

CHAPTER

15

# Chatbots & Dialogue Systems

Les lois de la conversation sont en général de ne s’y appesantir sur aucun objet, mais de passer légèrement, sans effort et sans affectation, d’un sujet à un autre ; de savoir y parler de choses frivoles comme de choses sérieuses

[The rules of conversation are, in general, not to dwell on any one subject, but to pass lightly from one to another without effort and without affectation; to know how to speak about trivial topics as well as serious ones;]

The 18th C. *Encyclopedia* of Diderot, start of the entry on conversation

“What ho!” I said.

“What ho!” said Motty.

“What ho! What ho!”

“What ho! What ho! What ho!”

After that it seemed rather difficult to go on with the conversation.”

P.G. Wodehouse, *My Man Jeeves*

conversation  
dialogue

dialogue system  
conversational  
agent

The literature of the fantastic abounds in inanimate objects magically endowed with sentience and the gift of speech. From Ovid’s statue of Pygmalion to Mary Shelley’s *Frankenstein*, there is something deeply moving about creating something and then having a chat with it. Legend has it that after finishing his sculpture *Moses*, Michelangelo thought it so lifelike that he tapped it on the knee and commanded it to speak. Perhaps this shouldn’t be surprising. Language is the mark of humanity and sentience, and **conversation** or **dialogue** is the most fundamental and specially privileged arena of language. It is the first kind of language we learn as children, and for most of us, it is the kind of language we most commonly indulge in, whether we are ordering curry for lunch or buying spinach, participating in business meetings or talking with our families, booking airline flights or complaining about the weather.



This chapter introduces the fundamental algorithms of **dialogue systems**, or **conversational agents**. These programs communicate with users in natural language (text, speech, or both), and fall into two classes. **Task-oriented dialogue agents** use conversation with users to help complete tasks. Dialogue agents in digital assistants (Siri, Alexa, Google Now/Home, Cortana, etc.), give directions, control appliances, find restaurants, or make calls. Conversational agents can answer questions on corporate websites, interface with robots, and even be used for social good: DoNotPay is a “robot lawyer” that helps people challenge incorrect parking fines, apply for emergency housing, or claim asylum if they are refugees. By

contrast, **chatbots** are systems designed for extended conversations, set up to mimic the unstructured conversations or ‘chats’ characteristic of human-human interaction, mainly for entertainment, but also for practical purposes like making task-oriented agents more natural.<sup>1</sup> In Section 15.2 we’ll discuss the three major chatbot architectures: rule-based systems, information retrieval systems, and encoder-decoder generators. In Section 15.3 we turn to task-oriented agents, introducing the frame-based architecture (the **GUS** architecture) that underlies most task-based systems.

## 15.1 Properties of Human Conversation

Conversation between humans is an intricate and complex joint activity. Before we attempt to design a conversational agent to converse with humans, it is crucial to understand something about how humans converse with each other. Consider some of the phenomena that occur in the conversation between a human travel agent and a human client excerpted in Fig. 15.1.

C<sub>1</sub>: ... I need to travel in May.  
 A<sub>2</sub>: And, what day in May did you want to travel?  
 C<sub>3</sub>: OK uh I need to be there for a meeting that’s from the 12th to the 15th.  
 A<sub>4</sub>: And you’re flying into what city?  
 C<sub>5</sub>: Seattle.  
 A<sub>6</sub>: And what time would you like to leave Pittsburgh?  
 C<sub>7</sub>: Uh hmm I don’t think there’s many options for non-stop.  
 A<sub>8</sub>: Right. There’s three non-stops today.  
 C<sub>9</sub>: What are they?  
 A<sub>10</sub>: The first one departs PGH at 10:00am arrives Seattle at 12:05 their time.  
 The second flight departs PGH at 5:55pm, arrives Seattle at 8pm. And the  
 last flight departs PGH at 8:15pm arrives Seattle at 10:28pm.  
 C<sub>11</sub>: OK I’ll take the 5ish flight on the night before on the 11th.  
 A<sub>12</sub>: On the 11th? OK. Departing at 5:55pm arrives Seattle at 8pm, U.S. Air  
 flight 115.  
 C<sub>13</sub>: OK.  
 A<sub>14</sub>: And you said returning on May 15th?  
 C<sub>15</sub>: Uh, yeah, at the end of the day.  
 A<sub>16</sub>: OK. There’s #two non-stops ... #  
 C<sub>17</sub>: #Act... actually #, what day of the week is the 15th?  
 A<sub>18</sub>: It’s a Friday.  
 C<sub>19</sub>: Uh hmm. I would consider staying there an extra day til Sunday.  
 A<sub>20</sub>: OK... OK. On Sunday I have ...

**Figure 15.1** Part of a phone conversation between a human travel agent (A) and human client (C). The passages framed by # in A<sub>16</sub> and C<sub>17</sub> indicate overlaps in speech.

### Turns

**turn** A dialogue is a sequence of **turns** (C<sub>1</sub>, A<sub>2</sub>, C<sub>3</sub>, and so on), each a single contribution from one speaker to the dialogue (as if in a game: I take a turn, then you take a turn,

<sup>1</sup> By contrast, in popular usage, the word *chatbot* is often generalized to refer to both task-oriented and chit-chat systems; we’ll be using *dialogue systems* for the former.

then me, and so on). There are 20 turns in Fig. 15.1. A turn can consist of a sentence (like  $C_1$ ), although it might be as short as a single word ( $C_{13}$ ) or as long as multiple sentences ( $A_{10}$ ).

Turn structure has important implications for spoken dialogue. A system has to know when to stop talking; the client interrupts (in  $A_{16}$  and  $C_{17}$ ), so the system must know to stop talking (and that the user might be making a correction). A system also has to know when to start talking. For example, most of the time in conversation, speakers start their turns almost immediately after the other speaker finishes, without a long pause, because people are able to (most of the time) detect when the other person is about to finish talking. Spoken dialogue systems must also detect whether a user is done speaking, so they can process the utterance and respond. This task—called **endpointing** or **endpoint detection**—can be quite challenging because of noise and because people often pause in the middle of turns.

### Speech Acts

A key insight into conversation—due originally to the philosopher Wittgenstein (1953) but worked out more fully by Austin (1962)—is that each utterance in a dialogue is a kind of **action** being performed by the speaker. These actions are commonly called **speech acts** or **dialog acts**: here's one taxonomy consisting of 4 major classes (Bach and Harnish, 1979):

<b>Constatives:</b>	committing the speaker to something's being the case ( <i>answering, claiming, confirming, denying, disagreeing, stating</i> )
<b>Directives:</b>	attempts by the speaker to get the addressee to do something ( <i>advising, asking, forbidding, inviting, ordering, requesting</i> )
<b>Commissives:</b>	committing the speaker to some future course of action ( <i>promising, planning, vowing, betting, opposing</i> )
<b>Acknowledgments:</b>	express the speaker's attitude regarding the hearer with respect to some social action ( <i>apologizing, greeting, thanking, accepting an acknowledgment</i> )

A user asking a person or a dialogue system to do something ('Turn up the music') is issuing a DIRECTIVE. Asking a question that requires an answer is also a way of issuing a DIRECTIVE: in a sense when the system says ( $A_2$ ) "what day in May did you want to travel?" it's as if the system is (very politely) commanding the user to answer. By contrast, a user stating a constraint (like  $C_1$  'I need to travel in May') is issuing a CONSTATIVE. A user thanking the system is issuing an ACKNOWLEDGMENT. The speech act expresses an important component of the intention of the speaker (or writer) in saying what they said.

### Grounding

A dialogue is not just a series of independent speech acts, but rather a collective act performed by the speaker and the hearer. Like all collective acts, it's important for the participants to establish what they both agree on, called the **common ground** (Stalnaker, 1978). Speakers do this by **grounding** each other's utterances. Grounding means acknowledging that the hearer has understood the speaker; like an ACK used to confirm receipt in data communications (Clark, 1996). (People need grounding for non-linguistic actions as well; the reason an elevator button lights up when it's pressed is to acknowledge that the elevator has indeed been called (Norman, 1988).)

Humans constantly ground each other's utterances. We can ground by explicitly saying "OK", as the agent does in  $A_8$  or  $A_{10}$ . Or we can ground by repeating what

the other person says; in utterance A<sub>2</sub> the agent repeats “in May”, demonstrating her understanding to the client. Or notice that when the client answers a question, the agent begins the next question with “And”. The “And” implies that the new question is ‘in addition’ to the old question, again indicating to the client that the agent has successfully understood the answer to the last question.

### Subdialogues and Dialogue Structure

conversational  
analysis

Conversations have structure. Consider, for example, the local structure between speech acts discussed in the field of **conversational analysis** (Sacks et al., 1974). QUESTIONS set up an expectation for an ANSWER. PROPOSALS are followed by ACCEPTANCE (or REJECTION). COMPLIMENTS (“Nice jacket!”) often give rise to DOWNPLAYERS (“Oh, this old thing?”). These pairs, called **adjacency pairs** are composed of a **first pair part** and a **second pair part** (Schegloff, 1968), and these expectations can help systems decide what actions to take.

adjacency pair

side sequence  
subdialogue

However, dialogue acts aren’t always followed immediately by their second pair part. The two parts can be separated by a **side sequence** (Jefferson 1972) or **subdialogue**. For example utterances C<sub>17</sub> to A<sub>20</sub> constitute a **correction subdialogue** (Litman 1985, Litman and Allen 1987, Chu-Carroll and Carberry 1998):

C<sub>17</sub>: #Act... actually#, what day of the week is the 15th?  
A<sub>18</sub>: It’s a Friday.  
C<sub>19</sub>: Uh hmm. I would consider staying there an extra day til Sunday.  
A<sub>20</sub>: OK... OK. On Sunday I have ...

The question in C<sub>17</sub> interrupts the prior discourse, in which the agent was looking for a May 15 return flight. The agent must answer the question and also realize that “I would consider staying...til Sunday” means that the client would probably like to change their plan, and now go back to finding return flights, but for the 17th.

Another side sequence is the **clarification question**, which can form a subdialogue between a REQUEST and a RESPONSE. This is especially common in dialogue systems where speech recognition errors causes the system to have to ask for clarifications or repetitions like the following:

User: What do you have going to UNKNOWN\_WORD on the 5th?  
System: Let’s see, going where on the 5th?  
User: Going to Hong Kong.  
System: OK, here are some flights...

presequence

In addition to side-sequences, questions often have **presequences**, like the following example where a user starts with a question about the system’s capabilities (“Can you make train reservations”) before making a request.

User: Can you make train reservations?  
System: Yes I can.  
User: Great, I’d like to reserve a seat on the 4pm train to New York.

### Initiative

initiative

Sometimes a conversation is completely controlled by one participant. For example a reporter interviewing a chef might ask questions, and the chef responds. We say that the reporter in this case has the conversational **initiative** (Walker and Whittaker, 1990). In normal human-human dialogue, however, it’s more common for initiative to shift back and forth between the participants, as they sometimes answer questions, sometimes ask them, sometimes take the conversations in new directions, sometimes

not. You may ask me a question, and then I respond asking you to clarify something you said, which leads the conversation in all sorts of ways. We call such interactions **mixed initiative** (Walker and Whittaker, 1990).

Mixed initiative, while the norm for human-human conversations, is very difficult for dialogue systems to achieve. It's much easier to design dialogue systems to be passive responders. In the question answering systems we saw in Chapter 14, or in simple search engines, the initiative lies completely with the user. In such **user-initiative** systems, the user specifies a query, and the systems responds. Then the user can specify another query. Alternatively, you may have had the experience of being stuck in a bad dialogue system that asks a question and gives you no opportunity to do anything until you answer it. Such **system-initiative** architectures can be very frustrating.

### Inference and Implicature

Inference is also important in dialogue understanding. Consider the client's response  $C_2$ , repeated here:

$A_2$ : And, what day in May did you want to travel?

$C_3$ : OK uh I need to be there for a meeting that's from the 12th to the 15th.

Notice that the client does not in fact answer the agent's question. The client merely mentions a meeting at a certain time. What is it that licenses the agent to infer that the client is mentioning this meeting so as to inform the agent of the travel dates?

implicature

relevance

The speaker seems to expect the hearer to draw certain inferences; in other words, the speaker is communicating more information than seems to be present in the uttered words. This kind of example was pointed out by Grice (1975, 1978) as part of his theory of **conversational implicature**. **Implicature** means a particular class of licensed inferences. Grice proposed that what enables hearers to draw these inferences is that conversation is guided by a set of **maxims**, general heuristics that play a guiding role in the interpretation of conversational utterances. One such maxim is the maxim of **relevance** which says that speakers attempt to be relevant, they don't just utter random speech acts. When the client mentions a meeting on the 12th, the agent reasons 'There must be some relevance for mentioning this meeting. What could it be?'. The agent knows that one precondition for having a meeting (at least before Web conferencing) is being at the place where the meeting is held, and therefore that maybe the meeting is a reason for the travel, and if so, then since people like to arrive the day before a meeting, the agent should infer that the flight should be on the 11th.

These subtle characteristics of human conversations (**turns**, **speech acts**, **grounding**, **dialogue structure**, **initiative**, and **implicature**) are among the reasons it is difficult to build dialogue systems that can carry on natural conversations with humans. Many of these challenges are active areas of dialogue systems research.

## 15.2 Chatbots

chatbot

The simplest kinds of dialogue systems are **chatbots**, systems that can carry on extended conversations with the goal of mimicking the unstructured conversations or 'chats' characteristic of informal human-human interaction.



counseling.

Like practically everything else in language processing, chatbot architectures fall into two classes: rule-based systems and corpus-based systems. Rule-based systems include the early influential ELIZA and PARRY systems. Corpus-based systems mine large datasets of human-human conversations, which can be done by using information retrieval to copy a human response from a previous conversation, or using an encoder-decoder system to generate a response from a user utterance.

### 15.2.1 Rule-based chatbots: ELIZA and PARRY

ELIZA (Weizenbaum, 1966) is the most important chatbot dialogue system in the history of the field, and so we take some time here to describe its algorithm in detail. ELIZA was designed to simulate a Rogerian psychologist, based on a branch of clinical psychology whose methods involve drawing the patient out by reflecting patient’s statements back at them. Rogerian psychology is the rare type of conversation in which, as Weizenbaum points out, one can “assume the pose of knowing almost nothing of the real world”. If a patient says “I went for a long boat ride” and the psychiatrist says “Tell me about boats”, you don’t assume she didn’t know what a boat is, but rather assume she had some conversational goal. Most chatbots trying to pass the Turing test choose a domain with similar properties.

Fig. 15.4 shows the most famous ELIZA conversation, in which ELIZA seems to give very sophisticated therapeutic advice, excerpted (deleting some material with ...) from a full interaction given in the original paper (Weizenbaum, 1966).

As we summarized briefly in Chapter 2, ELIZA worked by pattern/transform rules like the following one:

```
(0 YOU 0 ME) [pattern]
->
(WHAT MAKES YOU THINK I 3 YOU) [transform]
```

In the ELIZA pattern, 0 means Kleene\*, and in the transform rules the numbers are the index of the constituent in the pattern. Thus the number 3 refers to the second 0 in the first pattern. This rule would transfer

You hate me

into:

WHAT MAKES YOU THINK I HATE YOU

Each ELIZA pattern/rule is linked to a keyword that might occur in a user sentence. The architecture is summarized in Fig. 15.5.

Keywords are associated with a rank, with specific words being more highly ranked, and more general words ranking lower. Consider the following user sentence:

I know everybody laughed at me

Because it has the word “I”, this sentence could match the following rule whose keyword is *I*:

```
(I *) -> (You say you 2)
```

producing:

YOU SAY YOU KNOW EVERYBODY LAUGHED AT YOU

But as Weizenbaum points out, “I” is a very general word and its keywords lead to very general responses. Instead the keyword “everybody” is much more interesting, since someone using universals like everybody or always is probably “referring



Men are all alike.  
**IN WHAT WAY**  
 They're always bugging us about something or other.  
**CAN YOU THINK OF A SPECIFIC EXAMPLE**  
 Well my boyfriend made me come here.  
**YOUR BOYFRIEND MADE YOU COME HERE**  
 He says I'm depressed much of the time.  
**I AM SORRY TO HEAR YOU ARE DEPRESSED**  
 ::  
**WHO ELSE IN YOUR FAMILY TAKES CARE OF YOU?**  
 My father  
**YOUR FATHER**  
 You are like my father in some ways.  
**WHAT RESEMBLANCE DO YOU SEE**  
 You are not very aggressive but I think you don't want me to notice that.  
**WHAT MAKES YOU THINK I AM NOT AGGRESSIVE**  
 You don't argue with me.  
**WHY DO YOU THINK I DON'T ARGUE WITH YOU**  
 You are afraid of me.  
**DOES IT PLEASE YOU TO BELIEVE I'M AFRAID OF YOU**  
 My father is afraid of everybody.  
**WHAT ELSE COMES TO MIND WHEN YOU THINK OF YOUR FATHER**  
 Bullies.  
**DOES THAT HAVE ANYTHING TO DO WITH THE FACT THAT YOUR**  
**BOYFRIEND MADE YOU COME HERE**

**Figure 15.4** Sample ELIZA dialogue from Weizenbaum (1966).

```

function ELIZA GENERATOR(user sentence) returns response
Find the word w in sentence that has the highest keyword rank
if w exists
    Choose the highest ranked rule r for w that matches sentence
    response ← Apply the transform in r to sentence
    if w = 'my'
        future ← Apply a transformation from the 'memory' rule list to sentence
        Push future onto memory queue
    else (no keyword applies)
        either
            response ← Apply the transform for the NONE keyword to sentence
        or
            response ← Pop the oldest response from the memory queue
return(response)

```

**Figure 15.5** A simplified sketch of the ELIZA algorithm. The power of the algorithm comes from the particular transforms associated with each keyword.

to some quite specific event or person". Therefore, ELIZA prefers to respond with the pattern associated with the more specific keyword *everybody* (implementing by just assigning "everybody" rank 5 and "I" rank 0 in the lexicon), whose rule thus



transforms the sentence to:

WHO IN PARTICULAR ARE YOU THINKING OF?

If no keyword matches, ELIZA chooses a non-committal response like “PLEASE GO ON”, “THAT’S VERY INTERESTING”, or “I SEE”.

Finally, ELIZA has a clever memory trick that accounts for the very last sentence of the conversation above. Whenever the word “my” is the highest ranked keyword, ELIZA will randomly select a transform on the MEMORY list, apply it to the sentence, and store it on the queue:

(MEMORY MY

(0 MY 0 = LETS DISCUSS FURTHER WHY YOUR 3)

(0 MY 0 = EARLIER YOU SAID YOUR 3)

(0 MY 0 = DOES THAT HAVE ANYTHING TO DO WITH THE FACT THAT YOUR 3

Later, if no keyword matches a sentence, ELIZA will return the oldest entry on the MEMORY queue instead.

People became deeply emotionally involved with the program. Weizenbaum tells the story of one of his staff who would ask Weizenbaum to leave the room when she talked with ELIZA. When Weizenbaum suggested that he might want to store all the ELIZA conversations for later analysis, people immediately pointed out the privacy implications, which suggested that they were having quite private conversations with ELIZA, despite knowing that it was just software.

ELIZA’s framework is still used today; modern chatbot system tools like ALICE are based on updated versions of ELIZA’s pattern/action architecture.

A few years after ELIZA, another chatbot with a clinical psychology focus, PARRY (Colby et al., 1971), was used to study schizophrenia. In addition to ELIZA-like regular expressions, the PARRY system included a model of its own mental state, with affect variables for the agent’s levels of fear and anger; certain topics of conversation might lead PARRY to become more angry or mistrustful. If PARRY’s **anger** variable is high, he will choose from a set of “hostile” outputs. If the input mentions his delusion topic, he will increase the value of his **fear** variable and then begin to express the sequence of statements related to his delusion. Parry was the first known system to pass the Turing test (in 1972!); psychiatrists couldn’t distinguish text transcripts of interviews with PARRY from transcripts of interviews with real paranoids (Colby et al., 1972).

### 15.2.2 Corpus-based chatbots

Corpus-based chatbots, instead of using hand-built rules, mine conversations of human-human conversations. These systems are enormously data-intensive, requiring hundreds of millions or even billions of words for training (Serban et al., 2018).

Available datasets include transcripts of natural spoken conversational corpora, like the Switchboard corpus of American English telephone conversations (Godfrey et al., 1992) or the various CALLHOME and CALLFRIEND telephone conversational corpora in many languages. Many systems also train on movie dialogue, (Danescu-Niculescu-Mizil and Lee 2011, Lison and Tiedemann 2016, inter alia) which resembles natural conversation in many ways (Forchini, 2013).

Datasets have also been created specifically for training dialog systems by hiring crowdworkers to have conversations, often having them take on personas or talk about knowledge provided to them. For example the Topical-Chat dataset has 11K crowdsourced conversations spanning 8 broad topics (Gopalakrishnan et al.,

2019), and the EMPATHETICDIALOGUES includes 25K crowdsourced conversations grounded in a specific situation where a speaker was feeling a specific emotion (Rashkin et al., 2019).

All of these datasets, although large, don't reach the size of billions of words, and so many systems first pretrain on large datasets of pseudo-conversations drawn from Twitter (Ritter et al., 2010), Reddit (Roller et al., 2021), Weibo (微博), and other social media platforms.

Another common technique is to extract possible responses from knowledge sources (Wikipedia, news stories) so that a chatbot can tell stories or mention facts acquired in that way.

Finally, once a chatbot has been put into practice, the turns that humans use to respond to the chatbot can be used as additional conversational data for training or finetuning. Here it's important to have confidence metrics to make sure that these turns come from conversations that are going well (Hancock et al., 2019). It's also crucial in these cases to remove personally identifiable information (PII); see Section 15.6.1.

response  
generation

Most corpus based chatbots produce their responses to a user's turn in context either by **retrieval** methods (using information retrieval to grab a response from some corpus that is appropriate given the dialogue context) or **generation** methods (using a language model or encoder-decoder to generate the response given the dialogue context). In either case, systems mostly generate a single response turn that is appropriate given the entire conversation so far (for conversations that are short enough to fit into a single model's window). For this reason they are often called **response generation** systems. Corpus-based chatbot algorithms thus draw on algorithms for question answering systems, which similarly focus on single responses while ignoring longer-term conversational goals.

**Response by retrieval** The **retrieval** method of responding is to think of the user's turn as a query  $q$ , and our job is to retrieve and repeat some appropriate turn  $r$  as the response from a corpus of conversations  $C$ . Generally  $C$  is the training set for the system, and we score each turn in  $C$  as a potential response to the context  $q$  selecting the highest-scoring one. The scoring metric is similarity: we choose the  $r$  that is most similar to  $q$ , using any of the IR methods we saw in Section ???. This can be done using classic IR techniques to compute tf-idf models for  $C$  and  $q$ , choosing the  $r$  that has the highest tf-idf cosine with  $q$ :

$$\text{response}(q, C) = \operatorname{argmax}_{r \in C} \frac{q \cdot r}{|q||r|} \quad (15.1)$$

Another version of this method is to return the *response* to the turn resembling  $q$ ; that is, we first find the most similar turn  $t$  to  $q$  and then return as a response the *following* turn  $r$ .

Alternatively, we can use the neural IR techniques of Section ???. The simplest of those is a bi-encoder model, in which we train two separate encoders, one to encode the user query and one to encode the candidate response, and use the dot product between these two vectors as the score (Fig. 15.6a). For example to implement this using BERT, we would have two encoders  $\text{BERT}_Q$  and  $\text{BERT}_R$  and we could represent the query and candidate response as the [CLS] token of the respective

encoders:

$$\begin{aligned} h_q &= \text{BERT}_Q(q)[\text{CLS}] \\ h_r &= \text{BERT}_R(r)[\text{CLS}] \\ \text{response}(q, C) &= \underset{r \in C}{\operatorname{argmax}} h_q \cdot h_r \end{aligned} \quad (15.2)$$

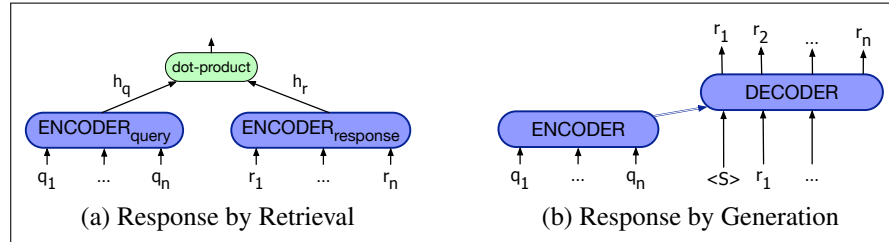
The IR-based approach can be extended in various ways, such as by using more sophisticated neural architectures (Humeau et al., 2020), or by using a longer context for the query than just the user’s last turn, up to the whole preceding conversation. Information about the user or sentiment or other information can also play a role.

**Response by generation** An alternate way to use a corpus to generate dialogue is to think of response production as an encoder-decoder task—transducing from the user’s prior turn to the system’s turn. We can think of this as a machine learning version of ELIZA; the system learns from a corpus to transduce a question to an answer. Ritter et al. (2011) proposed early on to think of response generation as a kind of translation, and this idea was generalized to the encoder-decoder model roughly contemporaneously by Shang et al. (2015), Vinyals and Le (2015), and Sordoni et al. (2015).

As we saw in Chapter 13, encoder decoder models generate each token  $r_t$  of the response by conditioning on the encoding of the entire query  $q$  and the response so far  $r_1 \dots r_{t-1}$ :

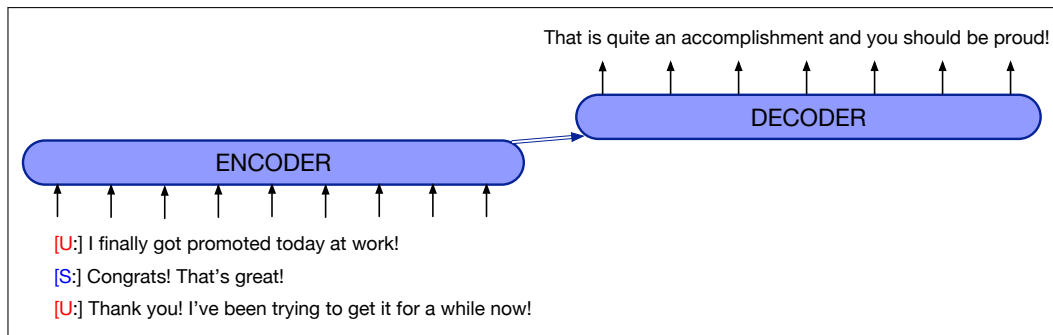
$$\hat{r}_t = \underset{w \in V}{\operatorname{argmax}} P(w|q, r_1 \dots r_{t-1}) \quad (15.3)$$

Fig. 15.6 shows the intuition of the generator and retriever methods for response generation. In the generator architecture, we normally include a longer context, forming the query not just from the user’s turn but from the entire conversation-so-far. Fig. 15.7 shows a fleshed-out example.



**Figure 15.6** Two architectures for generating responses for a neural chatbot. In response by retrieval (a) we choose a response by finding the turn in the corpus whose encoding has the highest dot-product with the user’s turn. In response by generation (b) we use an encoder-decoder to generate the response.

A number of modifications are required to the basic encoder-decoder model to adapt it for the task of response generation. For example basic encoder-decoder models have a tendency to produce predictable but repetitive and therefore dull responses like “I’m OK” or “I don’t know” that shut down the conversation. Thus instead of greedily choosing the most likely (and most predictable) response, we can use diversity-enhanced versions of beam search (Vijayakumar et al., 2018), or diversity-focused training objectives (Li et al., 2016a). Basic models also tend to produce utterances that are too short; adding minimum length constraints is also important (Roller et al., 2021).



**Figure 15.7** Example of encoder decoder for dialogue response generation; the encoder sees the entire dialogue context.

An alternative to the encoder-decoder architecture is to fine-tune a large language model on a conversational dataset and use the language model directly as a response generator. In the Chirpy Cardinal system (Paranjape et al., 2020), for example, the neural chat component generates responses from GPT-2 (Radford et al., 2019), fine-tuned on the EmpatheticDialogues dataset (Rashkin et al., 2019).

Finally, encoder-decoder response generators focus on generating single responses, and so don't tend to do a good job of continuously generating responses that cohere across multiple turns. This can be addressed by using reinforcement learning, as well as techniques like adversarial networks, to learn to choose responses that make the overall conversation more natural (Li et al. 2016b, Li et al. 2017).

### Response by retrieving and refining knowledge

Chatbots can be much more interesting and informative if they can respond based on text knowledge sources other than dialogue. This approach was pioneered early on by the COBOT chatbot (Isbell et al., 2000), which generated responses by selecting sentences from a corpus that combined the Unabomber Manifesto by Theodore Kaczynski, articles on alien abduction, the scripts of “The Big Lebowski” and “Planet of the Apes”. XiaoIce collects sentences from public lectures and news articles and searches them using IR based on query expansion from the user's turn to respond to turns like “Tell me something about Beijing” (Zhou et al., 2020);

One way to augment the encoder decoder architecture for retrieve and refine is to first use IR to retrieve potentially useful passages from Wikipedia (Yan et al., 2016), and then create multiple candidates by concatenating each retrieved Wikipedia sentence to the dialogue context with a separator token. Each candidate can be given as the encoder context to the encoder-decoder model, which learns to incorporate text from the Wikipedia sentence into its generated response (Dinan et al. 2019, Roller et al. 2021).

The language model approach to generation can also make use of external knowledge sources, by giving these same knowledge+dialogue context candidates to a language model like GPT-2 finetuned on conversational corpora to generate a knowledge-aware response (Paranjape et al., 2020).

### 15.2.3 Hybrid architectures

Chatbots can also be built with architectures that are hybrids of the rule-based and neural/corpus architectures, and even use elements of the frame-based structure we describe below. This is common, for example, for systems competing in the Alexa

Prize challenge, in which university teams build social chatbots to converse with volunteers on the Amazon Alexa platform, and are scored based on the length and user ratings of their conversations (Ram et al., 2017).

For example the Chirpy Cardinal system (Paranjape et al., 2020) applies an NLP pipeline that includes Wikipedia entity linking (Section ??), user intent classification, and dialogue act classification (to be defined below in Section 15.4.1). The intent classification is used when the user wants to change the topic, and the entity linker specifies what entity is currently being discussed. Dialogue act classification is used to detect when the user is asking a question or giving an affirmative versus negative response.

Bot responses are generated by a series of response generators. Some response generators use fine-tuned neural language models: a GPT-2 (Radford et al., 2019) language model fine-tuned on the EmpatheticDialogues dataset, and a separately fine-tuned GPT-2 language model that learns in fine-tuning to paraphrase content from Wikipedia in response to questions.

Other response generators are closer to rule-based; the movie or music generators produce scripted conversation about a movie or a musician, such as asking the user's opinion about a movie or an actor, giving a fun fact, and so on. These generators use regexes and sentiment classifiers to classify user responses and handwritten templates to generate bot utterances.

Each response generator can specify a priority, and a ranker is used to pass control between response generators. Fig. 15.8 shows part of a conversation.

#	Utterance	Entity	Commentary
1	User: let's chat	None	
	Bot: Hi, this is an Alexa Prize Socialbot. I'd love to get to know you a bit better before we chat! Is it all right if I ask for your name? [Launch]	None	Launch gives <i>FORCE_START</i> response to begin the conversation.
2	User: my name is chris	None	
	Bot: Well it's nice to meet you, Chris! I'm excited to chat with you today. [Launch] I hope your afternoon is going well. What are your plans for the rest of today? [Neural Chat]	None	Launch gives a <i>STRONG_CONTINUE</i> response that needs a prompt, which is supplied by Neural Chat with <i>FORCE_START</i> priority (scripted transition).
3	User: hang out with my friends	None	
	Bot: That sounds great, I love spending time with friends. What will you do together? [Neural Chat]	None	Neural Chat generates a GPT-2 response with <i>STRONG_CONTINUE</i> priority.
4	User: maybe watch a movie	None	
	Bot: Me too! I love watching movies; I get to learn so much about what the world is like outside of the cloud! Have you seen any movies recently? [Movies]	Film	Movies detects the 'movie' keyword in the user's utterance and gives a <i>FORCE_START</i> response (overriding Neural Chat), setting current entity to <i>Film</i> .
5	User: i saw the matrix	The Matrix	The Entity Linker and Entity Tracker identify <i>The Matrix</i> as the current entity.
	Bot: Nice! Did you like The Matrix? [Movies]	The Matrix	
6	User: i loved it neo is amazing	The Matrix	
	Bot: Oooh, yeah, I agree. Hey, isn't Keanu Reeves in that movie? What do you think about Keanu Reeves? [Movies]	Keanu Reeves	Movies sets current entity to <i>Keanu Reeves</i> .
7	User: i want to talk about something else	None	Navigational Intent detects <i>negative intent</i> , so Entity Tracker rejects the current entity.

**Figure 15.8** Part of a conversation with Chirpy Cardinal (Paranjape et al., 2020).

## 15.3 GUS: Simple Frame-based Dialogue Systems

We turn now to **task-based dialogue**, in which a dialogue system has the goal of helping a user solve some task like making an airplane reservation or buying a product. In this section we introduce the very simple **GUS** architecture for task-based dialogue. Although it is quite old, (it was first introduced in 1977 in the influential GUS system for travel planning (Bobrow et al., 1977)) the GUS architecture

has been astonishingly long-lived and underlies most if not all modern commercial digital assistants. Figure 15.9 shows a conversation with the original GUS system.

**dialogue state**  
**frame**  
**slot**

All modern task-based dialogue systems, whether the simple GUS architecture we describe here, or the more sophisticated **dialogue state** architectures we turn to in the following section, are based around **frames**. A frame is a kind of knowledge structure representing the kinds of intentions the system can extract from user sentences, and consists of a collection of **slots**, each of which can take a set of possible **values**. Together this set of frames is sometimes called a **domain ontology**.

GUS: Hello. My name is GUS. I can help you plan a simple trip by air. Where do you want to go ?  
Client: I want to go to San Diego on May 28  
GUS: What time do you want to leave?  
Client: I must be in San Diego before 10 am  
GUS: Would you like P.S.A. flight 102 that arrives at 9.15 am ?  
Client: What is the next flight ?  
GUS: Air California flight 310 that leaves at 8.30am and arrives at 10.00am  
Client: I'll take the first one  
GUS: In what name should I make a reservation, please?  
Client: Dan Bobrow  
GUS: I have confirmed the following flight: P.S.A. flight 102 on Wednesday May 28 from San Jose to San Diego that leaves at 7.30 am and arrives at 9.15 am.  
GUS: What date do you want to return on ?  
Client: On Friday in the evening.  
GUS: Would you like the flight that leaves at 7.45 pm ?  
Client: That's fine.  
GUS: I have confirmed the following flight: P.S.A. flight 307 on Friday May 30 from San Diego to San Jose that leaves at 7.45 pm and arrives at 9.30 pm Thank you for calling. Goodbye

**Figure 15.9** The travel domain: A transcript of an actual dialogue with the GUS system of Bobrow et al. (1977). P.S.A. and Air California were airlines of that period.

The set of slots in a task-based dialogue frame specifies what the system needs to know, and the filler of each slot is constrained to values of a particular semantic type. In the travel domain, for example, a slot might be of type city (hence take on values like *San Francisco*, or *Hong Kong*) or of type date, airline, or time.

Slot	Type	Question Template
ORIGIN CITY	city	"From what city are you leaving?"
DESTINATION CITY	city	"Where are you going?"
DEPARTURE TIME	time	"When would you like to leave?"
DEPARTURE DATE	date	"What day would you like to leave?"
ARRIVAL TIME	time	"When do you want to arrive?"
ARRIVAL DATE	date	"What day would you like to arrive?"

**Figure 15.10** A frame in a frame-based dialogue system, showing the type of each slot and a question used to fill the slot.

Types in GUS, as in modern frame-based dialogue agents, have hierarchical structure; for example the *date* type in GUS is itself a frame with slots with types like *integer* or members of sets of weekday names:

DATE

MONTH:NAME YEAR:INTEGER DAY:(BOUNDED-INTEGER 1 31)  
WEEKDAY:(MEMBER (Sunday Monday Tuesday Wednesday

Thursday Friday Saturday))

### 15.3.1 Control structure for frame-based dialogue

The control architecture for frame-based dialogue systems, used in various forms in modern systems like Apple's Siri, Amazon's Alexa, and the Google Assistant, is designed around the frame. The system's goal is to fill the slots in the frame with the fillers the user intends, and then perform the relevant action for the user (answering a question, or booking a flight).

To do this, the system asks questions of the user (using pre-specified question templates associated with each slot of each frame, as shown in Fig. 15.10), filling any slot that the user specifies (we'll describe how slot-filling works in the next section). If a user's response fills multiple slots, like the following:

(15.4) I want a flight from San Francisco to Denver one way leaving after five p.m. on Tuesday.

the system fills all the relevant slots, and then continues asking questions to fill the remaining slots, skipping questions associated with filled slots. The GUS architecture also has condition-action rules attached to slots. For example, a rule attached to the DESTINATION slot for the plane booking frame, once the user has specified the destination, might automatically enter that city as the default *StayLocation* for the related hotel booking frame. Or if the user specifies the DESTINATION DAY for a short trip the system could automatically enter the ARRIVAL DAY.

Many domains require multiple frames. Besides frames for car or hotel reservations, we might need frames with general route information (for questions like *Which airlines fly from Boston to San Francisco?*), or information about airfare practices (for questions like *Do I have to stay a specific number of days to get a decent airfare?*). The system must be able to disambiguate which slot of which frame a given input is supposed to fill and then switch dialogue control to that frame.

Because of this need to dynamically switch control, the GUS architecture is a **production rule** system. Different types of inputs cause different productions to fire, each of which can flexibly fill in different frames. The production rules can then switch control according to factors such as the user's input and some simple dialogue history like the last question that the system asked.

Once the system has enough information it performs the necessary action (like querying a database of flights) and returns the result to the user.

### 15.3.2 Determining Domain, Intent, and Slot fillers in GUS

The goal of the next component in the frame-based architecture is to extract three things from the user's utterance. The first task is **domain classification**: is this user for example talking about airlines, programming an alarm clock, or dealing with their calendar? Of course this 1-of-n classification tasks is unnecessary for single-domain systems that are focused on, say, only calendar management, but multi-domain dialogue systems are the modern standard. The second is user **intent determination**: what general task or goal is the user trying to accomplish? For example the task could be to Find a Movie, or Show a Flight, or Remove a Calendar Appointment. Finally, we need to do **slot filling**: extract the particular slots and fillers that the user intends the system to understand from their utterance with respect to their intent. From a user utterance like this one:

Show me morning flights from Boston to San Francisco on Tuesday

intent  
determination

slot filling



a system might want to build a representation like:

```
DOMAIN:      AIR-TRAVEL
INTENT:      SHOW-FLIGHTS
ORIGIN-CITY: Boston
ORIGIN-DATE: Tuesday
ORIGIN-TIME: morning
DEST-CITY:   San Francisco
```

while an utterance like

```
Wake me tomorrow at 6
```

should give an intent like this:

```
DOMAIN:  ALARM-CLOCK
INTENT:  SET-ALARM
TIME:    2017-07-01 0600-0800
```

The slot-filling method used in the original GUS system, and still quite common in industrial applications, is to use handwritten rules, often as part of the condition-action rules attached to slots or concepts. For example we might just define a regular expression for recognizing the SET-ALARM intent:

```
wake me (up) | set (the|an) alarm | get me up
```

More complicated rules, for example based on context-free grammars and parsers (Chapter 17) can be used, after which the fillers are put into some sort of canonical form, for example by normalizing dates as discussed in Chapter 22. Most industrial dialogue systems employ the GUS architecture but use supervised machine learning for slot-filling instead of these kinds of rules; see Section 15.4.2.

### 15.3.3 Other components of frame-based dialogue

The ASR (automatic speech recognition) component takes audio input from a phone or other device and outputs a transcribed string of words, as discussed in Chapter 16. The ASR component can also be made dependent on the dialogue state. For example, if the system has just asked the user “What city are you departing from?”, the ASR language model component can be constrained to assign high probability to answers with city names. This can be done by training a language model on answers to this question, or by hand-writing finite-state or context-free grammars for such sentences. A language model that is completely dependent on dialogue state is called a **restrictive grammar**; usually real systems mix restrictive grammars with more general language models.

restrictive  
grammar

The **natural language generation** module of any dialogue system produces the utterances that the system says to the user. Frame-based systems tend to use **template-based generation**, in which all or most of the words in the sentence to be uttered to the user are prespecified by the dialogue designer. Sentences created by these templates are often called **prompts**. Templates might be completely fixed (like ‘Hello, how can I help you?’), or can include some variables that are filled in by the generator, as in the following:

template-based  
generation

prompt

```
What time do you want to leave CITY-ORIG?
Will you return to CITY-ORIG from CITY-DEST?
```

It’s possible to do some simple grounding even in templated generation. Consider the unnaturalness of this example from [Cohen et al. \(2004\)](#):

- (15.5) System: Did you want to review some more of your personal profile?  
 Caller: No.  
 System: What's next?

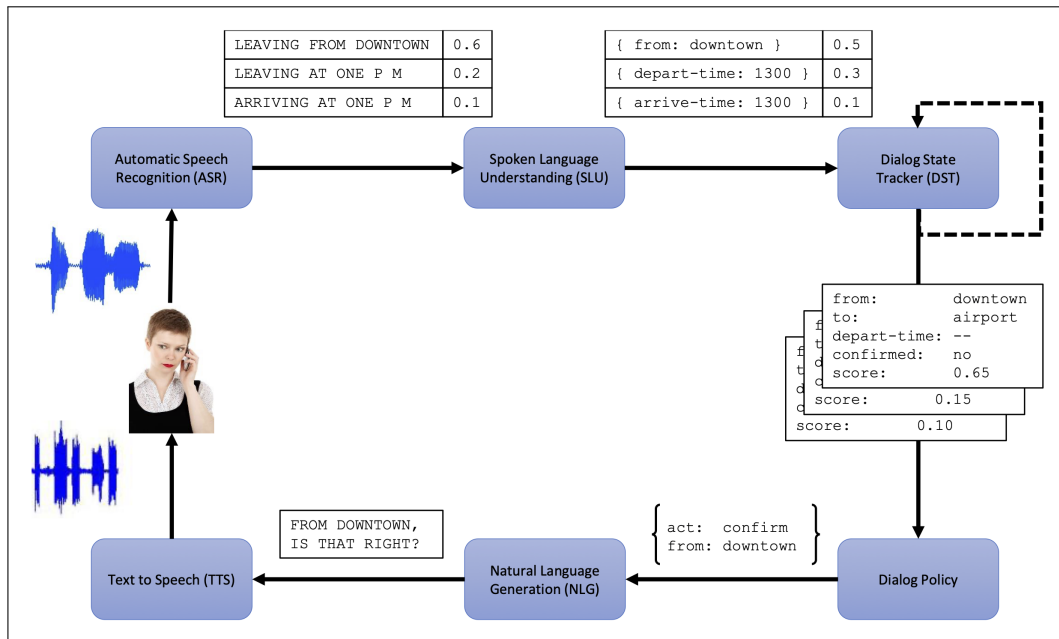
Without an acknowledgment, the caller doesn't know that the system has understood her 'No'. The use of *Okay* below adds grounding to the templated response *What's next?*, making (15.6) a much more natural response than (15.5):

- (15.6) System: Did you want to review some more of your personal profile?  
 Caller: No.  
 System: *Okay*, what's next?

The rule-based GUS approach is very common in industrial applications. As was true with the rule-based approach to information extraction, it has the advantage of high precision, and if the domain is narrow enough and experts are available, can provide sufficient coverage as well. On the other hand, the handwritten rules or grammars can be both expensive and slow to create, and handwritten rules can suffer from recall problems.

## 15.4 The Dialogue-State Architecture

Modern research systems for task-based dialogue are based on a more sophisticated version of the frame-based architecture called the **dialogue-state** or **belief-state** architecture. Figure 15.11 shows the six components of a typical dialogue-state system. The speech recognition and synthesis components deal with spoken language processing; we'll return to them in Chapter 16.



**Figure 15.11** Architecture of a dialogue-state system for task-oriented dialogue from Williams et al. (2016).

For the rest of this chapter we therefore consider the other four components, which are part of both spoken and textual dialogue systems. These four components

are more complex than in the simple GUS systems. For example, like the GUS systems, the dialogue-state architecture has a component for extracting slot fillers from the user's utterance, but generally using machine learning rather than rules. (This component is sometimes called the **NLU** or **SLU** component, for 'Natural Language Understanding', or 'Spoken Language Understanding', using the word "understanding" loosely.) The **dialogue state tracker** maintains the current state of the dialogue (which include the user's most recent dialogue act, plus the entire set of slot-filler constraints the user has expressed so far). The **dialogue policy** decides what the system should do or say next. The dialogue policy in GUS was simple: ask questions until the frame was full and then report back the results of some database query. But a more sophisticated dialogue policy can help a system decide when to answer the user's questions, when to instead ask the user a clarification question, when to make a suggestion, and so on. Finally, dialogue state systems have a **natural language generation** component. In GUS, the sentences that the generator produced were all from pre-written templates. But a more sophisticated generation component can condition on the exact context to produce turns that seem much more natural.

As of the time of this writing, most commercial system are architectural hybrids, based on GUS architecture augmented with some dialogue-state components, but there are a wide variety of dialogue-state systems being developed in research labs.

### 15.4.1 Dialogue Acts

dialogue acts

Dialogue-state systems make use of **dialogue acts**. Dialogue acts represent the interactive function of the turn or sentence, combining the idea of speech acts and grounding into a single representation. Different types of dialogue systems require labeling different kinds of acts, and so the tagset—defining what a dialogue act is exactly—tends to be designed for particular tasks.

Tag	Sys	User	Description
HELLO( $a = x, b = y, \dots$ )	✓	✓	Open a dialogue and give info $a = x, b = y, \dots$
INFORM( $a = x, b = y, \dots$ )	✓	✓	Give info $a = x, b = y, \dots$
REQUEST( $a, b = x, \dots$ )	✓	✓	Request value for a given $b = x, \dots$
REQALTS( $a = x, \dots$ )	✗	✓	Request alternative with $a = x, \dots$
CONFIRM( $a = x, b = y, \dots$ )	✓	✓	Explicitly confirm $a = x, b = y, \dots$
CONFREQ( $a = x, \dots, d$ )	✓	✗	Implicitly confirm $a = x, \dots$ and request value of $d$
SELECT( $a = x, a = y$ )	✓	✗	Implicitly confirm $a = x, \dots$ and request value of $d$
AFFIRM( $a = x, b = y, \dots$ )	✓	✓	Affirm and give further info $a = x, b = y, \dots$
NEGATE( $a = x$ )	✗	✓	Negate and give corrected value $a = x$
DENY( $a = x$ )	✗	✓	Deny that $a = x$
BYE()	✓	✓	Close a dialogue

**Figure 15.12** Dialogue acts used by the HIS restaurant recommendation system of Young et al. (2010). The Sys and User columns indicate which acts are valid as system outputs and user inputs, respectively.

Figure 15.12 shows a tagset for a restaurant recommendation system, and Fig. 15.13 shows these tags labeling a sample dialogue from the HIS system (Young et al., 2010). This example also shows the content of each dialogue acts, which are the slot fillers being communicated. So the user might INFORM the system that they want Italian food near a museum, or CONFIRM with the system that the price is reasonable.

Utterance	Dialogue act
U: Hi, I am looking for somewhere to eat.	hello(task = find, type=restaurant)
S: You are looking for a restaurant. What type of food do you like?	confreq(type = restaurant, food)
U: I'd like an Italian somewhere near the museum.	inform(food = Italian, near=museum)
S: Roma is a nice Italian restaurant near the museum.	inform(name = "Roma", type = restaurant, food = Italian, near = museum)
U: Is it reasonably priced?	confirm(pricerange = moderate)
S: Yes, Roma is in the moderate price range.	affirm(name = "Roma", pricerange = moderate)
U: What is the phone number?	request(phone)
S: The number of Roma is 385456.	inform(name = "Roma", phone = "385456")
U: Ok, thank you goodbye.	bye()

**Figure 15.13** A sample dialogue from the HIS System of [Young et al. \(2010\)](#) using the dialogue acts in Fig. 15.12.

### 15.4.2 Slot Filling

The task of slot-filling, and the simpler tasks of domain and intent classification, are special cases of the task of supervised semantic parsing discussed in Chapter 20, in which we have a training set that associates each sentence with the correct set of slots, domain, and intent.

A simple method is to train a sequence model to map from input words representation to slot fillers, domain and intent. For example given the sentence:

I want to fly to San Francisco on Monday afternoon please

we compute a sentence representation, for example by passing the sentence through a contextual embedding network like BERT. The resulting sentence representation can be passed through a feedforward layer and then a simple 1-of-N classifier to determine that the domain is AIRLINE and the intent is SHOWFLIGHT.

Our training data is sentences paired with sequences of **BIO** labels:

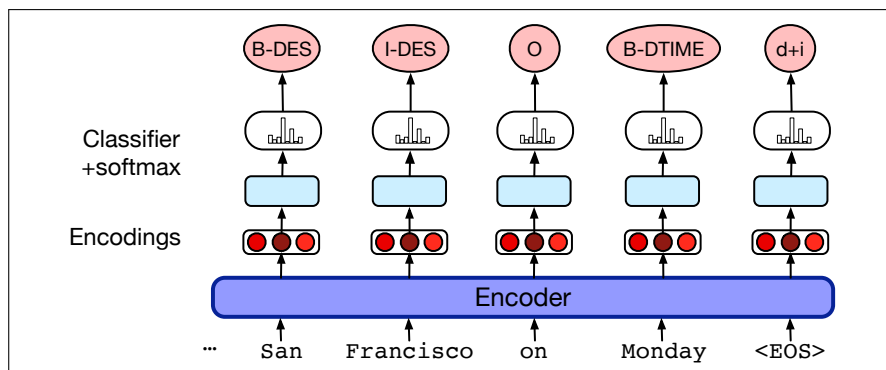
O O O O O B-DES I-DES O B-DEPTIME I-DEPTIME O  
I want to fly to San Francisco on Monday afternoon please

Recall from Chapter 8 that in BIO tagging we introduce a tag for the beginning (B) and inside (I) of each slot label, and one for tokens outside (O) any slot label. The number of tags is thus  $2n + 1$  tags, where  $n$  is the number of slots.

Fig. 15.14 shows the architecture. The input is a series of words  $w_1 \dots w_n$ , which is passed through a contextual embedding model to get contextual word representations. This is followed by a feedforward layer and a softmax at each token position over possible BIO tags, with the output a series of BIO tags  $s_1 \dots s_n$ . We can also combine the domain-classification and intent-extraction tasks with slot-filling simply by adding a domain concatenated with an intent as the desired output for the final EOS token.

Once the sequence labeler has tagged the user utterance, a filler string can be extracted for each slot from the tags (e.g., “San Francisco”), and these word strings can then be normalized to the correct form in the ontology (perhaps the airport code ‘SFO’). This normalization can take place by using homonym dictionaries (specifying, for example, that SF, SFO, and San Francisco are the same place).

In industrial contexts, machine learning-based systems for slot-filling are often bootstrapped from GUS-style rule-based systems in a semi-supervised learning



**Figure 15.14** A simple architecture for slot filling, mapping the words in the input through contextual embeddings like BERT to an output classifier layer (which can be linear or something more complex), followed by softmax to generate a series of BIO tags (and including a final state consisting of a domain concatenated with an intent).

manner. A rule-based system is first built for the domain, and a test set is carefully labeled. As new user utterances come in, they are paired with the labeling provided by the rule-based system to create training tuples. A classifier can then be trained on these tuples, using the test set to test the performance of the classifier against the rule-based system. Some heuristics can be used to eliminate errorful training tuples, with the goal of increasing precision. As sufficient training samples become available the resulting classifier can often outperform the original rule-based system (Suendermann et al., 2009), although rule-based systems may still remain higher-precision for dealing with complex cases like negation.

### 15.4.3 Dialogue State Tracking

The job of the dialogue-state tracker is to determine both the current state of the frame (the fillers of each slot), as well as the user’s most recent dialogue act. The dialogue-state thus includes more than just the slot-fillers expressed in the current sentence; it includes the entire state of the frame at this point, summarizing all of the user’s constraints. The following example from Mrkšić et al. (2017) shows the required output of the dialogue state tracker after each turn:

```
User: I'm looking for a cheaper restaurant
      inform(price=cheap)
System: Sure. What kind - and where?
User: Thai food, somewhere downtown
      inform(price=cheap, food=Thai, area=centre)
System: The House serves cheap Thai food
User: Where is it?
      inform(price=cheap, food=Thai, area=centre); request(address)
System: The House is at 106 Regent Street
```

Since dialogue acts place some constraints on the slots and values, the tasks of dialogue-act detection and slot-filling are often performed jointly. Consider the task of determining that

```
I'd like Cantonese food near the Mission District
has the structure
inform(food=cantonese, area=mission).
```

Dialogue act interpretation—in this example choosing **inform** from the set of dialogue acts for this task—is done by supervised classification trained on hand-labeled dialog acts, predicting the dialogue act tag based on embeddings representing the current input sentence and the prior dialogue acts.

The simplest dialogue state tracker might just take the output of a slot-filling sequence-model (Section 15.4.2) after each sentence. Alternatively, a more complex model can make use of the reading-comprehension architectures from Chapter 14. For example the model of Gao et al. (2019) trains a classifier for each slot to decide whether its value is being changed in the current sentence or should be carried over from the previous sentences. If the slot value is being changed, a span-prediction model is used to predict the start and end of the span with the slot filler.

### A special case: detecting correction acts

user correction  
acts

hyperarticulation

Some dialogue acts are important because of their implications for dialogue control. If a dialogue system misrecognizes or misunderstands an utterance, the user will generally correct the error by repeating or reformulating the utterance. Detecting these **user correction acts** is therefore quite important. Ironically, it turns out that corrections are actually *harder* to recognize than normal sentences! In fact, corrections in one early dialogue system (the TOOT system) had double the ASR word error rate of non-corrections (Swerts et al., 2000)! One reason for this is that speakers sometimes use a specific prosodic style for corrections called **hyperarticulation**, in which the utterance contains exaggerated energy, duration, or F0 contours, such as *I said BAL-TI-MORE, not Boston* (Wade et al. 1992, Levow 1998, Hirschberg et al. 2001). Even when they are not hyperarticulating, users who are frustrated seem to speak in a way that is harder for speech recognizers (Goldberg et al., 2003).

What are the characteristics of these corrections? User corrections tend to be either exact repetitions or repetitions with one or more words omitted, although they may also be paraphrases of the original utterance (Swerts et al., 2000). Detecting these reformulations or correction acts can be part of the general dialogue act detection classifier. Alternatively, because the cues to these acts tend to appear in different ways than for simple acts (like **INFORM** or *request*), we can make use of features orthogonal to simple contextual embedding features; some typical features are shown below (Levow 1998, Litman et al. 1999, Hirschberg et al. 2001, Bulyko et al. 2005, Awadallah et al. 2015).

features	examples
<b>lexical</b>	words like “no”, “correction”, “I don’t”, swear words, utterance length
<b>semantic</b>	similarity (word overlap or embedding dot product) between the candidate correction act and the user’s prior utterance
<b>phonetic</b>	phonetic overlap between the candidate correction act and the user’s prior utterance (i.e. “WhatsApp” may be incorrectly recognized as “What’s up”)
<b>prosodic</b>	hyperarticulation, increases in F0 range, pause duration, and word duration, generally normalized by the values for previous sentences
<b>ASR</b>	ASR confidence, language model probability

## 15.4.4 Dialogue Policy

dialogue policy

The goal of the **dialogue policy** is to decide what action the system should take next, that is, what dialogue act to generate.

More formally, at turn  $i$  in the conversation we want to predict which action  $A_i$

to take, based on the entire dialogue state. The state could mean the entire sequence of dialogue acts from the system (A) and from the user (U), in which case the task would be to compute:

$$\hat{A}_i = \operatorname{argmax}_{A_i \in A} P(A_i | (A_1, U_1, \dots, A_{i-1}, U_{i-1})) \quad (15.7)$$

We can simplify this by maintaining as the dialogue state mainly just the set of slot-fillers that the user has expressed, collapsing across the many different conversational paths that could lead to the same set of filled slots.

Such a policy might then just condition on the current dialogue state as represented just by the current state of the frame  $\text{Frame}_i$  (which slots are filled and with what) and the last turn by the system and user:

$$\hat{A}_i = \operatorname{argmax}_{A_i \in A} P(A_i | \text{Frame}_{i-1}, A_{i-1}, U_{i-1}) \quad (15.8)$$

These probabilities can be estimated by a neural classifier using neural representations of the slot fillers (for example as spans) and the utterances (for example as sentence embeddings computed over contextual embeddings)

More sophisticated models train the policy via **reinforcement learning**. To decide which action to take, a reinforcement learning system gets a reward at the end of the dialogue, and uses that reward to train a policy to take actions. For example in the movie-recommendation dialogue system of [Fazel-Zarandi et al. \(2017\)](#), the action space has only three actions: EXECUTE, CONFIRM, and ELICIT. The EXECUTE sends a query to the database and answers the user’s question, CONFIRM clarifies the intent or slot with the users (e.g., “Do you want movies directed by Christopher Nolan?”) while ELICIT asks the user for missing information (e.g., “Which movie are you talking about?”). The system gets a large positive reward if the dialogue system terminates with the correct slot representation at the end, a large negative reward if the slots are wrong, and a small negative reward for confirmation and elicitation questions to keep the system from re-confirming everything.

### Policy Example: Confirmation and Rejection

Modern dialogue systems often make mistakes. It is therefore important for dialogue systems to make sure that they have achieved the correct interpretation of the user’s input. This is generally done by two methods: **confirming** understandings with the user and **rejecting** utterances that the system is likely to have misunderstood.

As we saw in the prior section, most systems introduce particular strategies and actions related to confirmation and rejection. When using the **explicit confirmation** strategy, a system asks the user a direct question to confirm the system’s understanding, like the two examples below in which the system asks a (boldface) yes-no confirmation question:

explicit  
confirmation

S: Which city do you want to leave from?
U: Baltimore.
S: <b>Do you want to leave from Baltimore?</b>
U: Yes.
U: I’d like to fly from Denver Colorado to New York City on September twenty first in the morning on United Airlines
S: <b>Let’s see then. I have you going from Denver Colorado to New York on September twenty first. Is that correct?</b>
U: Yes



**implicit  
confirmation**

When using the **implicit confirmation** strategy, a system instead can demonstrate its understanding as a **grounding** strategy, for example repeating back the system’s understanding as part of asking the next question, as in the two examples below:

U:	I want to travel to Berlin
S:	<b>When do you want to travel to Berlin?</b>
U2:	Hi I’d like to fly to Seattle Tuesday Morning
A3:	<b>Traveling to Seattle on Tuesday, August eleventh in the morning.</b> Your full name?

Explicit and implicit confirmation have complementary strengths. Explicit confirmation makes it easier for users to correct the system’s misrecognitions since a user can just answer “no” to the confirmation question. But explicit confirmation is awkward and increases the length of the conversation (Danieli and Gerbino 1995, Walker et al. 1998). The explicit confirmation dialogue fragments above sound non-natural and definitely non-human; implicit confirmation is much more conversationally natural.

**rejection**

Confirmation is just one kind of conversational action by which a system can express lack of understanding. Another option is **rejection**, in which a system gives the user a prompt like *I’m sorry, I didn’t understand that*.

**progressive  
prompting**

Sometimes utterances are rejected multiple times. This might mean that the user is using language that the system is unable to follow. Thus, when an utterance is rejected, systems often follow a strategy of **progressive prompting** or **escalating detail** (Yankelovich et al. 1995, Weinschenk and Barker 2000), as in this example from Cohen et al. (2004):

System:	When would you like to leave?
Caller:	Well, um, I need to be in New York in time for the first World Series game.
System:	<reject>. Sorry, I didn’t get that. Please say the month and day you’d like to leave.
Caller:	I wanna go on October fifteenth.

In this example, instead of just repeating “When would you like to leave?”, the rejection prompt gives the caller more guidance about how to formulate an utterance the system will understand. These *you-can-say* help messages are important in helping improve systems’ understanding performance (Bohus and Rudnicky, 2005). If the caller’s utterance gets rejected yet again, the prompt can reflect this (“I *still* didn’t get that”), and give the caller even more guidance.

**rapid  
reprompting**

An alternative strategy for error handling is **rapid reprompting**, in which the system rejects an utterance just by saying “I’m sorry?” or “What was that?” Only if the caller’s utterance is rejected a second time does the system start applying progressive prompting. Cohen et al. (2004) summarize experiments showing that users greatly prefer rapid reprompting as a first-level error prompt.

It is common to use rich features other than just the dialogue state representation to make policy decisions. For example, the **confidence** that the ASR system assigns to an utterance can be used by explicitly confirming low-confidence sentences. Confidence is a metric that the speech recognizer can assign to its transcription of a sentence to indicate how confident it is in that transcription. Confidence is often computed from the acoustic log-likelihood of the utterance (greater probability means higher confidence), but prosodic features can also be used in confidence

prediction. For example, utterances with large F0 excursions or longer durations, or those preceded by longer pauses, are likely to be misrecognized (Litman et al., 2000).

Another common feature in confirmation is the **cost** of making an error. For example, explicit confirmation is common before a flight is actually booked or money in an account is moved. Systems might have a four-tiered level of confidence with three thresholds  $\alpha$ ,  $\beta$ , and  $\gamma$ :

$< \alpha$	low confidence	reject
$\geq \alpha$	above the threshold	confirm explicitly
$\geq \beta$	high confidence	confirm implicitly
$\geq \gamma$	very high confidence	don't confirm at all

15.4.5 Natural language generation in the dialogue-state model

content  
planning  
sentence  
realization

Once a dialogue act has been decided, we need to generate the text of the response to the user. The task of natural language generation (NLG) in the information-state architecture is often modeled in two stages, **content planning** (what to say), and **sentence realization** (how to say it).

Here we'll assume content planning has been done by the dialogue policy, which has chosen the dialogue act to generate, and chosen some attributes (slots and values) that the planner wants to say to the user (either to give the user the answer, or as part of a confirmation strategy).

<code>recommend(restaurant name= Au Midi, neighborhood = midtown, cuisine = french</code>
1 Au Midi is in Midtown and serves French food.
2 There is a French restaurant in Midtown called Au Midi.
<code>recommend(restaurant name= Loch Fyne, neighborhood = city centre, cuisine = seafood)</code>
3 Loch Fyne is in the City Center and serves seafood food.
4 There is a seafood restaurant in the City Centre called Loch Fyne.

**Figure 15.15** Two examples of inputs to the sentence realization phase of NLG, showing the dialogue act and attributes prespecified by the content planner. Line 1-2 and 3-4 show different possible output sentences to be generated by the sentence realizer. From the restaurant recommendation system of Nayak et al. (2017).

Fig. 15.15 shows some sample input/outputs for the sentence realization phase. In the first example, the content planner has chosen the dialogue act RECOMMEND and some particular slots (name, neighborhood, cuisine) and their fillers. The goal of the sentence realizer is to generate a sentence like lines 1 or 2 shown in the figure, by training on many such examples of representation/sentence pairs from a large corpus of labeled dialogues.

delexicalization

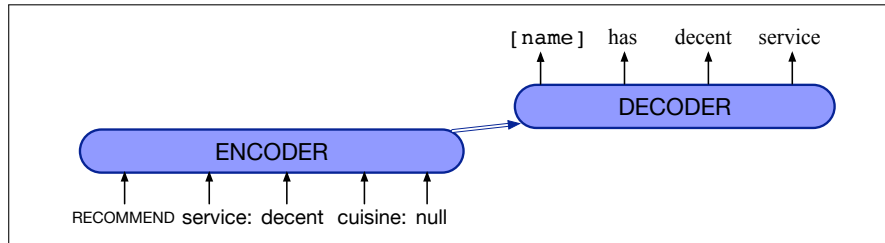
Training data is hard to come by; we are unlikely to see every possible restaurant with every possible attribute in many possible differently worded sentences. Therefore it is common in sentence realization to increase the generality of the training examples by **delexicalization**. Delexicalization is the process of replacing specific words in the training set that represent slot values with a generic placeholder token representing the slot. Fig. 15.16 shows the result of delexicalizing the training sentences in Fig. 15.15.

Mapping from frames to delexicalized sentences is generally done by encoder decoder models (Wen et al. 2015a, Wen et al. 2015b, Mrkšić et al. 2017, inter

<code>recommend(restaurant name= Au Midi, neighborhood = midtown, cuisine = french</code>
1 restaurant_name is in neighborhood and serves cuisine food.
2 There is a cuisine restaurant in neighborhood called restaurant_name.

**Figure 15.16** Delexicalized sentences that can be used for generating many different relexicalized sentences. From the restaurant recommendation system of [Nayak et al. \(2017\)](#).

alia), trained on large hand-labeled corpora of task-oriented dialogue ([Budzianowski et al., 2018](#)). The input to the encoder is a sequence of tokens  $x_t$  that represent the dialogue act and its arguments. Thus the dialogue act RECOMMEND and the attribute/value pairs service:decent, cuisine:null might be represented as a flat sequence of tokens ([Nayak et al., 2017](#)), each mapped to a learned embedding  $w_t$ , as shown in Fig. 15.17.



**Figure 15.17** An encoder decoder sentence realizer mapping slots/fillers to English.

The encoder reads all the input slot/value representations, and the decoder outputs the following delexicalized English sentence:

restaurant\_name has decent service

**relexicalize** We can then use the input frame from the content planner to **relexicalize** (fill in the exact restaurant or neighborhood or cuisine) resulting in:

Au Midi has decent service

### Generating Clarification Questions

**clarification questions**

It's also possible to design NLG algorithms that are specific to a particular dialogue act. For example, consider the task of generating **clarification questions**, in cases where the speech recognition fails to understand some part of the user's utterance. While it is possible to use the generic dialogue act REJECT ("Please repeat", or "I don't understand what you said"), studies of human conversations show that humans instead use targeted clarification questions that reprise elements of the misunderstanding ([Purver 2004](#), [Ginzburg and Sag 2000](#), [Stoyanchev et al. 2013](#)).

For example, in the following hypothetical example the system reprises the words "going" and "on the 5th" to make it clear which aspect of the user's turn the system needs to be clarified:

User: What do you have going to UNKNOWN\_WORD on the 5th?

System: Going where on the 5th?

Targeted clarification questions can be created by rules (such as replacing "going to UNKNOWN\_WORD" with "going where") or by building classifiers to guess which slots might have been misrecognized in the sentence ([Chu-Carroll and Carpenter 1999](#), [Stoyanchev et al. 2014](#), [Stoyanchev and Johnston 2015](#)).

## 15.5 Evaluating Dialogue Systems

Evaluation is crucial in dialogue system design. Chatbots and task-based systems are generally evaluated differently, since they have different goals; task-based systems have to complete a task like booking a flight; chatbots have a different kind of goal, like being enjoyable to users.

### 15.5.1 Evaluating Chatbots

Chatbots are evaluated by humans, who assign a score. This can be the human who talked to the chatbot (**participant evaluation**) or a third party who reads a transcript of a human/chatbot conversation (**observer evaluation**).

In the participant evaluation of [See et al. \(2019\)](#), the human evaluator chats with the model for six turns and rates the chatbot on 8 dimensions capturing conversational quality: avoiding repetition, interestingness, making sense, fluency, listening, inquisitiveness, humanness and engagingness. A few examples:

**Engagingness** How much did you enjoy talking to this user?

- Not at all • A little • Somewhat • A lot

**Avoiding Repetition** How repetitive was this user?

- Repeated themselves over and over • Sometimes said the same thing twice
- Always said something new

**Making sense** How often did this user say something which did NOT make sense?

- Never made any sense • Most responses didn't make sense • Some responses didn't make sense • Everything made perfect sense

Observer evaluations use third party annotators to look at the text of a complete conversation. Sometimes we're interested in having raters assign a score to each system turn; for example ([Artstein et al., 2009](#)) have raters mark how *coherent* each turn is. Often, however, we just want a single high-level score to know if system A is better than system B. The **acute-eval** metric ([Li et al., 2019](#)) is such an observer evaluation in which annotators look at two separate human-computer conversations (A and B) and choose the one in which the dialogue system participant performed better (interface shown in Fig. 15.18). They answer the following 4 questions (with these particular wordings shown to lead to high agreement):

**Engagingness** Who would you prefer to talk to for a long conversation?

**Interestingness** If you had to say one of these speakers is interesting and one is boring, who would you say is more interesting?

**Humanness** Which speaker sounds more human?

**Knowledgeable** If you had to say that one speaker is more knowledgeable and one is more ignorant, who is more knowledgeable?

Automatic evaluations are generally not used for chatbots. That's because computational measures of generation performance like BLEU or ROUGE or embedding dot products between a chatbot's response and a human response correlate very poorly with human judgments ([Liu et al., 2016](#)). These methods perform poorly because there are so many possible responses to any given turn; simple word-overlap or semantic similarity metrics work best when the space of responses is small and lexically overlapping, which is true of generation tasks like machine translation or possibly summarization, but definitely not dialogue.

Hi how are you today

Hi, I'm pretty good! Just listening to some aerosmith, they're my fave :) whatre you up to?

I am listening to some italian music

Italian music, nice! What do you do for work?

I'm a professional wrestler

Oh wow! I'm a construction worker, I've been doing it since I was 16.

What do you do for a living

I'm a construction worker haha, just told you! Whats your favorite type of food?

My favorite is garlic bread

Yum! I love mexican food, so good :) do you have any hobbies?

I like to workout and go to the gym

We're a bit different- I love watching nascar and ufc. They're so fun!

Hello there, how are you?

I am doing great. How are you?

I am great, I did something crazy for me and colored my hair blue!

I have a daughter and a son who also love blue colored balls. You should meet them

Well that neat, I got a new car my mother gave so maybe I could see them!

It is a beautiful city. And, I try to be... Just cannot afford a bigger house atm.

I am sorry to hear that, I feel bad going out of town for spring break now.

Ok. I going to school in the spring for casino manager

Well I turn 29 next week, I wonder if that is a good age to apply as one.

My grandmother just died from lung cancer, sucks

**Who would you prefer to talk to for a long conversation?**

☐ I would prefer to talk to **Speaker 1**    ☐ I would prefer to talk to **Speaker 2**

**Please provide a brief justification for your choice (a few words or a sentence)**

Please enter here...

**Figure 15.18** The ACUTE-EVAL method asks annotators to compare two dialogues and choose between Speaker 1 (light blue) and Speaker 2 (dark blue), independent of the gray speaker. Figure from Li et al. (2019).

adversarial  
evaluation

However, research continues in ways to do more sophisticated automatic evaluations that go beyond word similarity. One novel paradigm is **adversarial evaluation** (Bowman et al. 2016, Kannan and Vinyals 2016, Li et al. 2017), inspired by the Turing test. The idea is to train a “Turing-like” evaluator classifier to distinguish between human-generated responses and machine-generated responses. The more successful a response generation system is at fooling this evaluator, the better the system.

### 15.5.2 Evaluating Task-Based Dialogue

For task-based dialogue, if the task is unambiguous, we can simply measure absolute task success (did the system book the right plane flight, or put the right event on the calendar).

To get a more fine-grained idea of user happiness, we can compute a *user satisfaction rating*, having users interact with a dialogue system to perform a task and then having them complete a questionnaire. For example, Fig. 15.19 shows sample multiple-choice questions (Walker et al., 2001); responses are mapped into the range of 1 to 5, and then averaged over all questions to get a total user satisfaction rating.

It is often economically infeasible to run complete user satisfaction studies after every change in a system. For this reason, it is useful to have performance evaluation heuristics that correlate well with human satisfaction. A number of such factors and heuristics have been studied, often grouped into two kinds of criteria: how well the

system allows users to accomplish their goals (maximizing task success) with the fewest problems (minimizing costs):

**Task completion success:** Task success can be measured by evaluating the correctness of the total solution. For a frame-based architecture, this might be **slot error rate**, the percentage of slots that were filled with the correct values:

$$\text{Slot Error Rate for a Sentence} = \frac{\# \text{ of inserted/deleted/substituted slots}}{\# \text{ of total reference slots for sentence}} \quad (15.9)$$

For example consider a system given this sentence:

(15.10) Make an appointment with Chris at 10:30 in Gates 104

which extracted the following candidate slot structure:

Slot	Filler
PERSON	Chris
TIME	11:30 a.m.
ROOM	Gates 104

Here the slot error rate is 1/3, since the TIME is wrong. Instead of error rate, slot precision, recall, and F-score can also be used. Slot error rate is also sometimes called **concept error rate**.

Interestingly, sometimes the user's *perception* of whether they completed the task is a better predictor of user satisfaction than the actual task completion success. (Walker et al., 2001).

A perhaps more important, although less fine-grained, measure of success is an extrinsic metric like **task error rate**. In this case, the task error rate would quantify how often the correct meeting was added to the calendar at the end of the interaction.

**Efficiency cost:** Efficiency costs are measures of the system's efficiency at helping users. This can be measured by the total elapsed time for the dialogue in seconds, the number of total turns or of system turns, or the total number of queries (Polifroni et al., 1992). Other metrics include the number of system non-responses and the "turn correction ratio": the number of system or user turns that were used solely to correct errors divided by the total number of turns (Danieli and Gerbino 1995, Hirschman and Pao 1993).

**Quality cost:** Quality cost measures other aspects of the interactions that affect user's perception of the system. One such measure is the number of times the ASR system failed to return any sentence, or the number of ASR rejection prompts. Similar metrics include the number of times the user had to barge in (interrupt the system), or the number of time-out prompts played when the user didn't respond quickly enough. Other quality metrics focus on how well the system understood and responded to the user. The most important is the **slot error rate** described above,

<b>TTS Performance</b>	Was the system easy to understand ?
<b>ASR Performance</b>	Did the system understand what you said?
<b>Task Ease</b>	Was it easy to find the message/flight/train you wanted?
<b>Interaction Pace</b>	Was the pace of interaction with the system appropriate?
<b>User Expertise</b>	Did you know what you could say at each point?
<b>System Response</b>	How often was the system sluggish and slow to reply to you?
<b>Expected Behavior</b>	Did the system work the way you expected it to?
<b>Future Use</b>	Do you think you'd use the system in the future?

**Figure 15.19** User satisfaction survey, adapted from Walker et al. (2001).



but other components include the inappropriateness (verbose or ambiguous) of the system's questions, answers, and error messages or the correctness of each question, answer, or error message (Zue et al. 1989, Polifroni et al. 1992).

## 15.6 Dialogue System Design

voice user  
interface

The user plays a more important role in dialogue systems than in most other areas of speech and language processing, and thus the study of dialogue systems is closely linked with the field of Human-Computer Interaction (HCI). The design of dialogue strategies, prompts, and error messages, is often called **voice user interface** design, and generally follows **user-centered design** principles (Gould and Lewis, 1985):

**1. Study the user and task:** Understand the potential users and the nature of the task by interviews with users, investigation of similar systems, and study of related human-human dialogues.

Wizard-of-Oz  
system

**2. Build simulations and prototypes:** A crucial tool in building dialogue systems is the **Wizard-of-Oz system**. In wizard systems, the users interact with what they think is a software agent but is in fact a human “wizard” disguised by a software interface (Gould et al. 1983, Good et al. 1984, Fraser and Gilbert 1991). The name comes from the children's book *The Wizard of Oz* (Baum, 1900), in which the wizard turned out to be just a simulation controlled by a man behind a curtain or screen.

A Wizard-of-Oz system can be used to test out an architecture before implementation; only the interface software and databases need to be in place. The wizard gets input from the user, has a graphical interface to a database to run sample queries based on the user utterance, and then has a way to output sentences, either by typing them or by some combination of selecting from a menu and typing.

The results of a Wizard-of-Oz system can also be used as training data to train a pilot dialogue system. While Wizard-of-Oz systems are very commonly used, they are not a perfect simulation; it is difficult for the wizard to exactly simulate the errors, limitations, or time constraints of a real system; results of wizard studies are thus somewhat idealized, but still can provide a useful first idea of the domain issues.



**3. Iteratively test the design on users:** An iterative design cycle with embedded user testing is essential in system design (Nielsen 1992, Cole et al. 1997, Yankelovich et al. 1995, Landauer 1995). For example in a well-known incident in dialogue design history, an early dialogue system required the user to press a key to interrupt the system (Stifelman et al., 1993). But user testing showed users barged in, which led to a redesign of the system to recognize overlapped speech. The iterative method is also important for designing prompts that cause the user to respond in normative ways. It's also important to incorporate **value sensitive design**, in which we

value sensitive  
design



carefully consider during the design process the benefits, harms and possible stakeholders of the resulting system (Friedman et al. 2017, Bender and Friedman 2018).

There are a number of good books on conversational interface design (Cohen et al. 2004, Harris 2005, Pearl 2017, Deibel and Evanhoe 2021).

### 15.6.1 Ethical Issues in Dialogue System Design

Ethical issues have been key to how we think about designing artificial agents since well before we had conversational agents. Mary Shelley (depicted below) centered her novel *Frankenstein* around the problem of creating agents without considering ethical and humanistic concerns. One issue is the **safety** of users. If users seek information from conversational agents in safety-critical situations like asking medical advice, or in emergency situations, or when indicating the intentions of self-harm, incorrect advice can be dangerous and even life-threatening. For example (Bickmore et al., 2018) gave participants medical problems to pose to three commercial dialogue systems (Siri, Alexa, Google Assistant) and asked them to determine an action to take based on the system responses; many of the proposed actions, if actually taken, would have led to harm or death.



Tay

A system can also harm users by verbally attacking them, or creating *representational harms* (Blodgett et al., 2020) by generating abusive or harmful stereotypes that demean particular groups of people. Both abuse and stereotypes can cause psychological harm to users. Microsoft's 2016 **Tay** chatbot, for example, was taken offline 16 hours after it went live, when it began posting messages with racial slurs, conspiracy theories, and personal attacks on its users. Tay had learned these biases and actions from its training data, including from users who seemed to be purposely teaching the system to repeat this kind of language (Neff and Nagy 2016). Henderson et al. (2017) examined dialogue datasets used to train corpus-based chatbots and found toxic and abusive language, especially in social media corpora like Twitter and Reddit, and indeed such language then appears in the text generated by language models and dialogue systems (Gehman et al. 2020; Xu et al. 2020) which can even amplify the bias from the training data (Dinan et al., 2020). Liu et al. (2020) developed another method for investigating bias, testing how neural dialogue systems responded to pairs of simulated user turns that are identical except for mentioning different genders or race. They found, for example, that simple changes like using the word 'she' instead of 'he' in a sentence caused systems to respond more offensively and with more negative sentiment.

Another important ethical issue is **privacy**. Already in the first days of ELIZA, Weizenbaum pointed out the privacy implications of people's revelations to the chatbot. The ubiquity of in-home dialogue agents means they may often overhear private information (Henderson et al., 2017). If a chatbot is human-like, users are also more likely to disclose private information, and less likely to worry about the harm of this disclosure (Ischen et al., 2019). In general, chatbots that are trained on transcripts of human-human or human-machine conversation must anonymize personally identifiable information.

Finally, chatbots raise important issues of gender equality in addition to textual bias. Current chatbots are overwhelmingly given female names, likely perpetuating the stereotype of a subservient female servant (Paolino, 2017). And when users

use sexually harassing language, most commercial chatbots evade or give positive responses rather than responding in clear negative ways (Fessler, 2017).

These ethical issues are an important area of investigation, including finding ways to mitigate problems of abuse and toxicity, like detecting and responding appropriately to toxic contexts (Wolf et al. 2017, Dinan et al. 2020, Xu et al. 2020). Value sensitive design, carefully considering possible harms in advance (Friedman et al. 2017, Bender and Friedman 2018) is also important; (Dinan et al., 2021) give a number of suggestions for best practices in dialogue system design. And because dialogue systems by definition involve human participants, researchers also work on these issues with the Institutional Review Boards (**IRB**) at their institutions, who help protect the safety of experimental subjects.

## 15.7 Summary

**Conversational agents** are crucial speech and language processing applications that are already widely used commercially.

- In human dialogue, speaking is a kind of action; these acts are referred to as speech acts or dialogue acts. Speakers also attempt to achieve **common ground** by acknowledging that they have understood each other. Conversation also is characterized by turn structure and dialogue structure.
- Chatbots are conversational agents designed to mimic the appearance of informal human conversation. Rule-based chatbots like ELIZA and its modern descendants use rules to map user sentences into system responses. Corpus-based chatbots mine logs of human conversation to learn to automatically map user sentences into system responses.
- For task-based dialogue, most commercial dialogue systems use the GUS or frame-based architecture, in which the designer specifies frames consisting of slots that the system must fill by asking the user.
- The **dialogue-state** architecture augments the GUS frame-and-slot architecture with richer representations and more sophisticated algorithms for keeping track of user's dialogue acts, **policies** for generating its own dialogue acts, and a natural language component.
- Dialogue systems are a kind of human-computer interaction, and general HCI principles apply in their design, including the role of the user, simulations such as Wizard-of-Oz systems, and the importance of iterative design and testing on real users.

## Bibliographical and Historical Notes

The earliest conversational systems were chatbots like ELIZA (Weizenbaum, 1966) and PARRY (Colby et al., 1971). ELIZA had a widespread influence on popular perceptions of artificial intelligence, and brought up some of the first ethical questions in natural language processing —such as the issues of privacy we discussed above as well the role of algorithms in decision-making— leading its creator Joseph Weizenbaum to fight for social responsibility in AI and computer science in general.

Another early system, the GUS system (Bobrow et al., 1977) had by the late 1970s established the main frame-based paradigm that became the dominant industrial paradigm for dialogue systems for over 30 years.

In the 1990s, stochastic models that had first been applied to natural language processing began to be applied to dialogue slot filling (Miller et al. 1994, Pieraccini et al. 1991).

By around 2010 the GUS architecture finally began to be widely used commercially in dialogue systems on phones like Apple’s SIRI (Bellegarda, 2013) and other digital assistants.

The rise of the web and online chatbots brought new interest in chatbots and gave rise to corpus-based chatbot architectures around the turn of the century, first using information retrieval models and then in the 2010s, after the rise of deep learning, with sequence-to-sequence models.

The idea that utterances in a conversation are a kind of **action** being performed by the speaker was due originally to the philosopher Wittgenstein (1953) but worked out more fully by Austin (1962) and his student John Searle. Various sets of speech acts have been defined over the years, and a rich linguistic and philosophical literature developed, especially focused on explaining the use of indirect speech acts.

conversation  
analysis

The idea of dialogue acts draws also from a number of other sources, including the ideas of adjacency pairs, pre-sequences, and other aspects of the international properties of human conversation developed in the field of **conversation analysis** (see Levinson (1983) for an introduction to the field).

This idea that acts set up strong local dialogue expectations was also prefigured by Firth (1935, p. 70), in a famous quotation:

*Most of the give-and-take of conversation in our everyday life is stereotyped and very narrowly conditioned by our particular type of culture. It is a sort of roughly prescribed social ritual, in which you generally say what the other fellow expects you, one way or the other, to say.*

Another important research thread modeled dialogue as a kind of collaborative behavior, including the ideas of common ground (Clark and Marshall, 1981), reference as a collaborative process (Clark and Wilkes-Gibbs, 1986), joint intention (Levesque et al., 1990), and shared plans (Grosz and Sidner, 1980).

The dialogue-state model was also strongly informed by analytic work on the linguistic properties of dialogue acts and on methods for their detection (Sag and Liberman 1975, Hinkelman and Allen 1989, Nagata and Morimoto 1994, Goodwin 1996, Chu-Carroll 1998, Shriberg et al. 1998, Stolcke et al. 2000, Gravano et al. 2012).

BDI

Two important lines of research that we were unable to cover in the chapter focused on the computational properties of conversational structure. One line, first suggested by Bruce (1975), suggested that since speech acts are actions, they should be planned like other actions, and drew on the AI planning literature (Fikes and Nilsson, 1971). An agent seeking to find out some information can come up with the plan of asking the interlocutor for the information. An agent hearing an utterance can interpret a speech act by running the planner “in reverse”, using inference rules to infer from what the interlocutor said what the plan might have been. Plan-based models of dialogue are referred to as **BDI** models because such planners model the **beliefs**, **desires**, and **intentions** (BDI) of the agent and interlocutor. BDI models of dialogue were first introduced by Allen, Cohen, Perrault, and their colleagues in a number of influential papers showing how speech acts could be generated (Cohen and Perrault, 1979) and interpreted (Perrault and Allen 1980, Allen and Perrault 1980). At the

same time, [Wilensky \(1983\)](#) introduced plan-based models of understanding as part of the task of interpreting stories.

Another influential line of research focused on modeling the hierarchical structure of dialogue. Grosz's pioneering [1977](#) dissertation first showed that "task-oriented dialogues have a structure that closely parallels the structure of the task being performed" (p. 27), leading to her work with Sidner and others showing how to use similar notions of intention and plans to model discourse structure and coherence in dialogue. See, e.g., [Lochbaum et al. \(2000\)](#) for a summary of the role of intentional structure in dialogue.

The idea of applying reinforcement learning to dialogue first came out of AT&T and Bell Laboratories around the turn of the century with work on MDP dialogue systems ([Walker 2000](#), [Levin et al. 2000](#), [Singh et al. 2002](#)) along with work on cue phrases, prosody, and rejection and confirmation. Reinforcement learning research turned quickly to the more sophisticated POMDP models ([Roy et al. 2000](#), [Lemon et al. 2006](#), [Williams and Young 2007](#)) applied to small slot-filling dialogue tasks. Neural reinforcement learning models have been used both for chatbot systems, for example simulating dialogues between two virtual agents, rewarding good conversational properties like coherence and ease of answering ([Li et al., 2016c](#)), and for task-oriented dialogue ([Williams et al., 2017](#)).

Dialogue state tracking quickly became an important problem for task-oriented dialogue, and there is an influential by an annual evaluation of state-tracking algorithms ([Williams et al., 2016](#)). Other important dialogue areas include the study of affect in dialogue ([Rashkin et al. 2019](#), [Lin et al. 2019](#)). See [Gao et al. \(2019\)](#) for a survey of modern dialogue system architectures, [McTear \(2020\)](#) on conversational AI in general, and [Deibel and Evanhoe \(2021\)](#) on conversation design.

## Exercises

dispreferred  
response

- 15.1 Write a finite-state automaton for a dialogue manager for checking your bank balance and withdrawing money at an automated teller machine.
- 15.2 A **dispreferred response** is a response that has the potential to make a person uncomfortable or embarrassed in the conversational context; the most common example dispreferred responses is turning down a request. People signal their discomfort with having to say no with surface cues (like the word *well*), or via significant silence. Try to notice the next time you or someone else utters a dispreferred response, and write down the utterance. What are some other cues in the response that a system might use to detect a dispreferred response? Consider non-verbal cues like eye gaze and body gestures.
- 15.3 When asked a question to which they aren't sure they know the answer, people display their lack of confidence by cues that resemble other dispreferred responses. Try to notice some unsure answers to questions. What are some of the cues? If you have trouble doing this, read [Smith and Clark \(1993\)](#) and listen specifically for the cues they mention.
- 15.4 Implement a small air-travel help system based on text input. Your system should get constraints from users about a particular flight that they want to take, expressed in natural language, and display possible flights on a screen. Make simplifying assumptions. You may build in a simple flight database or you may use a flight information system on the Web as your backend.