AirConcierge: Generating Task-Oriented Dialogue via Efficient Large-Scale Knowledge Retrieval

Chieh-Yang Chen[†] **Pei-Hsin Wang**[†] **Shih-Chieh Chang**[†]
Da-Cheng Juan[¶] **Wei Wei**[¶] **Jia-Yu Pan**[¶]
[†]National Tsing-Hua University [¶]Google Research
{darius107062542, peihsin}@gapp.nthu.edu.tw
scchang@cs.nthu.edu.tw
{dacheng, wewei, jypan}@google.com

Abstract

Despite recent success in neural task-oriented dialogue systems, developing such a real-world system involves accessing large-scale knowledge bases (KBs), which cannot be simply encoded by neural approaches, such as memory network mechanisms. To alleviate the above problem, we propose AirConcierge, an end-to-end trainable text-to-SQL guided framework to learn a neural agent that interacts with KBs using the generated SQL queries. Specifically, the neural agent first learns to ask and confirm the customer's intent during the multi-turn interactions, then dynamically determining when to ground the user constraints into executable SQL queries so as to fetch relevant information from KBs. With the help of our method, the agent can use less but more accurate fetched results to generate useful responses efficiently, instead of incorporating the entire KBs. We evaluate the proposed method on the AirDialogue dataset, a large corpus released by Google, containing the conversations of customers booking flight tickets from the agent. The experimental results show that AirConcierge significantly improves over previous work in terms of accuracy and the BLEU score, which demonstrates not only the ability to achieve the given task but also the good quality of the generated dialogues.

1 Introduction

The task-oriented dialogue system (Young et al., 2013) is one of the rapidly growing fields with many practical applications, attracting more and more research attention recently (Zhao and Eskénazi, 2016; Wen et al., 2016; Bordes et al., 2017; Dhingra et al., 2017; Eric and Manning, 2017; Liu and Lane, 2017). In order to assist users in solving a specific task while holding conversations with human, the agent needs to understand the intentions of a user during the conversation and fulfills the request. Such a process often involves interacting with external KBs to access task-related information. Figure 1 shows an example of a task-oriented dialogue between a user and an airline ticket reservation agent.

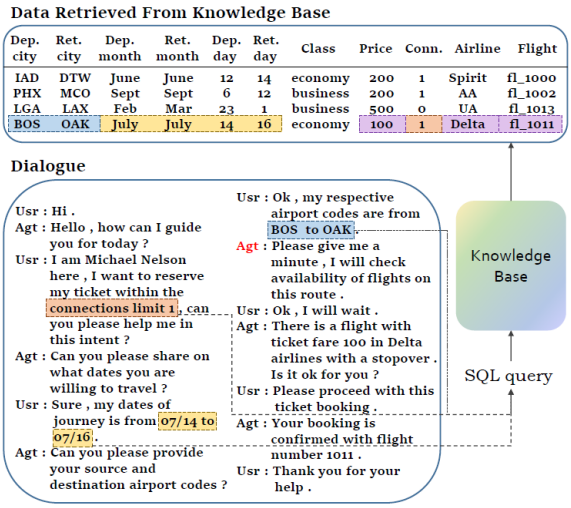

Figure 1: An example of the task-oriented dialogue that incorporates a knowledge base (KB) from the AirDialogue dataset. The agent ground the user constraints into executable SQL query at the turn annotated in red.

Traditional dialogue systems (Kim et al., 2008; Deoras and Sarikaya, 2013) may rely on the predefined slot-filling pairs, where a set of slots needs to be filled during the conversation. In addition, some works (Sukhbaatar et al., 2015; Madotto et al., 2018; Wu et al., 2019) have considered integrating KBs in a task-oriented dialogue system to generate a suitable response and have achieved promising performance. However, these methods either are limited by predefined configurations or do not scale to large KBs. Since real-world KBs typically contain millions of records, end-to-end dialogue systems are not able to incorporate external KBs effectively, leading to unstable dialogue responses.

Moreover, very few research has attempted to

Findings of the Association for Computational Linguistics: EMNLP 2020, pages 884–897

explore how to efficiently cooperate with KBs or taken resource consumption, such as FLOPs or memory space, into consideration when designing the model. In order to solve the issues mentioned above, we propose AirConcierge, an SQL-guided task-oriented dialogue system that can efficiently work with real-world, large-scale KBs, by formulating SQL queries based on the context of the dialogue so as to retrieve relevant information from KBs.

We evaluate and demonstrate AirConcierge on AirDialogue (Wei et al., 2018), a large-scale airline reserving dataset published recently. AirDialogue has high complexity in contexts, creating the opportunity and the necessity of forming diverse task-oriented conversations. Our experiments show that AirConcierge achieves improvements in accuracy and resource usage compared to previous work.

2 Related Work

2.1 Task-oriented Dialogue System

Traditional task-oriented dialogue systems are usually accompanied by complex modular pipelines (Rudnicky et al., 1999; Zue, 2000; Zue et al., 2000). Each module is trained individually and follows by being pipelined for testing, so error from previous modules may propagate to downstream modules. Therefore, several jointed learning (Yang et al., 2017) and end-to-end reinforcement learning (RL) framework (Zhao and Eskénazi, 2016) are proposed to jointly train NLU and dialog manager using specifically collected supervised labels or user utterances to migrate the above problems. Other different end-to-end trainable dialogue systems (Wen et al., 2016; Li et al., 2017) have also been proposed and achieved successful performance by using supervised learning or RL. Compared to the pure end-to-end system, intermediate labels are still added to the model to train NLU and DST.

Existing pipeline methods to task-oriented dialogue systems still have problems of structural complexity and fragility. For example, NLU typically detects dialog domains by parsing user utterances, then classifying user intentions, and filling a set of slots to form domain-specific semantic frames. These models may highly rely on manual feature engineering, which makes them laborious and time-consuming and are difficult to adapt to new domains. Therefore, more and more research (Manning and Eric, 2017; Sukhbaatar et al., 2015; Dodge et al., 2016; Serban et al., 2016; Bordes et al., 2017; Eric and Manning, 2017) dedicated to building end-to-end dialogue systems, in which all their components are trained entirely from the utterances themselves without the need to assume domains or dialog state structure, so it is easy to automatically extend to new domains and free it from manually designed pipeline modules. For example, (Bordes et al., 2017) treated dialogue system learning as the problem of learning a mapping from dialogue histories to system responses.

The common point of the pipeline and end-to-end methods is that they both need to acquire knowledge from the knowledge base to produce more contentful responses. For instance, (Eric and Manning, 2017) represent each entry as several key-value tuples and attend on each key to extract useful information from a KB in an end-to-end fashion, KB-InfoBot (Dhingra et al., 2017) directly model posterior distributions over KBs according to the user input and a prior distribution, and GLMP (Wu et al., 2019) use a global to local memory network (Weston et al., 2014; Sukhbaatar et al., 2015) to encode KBs and query it in a continuous neural. However, as the KBs continue to grow in the real-world scenarios, such end-to-end methods of directly encoding and integrating whole KBs will eventually result in inefficiency and incorrect responses.

On the other hand, some works may put the user utterances through a semantic parser to obtain executable logical forms and apply this symbolic query to the KB to retrieve entries based on their attributes. A common practice for generating queries is to record the slot values that appeared in each dialogue turn. For instance, (Lei et al., 2018) design text spans named belief spans to track dialogue beliefs and record informable and requestable slots[1], then converting them into a query with human efforts. Additionally, (Bordes et al., 2017) generate API calls from predefined candidates. Use such pipeline methods can interact and cooperate with the knowledge base efficiently by issuing API calls such as SQL-like queries. However, such symbolic operations break the differentiability of the system and prevent end-to-end training of neural dialogue agents.

In particular, it is unclear if end-to-end models can completely replace and perform better than pipeline methods in a task-directed setting. In comparison, our end-to-end trainable text-to-SQL

[1]Informable slots are slots that users can use to constrain the search, while requestable slots are slots that users can ask a value for.

guided framework balances the strengths and the weaknesses of the two research methods. We first introduce the natural-language-to-SQL concept into task-oriented systems that map context dialogue histories and table schema to a SQL query and choose instead to rely on learned neural representations for implicit modeling of user intent and current state. Moreover, we provide more efficient labeling by only generating a query at an appropriate timing based on current state representations, instead of recording each slot values at each time step. By doing this, we do not need predefined slot-value pair or domain ontology, but just input dialogue histories and table schema and output synthesized SQL queries. Then we use a memory network to encode the results retrieved from KBs. Thus, we can access KBs more efficiently and achieve a high task success rate.

2.2 Semantic Parsing in SQL

Another related research is text-to-SQL, a sub-task of semantic parsing that aims at synthesizing SQL queries from natural language. The widely adopted dataset is the WikiSQL (Zhong et al., 2017). The task goal is to generate a corresponding SQL query given a natural language question and sets of table schema (Xu et al., 2018; Yu et al., 2018a; McCann et al., 2018; Hwang et al., 2019). Furthermore, cross-domain semantic parsing in text-to-SQL has been investigated (Yu et al., 2019b, 2018b, 2019a). In comparison, the SQL generator in our model is a task-oriented dialogue-to-SQL generator, which aims to help users accomplish a specific task, and dynamically determines whether to ground the dialogue context to an executable SQL.

3 The Proposed Framework

Our design of the AirConcierge system addresses the following challenges in developing an effective task-oriented dialogue system, including

- When should the system access the KBs to obtain task-relevant information during a conversation?
- How does the system formulate a query that retrieves task-relevant data from the KBs?

3.1 System Architecture of AirConcierge

AirConcierge is a task-oriented dialogue system for flight reservations and therefore depends on flight information in large external KBs to fulfill user requests. Unlike previous work that directly encodes the entire KBs, AirConcierge issues API calls to the KBs at the appropriate time to retrieve the information relevant to the task. Besides, during the dialogue with a user, AirConcierge actively prompts and guides the user for key information, and responds with informative and human-comprehensible sentences based on the retrieved results from the KBs. In particular, the "dialogue-to-SQL-to-dialogue" approach, which we implement in AirConcierge allows it to integrate with large-scale, real-world KBs.

Figure 2 shows the system architecture of AirConcierge. During a dialogue with a user, AirConcierge processes the dialogue lines in the following procedures: For each new line of a dialogue, it serves as an input to the Dialogue Encoder, which encodes the conversation history. The hidden states of Dialogue Encoder are next used by the Dialogue State Tracker to determine the phase of the dialogue (e.g., greeting phase or the problem-solving phase). If the system determines that enough information about the user's request has been collected, the SQL generator then generates a SQL query, according to the context of the dialogue so far, to retrieve information from KBs. Next, the retrieved results are encoded and stored in a Memory Network. With the encoded dialogue and the memory readout, a context-aware Dialogue Decoder generates a corresponding response. In addition to the process described above, there is a Dialogue Goal Generator which predicts the final status of the full dialogue, given the entire conversation history, to measure the agent performance.

3.2 Dialogue Encoder

We implement the Dialogue Encoder using a RNN with a gated recurrent unit (GRU) (Chung et al., 2014). Given a sequence of the conversation history $X = \{x_1, x_2, ..., x_t\}$, a word embedding matrix W_{emb} embeds each token x_t. A GRU then models the sequence of tokens by taking the embedded token $W^{emb}(x_{t-1})$ and the hidden state h^e_{t-1} from time step $t-1$ as inputs at the next time step t:

$$h^e_t = GRU(W_{emb}(x_{t-1}), h^e_{t-1}) \quad (1)$$

The whole dialogue history is encoded into the hidden states $H = (h^e_1, \ldots, h^e_T)$, where T is the total number of time steps.

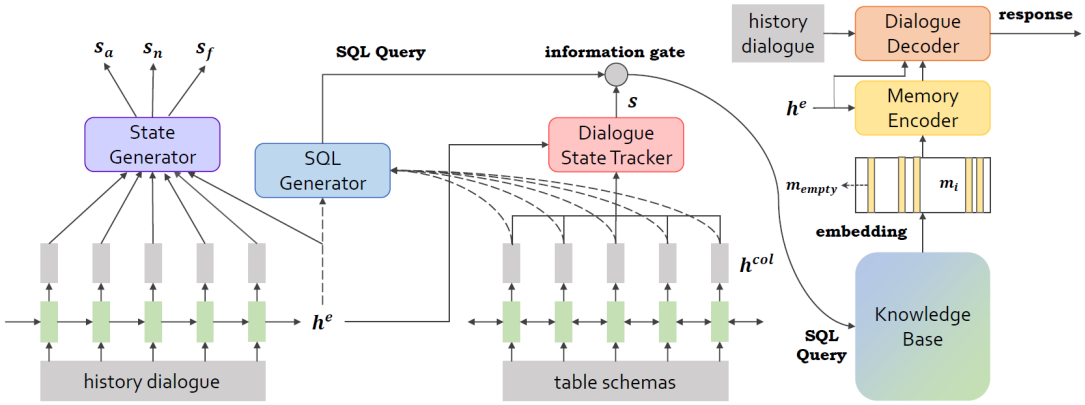

Figure 2: An overview of the system architecture of AirConcierge.

3.3 Dialogue State Tracker (Information Gate Module)

In order to determine whether a dialogue has reached a state where the system has received enough initial information about a user's need and transitioned from the "greeting state" into the "problem-solving state", we design a Dialogue State Tracker to model such a transition of states. This is a module introduced by AirConcierge to determine when to retrieve and incorporate data from the KBs into the dialogue, so we also consider it as an "information gate". The Dialogue State Tracker takes the information about the schema of KBs as an input to the model. Intuitively, by matching the information in the dialogue history with the available columns in the KBs, a better decision can be made about whether it is the right time to start querying the KBs. This module takes the last hidden state h^e_T from the Dialogue Encoder and outputs a binary value $s \in \{0, 1\}$ indicating whether the current information is sufficient to generate a query. Let $P(s)$ denote the probability that the agent would send a query:

$$P(s|h^e_T, x^{col}_{1:J}) = \sigma(W^s_2(W^s_1 h^e_T + \Sigma U_2 W_{emb}(x^{col}_{1:J}))), \quad (2)$$

where $x^{col}_{1:J}$ denotes the tokens of the J column names; W_{emb} is the word embedding matrix as in Equation (1); $U_2 \in \mathbb{R}^{d_{enc} \times d_{enc}}$ is a bidirectional LSTM; W^s_1 and W^s_2 are fully-connected layers with size $d_{enc} \times d_{enc}$; and σ is the sigmoid function. Note that we denote $U_2 W_{emb}(x^{col}_{1:J})$ as h^{col} in Figure 2.

3.4 SQL Generator

In order to enable AirConcierge to handle large-scale KBs, we devise a SQL Generator and deployed it in AirConcierge. If the state s from the Dialogue State Tracker is "problem-solving state", AirConcierge will activate the SQL Generator and generate a SQL query to access the KBs. A SQL query is in the form of `SELECT * FROM KBs WHERE $COL $OP $VALUE (AND $COL $OP $VALUE)*`, where `$COL` is a column name. Here we focus on predicting the constraints in the `WHERE` clause.

To predict the column `$COL`, we follow the sequence-to-set idea from SQLNet (Xu et al., 2018). That is, given the encoded column names $\{h^{col}_j\}_{j=1...J}$ and the last encoding of the dialogue history h^e_T, the model computes the probability $P_{col}(x^{col}_j)$ of column j to appear in the SQL query:

$$P_{col}(x^{col}_j | h^{col}_j, h^e_T) = \sigma(W^{col}_1 h^{col}_j + W^{col}_2 h^e_T) \quad (3)$$

The `$OP` slots are predicted using similar architecture:

$$P_{op}(x^{op}_j | h^{col}_j, h^e_T) = \sigma(W^{op}_1 h^{col}_j + W^{op}_2 h^e_T) \quad (4)$$

As for predicting the `$VALUE` slot for a particular `$COL`, we model it as a classification problem. Let v^j_i be the i-th value of the j-th column. The predicted probability of the value v^j_i is:

$$P_{value}(v^j_i | h^{col}_j, h^e_T) = Softmax\left(W^{val}_1 (W^{val}_2 h^e_T + W^{val}_3 h^{col}_j)\right) \quad (5)$$

where all $W^{col}_{1,2}$, $W^{op}_{1,2}$ and $W^{val}_{1,2,3}$ are trainable matrices of size $d_{enc} \times d_{enc}$.

3.5 Knowledge Base Memory Encoder

We encode the retrieved data from the KBs with a memory network mechanism. Unlike previous

work (Wei et al., 2018) which applies a hierarchical RNN to encode the entire KBs directly, we only model the retrieved results from the KBs. Thanks to the SQL Generator module that filters out most of the irrelevant data in KBs, AirConcierge is needless to encode the entire KBs and can focus on the small set of relevant data records.

Let the data records of flights retrieved from the KBs be $\{f_1, .., f_F\}$, each flight containing 12 column attributes and one additional "flight number" column attribute. These records are converted into memory vectors $\{m_1, ..., m_F\}$ using a set of trainable embedding matrices $C = \{C^1, \ldots, C^{K+1}\}$, where $C^k \in \mathbb{R}^{|V| \times d_{emb}}$ and K is the number of hops. Note that we additionally add an empty flight vector m_{empty} to represent the case where no flight in the KBs meets the customer's intent.

An initial query vector q^0 is defined to be the output of the dialogue encoder h_T^e. Then, the query vector is passed through a few "hops" where, at each hop k, a vector q^k is computed as attention weights with respect to each memory vector m_i:

$$p_i^k = \text{Softmax}((q^k)^T c_i^k) \tag{6}$$

where $c_i^k = B(C^k(f_i))$ is the embedding vector at the i^{th} memory position, and $B(\cdot)$ is a bag-of-word function. Here, p_i^k decides which ticket has higher relevance to the customer intent. Then, the memory readout o^k is summed over c^{k+1} weighted by p^k as:

$$o^k = \sum_{i=1}^{F} p_i^k c_i^{k+1} \tag{7}$$

To continue to the next hop, the query vector is updated by $q^{k+1} = q^k + o^k$.

We use the pointer $G = (g_1, \ldots, g_F)$ to pick the most relevant ticket and also filter out unimportant or unqualified tickets. K denotes the last hop.

$$\boldsymbol{g}_i^K = \text{Softmax}\left(\left(\boldsymbol{q}^K\right)^\top \boldsymbol{c}_i^K\right) \tag{8}$$

3.6 Dialogue Decoder

We adopt a GRU model as the Dialogue Decoder to generate the agent's response. At each time step, the Dialogue Decoder generates a token based on the encoded dialogue h_T^e and flight ticket information g_i^K, by calculating a probability over all tokens:

$$\begin{aligned} h_t^d &= GRU(W_{emb}(\hat{y}_{t-1}), h_{t-1}^d), \\ P(\hat{y}_t) &= \text{Softmax}(W_{dec} h_t^d) \end{aligned} \tag{9}$$

where $W_{dec} \in \mathbb{R}^{d_{enc} \times |V|}$ is a trainable matrix, and h_0 is initialized as a concatenation of q^K and h_T^e, $\hat{y}_t$ is output tokens at timestep t.

3.7 Dialogue Goal Generator

As stated in the AirDialogue (Wei et al., 2018), three final dialogue goals s_a, s_n, s_f are generated by the agent to examine the correctness at the end of conversations. s_n represents the name of the customer. The flight state s_f is the flight number selected from F flights in the KBs. The action s_a that accomplished at the end of a dialogue can be one of the following five choices: "booked", "changed", "no flight found", "no reservation" and "cancel". We feed h_T^e into three fully-connected layers, W_i^{goal}, to predict the three goals ($i \in \{\text{n,f,a}\}$), respectively:

$$P(s_i) = W_i^{goal} h_T^e. \tag{10}$$

3.8 Objective Function

In order to train the dialogue system in an end-to-end fashion, loss functions are defined for the above modules. The loss for Dialogue State Tracker, $\mathcal{L}_{gate}$, is the binary cross entropy (BCE). The loss for SQL generator consists of three parts: $\mathcal{L}_{SQL} = \mathcal{L}_{col} + \mathcal{L}_{op} + \mathcal{L}_{value}$. The loss for the `$COL` slots $\mathcal{L}_{col}$ is the BCE, and the loss for both `$OP` and `$VALUE` slots is CE. For the KB memory encoder, we use CE: $\mathcal{L}_{mem} = -\sum_{i=1}^{N}\sum_{j=1}^{F}(y_{ij} \cdot log(g_{ij}^K))$, where g_{ij}^K is the pointer, N is the number of samples, and F is the number of flights retrieved from KBs. For the state generator, CE is used for all three states, that is, $\mathcal{L}_{goal} = \mathcal{L}_{name} + \mathcal{L}_{flight} + \mathcal{L}_{action}$.

The overall loss function is formed by summing up the losses of all modules:

$$\mathcal{L} = \mathcal{L}_{gate} + \mathcal{L}_{SQL} + \mathcal{L}_{mem} + \mathcal{L}_{goal} \tag{11}$$

4 Experiments

4.1 Dataset

AirDialogue Dataset We evaluate the proposed framework on the AirDialogue dataset, a large-scale task-oriented dialogue dataset released by Google. The dataset contains 402,038 conversations, with an average length of 115. For data pre-processing, we follow the steps in the original paper (Wei et al., 2018) and their official code [2].

[2] https://github.com/google/airdialogue

Labels for State Tracker Since the original AirDialogue dataset lacks the labels for learning the Dialogue State Tracker, we devise a method to annotate each dialogue turn with a "ground-truth" state label. We define two dialogue states: At the beginning of a dialogue, while the customer expresses travel constraints and the agent asks for information, we define this as the "greeting state" of the dialogue. Once the agent receives adequate information from the user and decides to send a query, we define that the dialogue enters the "problem-solving state" and will remain in this state afterward.

We use a rule-based model to annotate. For most dialogues, the first turn of the "problem-solving state" is where the flight number is mentioned. With this observation, we label the turn where the flight number first occurs to be the starting point of the "problem-solving state". As for the dialogues that either issue multiple SQL queries or have no mention of the flight number, we apply a set of keywords to mark the problem-solving state.

Labels for SQL Generator In the original AirDialogue dataset, each dialogue is accompanied with an intention indicating the customer's travel constraints. We construct the "ground-truth query" based on the user's intention of each dialogue.

4.2 Training Details

We conduct experiments using one 2080 Ti GPU and the Pytorch (Paszke et al., 2017) environment. We use Adam (Kingma and Ba, 2015) to optimize the model parameters with a learning rate $1e^{-3}$ and a batch size of 32. The word embedding size and GRU hidden dimension are 256. The hop of the memory encoder K is set to 3. For Dialogue Decoder, a greedy strategy is used instead of beam-search. The accelerated training technique used in Wei et al. (2018) is also adopted in our model. The models are trained for 5 epochs, roughly equals to 44000 steps.

4.3 Evaluation

There are two important perspectives about the model: the quality of the dialogue and the correctness of the exact information. In order to properly evaluate these two, we use the BLEU score to evaluate the dialogues and use accuracy to evaluate the dialogue goals and SQL queries. While providing a human-like interaction with the customers is important, it is even more critical to guarantee that all

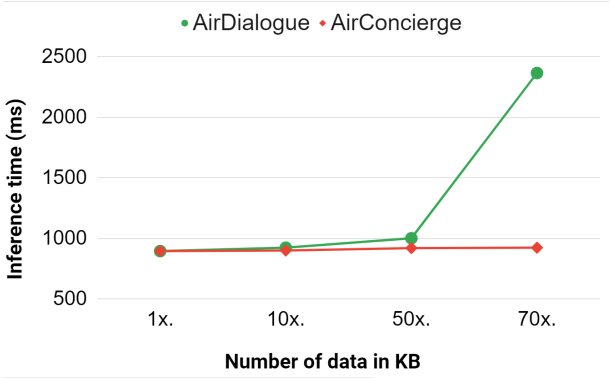

Figure 3: Inference time under different numbers of KB records on the AirDialogue dev set. "1x." denotes 30 records in the KBs, "10x." is 300 records, and so on.

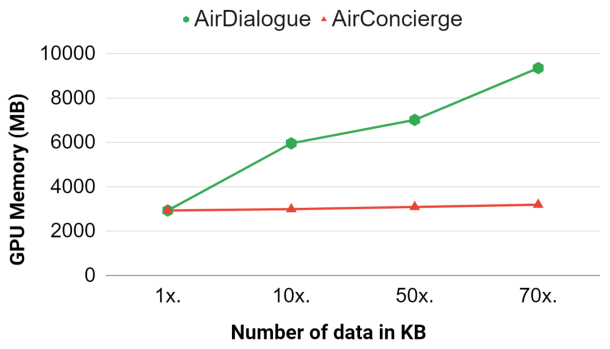

Figure 4: Memory consumption under different amounts of KB data on the AirDialogue dev set. "1x." denotes 30 records in the KBs, "10x." is 300 records, and so on.

of the provided information is correct.

For example, the agent might reply "We have found a flight number 1011 which meets your need. Should I book it?". Suppose the actual correct flight number is 1012, this sentence may have a high BLEU score while the provided information is misleading. Such an error further reveals the importance of the accuracy of Dialogue Goal Generator.

As for the correctness of the provided information, we evaluate the performance by SQL accuracy and state accuracy. The SQL accuracy is critical in filtering and accessing data from the KBs.

User simulator For self-play evaluation, we build a simulator to model a user's utterances. The simulator generates a response based on three things: a list of travel constraints, the user's intent ({"book", "change", "cancel"}), and the dialogue history. Similar to the previous work, we adopt a

Model	Name Acc.	Flight Acc.	State Acc.	BLEU
Supervised (2018) (AirDialogue dev)	0.9 %	1.2%	12%	23.26
RL (2018) (AirDialogue dev)	1%	4%	29%	19.65
AirConcierge (AirDialogue dev)	**100%**	**72.2%**	**90.0%**	**32.59**
Supervised (2018) (Synthesized dev)	0%	8%	32%	68.72
RL (2018) (Synthesized dev)	0%	35%	39%	62.71
AirConcierge (Synthesized dev)	**100%**	**58.9%**	**86.0%**	**73.51**
Human (AirDialogue test)	98%	91.4%	91.8%	-

Table 1: Dialogue performance under self-play evaluation. The agent model is the model in the first column, while the customer is the user simulator described in section 4.3. The supervised model and the Reinforcement Learning (RL) model are the baseline models reported in the original AirDialogue paper.

sequence-to-sequence model to build the simulator.

SQL evaluation We use logical-form accuracy (Acc_{lf}) and execution accuracy (Acc_{ex}) (Zhong et al., 2017) to measure the SQL quality. For Acc_{lf}, we directly compare the generated SQL query with the ground truth to check whether they match each other. For Acc_{ex}, we execute both the generated query and the ground truth and compare whether the retrieved results match each other. We also evaluate the accuracy of the 3 components (`$COL`, `$OP`, and `$VALUE`) of a WHERE condition: Acc_{col}, Acc_{op}, and Acc_{val}, respectively. For each dialogue, we evaluate only the SQL query at the turn when the "problem-solving state" first occurs.

4.4 Experimental Results: Accuracy

In Table 1, we compare the performance of AirConcierge with the baseline in the AirDialogue paper. On generating a response that matches the ground-truth dialogue line, AirConcierge achieves improvements on the BLEU score by 9.33 and 4.79 on the dev set and the synthesized set, respectively. In the self-play evaluation, AirConcierge achieves significant improvements on NameAcc, FlightAcc, and ActionAcc. We attribute the high accuracy to the correctness of SQL queries, since the data retrieved from KBs is correctly filtered and thus helps the agent make suitable and better predictions.

Besides the model's overall performance in accomplishing a user's task, we are interested in the accuracy of the SQL queries generated by AirConcierge based on the dialogue context. In this evaluation, we consider two cases: the accuracy of the 6 *essential* attributes (departure airport, return airport, departure month, return month, departure day, and return day), and the accuracy on all 12 attributes. The 6 essential attributes are the ones that are essential in identifying a ticket and therefore appear in nearly all dialogue samples.

Table 2 shows the model's accuracy in generating SQL queries. The model achieves outstanding accuracy in predicting the column-name slots, the operator slots, and the value slots. The metric Acc_{lf} evaluates whether two queries are exactly the same, so its value is typically smaller than Acc_{col}, Acc_{op}, or Acc_{val}, especially when more conditions are considered. This can be observed in the table, where the accuracy Acc_{lf} under 12 conditions is much smaller than that under only 6 essential conditions.

Furthermore, we break down the performance of overall SQL queries into each `$VALUE` slot, results presented in Table 3. AirConcierge achieves high accuracy on predicting the values of the 6 essential conditions, but performs not as good on the other 6 conditions (departure time, return time, class, price, connections, and airline). This may be due to that the essential 6 conditions are provided in nearly all dialogues, while the other conditions are only provided from time to time. Having fewer data about the other conditions makes it harder for the model to learn about them.

4.5 Experimental Results: Scalability

An important contribution of AirConcierge is the efficiency in cooperating with KBs. By employing the SQL Generator, AirConcierge increases the model's ability to handle large-scale KBs. In Figure 3, we show the model's inference time with respect to the number of data records in the KBs. The "1x." at the x-axis corresponds to having 30 data records in the KBs, and "10x." corresponds to 300 entries in the KBs, and so on. As shown in the

Experiment	Acc_{col}	Acc_{op}	Acc_{val}	Acc_{lf}	Acc_{ex}
AirConcierge†	98.96%	99.7%	97.9%	95.54%	96.44%
AirConcierge‡	97.24%	98.6%	61.4%	28.11%	86.28%

Table 2: Performance on the AirDialogue dataset. † indicates considering only 6 conditions, such as departure city, return city, departure month, return month, departure day, and return day. ‡ means considering all 12 conditions. The models of † and ‡ are the same. We report the average accuracy.

Experiment	dep. city	ret. city	dep. month	ret. month	dep. day	ret. day
AirConcierge	98.89%	97.93%	97.52%	97.49%	97.27%	97.29%
Experiment	dep. time	ret. time	class	price	connections	airline
AirConcierge	49.60%	52.46 %	42.74%	37.60%	95.36%	42.12%

Table 3: Performance of each `$VALUE` slot to be generated in the query.

figure, the inference time of AirConcierge remains short as the KBs grows larger. On the contrary, the baseline model, AirDialogue , requires obviously more inference time: when the KBs are 70 times larger, AirDialogue takes 5 times longer to complete the dialogue. We also compare the memory consumption of AirConcierge with that of AirDialogue. In Figure 4, it is shown that AirConcierge consumes a constant amount of memory regardless of the KBs size, while AirDialogue requires more memory as the KBs size grows. This indicates that AirConcierge is scalable from the aspect of memory consumption as well.

We inflate the size of KBs by augmenting additional data records. To generate a variant data record, we choose an existing ground-truth record and modify the values of some of its columns. The modified column value is sampled from a prior distribution defined for that column. We experiment with different numbers of columns to modify. For an augmentation where the *last i columns* subject to variations, we denote such an augmentation as "#Augment-column-i".

Intuitively, the more columns are subject to variations, the more diverse the records are. Therefore, fewer records will match the query when more columns are subject to variations. This is shown in Figure 5. When more records are added in the KBs, for an augmentation that has more variant columns (e.g., #Augment-column-10), the growth of the number of records returned for a SQL query is slower than the growth experienced by augmentation with fewer variation columns (e.g., #Augment-column-6). This also illustrates the importance of having a high-quality SQL Generator. Since generating precise SQL queries can effectively cut down the data records to be considered.

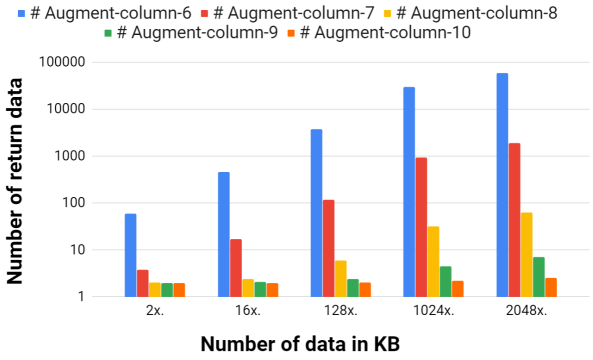

Figure 5: Number of returned data from different augment types of KBs using SQL queries generated by our model.

5 Conclusions

We propose AirConcierge, a task-oriented dialogue system that has high accuracy in achieving the user's tasks. By employing a subsystem, including a Dialogue State Tracker and a SQL Generator, AirConcierge can issue a precise SQL query at the right time during a dialogue and retrieve relevant data from KBs. As a result, AirConcierge can handle large-scale KBs efficiently, in terms of shorter processing time and less memory consumption. Using a precise SQL query also filters out noise and irrelevant data from the KBs, which improves the quality of the dialogue responses. Our experiments demonstrate the better performance and efficiency of AirConcierge, over the previous work.

References

Antoine Bordes, Y-Lan Boureau, and Weston Jason. 2017. Learning end-to-end goal-oriented dialog. In *ICLR*.

Junyoung Chung, Çaglar Gülçehre, Kyunghyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *ArXiv*, abs/1412.3555.

Anoop Deoras and Ruhi Sarikaya. 2013. Deep belief network based semantic taggers for spoken language understanding. In *INTERSPEECH*.

Bhuwan Dhingra, Lihong Li, Xiujun Li, Jianfeng Gao, Yun-Nung Chen, Faisal Ahmed, and Li Deng. 2017. Towards end-to-end reinforcement learning of dialogue agents for information access. In *ACL*.

Jesse Dodge, Andreea Gane, Xiang Zhang, Antoine Bordes, Sumit Chopra, Alexander H. Miller, Arthur Szlam, and Jason Weston. 2016. Evaluating prerequisite qualities for learning end-to-end dialog systems. *CoRR*, abs/1511.06931.

Mihail Eric and Christopher D. Manning. 2017. Key-value retrieval networks for task-oriented dialogue. In *SIGDIAL*.

Wonseok Hwang, Jinyeung Yim, Seunghyun Park, and Minjoon Seo. 2019. A comprehensive exploration on wikisql with table-aware word contextualization. *arXiv preprint arXiv:1902.01069*.

Kyungduk Kim, Cheongjae Lee, Sangkeun Jung, and Gary Geunbae Lee. 2008. A frame-based probabilistic framework for spoken dialog management using dialog examples. In *SIGDIAL Workshop*.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Wenqiang Lei, Xisen Jin, Zhaochun Ren, Xiangnan He, Min-Yen Kan, and Dawei Yin. 2018. Sequicity: Simplifying task-oriented dialogue systems with single sequence-to-sequence architectures. In *ACL*.

Xiujun Li, Yun-Nung Chen, Lihong Li, Jianfeng Gao, and Asli Çelikyilmaz. 2017. End-to-end task-completion neural dialogue systems. *ArXiv*, abs/1703.01008.

Bing Liu and Ian Lane. 2017. An end-to-end trainable neural network model with belief tracking for task-oriented dialog. *ArXiv*, abs/1708.05956.

Andrea Madotto, Chien-Sheng Wu, and Pascale Fung. 2018. Mem2seq: Effectively incorporating knowledge bases into end-to-end task-oriented dialog systems. *ArXiv*, abs/1804.08217.

Christopher D. Manning and Mihail Eric. 2017. A copy-augmented sequence-to-sequence architecture gives good performance on task-oriented dialogue. In *EACL*.

Bryan McCann, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. 2018. The natural language decathlon: Multitask learning as question answering. *arXiv preprint arXiv:1806.08730*.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary Devito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch. In *NIPS-W*.

Alexander I. Rudnicky, Eric H. Thayer, Paul C. Constantinides, Chris Tchou, R. Shern, Kevin A. Lenzo, Weiyang Xu, and Alice H. Oh. 1999. Creating natural dialogs in the carnegie mellon communicator system. In *EUROSPEECH*.

Iulian Serban, Alessandro Sordoni, Yoshua Bengio, Aaron C. Courville, and Joelle Pineau. 2016. Building end-to-end dialogue systems using generative hierarchical neural network models. In *AAAI*.

Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. 2015. End-to-end memory networks. In *NIPS*.

Wei Wei, Quoc V. Le, Andrew M. Dai, and Jia Li. 2018. Airdialogue: An environment for goal-oriented dialogue research. In *EMNLP*.

Tsung-Hsien Wen, David Vandyke Lina Maria Rojas-Barahona, Milica Gasic, Nikola Mrksic, Pei hao Su, Stefan Ultes, and Steve J. Young. 2016. A network-based end-to-end trainable task-oriented dialogue system. In *EACL*.

Jason Weston, Sumit Chorpa, and Antoine Bordes. 2014. Memory networks. *arXiv:1410.3916*.

Chien-Sheng Wu, Richard Socher, and Caiming Xiong. 2019. Global-to-local memory pointer networks for task-oriented dialogue. *ArXiv*, abs/1901.04713.

Xiaojun Xu, Chang Liu, and Dawn Song. 2018. Sqlnet: Generating structured queries from natural language without reinforcement learning. In *ICLR*.

Xuesong Yang, Yun-Nung Chen, Dilek Z. Hakkani-Tür, Paul Crook, Xiujun Li, Jianfeng Gao, and Li Deng. 2017. End-to-end joint learning of natural language understanding and dialogue manager. *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5690–5694.

Steve J. Young, Milica Gasic, Blaise Thomson, and Jason D. Williams. 2013. Pomdp-based statistical spoken dialog systems: A review. *Proceedings of the IEEE*, 101:1160–1179.

Tao Yu, Zifan Li, Zilin Zhang, Rui Zhang, and Dragomir Radev. 2018a. Typesql: Knowledge-based type-aware neural text-to-sql generation. In *NAACL*.

Tao Yu, Rui Zhang, He Yang Er, Suyi Li, Eric Xue, Bo Pang, Xi Victoria Lin, Yi Chern Tan, Tianze Shi, Zihan Li, Youxuan Jiang, Michihiro Yasunaga, Sungrok Shim, Tao Chen, Alexander R. Fabbri, Zifan Li, Luyao Chen, Yuwen Zhang, Shreya Dixit, Vincent Zhang, Caiming Xiong, Richard Socher, Walter S. Lasecki, and Dragomir R. Radev. 2019a. Cosql: A conversational text-to-sql challenge towards cross-domain natural language interfaces to databases. In *EMNLP/IJCNLP*.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. Radev. 2018b. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *EMNLP*.

Tao Yu, Rui Zhang, Michihiro Yasunaga, Yi Chern Tan, Xi Victoria Lin, Suyi Li, Heyang Er, Irene Li, Bo Pang, Tao Chen, Emily Ji, Shreya Dixit, David N Proctor, Sungrok Shim, Jonathan Kraft, Vincent Zhang, Caiming Xiong, Richard Socher, and Dragomir R. Radev. 2019b. Sparc: Cross-domain semantic parsing in context. In *ACL*.

Tiancheng Zhao and Maxine Eskénazi. 2016. Towards end-to-end learning for dialog state tracking and management using deep reinforcement learning. In *SIGDIAL Conference*.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *ArXiv*, abs/1709.00103.

Victor Zue. 2000. Conversational interfaces: advances and challenges. *Proceedings of the IEEE*, 88:1166–1180.

Victor Zue, Stephanie Seneff, James R. Glass, Joseph Polifroni, Christine Pao, Timothy J. Hazen, and I. Lee Hetherington. 2000. Juplter: a telephone-based conversational interface for weather information. *IEEE Trans. Speech Audio Process.*, 8:85–96.

A Appendices

A.1 Data Statistics

For the data records in the KBs, each of them is generated using the prior distributions defined in Table 4. In section 4.5, we conduct experiments under different scales of the KBs, where the newly augmented records are generated according to these prior distributions. The original AirDialogue dataset contains 30 records in the KBs, and we augment the KBs to "10x.", "50x.", and "70x.". That is, we additionally add 270 records, sampled according to the prior distributions, into the "10x." KBs. Similar things are done to the "50x." KBs and "70x." KBs.

A.2 Qualitative Analysis

We provide samples of dialogues generated by our agent and the user simulator under the self-play evaluation. The user simulator has a pre-defined intent that belongs to one of the three: "book", "change", "'cancel', as well as a list of travel constraints. On the other hand, responses provided by the agent may result in one of the five actions: booked", "changed", "cancelled", "no flight found", "no reservation". The user intent "book" could lead to the agent action "booked" or "no flight found", while both "change" and "cancel" may lead to "no reservation". However, the user intent "change" could be successfully achieved, and result in the agent action "changed". Similarly, "cancel" could lead to "cancelled".

We show several samples according to the agent's action. First, Table 5 shows the two samples of the agent action "booked". We see that the user tends to provide the destination and return airport codes spontaneously, followed by the agent requiring the travel dates. After the ticket is found, the agent informs the user about the flight details, which is a human-like behaviour. Finally, the ticket is confirmed by the user, and both the user and agent ends the dialogue through the thankfulness.

Table 6 shows the samples for the action "changed". At the beginning, the user and the agent greets with each other. Then, the user not only expresses the intent to change the flight, but also gives a reason for changing. We see that the agent learns to judge whether the user has provided his/her name. In the first, or say upper, sample, the user mentioned his/her name right after greeting, and hence the agent go through to check the KBs. However, in the second, or say lower, sample, the agent identified that the user hasn't told his/her name yet, so the agent requires the name before querying the KBs.

For the action "cancelled", samples are provided in Table 7. We observe similar patterns to the action "changed". The user first describes the need to cancel the ticket, and followed by the agent asking the name if necessary. Lastly, the agent found the ticket and confirm the cancellation with the user.

Table 8 provides the samples of the action "no flight found". Similar to the samples of "booked", the user describes the travel constraints and ask to book a ticket. The difference is that the agent could not find a matched flight, and thus responds with no flight available. One thing special is that

feature	dep./ret.city	dep./ret. month	dep./ret. day	dep./ret. time
range	categorical	1-12	1-31	00-23
prob.	uniform	uniform	uniform	uniform
feature	class	price	connections	airline
range	business,economy	0-5000	0,1,2	categorical
prob.	economy (7%) business (3%) any (90%)	≤200 (25%) ≤500 (25%) ≤1000 (25%) any (25%)	0 (7%) 1 (90%) any (3%)	standard fare (5%) UA, Delta AA, Hawaiian any (95%)

Table 4: Flight features of the AirDialogue dataset.

Samples of dialogues with state "booked".	
Usr	Hello .
Agt	Hello . How can I help you ?
Usr	I am Dennis Carter . I am planning to visit Detroit , can you please book me a ticket from AUS to DTW ?
Agt	Sure , I will help you to book a flight ticket . Can you please share me your travelling dates ?
Usr	My preferable journey dates are from 03/19 to 03/21 .
Agt	Ok , give me a moment .
Usr	Sure , take your own time .
Agt	Here , I found a flight for you with single connection . flight number is <fl_1020> .
Usr	Please book my ticket with this airlines .
Agt	Your ticket is booked .
Usr	Thank you for booking my ticket .
Agt	Welcome .
Usr	Hello .
Agt	Hello , how may I help you ?
Usr	I am Andrew Young . I want to book a flight ticket from DCA to MSP . Could you help me in booking ?
Agt	Sure , may I know your trip dates please ?
Usr	My travelling dates are from May 10 - 12 .
Agt	Do you have any other requirements ?
Usr	I need a connecting flight .
Agt	Ok , please wait .Ok , please wait .
Usr	Sure .
Agt	Thank you for waiting and we have found a flight in economy class with 1 connection in Spirit airlines and the fare of your flight will be 200 . Do you like me to proceed your booking with this flight ? flight number is <fl_1026> .
Usr	Please proceed .
Agt	Ok , your ticket is booked .
Usr	Thank you .
Agt	Most welcome .

Table 5: Samples of dialogues with state "booked".

Samples of dialogue with state “changed”.	
Usr	Hello .
Agt	Hello . How can I help you ?
Usr	I am Jeffrey Miller . I want to change my existing reservation due to postpone of my exams . Can you please help me with this ?
Agt	Sure , please wait for a while .
Usr	sure .
Agt	I found your reservation , can you please share your travelling dates ?
Usr	My travelling dates are July 28 and July 30 .
Agt	Can I know your airport codes ?
Usr	My airport codes are PHL and MSP .
Agt	Do you have any other specifications ?
Usr	No .
Agt	Ok , please wait for a while .
Usr	Sure .
Agt	There is a flight with single connection in UA airlines and the ticket fare is 200 only . Can I book the ticket for you ? flight number is <fl_1025> .
Usr	You can proceed .
Agt	Your ticket is confirmed in economy class .
Usr	Thank you .
Agt	Thank you for choosing us .
Usr	Hello .
Agt	Hello . How can I help you ?
Usr	I want to change my existing reservation due to postpone of my exams . Can you please help me with this ?
Agt	Sure , I will help you to change your reservation .
Usr	Thank you .
Agt	May I know your name please ?
Usr	My name is David Martinez .
Agt	Ok , please wait a moment .
Usr	Sure .
Agt	Thank you for waiting , I have found your reservation and cancelled it successfully .
Usr	Thank you for your help .
Agt	Thank you for reaching us .

Table 6: Samples of dialogue with state “changed”.

Samples of dialogue with state “cancelled”.	
Usr	Hello .
Agt	Hello . How can I help you ?
Usr	I am Amy White . I want to cancel my recent reservation due to sudden cancellation of my trip . Can you help me ?
Agt	Sure , I will help you to cancel your reservation .
Usr	Thank you .
Agt	Happy to inform you that we found a reservation on your name . Shall I cancel it ?
Usr	Yes , please cancel it .
Agt	Your ticket has been cancelled .
Usr	Thank you for your help .
Agt	Welcome .
Usr	Hello .
Agt	Hello . How can I help you ?
Usr	I am Jacob Clark . I want to cancel my recent reservation due to sudden cancellation of my trip . Can you help me ?
Agt	Sure , I will help you to cancel your reservation .
Usr	Thank you .
Agt	Please give me a moment .
Usr	Sure .
Agt	I found your reservation and I have successfully cancelled your reservation .
Usr	Thank you for your help .
Agt	Welcome .

Table 7: Samples of dialogue with state “cancelled”.

Samples of dialogue with state “no flight found”.	
Usr	Hello .
Agt	Hello , how may I help you ?
Usr	I am Paul Phillips . I am planning to attend music concert at CLT , can you please help me with booking a flight ticket from BOS ?
Agt	Sure , may I know your trip dates please ?
Usr	Sure , I would like to take off on Mar 10 and return on Mar 12 .
Agt	Do you have any other requirements ?
Usr	Yes , I need a connecting flight .
Agt	Please wait .
Usr	Sure , take your time .
Agt	Sorry , there are no flights available on your preferred dates .
Usr	Ok , no issues . Thank you for checking .
Agt	Thank you for choosing us .
Usr	Hello .
Agt	Hello , how may I help you ?
Usr	I am Robert Lee . I am planning to visit Atlanta , will you help me in booking the flight ticket from PHL to ATL ?
Agt	Sure , can I know your travelling dates ?
Usr	My travelling dates are 10/03 and 10/05 .
Agt	Sorry , there is no flight available on your route .
Usr	That ’s ok , thank you for checking .
Agt	Most welcome .

Table 8: Samples of dialogue with state “no flight found”.

Samples of dialogue with state “no reservation”.	
Usr	Hello .
Agt	Hello . How can I help you ?
Usr	I am Steven Allen . I want to change my existing reservation due to some health issues , can you please help me with that ?
Agt	Sure , I will help you to change your reservation .
Usr	Thank you .
Agt	Please give me a moment .
Usr	Sure .
Agt	Sorry , there is no reservation found on your name .
Usr	Ok , no problem . Thank you for your information .
Agt	Welcome .
Usr	Hello .
Agt	Hello , how may I help you ?
Usr	I am Karen Gonzalez . I want to cancel my recent reservation due to sudden cancellation of my trip . Can you help me ?
Agt	Sure , please wait for a moment .
Usr	Ok .
Agt	Sorry , there is no reservation found on your name .
Usr	No problem , thank you for the information .
Agt	Thank you for reaching us .

Table 9: Samples of dialogue with state “no reservation”.

the agent responds no matching flight along with a reason. For instance, the agent in the upper sample mentions that no matching flights found is due to the mismatching dates.

For "no reservation", Table 9 shows the corresponding samples, where the upper sample is with the user intent "change" and the lower sample is with the intent "cancel". We see similar patterns to samples of "changed" and "cancelled". At the beginning, the user says the intent of changing, or cancelling, the ticket with some reason. The agent asks for the name if needed, and confirm the action of changing, or cancel, with the user.