestions results are merged with the search results. Our goal with suggestions is not to learn a recommender system, but to learn the timing of recommendation, hence the randomness of the mechanism.

4 STATISTICS OF THE CORPUS

We used the data collection process described in the previous section to collect *1369 dialogues*. Figure 1a shows the distribution of dialogue length in the corpus. The average number of turns is 15, for a total of *19986 turns* in the dataset. A turn is defined as a Slack message sent by either a user or a wizard. A user turn is always followed by a wizard turn and vice versa.

Figure 1b shows the number of acts per dialogue turn. About 25% of the dialogue turns have more than one dialogue act. The turns with 0 dialogue acts are turns where the user asked for something that the wizard could not provide, *e.g.*, because it was not part of the database. An example in the dataset is: "Would my room have a view of the city? How much would it cost to upgrade to a room with a view?". We left such user turns unannotated and they are usually followed up by the wizard saying she cannot provide the required information.

Figure 1c shows the distribution of user ratings. More than 70% of the dialogues have the maximum rating of 5. Figure 1d shows the occurrences of dialogue acts in the corpus. The dialogue acts are described in Table 8 in Appendix B. We present the annotation scheme in the following section.

5 Dialogue Annotation Scheme

We annotated the Frames dataset with three types of labels:

1. Dialogue acts, slot types, slot values, and references to other frames for each utterance.
2. The id of the currently active frame.
3. Frame tracking labels which were automatically computed based on the previous two sets of labels.

5.1 Dialogue Acts, Slot Types, and Slot Values

Most of the dialogue acts used for annotation are acts which are usually encountered in the goal-oriented setting such as `inform` and `offer`. We also introduced dialogue acts which are specific to our frame tracking setting such as `switch_frame` and `request_compare`. The dialogue acts are listed in Table 8.

Our annotation uses three sets of slot types. The first set, listed in Tables 6 and 7, corresponds to the fields of the database. The second set is listed in Table 9 and contains the slot types which we defined in order to describe specific aspects of the dialogue such as `intent` and `action`. An `intent` indicates whether or not the user wants to book a package, whereas an `action` indicates whether or not the wizard should, or did, book it. We also introduced several `count` slot types which were used most often by wizards to summarize information in the database, *e.g.*, "I have 2 hotels in Paris". In this case, the wizard informs that the count for hotels is 2.

The remaining slot types in Table 9 were introduced to describe frames and cross-references between them. Before we discuss these slot types, we define frames more formally in the following section.

5.2 Frames

Each dialogue starts in frame 1. New frames are introduced when the wizard offers or suggests something, or when the user modifies pre-established slots. Frames can be seen as checkpoints in a dialogue, to which the user can return. The frames contain the slot values set by the user, or by information given by the wizard. An example is given in Table 1. In this example, the frame number is changed when the user changes several slot values: the destination city, the number of adults for the trip, and the budget. Though frames are created for each offer or suggestion made by the wizard, the *active* frame can only be changed by the user. If the user asks for more information about a specific offer or suggestion, the active frame is changed to the frame introduced with that offer or suggestion. This change of frame is indicated by a `switch_frame` act (see Appendix B). The rules for creating and switching to frames are summarized in Table 2.

Table 1: Dialogue excerpt with active frame annotation

Author	**Utterance**	**Frame**
User	I'd like to book a trip to boston from London on Saturday, August 13, 2016 for 8 adults. I have a tight budget of 1700.	1
Wizard	Hi...I checked a few options for you, and unfortunately, we do not currently have any trips that meet this criteria. Would you like to book an alternate travel option?	1
User	Yes, how about going to Detroit from London on August 13, 2016 for 5 adults. For this trip, my budget would be 1900.	2
Wizard	I checked the availability for those dates and there were no trips available. Would you like to select some alternate dates?	2

We introduced specific slot types for recording the creation and modification of frames. These slot types are `id`, `ref`, `read`, and `write` (see Table 9 in Appendix B). Each offer or suggestion made by a wizard creates a new frame. The dialogue only switches to that frame if the user considers the

Table 2: Frequency of frame creation and switching events

Rule Type	Author	Rule Description	Frequency	
			Relative	Absolute
Creation	User	Changing the value of a slot	31 %	2092
	Wizard	Making an offer or a suggestion	69 %	4762
Switching	User	Changing the value of a slot (it causes the dialogue to switch to that frame)	50 %	2092
		Considering a wizard offer or suggestion	39 %	1635
		Switching to an earlier frame by mentioning its slot values	11 %	458

offer or suggestion. Therefore, when the wizards makes the offer or suggestion, the new frame gets an `id`. This id is used to switch to this frame when the user decides to do so.

The other slot types – `ref`, `read`, and `write` – are used to annotate cross-references between frames, which are a crucial component of the recorded dialogues. A reference has two parts: the number of the frame it is referring to and the slots and values that are used to refer to that frame (if any). For instance, `ref[1{name=Tropic}]` means that frame 1 is being referred to by the hotel name *Tropic*. If anaphora is used to refer to a frame, we annotated this with the slot `ref_anaphora` (*e.g.*, "This is too long" – `inform(duration=too long,ref_anaphora=this)`). Inside an `offer` dialogue act, a `ref` means that the frame corresponding to the offer is derived from another frame. For example, here is an utterance from the corpus, written by a wizard:

> "Here are a couple of options. The first option is a 3.0 star hotel (the Tropic), with a guest rating of 4.77/10 and a business class flight. The cost is 1002.27 USD. Or, if you prefer, you could choose the same 3.0 star hotel with a guest rating of 4.77/10 (the Tropic) and an economy flight, for 812.69."

This utterance is annotated with the following dialogue acts:

- `offer(category=3.0,name=Tropic,gst_rating=4.77/10,id=6);`
- `offer(ref=[6],seat=business,price=1002.27 USD,id=7);`
- and `offer(ref=[6],seat=economy,price=812.69,id=8).`

Here, the frames corresponding to the last two offers are derived from the first one by inheriting all values.

The slot types `read` and `write` only occur inside a wizard's `inform` act and are used to transfer slot values between frames: `read` is used to indicate which frame the values are coming from (and which slots are used to refer to this frame, if any), while `write` indicates the frame where the slot values are to be written (and which slot values are used to refer to this frame, if any). If there is a `read` without a `write`, the current frame is assumed as the storage for the slot values. A slot type without a value indicates that the value is the same as in the referenced frame, but was not mentioned explicitly *i.e.*, "for the same price".

Table 3 gives an example of how these slot types are used in practice: `inform(` *`read`*`=[7{dst_city=Havana, category=2.5}]` means that the values *2.5* and *Havana* are to be read from frame 7, and to be written in the current frame. At this turn of the dialogue, the wizard repeats information from frame 7.

The annotation `inform(breakfast=False,` *`write`*`=[7{name=El Mar}])` means that the value *False* for breakfast is written in frame 7 and that frame 7 was identified in this utterance by the name of the hotel *El Mar*.

The average number of frames created per dialogue is 6.71 and the average number of frame switches is 3.58. Figure 2 shows boxplots for the number of frame creations and the number of frame changes in the corpus.

Table 3: Annotation example with the `write` and `read` slot types

Author	Utterance	Frame	Annotation
Wizard	I am only able to find hotels with a 2.5 star rating in Havana for that time.	6	inform(*read*=[7{dst_city=Havana, category=2.5}])
User	2.5 stars will do. Can you offer any additional activities?	11	inform(category=2.5)
Wizard	Unfortunately I am not able to provide this information.	11	sorry, canthelp
User	How about breakfast?	11	request(breakfast)
Wizard	El Mar does not provide breakfast.	11	inform(breakfast=False, *write*=[7{name=El Mar}])

5.3 Frame Tracking Labels

Beside dialogue acts, slot types, slot values, and frame annotations, we propose labels for frame tracking. These labels are turn-based and contain four parts:

- User constraints in the currently active frame.
- User comparison requests.
- User requests.
- User binary questions.

The user constraints are all the slots which have been set to a particular value by the user. Any field in the database (see Tables 6 and 7 in Appendix A) can be set by the user.

The other three labels are designed to keep track of user questions. We distinguish three types of questions. The first type, comparison requests, corresponds to the `request_compare` dialogue act. This dialogue act is used to annotate turns when a user asks to compare different results, for instance: "*Could you tell me which of these resorts offers free wifi?*". These questions relate to several frames. The second type of question is user requests, corresponding to the `request` act. These are questions related to one specific frame, for instance "*how much will it cost?*". Finally, binary questions are questions with slot types and slot values, *e.g.*, "*Is this hotel in the downtown area of the city?*" (`request` act), or "*Is the trip to Paris cheaper than to Rome?*" (`request_compare` act), as well as all `confirm` acts.

The next section presents the frame tracking task as well as the metrics we compute using the frame tracking labels.

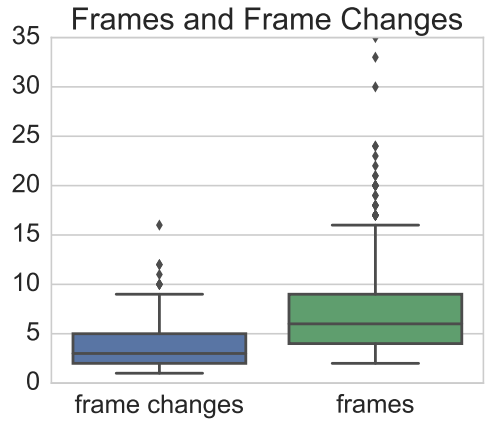

Figure 2: Number of frames and frame switches in the corpus

6 RESEARCH TOPICS

Frames can be used to research many aspects of goal-oriented dialogue, from natural language understanding to natural language generation. In this section, we propose three topics that we believe are new and representative of this dataset.

6.1 FRAME TRACKING

Frame tracking extends state tracking (Henderson, 2015) to a setting where several semantic frames are tracked simultaneously. In state tracking, every new slot value overwrites the previous one. In frame tracking, a new value creates a new semantic frame. The frame tracking task is significantly harder as it requires, for each user utterance, identifying the active frame as well as all the frames which are modified by the utterance.

Definition 1 (Frame Tracking)**.** At each user turn t, we assume access to the full dialogue history $H = \{f_1, ..., f_{n_{t-1}}\}$, where f_i is a frame and n_{t-1} is the number of frames created so far in the dialogue. Given the user utterance for time t, the purpose of frame tracking is to update the list of frames. This requires predicting a new list of frames given the user utterance and the ground truth H.

Predicting an updated frame set given the previous frame set and the current user utterance requires detecting if a new frame is created by this utterance, if values need to be added to previous frames, and if new questions need to be added or removed. Thus, frame tracking requires both performing natural language understanding[3] and properly distributing the slot values among the frames.

We measure the accuracy of frame tracking by comparing the frames described by the ground truth labels (described in Section 5) and the frames output by the frame tracker. We then compute a separate accuracy for each of the four types of frame tracking labels. Section 8 describes our baseline model for frame tracking.

6.2 DIALOGUE MANAGEMENT

One of the notable aspects of this dataset is that memory is not only a matter of frame tracking. Most of the time, the wizard would speak about the current frame to ask or answer questions. However, sometimes, the wizard would talk about previous frames. We can see it as appealing to memories in a conversation. An example is given in Table 4. In the bold utterance in this dialogue, even though the active frame is frame 4, the wizard mentions a previous frame (frame 2). In order to reproduce this kind of behaviour, a dialogue manager would need to be able to identify potentially relevant frames for the current turn and to output actions for these frames.

Table 4 also illustrates another novelty. In the utterance in italic, the wizard actually performs two actions. The first action consists of informing the user about the price of the *regal resort* and the second action consists of proposing another option, *Hotel Globetrotter*. Reinforcement learning has been used with great success for dialogue management (Fatemi et al., 2016; Gašić et al., 2012; Lemon et al., 2006). To our knowledge, there has not yet been a case where several actions could be output in one dialogue turn.

6.3 NATURAL LANGUAGE GENERATION

An interesting behaviour observed in our dataset is that wizards often tended to summarize database results. An example is the wizard saying: *"The cheapest available flight is 1947.14USD."* In this case, the wizard informs the user that the database has no cheaper result than the one she is proposing. To imitate this behaviour, a dialogue system would need to reason over the database and decide how to present the results to the user.

[3] *i.e.*, extracting dialogue acts, slot types, and slot values from the current user utterance.

Table 4: Dialogue excerpt where the wizard talks about a frame other than the active frame.

Author	Utterance	Frame
User	i need a vacation	1
Wizard	How can I help?	1
User	i've got a few days off from august 26-august 31. im not flexible on this, but i still want to somehow treat myself with an 8 day trip (??) im leaving Miami and i wanna check out berlin	1
Wizard	would a 5 day trip suffice?	1
User	sure dude	2
Wizard	A 5 star hotel called the Regal Resort, it has free wifi and a spa.	2
User	dates?	3
Wizard	Starts on august 27th until the 30th	3
User	ok that could work. I would like to see my options in Curitiba as well	4
Wizard	*regal resort goes for $2800 or there is the Hotel Globetrotter in Curitiba it has 3 stars and comes with breakfast and wifi, it leaves on the 25th and returns on the 30th! all for $2000*	*4*
User	ahh I can't leave until august 26 though	4
Wizard	**then i guess you might have to choose the Regal resort**	**4**
User	yeah. I will book it	3
Wizard	Thank you!	3

7 Dataset Format

7.1 Dialogues

We provide the Frames dialogues in JSON format. Each dialogue has five main fields: `turns`, `userSurveyRating`, `user_id`, `wizard_id`, and `id`. The ids are unique for each dialogue (`id`), each user (`user_id`), and each wizard (`wizard_id`). The field `userSurveyRating` is the user rating of wizard cooperativity on a scale of 1 to 5 (see Section 3).

The `turns` have the following fields:

- `author` "user" or "wizard".
- `text` the author's utterance.
- `labels` the id of the currently active `frame` as well as a list of dialogue acts, each with a `name`, and arguments (`args`). Each argument has a list of key-value pairs with a slot type `key`, and an optional slot value `val`. A special value of "-1" is used to express the fact that the user has no preference for the slot. The *False* value is also used to mark empty sets. For instance, `vicinity=False` means that the hotel is not close to any point of interest.
- `timestamp` timestamp for when the message was sent.
- `user_enquiries` (user turns only) requests, comparison requests, and binary questions (see frame tracking labels in Section 5).
- `user_goal_labels` (user turns only) user constraints (see frame tracking labels in Section 5). Note that each slot can have multiple values, which accumulate as long as the frame does not change. For example, price can be both "1000 USD" and "cheapest". Each value has a boolean property "negated", expressing whether the user negated the value of the corresponding slot, for instance *"I don't want to stay 3 days"* (`negate(duration=3)`), or negated an explicit confirmation question. When a user switches to a frame, we assume the user accepts all information provided by the wizard for that frame as "constraints". We drop these additional constraints when a constraint is modified by the user, or the user requests alternatives. Our motivation for this scheme is to make frames more distinguishable and encourage methods which correctly identify frame switches. Additionally to slots and their values, we added the following fields to keep track of specific aspects of the dialogue:
  - `REJECTED` a boolean value expressing if the user negated or affirmed an offer made by the wizard (corresponds to a `negate` act that does not follow a question).

- `MOREINFO` a boolean value expressing whether the user wants to know more about this frame, which happens if the wizard withholds detail information (see `moreinfo` act).

- `db` (wizard turns only) list of search queries made by the wizard with the associated search results.

7.2 HOTELS

The vacation packages were generated randomly. A database of packages can be created by using the search results in the JSON files containing the dialogues. The hotels in these search results have all the fields listed in the *Hotel Properties* section of Table 7 in Appendix A. Note that amenities or points of interest in the vicinity of the hotel are only listed in a hotel's description if they are true. For instance, the field *breakfast* is only present for hotels proposing free breakfast. Figure 3 shows statistics for these boolean values.

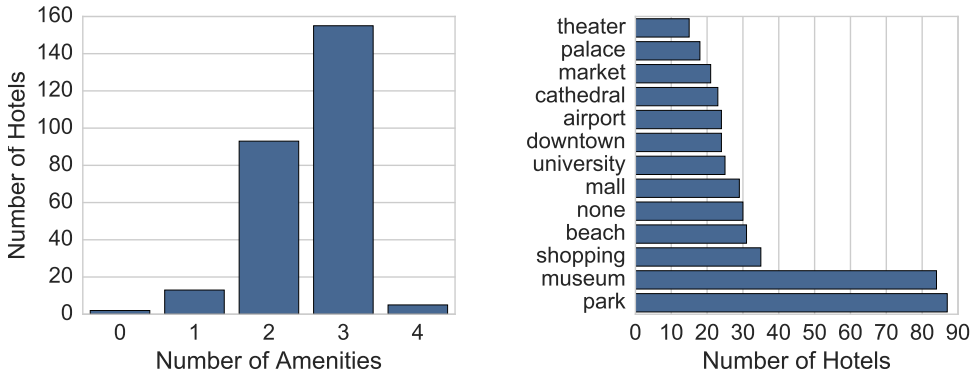

Figure 3: Left – number of amenities per hotel. Right – number of hotels in the vicinity of points of interest (none if the hotel is not close to any point of interest).

8 FRAME TRACKING BASELINE

The proposed baseline for the frame tracking task is built from two components:

- An IOB tagging model to predict user acts, slot types, and slot values.
- A rule-based frame tracker.

The IOB tag format (Inside, Outside, Beginning) is a common word-level annotation format for natural language understanding. A word tagged with *O* means that this word is not part of any slot value. The *B* and *I* tags are used for slot values. Every word which is the beginning of a slot value is tagged with *B* and the *I* tag is used for subsequent words until the end of the slot value. For instance, "I need to go to New York for business" would be tagged as "I (O) need (O) to (O) go (O) to (O) New (B) York (I) for (O) business (O)". We generated these word-level tags by matching the slot values in the manual annotations with the corresponding textual utterances.

8.1 IOB TAGGING MODEL

The IOB tagging model is illustrated in figure 4. It operates on character trigrams and is based on a robust named entity recognition model (Arnold et al., 2016). We modify this model by using a convolutional network to predict, for each word of the utterance, an n-gram tag for different values of n. The model splits into two parts: one part is trained to predict n-gram-wise dialogue acts and the other part is trained to predict n-gram-wise slot types (at this stage, we predict either a slot type or an *O* tag). These two parts share an embedding matrix for the input character trigrams. We found that it was not possible for the model to predict the n-gram-wise dialogue acts without using the slot type prediction task to help drive training of the character trigram embeddings.

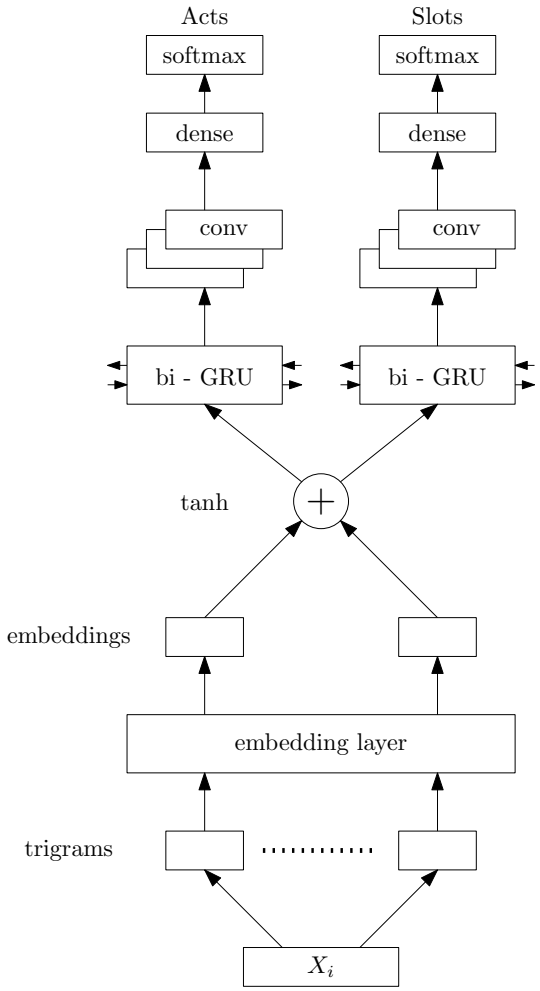

Figure 4: Illustration of the IOB-tagging model for slots and acts prediction, taking input word X_i and outputting labels for slots and acts. The model splits into slots-specific and acts-specific predictors after the word embedding layer (which computes a non-linearity on top of the per-word sum of character trigram embeddings). Convolutional layers compute n-gram outputs for values of n of either 1, 2, or 3, and the predictors output labels for each n-gram.

The two parts of the model are trained simultaneously, using a modified categorical crossentropy loss for either set of outputs. We modify the categorical crossentropy loss to ignore *O* labels that are already predicted correctly by the model. We introduce this modification because *O* labels are far more frequent than other labels, and not limiting their contribution to the loss causes the model to get stuck in a mode where it predicts *O* labels for every n-gram, for any value of n. The loss for the two parts of the model are added together, and the cobined objective is optimized using the ADAM optimizer (Kingma and Ba, 2014).

IOB tags are then extracted from the n-gram-wise slot type predictions by choosing the n-gram prediction for the largest n that is not an *O*, or an *O* if not applicable. The IOB tags served as input for the rule-based frame tracker.

8.2 RULE-BASED FRAME TRACKER

The rule-based frame tracker takes as input the predictions from the IOB tagging model and, according to hand-designed rules, determines which frame-related operations to perform. The following operations are considered:

- *Create a new frame:* if the user informs the wizard of a new constraint for a slot that has previously been filled.
- *Switch to a previous frame:* if the IOB tagging model predicts a `switch_frame` dialogue act.
- *Add information to the current frame:* if none of the above holds.

If neither of the first two conditions hold, then information is added to the current frame. Information is maintained by the frame tracker in two distinct sets: user constraints and wizard information. The user constraints are active constraints set by the user. The wizard information is information related to an offer or suggestion made by the wizard and that the user is currently considering. The frame tracker combines these two types of information: if the user is considering an offer or suggestion, then the currently active frame contains both user constraints and wizard information. If wizard information conflicts with user constraints, the wizard information is kept. The reason for this is that the user chose to consider the offer so it means that her constraints have changed to the ones given in the offer.

In addition, the frame tracker keeps track of any active question asked by the user (see Section 5). A question is considered to have been answered as soon as one of the following conditions holds:

- The wizard gives information regarding the slot that was asked about.
- The wizard negates or affirms.
- The user asks a new question.

For each tracked frame, performance is measured as the proportion of correct slots having approximately correct values over the total number of possible slots. A "correct slot" is a slot that is present in the ground truth frame. An "approximately correct value" is a value which matches semantically, such as "NY" and "New York" or "Five" and "5.0". Similarity is measured with a hand-crafted function. The "total number of possible slots" is the number of slots present in both the predicted frame and the ground truth frame, as well as the slots present in the predicted frame but not the ground truth frame, or vice-versa.

For questions, we compute 3 different scores, corresponding to the 3 possible types of questions (see Section 5). For each type of question, the raw score is the number of correctly identified frames (*i.e.*, frames where this question is active in the ground truth) to which we add the number of correct key-value pairs (*i.e.*, the slot type and, if applicable, slot value in the question). Performance for each question type is normalized by dividing this raw score by the maximum achievable raw score.

Table 5 presents results for this baseline. We tested the model by performing leave-one-user-out testing. We had a total of 11 participants in the user role during data collection. Two participants performed significantly fewer dialogues than the others. We merged the dialogues generated by these two participants. For each of the resulting 10 users, we split the nine others into training (80%) and validation (20%) users, and then tested on the dialogues from the held-out user.

Table 5: Accuracy of the frame tracking baselines.

	Accuracy (%)			
	User Constraints	Comparison Requests	User Requests	Binary Questions
Rule-Based Baseline	60.4	31.3	22.3	25.8
Random Performance	37.0	20.9	19.1	19.2

We compare this baseline model to random performance. Random performance is computed as follows. For user constraints, we randomly select whether or not a slot value is correct. For questions, we randomly select frames (possibly none) from the set of all possible frames. Additionally, when applicable, we randomly change the key for the question being asked. The results in Table 5 show that the baseline model performs significantly better than random on each subtask. Compared to random, the gain in performance on the three types of questions is not high. In general, these results suggest that there is significant room for improvement on frame tracking and that simple rules are far from adequate.

9 Conclusion and Future Work

In this paper we introduced the Frames dataset: a corpus of human-human dialogues for researching the role of memory in goal-oriented dialogue systems. We formalized the frame tracking task, which extends the state tracking task to a setting where several semantic frames are simultaneously tracked throughout the dialogue. We proposed a baseline for this task and we showed that there is a lot of room for improvement. Finally, we showed that Frames can be used to research other interesting aspects of dialogue such as the use of memory for dialogue management and information presentation through natural language generation. We propose adding memory as a first milestone towards goal-oriented dialogue systems that support more complex dialogue flows. Future work will consist of proposing models for frame tracking as well as proposing a methodology to scale up data collection and annotation.

References

Arnold, S., F. A. Gers, T. Kilias, and A. Löser (2016). "Robust Named Entity Recognition in Idiosyncratic Domains". In: arXiv: 1608.06757 [cs.CL].

Bordes, Antoine and Jason Weston (2016). "Learning End-to-End Goal-Oriented Dialog". In: arXiv: 1605.07683 [cs.CL].

Fatemi, Mehdi, Layla El Asri, Hannes Schulz, Jing He, and Kaheer Suleman (2016). "Policy Networks with Two-Stage Training for Dialogue Systems". In: *Proc. of SIGDIAL*.

Gašić, M., M. Henderson, B. Thomson, P. Tsiakoulis, and S. Young (2012). "Policy optimisation of POMDP-based dialogue systems without state space compression". In: *Proc. of SLT*.

Grice, Paul (1989). "Studies in the Way of Words". In: Harvard University Press, Cambridge MA. Chap. Logic and Conversation.

Henderson, Matthew (2015). "Machine Learning for Dialog State Tracking: A Review". In: *First International Workshop on Machine Learning in Spoken Language Processing*.

Henderson, Matthew, Blaise Thomson, and Jason D. Williams (2014). "The Second Dialog State Tracking Challenge". In: *Proc. of SIGDIAL*.

Kelley, John F. (1984). "An iterative design methodology for user-friendly natural language office information applications". In: *ACM Transaction on Information Systems*.

Kingma, D. and J. Ba (2014). "Adam: A Method for Stochastic Optimization". In: arXiv: 1412.6980 [cs.LG].

Lemon, Oliver and Olivier Pietquin (2007). "Machine Learning for Spoken Dialogue Systems". In: *Proc. of Interspeech*, pp. 2685–2688.

Lemon, Oliver, Kallirroi Georgila, James Henderson, and Matthew Stuttle (2006). "An ISU Dialogue System Exhibiting Reinforcement Learning of Dialogue Policies: Generic Slot-filling in the TALK In-car System". In: *Proc. of EACL*.

Moe, Wendy W. and Peter S. Fader (2001). "Uncovering Patterns in Cybershopping". In: *California Management Review*.

Raux, Antoine, Brian Langner, Allan Black, and Maxine Eskenazi (2003). "LET'S GO: Improving Spoken Dialog Systems for the Elderly and Non-natives". In: *Proc. of Eurospeech*.

Rieser, Verena, Ivana Kruijff-Korbayov, and Oliver Lemon (2005). "A corpus collection and annotation framework for learning multimodal clarification strategies". In: *Proc. of SIGDIAL*.

Singh, Satinder, Michael Kearns, Diane Litman, and Marilyn Walker (2002). "Optimizing dialogue management with reinforcement learning: experiments with the NJFun System". In: *Journal of Artificial Intelligence Research* 16, pp. 105–133.

Walker, Marilyn A., Jeanne C. Fromer, and Shrikanth Narayanan (1998). "Learning Optimal Dialogue Strategies: A Case Study of a Spoken Dialogue Agent for Email". In: *Proc. of COLING/ACL*, pp. 1345–1352.

Wen, Tsung-Hsien, Milica Gasic, Nikola Mrksic, Lina Maria Rojas-Barahona, Pei-Hao Su, Stefan Ultes, David Vandyke, and Steve J. Young (2016). "A Network-based End-to-End Trainable Task-oriented Dialogue System". In: arXiv: 1604.04562 [cs.CL].

Williams, Jason D. and Geoffrey Zweig (2016). "End-to-end LSTM-based dialog control optimized with supervised and reinforcement learning". In: arXiv: 1606.01269 [cs.CL].

A Database Overview

Table 6: Searchable fields in the database of packages

Field	Description
PRICE_MAX	Maximum price the user is willing to pay
PRICE_MIN	Minimum price defined by the user
DESTINATION_CITY	Destination city
MAX_DURATION	Maximum number of days for the trip
NUM_ADULTS	Number of adults
NUM_CHILDREN	Number of children
START_DATE	Start date for the trip
END_DATE	End date for the trip
ARE_DATES_FLEXIBLE	Boolean value indicating whether or not the user's dates are flexible. If True, then the search is broadened to 2 days before START_DATE and 2 days after END_DATE.
ORIGIN_CITY	Origin city

Table 7: Non-searchable fields in the database of packages

Field	Description
Global Properties	
PRICE	Price of the trip including flights and hotel
DURATION	Duration of the trip
Hotel Properties	
NAME	Name of the hotel
COUNTRY	Country where the hotel is located
CATEGORY	Rating of the hotel (in number of stars)
CITY	City where the hotel is located
GUEST_RATING	Rating of the hotel by guests (in number of stars)
BREAKFAST, PARKING, WIFI, GYM, SPA	Boolean value indicating whether or not the hotel offers this amenity.
PARK, MUSEUM, BEACH, SHOPPING, MARKET, AIRPORT, UNIVERSITY, MALL, CATHEDRAL, DOWNTOWN, PALACE, THEATRE	Boolean value indicating whether or not the hotel is in the vicinity of one of these.
Flights Properties	
SEAT	Seat type (economy or business)
DEPARTURE_DATE_DEP	Date of departure to destination
DEPARTURE_DATE_ARR	Date of return flight
DEPARTURE_TIME_DEP	Time of departure to destination
ARRIVAL_TIME_DEP	Time of arrival to destination
DEP_AIRP	Airport in origin city
ARR_AIRP	Airport in destination city
DEPARTURE_TIME_ARR	Time of departure from destination
ARRIVAL_TIME_ARR	Time of arrival to origin city
DURATION_DEP	Duration of flight to destination
DURATION_ARR	Duration of return flight

B Dialogue Acts and Slot Types

Table 8: List of dialogue acts in the annotation of Frames

Dialogue Act	Speaker	Description
inform	User/Wizard	Inform a slot value
offer	Wizard	Offer a package to the user
request	User/Wizard	Ask for the value of a particular slot
switch_frame	User	Switch to a frame
suggest	Wizard	Suggest a slot value or package that does not match the user's constraints
no_result	Wizard	Tell the user that the database returned no results
thankyou	User/Wizard	Thank the other speaker
sorry	Wizard	Apologize to the user
greeting	User/Wizard	Greet the other speaker
affirm	User/Wizard	Affirm something said by the other speaker
negate	User/Wizard	Negate something said by the other speaker
confirm	User/Wizard	Ask the other speaker to confirm a given slot value
moreinfo	User	Ask for more information on a given set of results
goodbye	User/Wizard	Say goodbye to the other speaker
request_alts	User	Ask for other possibilities
request_compare	User	Ask the wizard to compare packages
hearmore	Wizard	Ask the user if she'd like to hear more about a given package
you_are_welcome	Wizard	Tell the user she is welcome
canthelp	Wizard	Tell the user you cannot answer her request
reject	Wizard	Tell the user you did not understand what she meant

Table 9: List of slot types not present in the database

Slot Type	Description
count	Number of different packages
count_amenities	Number of amenities
count_name	Number of different hotels
count_dst_city	Number of destination cities
count_seat	Number of seat options (for flights)
count_category	Number of star ratings
id	Id of the frame created (for offers and suggestions)
vicinity	Vicinity of the hotel
amenities	Amenities of the hotel
ref_anaphora	Words used to refer to a frame *e.g.*, "the second package'
impl_anaphora	Used when a slot type is not specifically mentionned *e.g.*, "What is the price for Rio?"..."And for Cleveland?"
ref	Id of the frame that the speaker is referring to
read	Reads slot values specified in another frame and writes them in the current frame
write	Writes slot values in a given frame
intent	User intent (*e.g.*, book)
action	Wizard action (*e.g.*, book)