# CookDial: A dataset for task-oriented dialogs grounded in procedural documents

Yiwei Jiang, Klim Zaporojets, Johannes Deleu, Thomas Demeester and Chris Develder

IDLab, Ghent University – imec, Technologiepark Zwijnaarde 126, 9052 Ghent, Belgium.

*Corresponding author(s). E-mail(s): yiwei.jiang@ugent.be;
Contributing authors: klim.zaporojets@ugent.be; johannes.deleu@ugent.be; thomas.demeester@ugent.be; chris.develder@ugent.be;

**Abstract**

This work presents a new dialog dataset, CookDial, that facilitates research on task-oriented dialog systems with procedural knowledge understanding. The corpus contains 260 human-to-human task-oriented dialogs in which an agent, given a recipe document, guides the user to cook a dish. Dialogs in CookDial exhibit two unique features: (i) procedural alignment between the dialog flow and supporting document; (ii) complex agent decision-making that involves segmenting long sentences, paraphrasing hard instructions and resolving coreference in the dialog context. In addition, we identify three challenging (sub)tasks in the assumed task-oriented dialog system: (1) User Question Understanding, (2) Agent Action Frame Prediction, and (3) Agent Response Generation. For each of these tasks, we develop a neural baseline model, which we evaluate on the CookDial dataset. We publicly release the CookDial dataset, comprising rich annotations of both dialogs and recipe documents, to stimulate further research on domain-specific document-grounded dialog systems.

**Keywords:** dialog system, procedural knowledge, neural network modeling

# 1 Introduction

The last decade has seen a surge of work dedicated to building conversational agents (CA) via annual challenges (e.g., Dialog System Technology Challenges [1]) or benchmark datasets (e.g., WoZ 2.0 [2], MultiWoZ [3], SGD [4]). To provide meaningful responses, such conversational agents typically rely on some form of background knowledge. In question answering (QA), that knowledge often takes the form of descriptive texts from which an answer is distilled (e.g., SQuAD [5]), while in more complicated settings (possibly involving multi-hop reasoning), the system reasons over knowledge graphs (e.g., KdConv [6]). Thus, these question answering works are typically document-grounded. Recent QA works further consider more advanced settings involving multi-turn conversations on a given topic rather than single questions (e.g., CoQA [7] and QuAC [8]), and/or span various domains (e.g., DoQA [9]. ). Logical extensions of this line of work pertain to dialogs including follow-up questions (e.g., ShARC [10]), which may also be initiated by the agent itself and require the system to track both the dialog and the document context (e.g., doc2dial [11]). Going beyond answering user questions, task-oriented dialog systems assist with, e.g., flight, hotel and restaurant reservations, and similarly are grounded in supporting knowledge. The latter typically takes the form of a well-structured database, from which the system retrieves information relevant to the user's task at hand (e.g., WoZ 2.0 [2]).

Despite this growing body of work, we note that an under-explored use case in task-oriented systems is that of assisting the user to follow a given procedure. A notable exception is the work of Raghu et al. [12], which is based on flowcharts describing the procedure. To facilitate further research on procedural assistance dialog systems, we focus on establishing an exemplary dialog dataset and baseline solutions for a conversational agent grounded in textual procedural documents. Thus, our work relates to aforementioned document-grounded systems. Yet, whereas in these state-of-the-art works the documents contain a descriptive text on a particular subject, in our case those documents describe specific instructions. The procedural task dialogs we consider thus concern entities that are manipulated with certain tools, and steps that need to be executed in a particular order, etc.

Specifically, we introduce a **cook**ing **dial**og (CookDial) dataset, comprising conversations where a conversational agent instructs a user to follow a given (textual) recipe. We consider this recipe-based dataset as prototypical for procedural assistance conversations. The CookDial dialogs thus reflect the required capabilities of such a procedural CA, implying understanding the procedure's structure, and particularly the chronological steps it comprises, as well as tracking the user's state in relation to it, i.e., which step the user has reached, and relating the entities/tools in user questions to those in the recipe. Thus, to build such procedural CA system we basically need to solve three fundamental challenges: (C1) understanding the procedural documents (i.e., recipes), (C2) aligning a dynamic conversation with the accompanying document and, (C3) implementing a decision-making process for the CA to generate

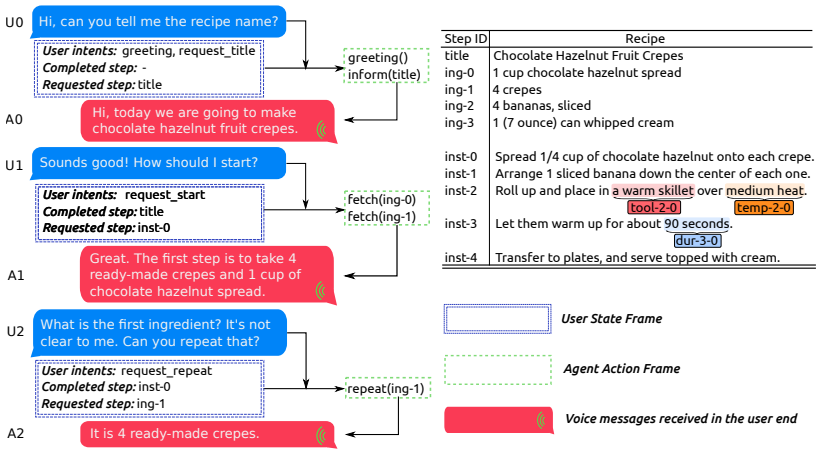


**Fig. 1** The left part shows a cooking dialog sample in which U and A represent the user and agent respectively. The upper right table is the corresponding recipe document. Identifiers `ing` and `inst` denote "ingredient" and "instruction" respectively. These identifiers are listed in the "Step ID" column. Within the recipe text, only three entities are highlighted (see full recipe annotations in Fig. 3). Further, `temp` and `dur` are shorthands for "temperature" and "duration".

the most appropriate utterance, constrained by the grounded document. For challenge (C1), we refer to our earlier work on information extraction from procedural texts [13]. In the current paper, we focus on challenges (C2)–(C3).

To illustrate these challenges, Fig. 1 lists an exemplary conversation for an annotated crepe recipe shown in the right part. Note that we assume the recipe has already been processed [13] to identify the constituent ingredients and instruction steps (respectively marked with `ing-`$i$ and `inst-`$j$), as well as entities such as tools and attributes (e.g., temperature, duration) mentioned within those steps. In challenge (C2), we need to determine the *User State Frame* for each user utterance, i.e, to establish which recipe step the user has completed as well as the step which the user is requesting details about. This will be Task I, User Question Understanding. Next, the decision process of the agent's response is annotated in an *Agent Action Frame*, phrased as a list of at least one function-like string specifying (a) an *agent act*, and (b) an *argument pointer* linking to an item of the recipe (i.e., an ingredient, a full instruction, or an entity within it). We will split the challenge (C3) into two subtasks: Task II will be Agent Action Frame Prediction; Task III, Agent Response Generation, will be to generate natural responses given a user question, the dialog context and extra information from the agent action frame. For each of these tasks we will provide a corresponding baseline model, based on deep neural networks with pretrained language models.

In summary, this paper offers three contributions:

- We create the CookDial dataset, which to the best of our knowledge is the first that grounds the knowledge of dialog systems in procedural documents. The final dataset consists of 260 dialogs and 260 corresponding

recipes (Section 3). The dataset and codes are available at https://github.com/YiweiJiang2015/CookDial.

- We introduce an annotation scheme (Section 4) for such procedural document-grounded dialog systems, in particular, cooking recipe documents. Specifically, we propose symbolic annotations that assign unique identifiers to steps and entities to facilitate the alignment problem between dialogs and grounding documents. We further propose two annotation phases, separating the user state tracking problem and the agent decision-making process.
- We identify three challenging tasks (Section 5) and propose baseline models for each (Section 6), which we evaluate on the CookDial dataset: User Question Understanding (Task I), Agent Action Frame Prediction (Task II) and Agent Response Generation (Task III).

We thus aim to stimulate further research on procedural conversational agent systems, as a subfield of document-grounded dialog models.

**Table 1** Summary of the characteristics of CookDial compared to other document-grounded dialog datasets.

| | CookDial (ours) | doc2dial [11] | QuAC [8] | CoQA [7] | DoQA [9] | ShARC [10] |
|---|---|---|---|---|---|---|
| **Annotation Features** | | | | | | |
| Task-oriented dialogs | ✓ | ✓ | | | | ✓ |
| TTS setting† | ✓ | | | | | |
| Rephrase grounding spans | ✓ | ✓ | | | | |
| User intent annotation | ✓ | | | | | |
| Agent act annotation | ✓ | | | | | |
| Grounding span annotation | ✓ | ✓ | ✓ | ✓ | | |
| **System Abilities** | | | | | | |
| Procedural understanding | ✓ | ✓ | | | | |
| User intent identification | ✓ | | | | | |
| User question grounding | ✓ | ✓ | | | | |
| Agent response grounding | ✓ | ✓ | ✓ | ✓ | | ✓ |
| Document retrieval | | ✓ | | | ✓ | |
| **Statistics** | | | | | | |
| # Dialogs | 260 | 4,800 | 13,594 | 8,399 | 2,437 | 6,637 |
| # Utterances per dialog | 35 | 14 | 7.2 | 15.2 | NA | NA |
| # Grounding documents | 260 | 480 | 8,854 | 8,399 | 1,908 | 948 |

†: TTS, or Text-to-speech, implies that the agent's textual replies are converted to speech messages in the Wizard-of-Oz dialog collection process.; #: Number of; NA: Not available;

# 2 Related Work

As indicated above, we consider a conversational agent (CA) to assist a user to execute a given procedure, as described in a textual document. Thus, our work is a specific use case of a document-grounded dialog system (DGDS). Furthermore, a core capability required by our CA is to grasp the meaning of utterances in the dialog (and relate them to the document). Therefore, our work is also in line with the area of conversational semantic representation. Below we sketch related work in both areas, and position our proposed system against it.

## 2.1 Document-grounded dialog systems (DGDS)

Given that a reasonable fraction of a conversation in our procedural assistant setting amounts to resolving user questions, our work is closely related to the field of conversational question answering. In this area, recently various datasets have been released, including QuAC [8], CoQA [7], DoQA [9] and ShARC [10] .

CoQA and QuAC have similar settings where a multi-turn conversation is grounded in a text passage (e.g., a Wikipedia article). They go beyond reading comprehension challenges like SQuAD [5] by requiring the agent to resolve coreferences, ellipsis and contextual reasoning in the dialog history. Compared to QuAC and CoQA, we note that DoQA is more oriented towards addressing information retrieval needs, as it is collected from FAQ sites and covers multiple domains. In particular, DoQA includes conversational search tasks [14] that require models to retrieve relevant documents across different domains, given a limited dialog context. In ShARC, the information request is further assumed to be underspecified, requiring the CA to pose follow-up questions (based on rules/conditions expressed in the text documents containing the information). This implies that a substantial fraction of user utterances are relatively easy to understand (e.g., many yes/no answers to the CA's follow-up question), meaning that the system complexity mainly lies in understanding the supporting textual documents (i.e., the rules/conditions they express).

Aforementioned QuAC, CoQA and ShARC datasets require the CA to "understand" the textual documents, but mainly on the level of relatively short text spans. The more recent doc2dial [11] is a document-grounded dialog dataset in which addressing the user question necessitates reasoning across paragraph- and document-level structures. Furthermore, rather than pure information-oriented conversations (in the cases of QuAC, CoQA and DoQA), doc2dial is collected in a more goal/task-oriented setting (e.g., how to get certain financial benefits). This setting makes doc2dial the closest to our work. However, since doc2dial grounds dialogs in documents specifying administrative rules/regulations, the proposed models still lack clear understanding of step-wise procedures. Further, we note that the CA system's "understanding" of the supporting textual documents in above datasets closely follows that in machine reading comprehension works [5], by focusing on extracting text spans as answers from documents. However, they show less interest in pragmatic properties of dialogs such as dialog acts, which are important in the semantic

understanding of dialogs. Our work goes further by providing rich annotations of dialog acts, based on which we introduce semantic representations, i.e., user state frames and agent action frames that abstract away from the literal forms (we refer to Section 2.2 for further discussion).

To summarize the characteristics of above DGDS datasets, and how our newly created CookDial compares to them, Table 1 outlines their characteristics in terms of (i) annotation features, (ii) system abilities, and (iii) basic statistics. The unique features of our CookDial dataset are twofold. First, we use text-to-speech (TTS) in the collection process of the dataset, which impacts the dialog flow in that it induces user questions for clarification/repetition. Our motivation for assuming a voice interface is that not having to read the textual instructions avoids distraction from executing the procedural actions. Second, we also include agent act annotations: similar to the user intent, we also include a formal, functional notation of the CA response (cf. the Agent Action Frame in Fig. 1). This action frame consists of (a list of) function(s) that indicate the semantics to express in the response, with arguments grounded in the document (e.g., entities or steps from the recipe). Further, as in doc2dial, the agent annotator is allowed to paraphrase entities/statements of the grounding spans in the documents supporting the conversation (as opposed to purely copying document text spans).

In terms of system abilities, particularly the procedural understanding is both unique and crucial in our procedural CA setting. It implies that the CA needs to be able to clearly identify the distinct steps and their order in the procedure. The only other dataset that to some extent implies some "procedural" notions is doc2dial, which however pertains to identifying certain rules or conditions (e.g., requirements that need to be fulfilled to warrant benefit eligibility) rather than an ordered sequence of distinct actions. Finally, from the statistics it is clear that the size of CookDial is much more modest than others, largely because of time and budget constraints for our data collection. Still, CookDial covers all features but the document retrieval ability and has been carefully designed for building a DGDS with procedural knowledge, thus filling a notable gap in the current body of research works.

## 2.2 Conversational Semantic Representation

The level of "understanding" the user utterances in dialog systems gradually evolved in terms of the complexity of meaning representation. In specific domain contexts, such as task-oriented CAs to assist users in finding/booking restaurants etc., a slot-filling approach is commonly adopted [3, 15–17]: a predefined template of slots is to be filled by string values. To perform the task at hand, these slot values are used to send queries to a database system, whose response then allows the user to make a decision or require further information. This sequential slot-filling process stores dialog states in a flat semantic frame, which eases the data collection of annotations, yet significantly constrains the possible interaction among utterances in a complex dialog flow.

Further research in this domain focuses on representing utterances with richer semantic meanings by moving beyond the flat semantic frames. The authors of [18–20] extend the classical intent-slot framework by using nested intents in slots, thus allowing for more complex queries that contain several intents and corresponding slot-value pairs. These nested structures still rely on the static attributes of the predefined database, comprising, e.g., food type, price, restaurant location. Dialog systems based on such a static database need to fill in these attributes (slots) during the conversation. In fact, the CA cares little about the occurring ordering of the slots, as long as it eventually can form a valid query (e.g., the user can ask for the price and location in any order as he or she likes, without influencing the query result). However, in procedures such as recipes in case of CookDial, the order of the steps is crucial and needs to be strictly followed. Further, entities are not static across this procedure: in our CookDial case, ingredients are manipulated and change state, implying that the agent may need to be aware of these changes (i.e., tracking how the entity evolves throughout the procedure) to answer user queries (as opposed to, e.g., restaurant attributes which remain constant). Also, these procedural "actions", which are formally captured in semantic frames, may have varying numbers and types of arguments, thus implying a greater diversity than aforementioned pre-defined slots in more traditional task-oriented CA cases. This nature of procedural documents makes the development of a dialog system based on CookDial quite challenging.

# 3 CookDial Dataset Description

Our CookDial dataset contains 260 dialogs, each of which corresponds to one recipe document. To generate dialogs, we employed 3 paid workers to work in pairs, conversing using a web chat platform. Workers were asked to switch between the roles of user and agent after each of their conversations. Later, 2 experts performed annotations in two phases: (i) they first annotated all the entity identifiers within each recipe document (see Section 4.1 for more details), (ii) then they annotated the user state frames (Section 4.2) and agent action frames (Section 4.3) for each dialog. In total, it took approximately 600 hours to curate CookDial, of which 350 hours (∼117 hours per worker) were spent on generating the dialog conversations (∼40 minutes per dialog), 50 hours (∼25 hours per expert) for document annotations and 150 hours (∼75 hours per expert) for dialog annotations, excluding about 50 hours of initial annotations to develop the format and guidelines. This time cost reflects the difficulty of collecting human-to-human dialogs, especially in a document-grounded setting.

## 3.1 Recipe collection

We use recipe texts from the RISeC corpus[1] [13], since it contains already annotated recipes, where the entities involved (i.e., ingredients, tools) as well as the actions on them (semantically represented as relations between predicates

[1] https://github.com/YiweiJiang2015/RISeC

and their arguments) already have been rigorously identified. In CookDial, we extend the original RISeC annotations by assigning unique identifiers to entities. See Section 4.1 for more details.

## 3.2 Dialog collection

As indicated previously (Section 2), a substantial body of recent work has considered document-grounded dialog systems (DGDS), but largely focused on information-oriented conversations, mostly in question answering settings. Conversely, we focus on task-oriented dialogs, specifically focusing on procedural tasks involving a sequence of steps to complete the task, grounded in a document describing these steps textually. Therefore, we collected the data with two objectives in mind: (i) grounding dialogs in documents, i.e., recipes, and steering the conversation towards a specific goal, i.e., complete the cooking process; and (ii) collecting high-quality conversations that can be used for building a dialog system or its components.

To fulfill the first goal, we adopt the recipe corpus from our previous RiSEC work, given its already annotated entities and actions performed on them, allowing linking them further to the conversation utterances in the dialogs. In total, we gathered 260 dialogs, using the following procedure.

We adopted the Wizard-of-Oz method in which two workers talked to each other via a web chat platform. Each cooking conversation lasted for 20-40 minutes, depending on the length and difficulty of the recipe. We collected exactly one dialog per recipe. One of the workers mimicked the 'agent', thus having full access to a recipe text, and was responsible for assisting the other annotator, namely the 'user', to accomplish the cooking. In the remainder of this section, we will use 'agent' and 'user' to represent the corresponding workers. The user knew nothing about the recipe except its title. To complete the cooking task, the user had to keep asking questions until the recipe was done. (Note that in practice, the user did not actually perform any cooking — we simply asked the annotators to imagine an actual cooking setting as accurately as possible.)

In our chatting application, the agent's textual input was converted to speech messages (by using a free text-to-speech service[2]) while the user's input remained as is, in textual form. Such configuration resembles popular chatbot services like Alexa, Google Assistant and Cortana. This sometimes caused the 'user' to have difficulty fully understanding utterances from the 'agent', either because of text-to-speech quality issues or the complexity of the sentence/word itself. This impaired comprehension then drove the 'user' to ask questions for clarification or repetition. Further, to offer a more understandable answer, the 'agent' resorted to segmenting or paraphrasing the original recipe instructions. In this way, the generated dialogs benefited from these clarification questions in terms of enhanced language diversity and non-linearity of dialog flows. In contrast, in pilot experiments without the TTS setting, 'user' annotators were found to lazily ask mostly trivial procedural questions (e.g., "What is the next

[2] https://responsivevoice.org/

step?" or "What should I do now?"). Such lack of creative questions rendered the dialogs too linear and straightforward, and less representative of dialogs we can expect to encounter in the wild.

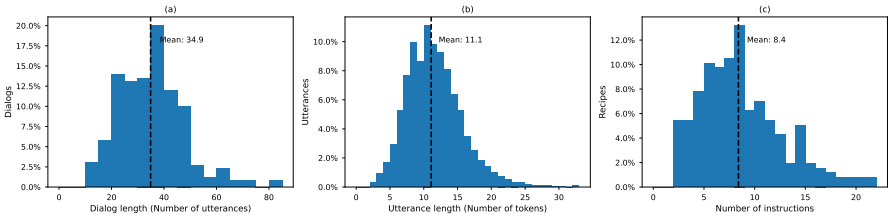


**Fig. 2** Statistics of dialogs and recipe documents in CookDial: (a) dialog distribution by the dialog length, (b) Utterance distribution by the utterance length, (c) Recipe distribution by the number of instructions.

After the first round of dialog generation, the expert annotators cleaned the data by fixing typos, wrong entities, etc., as marked by annotators after their session was finished or by our expert annotators. The final step was to merge the sentences from the same speaker into one utterance, as the user or agent sometimes sent a follow-up message to correct typos or mistakes. Fig. 2 shows statistics of the 260 dialogs and recipes. The average dialog length is 34.9 utterances. There are 9,068 utterances in total, with an average length of 11.1 tokens per utterance.

# 4 Annotation Details

This section introduces the annotation schema for our corpus regarding the recipe texts and the dialogs. For the recipes, we extend the original RISeC dataset [13] by assigning an identifier to each entity, in order to facilitate the dialog annotation, as described in Section 4.1. For the dialogs, we annotate both the user's and agent's utterances using dialog acts and entity identifiers from the recipe annotations.

## 4.1 Recipe annotation

The RISeC corpus [13] contains entity and predicate-argument relation annotations on 260 recipe texts. It focuses on revealing semantic structures within the instruction part of recipes, while ignoring the ingredient list that actually provides basic entities (i.e., ingredients and their quantities, or their mentions as part of descriptive instructions further down the recipe) for dialogs. However, in the context of recipe-grounded dialogs, these entities are key, since user-issued questions are often entity-centric (e.g., "Could you tell me the recipe title?" and "What is the next ingredient?"). Moreover, the user often asks for a whole instruction from the recipe (e.g., "That is done. What shall I do now?"), which cannot be uniquely identified by the original RISeC annotations.

To address these limitations, we extend the original RISeC annotations by assigning unique identifiers to title, ingredients, and instructions, as well as to the entities within the instructions. Fig. 3 shows a fully annotated recipe in which all identifiers are visualized in colored rectangular boxes. The title "California Chicken" is tagged as `title`, followed by the list of ingredients of which each ingredient (typically including an amount and description) is assigned an identifier formatted as `ing-`$i$, where $i$ indicates the rank of the considered ingredient in the list. For example, the last ingredient `ing-7` in Fig. 3 corresponds to "1 package Monterey Jack cheese". The ingredient list is followed by a list of instructions, each assigned an identifier `inst-`$j$, where $j$ denotes the instruction's index in the ranked list of instructions. In the instruction part, all the mentions of ingredients are assigned the corresponding identifier from the ingredient list, even when paraphrased or slightly changed. For example, the description "1 to 2 slices of tomato" is annotated as `ing-6`, referring to the ingredient originally introduced as "2 ripe tomatoes, sliced". Note that the RISeC originally introduces a number of other entity types for entities that appear in the instructions, such as `Action`, `Tool`, `Temperature`, etc. [13], as shown in Fig. 3(b). We extend these original annotations into identifiers formatted as `entity_type-`$j$-$k$, in which $j$ again refers to the instruction index (i.e., the same rank as in the identifier of the instruction where the entity mention appears), and $k$ is the rank of entities of the same type within that instruction. For example, in the first instruction `inst-0`, there are two temperature expressions, i.e., "350 degrees F" and "175 degrees C", which are assigned labels `temp-0-0` and `temp-0-1`, respectively. To wrap up, there are four sets of identifiers within each recipe: title identifier, ingredient identifier, instruction identifier and extended entity identifier.

In order to facilitate the connection between dialog and recipe annotations in the following sections, and for clearly defining prediction targets in Section 6, we define two supersets of identifiers: (i) the step identifiers, aggregating title, ingredient, and instruction identifiers, and (ii) the full set of all recipe identifiers comprising the step identifiers as well as the extended entity identifiers.

## 4.2 User question annotation

Understanding a user's utterance (which we will denote freely as a *question*) usually requires converting a natural utterance into a structured representation, i.e., a *User State Frame*. Its annotation task is structured as follows.

The first level of user question annotations identifies one or more suitable user intents. We considered 24 user intents listed in Table 2, some of which are borrowed from the work of [21] (e.g., greeting, confirm, negate) while others are designed specifically for the cooking dialogs (e.g., req_temperature, req_ingredient). Fig. 4(a) shows the relative frequency of each intent. The top five intents (confirm, req_instruction, req_ingredient, thank and other) together take up 75% of the total distribution while the others fall in the long tail region. A detailed description of all these user intents is listed in Table B.1, Appendix B.

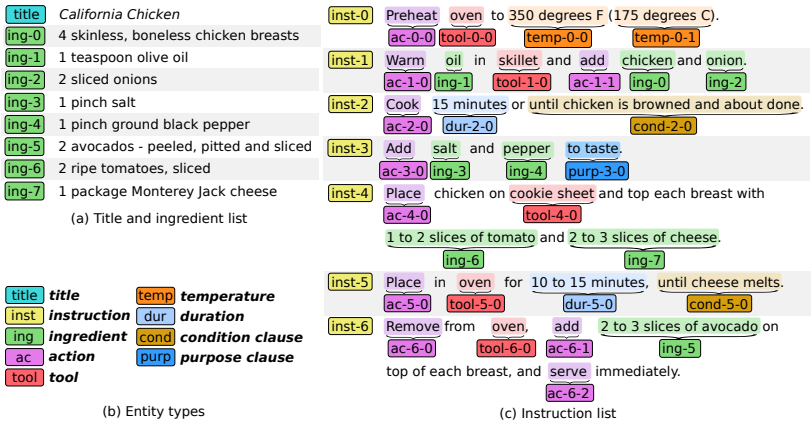


**Fig. 3** A fully annotated recipe text (viewing in color is recommended).

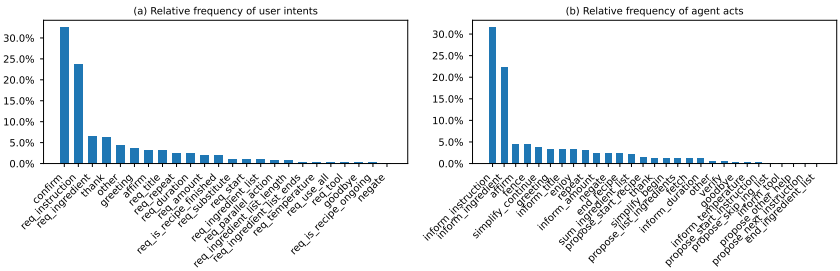


**Fig. 4** Statistics of user intents and agent acts.

The second level of user question annotations is designed to support user state tracking, and consists of annotating the 'completed step' and 'requested step' associated with the considered utterance. The 'completed step' annotation refers to the step the user has just completed or understood, whereas the 'requested step' denotes the step most directly following the user request. Both are annotated as a step identifier in the recipe (which can be the title, an ingredient, or an instruction ID, as explained in the previous section). For example, U1 in Fig. 1 can be interpreted as the user acknowledging the `title` (completed step), and asking for the first instruction `inst-0` (requested step).

## 4.3 Agent response annotation

Besides the user's questions, we also annotate the agent's responses, using a different representation: each response is labeled with an action frames consisting of an agent act and, as an argument, an optional pointer to a particular element in the recipe, under the form of an identifier from the full set of recipe identifiers. For example, simple phrases like "hello" and "yes" are annotated as greeting() and affirm() without any argument pointer. Taking A1 in Fig. 1 as another example, we annotate the response "Great. The first

**Table 2** dialog acts for the user and agent annotation. †: pointers are compulsory. ‡: no argument pointer.

| | Dialog Acts |
|---|---|
| User Intents | greeting, thank, confirm, negate, other, affirm, goodbye<br>req_start, req_temperature, req_instruction, req_repeat<br>req_amount, req_ingredient, req_use_all, req_title<br>req_is_recipe_finished, req_tool, req_duration<br>req_is_recipe_ongoing, req_substitute<br>req_ingredient_list, req_ingredient_list_length<br>req_ingredient_list_ends, req_parallel_action |
| Agent Acts† | inform_instruction, inform_ingredient, inform_title<br>inform_duration, inform_temperature, inform_amount<br>inform_tool, fetch, repeat, verify, simplify_begin<br>simplify_continue |
| Agent Acts‡ | greeting, goodbye, affirm, negate, enjoy, fence, end_recipe<br>thank, other, count_ingredient_list, propose_start_recipe<br>end_ingredient_list, propose_list_ingredients<br>propose_skip_ing_list, propose_next_instruction<br>propose_start_instruction, propose_other_help |

step is to take 4 ready-made crepes and 1 cup of chocolate hazelnut spread." as fetch(`ing-0`); fetch(`ing-1`)". Pointer annotations, i.e., `ing-0` and `ing-1`, are essential in this case since the agent act fetch alone cannot grasp the full semantic meaning of this response. Notice that the number of agent acts contained in agent action frames varies among different responses: for example, in Fig. 1, A0 and A1 have two acts while A2 only has one. Table 2 lists the 29 agent acts categorized by whether they need an argument pointer or not. Again, we borrow some dialog acts from [21] (e.g., affirm, negate) while others are invented specifically for the cooking domain (e.g., propose_start_recipe). We label the sentences that do not belong to any act as other. Detailed descriptions of the used agent acts are listed in Table B.2, Appendix B. The relative frequency of agent acts in Fig. 4(b) shows that inform_instruction and inform_ingredient are the major acts used by the agent.

As we will validate in Section 6.3, not only do the pointer annotations ground the agent answers, but they also help to improve the response generation quality in the final system. Since the argument pointers rely on the full set of recipe identifiers, the agent action frame annotations are grounded in a more fine-grained level of recipe texts compared to the step trackers in user state frame annotations.

## 4.4 Dialog flow

Since recipes comprise a sequence of ordered steps, the dialog is expected to roughly follow the steps of the corresponding recipe linearly. Fig. 5 illustrates such linearity by counting and normalizing the occurrences of (a) the requested recipe step identifier vs. the dialog turn number, and (b) the requested recipe step identifiers of a pair of two consecutive user questions. Note that the recipe step identifiers in Fig. 5 refer to a concatenation of the title and the instructions (i.e., ingredients are not considered) as the plotted statistics[3] were obtained from 86 dialogs in which the agent skips introducing the ingredient list.

Interestingly, most of the mass in the cone-shaped area marked in Fig. 5(a) lies below the diagonal line. If the mass entirely followed the diagonal line, it would imply that the user never has any problem in understanding the agent's utterances and just keeps asking "What is next?" through the conversation, which is apparently unrealistic. Instead, the user's continuous requests for clarification cause the distribution's mass to diverge from the diagonal line and shift downwards. Fig. 5(b) displays the transition between requested recipe step identifiers for the current user question and the next one. The two distinctive diagonal lines reflect that the grounding knowledge for the next question is mostly either in the same recipe step as the current question or in the step immediately following it.

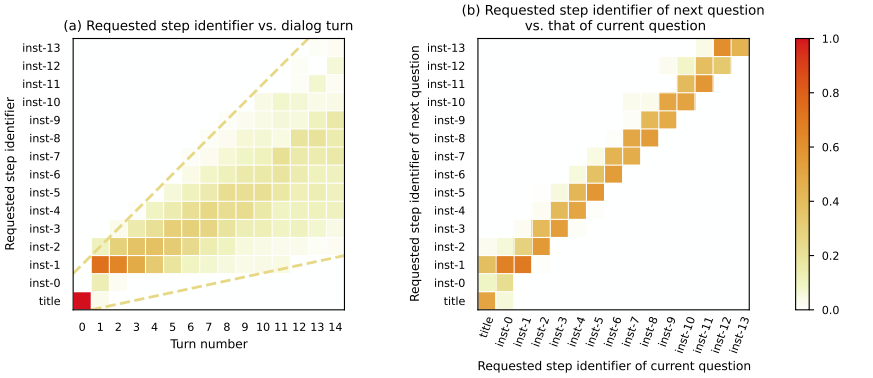


**Fig. 5** Heatmaps illustrating the dialogs flow along with the recipe steps. (a) Requested recipe step identifier vs. dialog turn number; (b) The next turn's requested recipe step identifier vs. the current turn's. (a) and (b) share the same color scale. The lengths of dialog turns (i.e., a pair of user and agent utterances) and recipe steps are truncated to 15 for simplicity.

[3]We performed vertical normalization on each cell by dividing its frequency by the sum of all the cell frequencies in the same column.

# 5 Task Definitions

As stated in Section 1, we focus on solving two challenges during the development of our CA system, i.e., challenge (C2), aligning a conversation and its document, and challenge (C3), simulating the CA's decision-making process to generate proper responses. We introduce three tasks to tackle these challenges as well as to assess the required language understanding abilities in a procedural document setting. Note that the challenge (C3) is split into two tasks, i.e., Task II and Task III. The definition of each task is listed below.

### Task I: User Question Understanding

Understanding a user's question requires the agent to resolve the user's intents and track the dialog state, i.e., the annotated *User State frame*. Every state frame contains at least one intent $\mathbf{y}_I$ and two state trackers pointing to the completed step $\mathbf{y}_C$ and the requested step $\mathbf{y}_R$ respectively. Predicting the requested step can be quite challenging, as it may refer to the dialog context rather than the document. For example, the question U2 in Fig. 1 asks for repeating the first mentioned ingredient in A1.

Formally, the input of Task I includes (1) the user question $Q$, (2) its dialog history $H$, i.e., a given number of utterances preceding the current user question that the CA has to respond to, and (3) the grounding recipe document $D$ including the span boundary indices of step identifiers $\{(s_j^{start}, s_j^{end})\}_{j=1}^{N_S}$ (where $N_S$ denotes the number of step identifiers in a recipe). (Note that we will experiment with a varying number of preceding turns for the history, see further.) The targets are an unordered set of intents ($\mathbf{y}_I$) and two state trackers (completed step $\mathbf{y}_C$, requested step $\mathbf{y}_R$). Predicting user intents can be viewed as a multi-label classification problem. The state tracker prediction is formulated as a span selection problem. Unlike the typical question answering task, e.g., SQuAD [5], our model does not predict the start or end indices of a span from a document. Instead, as the recipe step span indices are provided a priori in our setting, we just need to select the most likely one from these given candidate spans. Note that for each dialog, the number of span candidates varies with the recipe document.

### Task II: Agent Action Frame Prediction

As described in Section 4.3, every agent action frame is composed of agent acts (e.g., greeting) and argument pointers, i.e., identifiers of the grounding spans. Predicting the agent action frame can be viewed as end-to-end modeling of the agent's decision-making process in response to the user's questions. Although seemingly similar to Task I, Task II differs in two aspects: (i) the sequential order of agent acts (whereas user intent order is less important); (ii) whether or not there need to be argument pointers depends on the agent acts (i.e., some acts do not take any arguments).

Like Task I, the input of Task II includes (1) the user question $Q$, (2) the dialog history $H$ and (3) the recipe document $D$ along with the span indices of

the full set of all recipe identifiers $\{(s_j^{start}, s_j^{end})\}_{j=1}^{N_F}$ (where $N_F$ denotes the number of full set of all recipe identifiers in a recipe). The targets consist of agent acts $\mathbf{y}^{act}$ and the corresponding argument pointers $\mathbf{y}^{frag}$ in which we use a "dummy span" to pad those positions without any pointers (also called *null pointers*).

**Task III: Agent Response Generation**

Assuming that the grounding text spans are already given for agent turns, this task focuses on generating a natural language response. This agent response generation is challenging because (i) in many cases, the model needs to paraphrase the imperative expressions from recipes into different forms (e.g., "Preheat oven to 220 °C." → "Can you preheat the oven to 220 °C?"), and (ii) the system is supposed to resolve the coreferences within a dialog, e.g., "Could you repeat the last ingredient?".

Task III takes as input: (1) the user question $Q$, (2) its dialog history $H$, (3) a **Prompt** composed of agent acts of the agent's response, and (4) the argument pointer span tokens $G$ as the context. The target is the agent's gold response.

# 6 Baseline Models and Empirical Results

For each task in Section 5, we propose a baseline model leveraging neural networks and pre-trained language models. Detailed experimental settings and result analysis can be found in the subsequent subsections. For all our baseline models, the empirical results are averaged over 5 runs, each with a different random seed used for model weight initialization and mini-batch sampling. To reduce the impact of selection bias due to the limited number of conversations, a different random partition into train/dev/test sets (distributed as 80%, 10%, 10%) is constructed for each of these runs. Since each dialog is linked to a unique recipe text, we ensure that none of the recipes in the dev/test sets are part of the train set.

## 6.1 User Question Understanding (Task I)

Fig. 6(a) sketches our baseline model for Task I. For all the training instances, the grounding document $D$ is appended after the dialog history $H$ and user question $Q$ in the input. We experiment with the number of history utterances, denoted by $\#H$, ranging from 0 to 10. Note that our model in Fig. 6(a) takes the utterances in a chronological order, while doc2dial [11] also experimented with a reversed setting, in which the user question and dialog history are concatenated in reversed time order, i.e., the latest user question appears first in the input. Although doc2dial claimed that the reversed input outperformed the normal one, our pilot experiment results revealed that there was no significant difference between these two settings in our dataset.

The vanilla BERT encoder has a limited input size (i.e., there is a 512 token limit), which does not suffice to contain longer full documents. A common

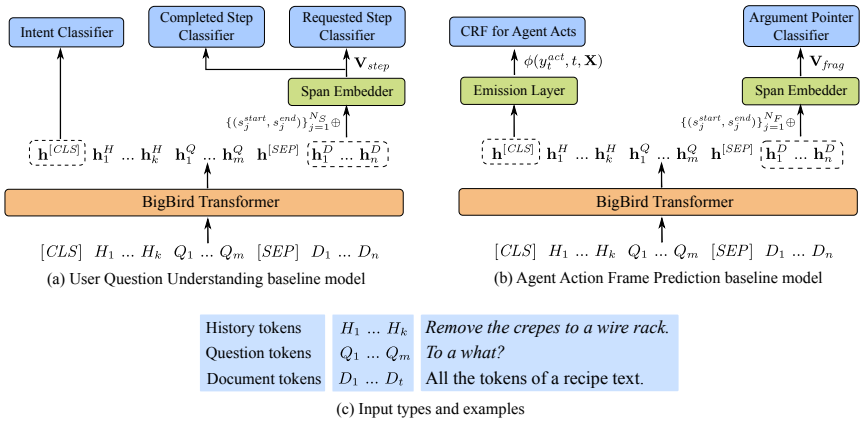


**Fig. 6** Baseline models for (a) Task I: User Question Understanding (b) Task II: Agent Action Frame Prediction. $H, Q, D$ denote tokens of dialog history, user question, and entire recipe text respectively. Example inputs are given in (c).

solution to circumvent this limitation is to segment the long input document and adopt a sliding window approach to do so [11, 22]. Since this means that a single original training instance is converted to multiple ones each covering part of the long input document, this implies a non-negligible computational overhead. Simply truncating the document would clearly lead to discarding potentially useful input information. Therefore, we use BigBird transformer [23] as the encoder, which has the advantage of sparse attention, which reduces the computation complexity from $O(n^2)$ to $O(n)$ in case of long inputs. As a result, the model can handle long documents more efficiently. Training instances are fed in batch to the encoder, which transforms tokens to hidden representations $\mathbf{h} \in \mathbb{R}^{b \times m \times s}$, as per Eq. (1); here, $b, m, s$ denote the batch size, length of input tokens and hidden size. We use the [CLS] token embedding $\mathbf{h}^{[CLS]} \in \mathbb{R}^{b \times m \times s}$ to represent the query vector. The document token embeddings $\{\mathbf{h}_i^D\}_{i=1}^{N_D}$ are passed to a span embedder ($N_D$ is the total number of document tokens), which concatenates the start and end token embeddings indexed by the step span tuples $\{(s_j^{start}, s_j^{end})\}_{j=1}^{N_S}$ to form step span vectors $\mathbf{V}_{step} \in \mathbb{R}^{N_S \times 2s}$.

$$\mathbf{h} = \text{BigBird}(H, Q, D) \tag{1}$$

$$\mathbf{V}_{step} = \text{SpanEmbedder}(\{\mathbf{h}_i^D\}_{i=1}^{N_D},\ \{(s_j^{start}, s_j^{end})\}_{j=1}^{N_S}) \tag{2}$$

On top of the encoder, a feed-forward neural network (FFNN)[4] maps $\mathbf{h}^{[CLS]}$ to a logit vector of which each dimension represents the unnormalized classification score of an intent. Given the multi-label nature of the user intent classification task, the probability $\hat{p}_I$ of each intent $I$ is modeled by applying a sigmoid function on the corresponding logit, denoted by Eq. (3). After the span embedder, span vectors are sent to two independent FFNNs and softmax layers

[4]By default, all FFNNs in this work are composed of 1 hidden layer activated by the GELU function and 1 output layer.

that predict the probability distributions $\hat{p}_c$, $\hat{p}_r$ for completed and requested state trackers, respectively, shown in Eqs. (4)–(5).

$$\hat{p}(I|X;\ \theta) = \text{sigmoid}(\text{FFNN}(\mathbf{h}^{[CLS]})) \tag{3}$$

$$\hat{p}(C|X;\ \theta) = \text{softmax}(\text{FFNN}(\mathbf{V}_{step})) \tag{4}$$

$$\hat{p}(R|X;\ \theta) = \text{softmax}(\text{FFNN}(\mathbf{V}_{step})) \tag{5}$$

During training, the model computes the cross entropy losses of the 3 probability distributions against their corresponding target distributions $p(\mathbf{y}_I), p(\mathbf{y}_C), p(\mathbf{y}_R)$. We implement joint training by minimizing the sum of three losses. For a given training instance $\mathbf{X}$, the local loss is:

$$\mathcal{L} = -\left[p(\mathbf{y}_I) \cdot \log \hat{p}(I|X;\ \theta) + p(\mathbf{y}_C) \cdot \log \hat{p}(C|X;\ \theta) + p(\mathbf{y}_R) \cdot \log \hat{p}(R|X;\ \theta)\right] \tag{6}$$

We use the F1 score for the intent prediction evaluation. The performance of state tracking is evaluated using the accuracy metric.

## Experimental Results

From Fig. 7, we note that the number of history utterances has less influence on the intent prediction than that on the accuracy of state tracking. The F1 score of predicting user intents stays around 91% for the test set. This stationary performance suggests that a single user question is informative enough for the model to infer the user's intent. The fact that there is no drastic drop of the F1 score when the history length is 0 also validates this assumption.

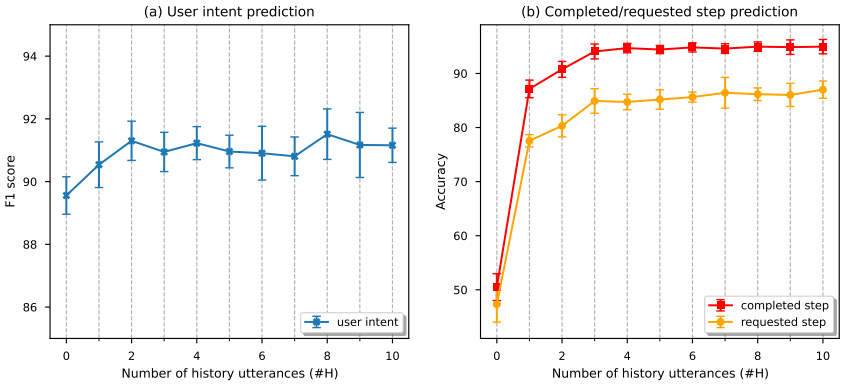


**Fig. 7** Experiment results of Task I User Question Understanding. The results are averaged over 5 runs with different randomly sampled splits of CookDial.

As for state tracking on the test set, the accuracy increases to 86.4% and 94.5% for the requested step and completed step tracker, respectively, by adding more history utterances. However, the F1 scores of both step tracker predictions

start plateauing from $\#H$=3 onward. This implies that, as intuitively expected, understanding a user question benefits little from its distant context. We note that predicting the completed step reaches higher performance than the requested step, which stems from the fact that the user's request can be rather ambiguous. For example, when the user is asking "What is next?", they may request an ingredient or instruction depending on the dialog context. Further, as expected, without any history ($\#H$=0) the model largely fails at state tracking (with accuracy dropping to around 50%).

## 6.2 Agent Action Frame Prediction (Task II)

Fig. 6(b) gives an overview of the baseline model for Task II. We again use BigBird as the encoder but fine-tune it independently from Task I. The span embedder creates a zero vector $\mathbf{V}_z$ as the dummy span embedding to represent null pointers. In addition, given the argument span tuples, the final output of the span embedder is $\mathbf{V}_{frag} = \{\mathbf{V}_z, \hat{\mathbf{V}}_{frag}\}$, where $\hat{\mathbf{V}}_{frag}$ is an end-point span vector.

$$\mathbf{h} = \text{BigBird}(H, Q, D) \tag{7}$$

$$\mathbf{V}_{frag} = \{\mathbf{V}_z, \hat{\mathbf{V}}_{frag}\},$$
$$\text{where } \hat{\mathbf{V}}_{frag} = \text{SpanEmbedder}\left(\{\mathbf{h}_i^D\}_{i=1}^{N_D},\ \{(s_j^{start}, s_j^{end})\}_{j=1}^{N_F}\right) \tag{8}$$

Again, the [CLS] token embedding $\mathbf{h}^{[CLS]}$ is used as the query vector. To predict the agent act sequence, we use conditional random fields (CRFs) [24] to model the dependency between agent acts.

Since in our dataset we observe at most 4 agent acts in the agent response, we simplify the model by fixing the CRF output length as 5 (appending an ¡eos¿ token after an act sequence). The potential function in the CRF layer is decomposed into an emission function $\phi$ and a transition function $\psi$. For the emission function $\phi$, we train an FFNN layer to transform the contextual vector $\mathbf{h}^{[CLS]}$ into the emission scores $\phi(y_k^{act}, k, \mathbf{X})$. The transition function $\psi$ is represented by a trainable matrix $\mathbf{W}_s$ maintaining the transition score between agent act tags and two extra tags i.e., ¡start¿ and ¡end¿. Finally, the predicted probability for an agent act sequence is computed as Eq. (10). As for the argument pointer prediction, we pass the argument pointer span vector $\mathbf{V}_{frag}$ to another FFNN layer. The output logit is then activated by a softmax function to predict the probability distribution $p(\mathbf{y}^{frag}|\mathbf{X};\ \theta)$ of the argument pointers, as shown in Eq. (11).

$$\phi(y_k^{act}, k, \mathbf{X}) = \text{FFNN}(\mathbf{h}^{[CLS]}), \quad \psi(y_{k-1}^{act}, y_k^{act}) = \left(y_{k-1}^{act}\right)^T \mathbf{W}_s\, y_k^{act} \tag{9}$$

$$\hat{p}(\mathbf{y}^{act}|\mathbf{X};\ \theta) = \frac{1}{Z(\mathbf{X})} \exp\left(\sum_{k=0}^{\tau} \phi(y_k^{act}, k, \mathbf{X}) + \sum_{k=1}^{\tau-1} \psi(y_{k-1}^{act}, y_k^{act})\right),$$

$$\text{where } Z(\mathbf{X}) = \sum_{\tilde{y}^{act}}^{\mathcal{Y}^{act}} \exp\left(\sum_{k=0}^{\tau} \phi(\tilde{y}_k^{act}, k, \mathbf{X}) + \sum_{k=1}^{\tau-1} \psi(\tilde{y}_{k-1}^{act}, \tilde{y}_k^{act})\right) \quad (10)$$

$$\hat{p}(\mathbf{y}^{frag}|\mathbf{X};\ \theta) = \text{softmax}(\text{FFNN}(\mathbf{V}_{frag})) \quad (11)$$

For one training instance, the local loss is computed by summing the negative log likelihood of $\hat{p}(\mathbf{y}^{act}|\mathbf{X};\ \theta)$ and the cross entropy of the predicted pointer likelihood against its target distribution $p(\mathbf{y}^{frag})$:

$$\mathcal{L} = -\left[\log \hat{p}(\mathbf{y}^{act}|\mathbf{X};\ \theta) + p(\mathbf{y}^{frag}) \cdot \log \hat{p}(\mathbf{y}^{frag}|\mathbf{X};\ \theta)\right] \quad (12)$$

For this task, we investigate the influence of the number of history utterances. Both of the act sequence and argument pointer predictions are evaluated by F1 scores.

## Experimental Results

Prediction results in Fig. 8 show that the length of dialog history more significantly impacts the performance of argument pointer prediction (F1 score drops by $>30\%$ for $\#H$ decreasing from 5 to 0) than agent act prediction (where F1 score drops by $\sim 6\%$). This difference is expected, since predicting the agent acts mainly relies on good understanding of the user intent embedded in the previous question and is less history dependent. For example, if the user intents are "confirm, req_instruction", we can presume an act sequence like "inform_instruction" as the most likely prediction without knowing extra information from distant utterances. In contrast, predicting the argument pointer sequence is strongly dependent on at least some history, since the model needs to update the user state (i.e., cooking progress) when the user is asking about a distant entity. When $\#H$ exceeds 3, the model performance plateaus, suggesting that it is not necessary to include all of the dialog history as input in our system.

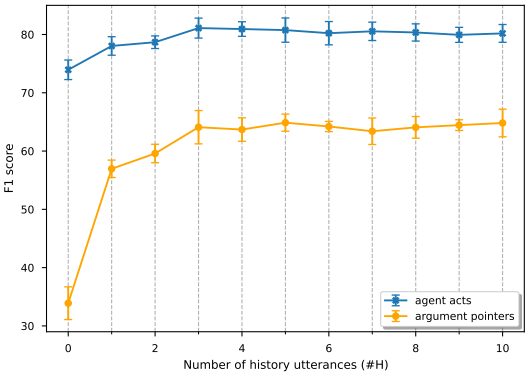


**Fig. 8** Experiment results of Task II Agent Action Frame Prediction. All the numbers are calculated over 5 random seeds.

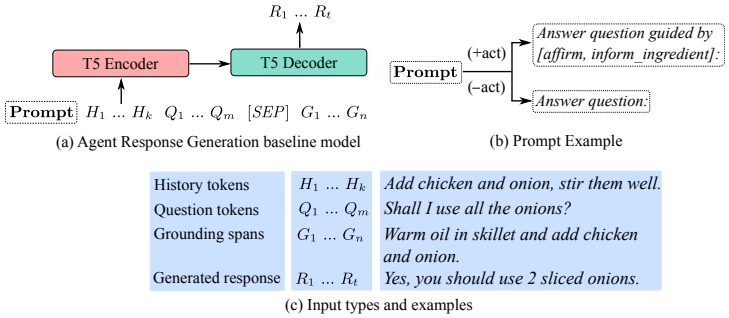


**Fig. 9** (a) Baseline model of Task III: Agent Response Generation. (b) Examples of dynamic and static **Prompt** depending on whether agent acts are used or not. (c) Input and output examples.

## 6.3 Agent Response Generation (Task III)

For the answer text generation, we rely on the pretrained T5-base model [25], which adopts a unified "text-to-text" approach casting NLP tasks as text generation and currently attaining state-of-the-art results on many of them. The pretrained encoder and decoder of T5-base are finetuned during our training. We use beam search with a beam width of 10 during decoding. Fig. 9(a) illustrates our baseline model. For the dialog history, we prepend the history tokens to the current question. The number of history utterances $\#H$ is fixed at 5 for all the models. Our model is designed to tackle the two challenges mentioned in Section 6.3 (i.e., paraphrasing and coreference resolving) by leveraging the semantic information from *Agent Action Frames*. Firstly, we hypothesize that generating a proper response can be enhanced by introducing the agent's dialog acts into the model input. More specifically, we integrate the agent acts into the prompt of the T5 model to inject essential semantic information, which is denoted as the (+act) setting as shown in Fig. 9(b). We also investigate another setting in which prompts contain no agent dialog act information, named as (−act) in our experiments. In the latter case (−act), the prompt remains static for all the training instances, i.e., "Answer question". In contrast, the (+act) setting makes the prompt dynamic, since the agent acts vary among agent utterances in a conversation. For example, the dynamic prompt used in Fig. 9(b) is "Answer question guided by [affirm, inform_ingredient]". Secondly, identifying the needed information from a long recipe is actually highly dependent on the grounding part, i.e., $G$ in Fig. 9(a). For the argument pointer spans, we also experiment with two settings: using gold argument pointer spans (+pointer) or not (−pointer). For the former, $G$ is in fact composed of the recipe text spans denoted by argument pointers within an *Agent Action Frame*. In the (−pointer) setting, we replace the gold spans with the entire recipe text, which makes it hard for the model to locate the key information precisely. Since the combination of (+act, +pointer) brings minimal noise into the input, we also call it the oracle model. We perform ablation experiments to asses the influence of providing agent dialog acts and

argument pointers as input to the generation model. Given a training instance $\mathbf{X}$ and its paired response $\mathbf{Z}$, the model minimizes cross-entropy loss:

$$\mathcal{L} = -\sum_{t=1}^{\mathbf{Z}} \log p(z_t|z_{t-1}, \ldots, z_1, \mathbf{X};\ \theta) \tag{13}$$

## Experimental Results

We adopt BLEU-1/2/3/4 and ROUGE-L as automatic evaluation metrics for the answer generation task. Table 3 reports the performance of our oracle model and its ablated variants on test set. Not surprisingly, the oracle model is the most competitive with 36.5 BLEU-4 and 54.4 ROUGE-L. Ablation of agent acts and argument pointer spans individually (row 2 and 3) shows little impact on the BLEU-4 score, while BLEU-1 and ROUGE-L results are more clearly affected. The larger decrease of ROUGE-L when agent acts are not used (−act, +pointer) implies that excluding the agent acts from the model input could harm the model's ability to generate responses or copy phrases from the document in a more appropriate way (see generation samples in Section 6.3). Moreover, when we remove both agent acts and pointer spans (−act, −pointer), the sharp drop of performance suggests the necessity of agent acts and document spans as inputs to improve the response generation quality of a document-grounded dialog system.

**Table 3** Experiment results of Task III Agent Response Generation. The results are averaged over 5 runs with different randomly sampled splits of CookDial. "act" and "pointer" denote agent acts and argument pointer spans respectively.

| | Settings | BLEU-1/4 | ROUGE-L |
|---|---|---|---|
| 1 | (+act, +pointer) oracle | **54.2±0.6 / 36.9±0.6** | **54.2±0.5** |
| 2 | (−act, +pointer) | 51.6±0.8 / 35.3±0.6 | 51.0±0.6 |
| 3 | (+act, −pointer) | 52.6±1.8 / 36.08±1.4 | 52.54±1.6 |
| 4 | (−act, −pointer) | 46.2±0.4 / 31.10±0.5 | 46.31±0.9 |

## Case study on generated response samples

Table 4 compares the generated responses sampled from two models, i.e., the (−act, −pointer) model and oracle model (+act, +pointer). In general, the oracle model generates more coherent responses than the ablated model does, which complies with our hypothesis that introducing the agent dialog acts into the model input can enhance the generation quality. On the other hand, the agent act is found to have great impact on steering the generative model. For example, in questions 1 and 4, the act "affirm" helps the oracle model to produce affirmative phrases like "yes, great" to respond smoothly to the user's yes/no questions, e.g., "Is all-purpose flour good?". In questions 2

and 3, without the guidance of agent acts, the ablated model gives totally wrong answers, which may confuse the user or even lead to a failure task in real applications. However, even when integrating agent acts (+act), the task remains highly challenging and responses are not perfect. In question 1, the oracle model ignores the act "simplify_continue" and does not mention the next ingredient "white sugar". In question 5, although the oracle model understands the meaning of "sum_ingredient_list", the total number is wrongly calculated. This numerical mistake reveals that our model lacks the ability to reason over the numerical information in the recipes. Mitigating this is left for future work.

The dialog snippets in Table 5 are consecutive turns generated by the two same models in Table 4. Without the assistance of *argument pointers*, the ablated model's generated responses are more erroneous and inconsistent. Compared to the ablated model, the oracle model succeeds in finding correct references for both user questions, i.e., "the beginning" and "what kind of sugar". This empirical observation implies that our *argument pointer* annotation helps the model to alleviate the difficulty in finding coreferences within dialogs. However, both models fail to answer the last question correctly, which requires the model to resolve long-distance anaphora. This limitation will be further discussed in Section 7.

**Table 4** Generated responses sampled from different experimental settings. The blue and italic font indicates a wrong response.

| Question | Gold Response | Agent Acts | Responses generated by (−act, −pointer) model | Responses generated by (+act, +pointer) model |
|---|---|---|---|---|
| 1. What kind of flour should it be? Is all-purpose flour good? | Yes it is! Also add 1 cup of white sugar. | affirm(); simplify_continue(ing-1); | *Sorry I cannot answer that question.* | Yes, it should be all-purpose flour. |
| 2. Should I add all of the egg? | I cannot answer that question. | fence(); | *Yes, you should add all of the eggs.* | Sorry I cannot answer that question. |
| 3. Something else to do still before putting them in the oven? | No that is it! | negate(); | *You can now bake it for 10 to 12 minutes.* | No. |
| 4. Ok no problem, can we start with the recipe? | Yes let us start! Do you have a slow cooker? | affirm(); verify(tool-0-0); | First place the chicken in a slow cooker. | Great! Can you take a slow cooker? |
| 5. Okay, what is the first thing I have to do? | There are 12 ingredients. Do you need me to list the ingredients? | sum_ingredient_list(); propose_list_ingredients(); | *Preheat the oven to 175 degrees Celsius.* | *This recipe has 9 ingredients.* Do you want to prepare them now? |

**Table 5** Dialog snippets from different experimental settings. The blue and italic font indicates a wrong response.

| Role | Gold dialog | Dialog generated by (−act, −pointer) model | Dialog turn generated by (+act, +pointer) model |
|---|---|---|---|
| *Agent* | Good. The next two are one and one third cup of white sugar and 2 eggs. | The next two ingredients are 1 1/3 cup of white sugar and 2 eggs. | The next two ingredients are 1 1/3 cup of white sugar and 2 eggs. |
| *User* | Could you repeat the beginning please? | | |
| *Agent* | The first ingredient is 1 and 1/3 cup of white sugar. | *2 eggs.* | 1 and 1/3 of a cup of white sugar. |
| *User* | Ok. Thanks for the clarification. What kind of sugar should it be? | | |
| *Agent* | It should be white sugar. | *sorry I cannot reply to that question.* | White sugar. |
| *User* | You said 2 eggs after that, right? | | |
| Agent | Yes, you also need to add 2 eggs | *1 tablespoon of baking powder.* | Yes, *you need 1 1/3 cup of white sugar.* |

# 7 Conclusion and Future Work

The main objective of this paper was to present a new dataset to facilitate research on document-grounded conversational agents (CAs) to assist in procedural tasks. Existing works on document-grounded CAs lacked the procedural aspect of the supporting documents, i.e., the fact that they are organized in a sequence of steps wherein entities undergo state changes that the CA needs to be aware of and be able to relate to one another. Indeed, for a CA it is crucial to relate the dialog context (e.g., which procedural step the user has arrived at) to the prescribed procedure to be able to generate an appropriate utterance in the conversation. Our newly created CookDial dataset contains 260 human-to-human dialogs that were collected based on recipe documents. We proposed an annotation approach for dialog systems grounded in procedural knowledge, with a schema that enables entity and state alignment between dialogs and corresponding documents. Our resulting annotated conversations exhibit non-trivial agent decision-making behavior, including responses containing a varying number of agent acts, segmentation of long instructions and paraphrasing the source document texts. From the CA perspective, we identified three major tasks, for which we also established baseline solutions: (i) user question understanding, (ii) agent action frame prediction, and (iii) agent response generation. We publicly release the dataset and the baseline models to spur further research. In terms of next steps, we highlight three directions that will guide our future work.

First, we want to explore generalization of our annotation method by applying it to other types of procedural tasks. Two more technical application domains of interest include chemical and mechanical operations. In these different domains, most definitions in our dialog-act taxonomy (Table 2) will still be useful like "req_instruction, req_subsititute". However, chemical processes are extremely sensitive to the element quantities, which raises the importance of accurate numerical understanding for the dialog system. On the other hand, in mechanical manuals, the spacial relation plays a significant role, e.g., in complex assembly tasks. Such mechanics related conversations may need intent annotations about the relative position between objects (e.g., "req_distance"). To facilitate such more refined acts, we envision a hierarchical structure of dialog acts to develop a cross-domain system. The hierarchy would start with domain-agnostic acts, then branch into domain-specific acts. Furthermore, it will be of great value to train a general language understanding model that can be applied for different procedural domains.

Second, our current solution lacks the ability to resolve ambiguous anaphora, which makes some user questions particularly hard to answer. A dialog snippet in Fig. 10 illustrates this. For question U1, the system needs to understand that the order reference "first and third" relates to the preceding utterance A1, rather than the ingredient order in the grounding recipe step. The user would be very confused if the agent answers wrongly with the first and last ingredients from `inst-2`, i.e., "butter and vanilla extract". Similarly, the subsequent user question U2 with "the one after vanilla extract" is even more ambiguous, and

**Fig. 10** A dialog snippet illustrating ambiguous anaphora across utterances.

**Dialog:**
Context: The agent already told the user to cream butter and sugar in a bowl.
A1: Now you can add 1 spoon vanilla extract, 3 eggs and 2 fresh strawberries.
U1: Sorry, can you repeat the first and third ingredients? What extract?
A2: No problem. 1 spoon vanilla extract and 2 fresh strawberries.
U2: Ok. But wait, I still did not catch the one after vanilla extract. What is it again?

**Grounding recipe step:**
inst-2: In a large bowl, cream butter and sugar until light and fluffy. Then add vanilla extract, beaten eggs and strawberries. Mix them well.

the CA should attach the context of A2 (rather than A1) to resolve it. Our current annotation relies on the absolute identifiers from the recipe, while coreferences within the dialog are not annotated (e.g., linking "the one after vanilla extract" to the preceding agent utterance's "2 fresh strawberries"). To overcome this problem, we envision extra annotations of coreference linking across utterances.

Third, we note that in our collected dialogs, users sometimes tend to ask for clarifications on items (ingredients, tools) that they are not familiar with. For now, our system performs poorly on answering those questions, given that the dialog agent's knowledge (besides the conversation itself) is limited to the grounded document. The most common answer observed in our collected dataset is "Sorry, I cannot answer your question". These uninformative answers normally do not disrupt the dialog flow but might damage the user satisfaction with (and therefor limit adoption of) the CA solution. This problem can be overcome by incorporating external knowledge sources (e.g., Wikipedia) or knowledge graphs (e.g., DBpedia). How to efficiently fuse such knowledge base information with grounded documents is another direction for our future work.

## Acknowledgements

We thank Maarten De Raedt and Amir Hadifar for their insightful suggestions in the initial data collection. The first author is supported by *China Scholarship Council* (No. 201906020194) and *Bijzonder Onderzoeksfonds (BOF) van Universiteit Gent* (No. 01SC0618). This research also receives funding from the Flemish Government under the *"Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen"* programme.

## Appendix A Experiment Settings

All the transformer modules in our models are implemented with the Huggingface library [26]. We conducted the experiments with a single Nvidia-Tesla-V100 (32GB) card. For all the tasks, we use the AdamW optimizer [27]. For both of Task I and Task II, we use two different learning rates depending on the layers to accelerate convergence: (i) $10^{-5}$ for the layers within the BigBird encoder; (ii) $10^{-3}$ for the top classifier layers (FFNNs and CRF). For Task III,

the learning rate for all the layers is set to $3 \times 10^{-4}$. The batch size is set to 8. The hidden size for all the FFNN layers is 128 except the intent classifier layer (64) in Task I. The dropout is set to 0.2 in the fine-tuning when needed.

# Appendix B User Intent and Agent Act Annotations

Elucidation on how we annotate the user intents and agent acts is presented in Table B.1 and Table B.2 respectively. For each intent or agent act, we also provide an annotation example except a few, i.e., other, repeat.

**Table B.1**: Annotation scheme for the user intents.

| User Intent | Meaning | Examples |
|---|---|---|
| greeting | User says "Hi" or "Hello" to start a conversation. | Hi! / Hello! / Good morning! |
| thank | User expresses gratitude. | Thank you. / Thanks. |
| confirm | User establishes the fact that he or she has accomplished one or several actions. | Done. / I have made it. / All of them are taken out of oven. |
| negate | User gives a negative statement. | No, I don't have any avocado. / Sorry, I cannot remove the cover. |
| affirm | User gives a positive statement. | Yes, I have an electrical beater. / Of course, lasagne is my favorite. |
| goodbye | User expresses good wishes at the end of a conversation. | Bye. / Goodbye. / See you. |
| req_start | User requests the agent to give the first guidance. | How should I start? / How do I make the secret pie? |
| req_temperature | User requests the exact temperature expression for an action. | What temperature shall I set for the baking? |
| req_instruction | User requests the next instruction after he or she confirms accomplishment of previous instructions. | What is next? / Next is? / What should I do now? |
| req_repeat | User asks for repeating a mentioned entity or instruction. | Can you repeat the first ingredient? / Could you tell me the last step again? |

*Continue on the next page*

**Table B.1**: (continued) Annotation scheme for the user intents.

| User Intent | Meaning | Examples |
|---|---|---|
| req_amount | User asks for the exact quantity of an entity. | How much sugar do I need? / How much does a package of cheese weigh to? |
| req_ingredient | User asks for the next ingredient. | What is next? / What is next ingredient? |
| req_use_all | User wants to know if a specific ingredient shall be used up. | Can I use all the pepper? / Shall I add all the berries? |
| req_title | User asks for the recipe title. | What are we making today? |
| req_is_recipe_finished | User asks if the cooking process ends or not. | Is it done? / Is the recipe finished? / Is it over? |
| req_tool | User requests a specific tool entity. | What should I use to make the star shape? |
| req_duration | User requests the time needed for an action. | How long shall I bake the cake? / When do I know it is ready? |
| req_is_recipe_ongoing | User asks if there are more steps to follow. | Are there more steps? / Anything else to do? |
| req_substitute | User wants to know if it is possible to use an alternative ingredient instead of the prescribed one. | Can I use white sugar instead of brown? |
| req_ingr_list | User requests a detailed list of ingredients before the instruction part starts. | Can you give me all the ingredients I need? |
| req_ingr_list_length | User requests the total number of ingredients. | How many ingredients do I need? |
| req_ingr_list_ends | User wants to know if all the ingredients have been introduced. | Is it over? / That is all the ingredients? |
| req_parallel_action | User asks for another plausible action while waiting for the current one to be finished. | What can I do in the meantime? |
| other | Used for user utterances that cannot be attributed to any of intents above. | - |

**Table B.2**: Annotation scheme for the agent acts.

| Agent Acts | Meaning | Examples |
|---|---|---|
| greeting | Agent responds to the user's "greeting". | Hi. / Greetings. |
| goodbye | Agent responds to the user's "goodbye". | Bye. / Have a nice day. |
| affirm | Agent gives a positive statement. | Yes, you need 6 eggs. / Yes, the recipe is finished. |
| negate | Agent gives a negative statement. | No, you still have to wait. |
| enjoy | Agent wishes the user to enjoy the food. | Enjoy the juicy burger. |
| end_recipe | Agent states that the cooking process is finished. | This is the last step. |
| thank | Agent acknowledges user's gratitude. | You are welcome. |
| fence | Agent cannot answer the user's question. | Sorry, I cannot answer that. / Sorry, it is beyond my knowledge. |
| count_ingredient_list | Agent gives the total number of all ingredients. | We will use 10 ingredients in total. |
| propose_start_recipe | Agent proposes to start the cooking process. | Shall we start now? / Are you prepared? |
| end_ingredient_list | Agent states that introducing ingredients is finished. | That is all the ingredients we need. / This is the last ingredient. |
| propose_list_ingr | Agent proposes to introduce ingredient details before the instruction part start. | Do you want to prepare the ingredients beforehand? |
| propose_skip_ing_list | Agent proposes to skip the ingredient list if it is too long. | Do you want me to announce the ingredients when needed? |
| propose_next_inst | Agent proposes to continue to give the next instruction. | Shall we continue? / Are you prepared for the next step? |
| propose_start_inst | Agent proposes to start the instruction part after finishing introducing all the ingredients. | Do you want to know the first step? |

*Continue on the next page*

**Table B.2**: (continued) Annotation scheme for the agent acts.

| Agent Acts | Meaning | Examples |
|---|---|---|
| propose_other_help | Agent asks the user if he or she needs extra help. (normally this occurs at the end of a conversation.) | Anything else I can do for you? |
| inform_instruction | Agent informs one instruction symbolized by its step identifier. | The next step is to remove the cake from oven. |
| inform_ingredient | Agent informs one ingredient symbolized by its step identifier. | You also need 1 bottle of honey. |
| inform_title | Agent informs the recipe title. | We will cook Flemish Stew today. |
| inform_duration | Agent informs time duration of an action. | Are there more steps? / Anything else to do? |
| inform_temperature | Agent informs a temperature expression. | Set the oven to 220 degrees C. |
| inform_amount | Agent informs quantity of an ingredient. | Please add 1/2 teaspoon of salt. |
| inform_tool | Agent informs a tool used in an action. | Prepare 2 feet long lining paper. |
| fetch | Agent asks the user to prepare necessary stuff for one instruction. | Take a large bowl. |
| repeat | Agent repeats a mentioned entity or instruction. It normally appears after the user intent "req_repeat". | - |
| verify | Agent checks if the user has the required tool or if the previous instruction is accomplished. | Do you have a slow cooker? / Are both sides of the breast browned? |
| simplify_begin | Agent segments a long instruction into a sequence of sub-instructions. This act remarks the first one. | First, add chopped onions. |
| simplify_continue | The other sub-instructions after "simplify_begin" | Then add salad oil and some spices as desired. |
| other | Used for agent responses that cannot be attributed to any of acts above. | - |

## References

[1] Gunasekara, C., Kim, S., D'Haro, L.F., et al.: Overview of the ninth dialog system technology challenge: DSTC9. In: Proceedings of the DSTC Workshop at AAAI, Online (2021)

[2] Wen, T.-H., Vandyke, D., Mrkšić, N., Gašić, M., Rojas-Barahona, L.M., Su, P.-H., Ultes, S., Young, S.: A network-based end-to-end trainable task-oriented dialogue system. In: Proceedings of EACL, Valencia, Spain, pp. 438–449 (2017). https://aclanthology.org/E17-1042

[3] Budzianowski, P., Wen, T.-H., Tseng, B.-H., Casanueva, I., Ultes, S., Ramadan, O., Gasic, M.: Multiwoz - a large-scale multi-domain wizard-of-oz dataset for task-oriented dialogue modelling. In: Proceedings of EMNLP, Brussels, Belgium, pp. 5016–5026 (2018). https://doi.org/10.18653/v1/D18-1547

[4] Rastogi, A., Zang, X., Sunkara, S., Gupta, R., Khaitan, P.: Towards scalable multi-domain conversational agents: The schema-guided dialogue dataset. In: Proceedings of AAAI, vol. 34. New York, USA, pp. 8689–8696 (2020). https://doi.org/10.1609/aaai.v34i05.6394

[5] Rajpurkar, P., Jia, R., Liang, P.: Know what you don't know: Unanswerable questions for SQuAD. In: Proceedings of ACL, vol. 2. Melbourne, Australia, pp. 784–789 (2018). https://doi.org/10.18653/v1/P18-2124

[6] Zhou, H., Zheng, C., Huang, K., Huang, M., Zhu, X.: KdConv: A Chinese multi-domain dialogue dataset towards multi-turn knowledge-driven conversation. In: Proceedings of ACL, Online, pp. 7098–7108 (2020). https://doi.org/10.18653/v1/2020.acl-main.635

[7] Reddy, S., Chen, D., Manning, C.D.: CoQA: A conversational question answering challenge. Transactions of the Association for Computational Linguistics **7**, 249–266 (2019). https://doi.org/10.1162/tacl_a_00266

[8] Choi, E., He, H., Iyyer, M., Yatskar, M., Yih, W.-t., Choi, Y., Liang, P., Zettlemoyer, L.: QuAC: Question answering in context. In: Proceedings of EMNLP, Brussels, Belgium, pp. 2174–2184 (2018). https://doi.org/10.18653/v1/D18-1241

[9] Campos, J.A., Otegi, A., Soroa, A., Deriu, J., Cieliebak, M., Agirre, E.: DoQA - accessing domain-specific FAQs via conversational QA. In: Proceedings of ACL, Online, pp. 7302–7314 (2020). https://doi.org/10.18653/v1/2020.acl-main.652

[10] Saeidi, M., Bartolo, M., Lewis, P., Singh, S., Rocktäschel, T., Sheldon, M., Bouchard, G., Riedel, S.: Interpretation of natural language rules

in conversational machine reading. In: Proceedings of EMNLP, Brussels, Belgium, pp. 2087–2097 (2018). https://doi.org/10.18653/v1/D18-1233

[11] Feng, S., Wan, H., Gunasekara, C., Patel, S., Joshi, S., Lastras, L.: doc2dial: A goal-oriented document-grounded dialogue dataset. In: Proceedings of EMNLP, Online, pp. 8118–8128 (2020). https://doi.org/10.18653/v1/2020.emnlp-main.652

[12] Raghu, D., Agarwal, S., Joshi, S., Mausam: End-to-end learning of flowchart grounded task-oriented dialogs. In: Proceedings of EMNLP, Online and Punta Cana, Dominican Republic, pp. 4348–4366 (2021). https://doi.org/10.18653/v1/2021.emnlp-main.357

[13] Jiang, Y., Zaporojets, K., Deleu, J., Demeester, T., Develder, C.: Recipe instruction semantics corpus (RISeC): Resolving semantic structure and zero anaphora in recipes. In: Proceedings of AACL, Online and Suzhou, China, pp. 821–826 (2020). https://aclanthology.org/2020.aacl-main.82

[14] Burtsev, M., Chuklin, A., Kiseleva, J., Borisov, A.: Search-oriented conversational AI (SCAI). In: Proceedings of ACM SIGIR ICTIR, Amsterdam, The Netherlands, pp. 333–334 (2017). https://doi.org/10.1145/3121050.3121111

[15] Henderson, M., Thomson, B., Williams, J.: The third dialog state tracking challenge. In: Proceedings of the SLT Workshop at IEEE, pp. 324–329 (2014)

[16] Wen, T.-H., Vandyke, D., Mrkšić, N., Gašić, M., Rojas-Barahona, L.M., Su, P.-H., Ultes, S., Young, S.: A network-based end-to-end trainable task-oriented dialogue system. In: Proceedings of EACL, vol. 1. Valencia, Spain, pp. 438–449 (2017). https://aclanthology.org/E17-1042

[17] El Asri, L., Schulz, H., Sharma, S., Zumer, J., Harris, J., Fine, E., Mehrotra, R., Suleman, K.: Frames: a corpus for adding memory to goal-oriented dialogue systems. In: Proceedings of SIGDIAL, Saarbrücken, Germany, pp. 207–219 (2017). https://doi.org/10.18653/v1/W17-5526

[18] Kollar, T., Berry, D., Stuart, L., Owczarzak, K., Chung, T., Mathias, L., Kayser, M., Snow, B., Matsoukas, S.: The Alexa meaning representation language. In: Proceedings of NAACL, vol. 3. New Orleans - Louisiana, pp. 177–184 (2018). https://doi.org/10.18653/v1/N18-3022

[19] Gupta, S., Shah, R., Mohit, M., Kumar, A., Lewis, M.: Semantic parsing for task oriented dialog using hierarchical representations. In: Proceedings of EMNLP, Brussels, Belgium, pp. 2787–2792 (2018). https://doi.org/10.18653/v1/D18-1300

[20] Aghajanyan, A., Maillard, J., Shrivastava, A., Diedrick, K., Haeger, M., Li, H., Mehdad, Y., Stoyanov, V., Kumar, A., Lewis, M., Gupta, S.: Conversational semantic parsing. In: Proceedings of EMNLP, Online, pp. 5026–5035 (2020). https://doi.org/10.18653/v1/2020.emnlp-main.408

[21] Bunt, H., Petukhova, V., Traum, D., Alexandersson, J.: Dialogue Act Annotation with the ISO 24617-2 Standard, pp. 109–135. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-42816-1_6

[22] Qu, C., Yang, L., Qiu, M., Zhang, Y., Chen, C., Croft, W., Iyyer, M.: Attentive history selection for conversational question answering. In: Proceedings of CIKM, Beijing, China, pp. 1391–1400 (2019). https://doi.org/10.1145/3357384.3357905

[23] Zaheer, M., Guruganesh, G., Dubey, K.A., Ainslie, J., Alberti, C., Ontanon, S., Pham, P., Ravula, A., Wang, Q., Yang, L., Ahmed, A.: Big bird: Transformers for longer sequences. In: Proceedings of NeurIPS, vol. 33. Online, pp. 17283–17297 (2020)

[24] Sutton, C., McCallum, A.: An introduction to conditional random fields. Foundations and Trends in Machine Learning **4**, 267–373 (2012). https://doi.org/10.1561/2200000013

[25] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., Liu, P.J.: Exploring the limits of transfer learning with a unified text-to-text transformer. Journal of Machine Learning Research **21**(140), 1–67 (2020)

[26] Huggingface: Transformers: State-of-the-art natural language processing. In: Proceedings of EMNLP: System Demonstrations, Online, pp. 38–45 (2020). https://doi.org/10.18653/v1/2020.emnlp-demos.6

[27] Loshchilov, I., Hutter, F.: Decoupled weight decay regularization. In: Proceedings of ICLR, Vancouver, BC, Canada (2019). https://openreview.net/forum?id=Bkg6RiCqY7