Predictable and Adaptive Goal-oriented Dialog Policy Generation

Nhat Le
University of California, Riverside
nhat@ucr.edu

A. B. Siddique
University of California, Riverside
msidd005@ucr.edu

Fuad Jamour
University of California, Riverside
fuadj@ucr.edu

Samet Oymak
University of California, Riverside
oymak@ece.ucr.edu

Vagelis Hristidis
University of California, Riverside
vagelis@cs.ucr.edu

***Abstract*—Most existing commercial goal-oriented chatbots are diagram-based; i.e., they follow a rigid dialog flow to fill the slot values needed to achieve a user's goal. Diagram-based chatbots are predictable, thus their adoption in commercial settings; however, their lack of flexibility may cause many users to leave the conversation before achieving their goal. On the other hand, state-of-the-art research chatbots use Reinforcement Learning (RL) to generate flexible dialog policies. However, such chatbots can be unpredictable, may violate the intended business constraints, and require large training datasets to produce a mature policy. We propose a framework that achieves a middle ground between the diagram-based and RL-based chatbots: we constrain the space of possible chatbot responses using a novel structure, the *chatbot dependency graph*, and use RL to dynamically select the best valid responses. Dependency graphs are directed graphs that conveniently express a chatbot's logic by defining the dependencies among slots: all valid dialog flows are encapsulated in one dependency graph. Our experiments in several domains show that our framework quickly adapts to user characteristics and achieves up to 23.77% improved success rate compared to a state-of-the-art RL model.**

I. Introduction

Dialog systems (a.k.a. chatbots) offer an intuitive way for humans to interact with machines. Building intelligent dialog systems has been one of the main goals of AI research since the inception of the field [1], [2]. Dialog systems can be categorized into two main groups: chit-chat and goal-oriented. Chit-chat (or social) bots such as Microsoft's Xiaoice [3] are designed to mimic open-ended human-to-human conversations for entertainment purposes, rather than to complete a particular task. Goal-oriented chatbots, which are the focus of this paper, facilitate automating repetitive tasks such as booking a flight ticket or ordering a pizza. The ultimate objective of a goal-oriented chatbot is to keep a user engaged until her task is achieved; a conversation is *successful* if the user reaches the end of the conversation, which typically results in concluding a business transaction.

A typical goal-oriented chatbot system has a modular architecture consisting of several components [4]: a natural language understanding (NLU) module [5], a dialog state tracker (DST) [6], a dialog policy (DP) unit [7], and a natural language generation (NLG) module [8]. NLU, DST, and NLG modules handle the bi-directional conversion between bot/user utterances and structured semantic representations. The dialog policy module is a key component that generates a chatbot's response at the semantic representation level, given a user utterance and a conversation context. The focus of this paper is *a new framework for building flexible dialog policies for goal-oriented chatbots*.

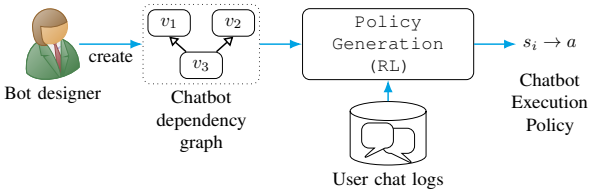


Fig. 1: Overview of our framework: A domain expert designs a chatbot dependency graph, and our framework uses this graph to restrict the actions of an RL model in the policy generation module. We train and evaluate adaptive RL models using simulated chat logs generated using our novel user simulator that considers user-specific characteristics, allowing the policy generation module to capture latent user profiles.

Despite the impressive advances in deep learning, commercial goal-oriented chatbots mainly rely on diagram-based dialog policies, where the conversation flow between the chatbot and the user is modeled as a simple finite state machine. The key reasons for the diagram-based model's popularity in state-of-the-art commercial chatbot platforms like Google's DialogFlow [9] and Amazon Lex [10] are their interpretability and predictability; i.e., the bot designer knows exactly what decision the bot will take. However, their rigid diagram-driven dialog policy does not take into account that different conversation flows may be better for different users. This lack of flexibility may cause users to leave the conversation before achieving their goal, resulting in lost revenue in commercially deployed chatbots. State-of-the-art research on goal-oriented chatbots, on the other hand, uses Reinforcement Learning (RL) to build flexible dialog policies [11], [7], [12]. However, such policies are difficult to interpret and tune, and they may violate the intended business logic, unless a large amount of high quality training data is available.

We propose a framework for building predictable and flexible dialog policies by coupling *chatbot dependency graphs* with RL – Figure 1 shows an overview of our framework. We propose chatbot dependency graphs to mitigate the rigidity of traditional chatbot diagrams, and provide chatbot designers with an intuitive way of integrating their domain-specific constraints into a flexible chatbot. A chatbot dependency graph is a directed graph, where nodes are slots or groups of slots, and edges are dependencies among slots; i.e., the strict order with which slots should be filled. Chatbot dependency graphs (*i*) are easier to design than traditional diagrams as the bot designer needs to only describe dependencies among slots rather than specifying exact dialog flows; (*ii*) encapsulate all valid conversation flows, which allows for dynamically optimizing dialog flows based on user characteristics; and (*iii*) provide an elegant way to restrict the action space of an RL model, resulting in faster convergence (i.e., less reliance on training data).

Our framework limits the action space of an RL model by assigning negative rewards to actions that violate the constraints imposed in the respective dependency graph. Coupling dependency graphs with RL achieves two desired goals: First, it enables an RL model to dynamically choose the best valid conversation flow based on the user behaviour and the conversation context. Second, it reduces the amount of training data required for the RL model to adapt to user characteristics and converge to a mature dialog policy. This is possible because the domain-specific constraints imposed by the dependency graph limit the action space of the RL model, and implicitly teach it what actions are illegal; hence, the RL model needs little to no training data with illegal actions.

We pre-train our framework using agenda-based user simulation [13], where each conversation simulates a user with a specific goal and an agenda of user actions. We train and evaluate our framework using our novel adaptive user simulator which, compared to existing simulators [13], is more realistic as it allows users to drop out of a conversation under certain circumstances based on their characteristics. In our simulator, a user is characterized with factors inspired by studies on online customer behaviour [14]. Each user profile determines the degree to which a user cares about a certain factor; e.g., a highly privacy-conscious user is likely to drop out of conversations when asked about her phone number without explaining why and how her number will be used. In summary:
(*i*) We propose chatbot dependency graphs to facilitate and simplify the design of flexible chatbot dialog flows (Section II).
(*ii*) We integrate dependency graphs with RL, which leads to faster convergence compared to existing RL policy generation models (Section III).
(*iii*) We train and evaluate our framework using our adaptive user simulator (Section IV); our extensive simulations show that our framework quickly adapts to user characteristics (Section V).

II. Chatbot Dependency Graphs

Diagram-based goal-oriented chatbots (or rule-based chatbots) acquire slot values from a user with the help of a diagram

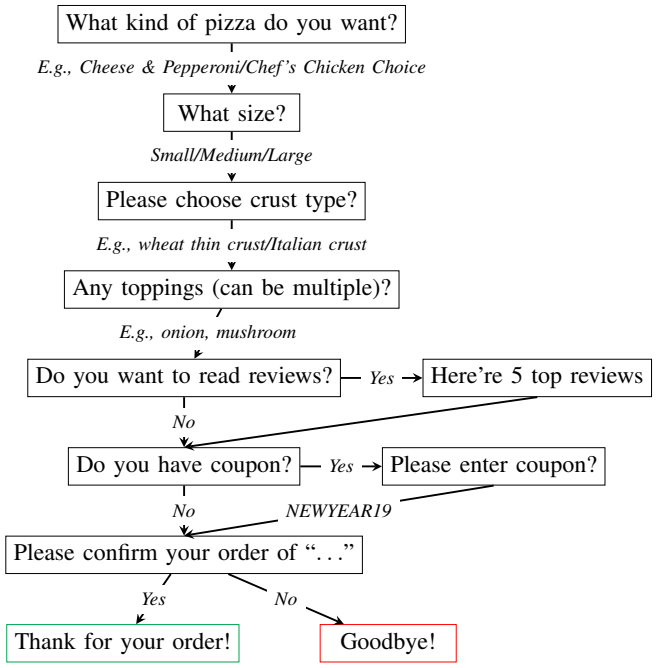


Fig. 2: Example chatbot diagram: pizza ordering.

(or a set of rules). The diagram guides a conversation by specifying which slots need to be filled and in what order. Consider the example of pizza ordering chatbot diagram in Figure 2. In this example, the diagram instructs the chatbot to ask for the slot values required to confirm a pizza order: pizza type, size, crust type, and toppings. Also, the diagram specifies that the bot has to offer reviews to the user only if he responds with *yes* to the question: "Do you want to read reviews?". The choice of presenting valid orderings or presenting/eliminating certain slots can be challenging for bot designers, because such choices should be driven by user characteristics, and can only be decided dynamically when the chatbot is interacting with the user. Our chatbot dependency graphs eliminate this burden from the bot designers, and assigns the choice to an RL model.

A dependency graph $G(V, E)$ is comprised of a set of *supernodes* V and a set of edges $E \subset V \times V$. A supernode $v \in V$ groups semantically related slots, similarly to frames in the frame-based paradigm [15]. The slots in a supernode should all be filled before filling slots from another supernode. A supernode may impose restrictions on the order of filling its slots by means of a traditional diagram (within a supernode), and supernodes can be optional, which we introduce to enable selectively providing and soliciting slot values that are not essential to the success of a conversation, but rather supplementary. Consider, for example, a price-conscious user trying to order a pizza. Offering the chance to enter a coupon code may encourage the user to finish her conversation. At the same time, presenting such coupon option may throw off an impatient user who does not care about discounts. Supernodes may contain information blocks, which are text messages or images that are used to facilitate providing the user with extra information, if necessary, such as product reviews. An edge

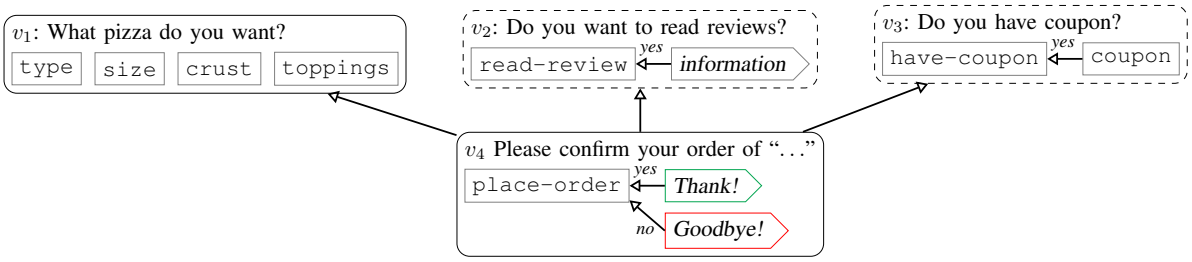


Fig. 3: Dependency graph for pizza ordering. A directed edge from v_j to v_i means that v_j has a dependency on v_i; i.e., v_j should only be presented to the user after v_i. Dotted borders denote optional supernodes.

$(v_i, v_j) \in E$ implies that supernode v_i depends on supernode v_j; that is, the slots in v_i should be presented/solicited before those in v_j. Figure 3 shows an example chatbot dependency graph. This dependency graph not only captures the dialog flow in Figure 2, but it also encapsulates all other valid dialog flows based on the specified business constraints presented in the form of dependencies among supernodes.

III. Dialog Policy Generation

In the state-space of our RL model, a state is a vector that captures the conversation history. A state contains the following information: (*i*) the current value for each slot, e.g., *pickup-location*="2894 Massachusetts Avenue"; (*ii*) a boolean variable for each slot specifying whether the slot has been filled or shown to the user; and (*iii*) three boolean variables for each supernode specifying whether it is optional, activated, or presented.

Our RL model's action space consists of all possible actions the chatbot can take. A bot action <*act*, *slot*> is a pair of *dialog act* and a slot, where a dialog act is a semantic description of the chatbot's intention at a dialog turn. For example, when a pizza ordering chatbot asks for the crust type, the action is the pair <*request*, *crust-type*>. We consider the following set of dialog acts: *request*, *inform*, *show*. Our framework can also be extended to support other domain-specific dialog acts, however, the dialog acts we currently use are essential to all domains.

The reward function in our RL model assigns a positive reward to the conversation success state (e.g., pizza order confirmation), and a negative reward otherwise. Dependency graphs restrict the possible actions at certain states by causing the reward function to return a negative reward when the RL agent takes an action that violates the dependency graph constraints. This way, the RL model learns to avoid violating actions. We describe next how violating actions are identified according to the constraints of a dependency graph.

First, actions involving slots from one supernode should be invoked together. For example, consider the case when a pizza ordering chatbot is at state s where the slots `type` and `size` have been filled, and the slots `crust`, `toppings`, and `read-reviews` have not been filled (supernode v_1 in Figure 3). In such a case, all the slots in supernode v_1 should be filled before moving to any other supernode. The dependency graph will cause the reward function to return a negative reward if the RL agent generates a response involving a slot from a supernode other than the active one (v_1 in our example). Second, the dependencies across supernodes restrict the next slots to present. If supernode v_i depends on supernode v_j, the slots in v_j should be presented before making any action involving slots from v_i. Thus, actions violating this constraint will result in a negative reward. For instance, when the chatbot is at the state s described above, the bot has not presented all the slots of supernode v_1, thus the RL agent would receive a negative reward if it takes action `<request, place-order>` or `<show, goodbye>` of supernode v_4. Finally, inside each supernode, actions involving a slot are valid only if the slot's dependencies have been satisfied (as in chatbot diagrams).

IV. User Simulation

Ideally, an RL agent learns from real-world feedback. In the context of goal-oriented chatbots, this means that the RL agent would have to perform a possibly large number of dialogues with real users before its dialogue policy becomes mature. Although crowd-sourcing services, such as Amazon MTurk, seemingly offer the possibility of training a chatbot by talking to MTurk workers, the time and financial cost can be prohibitive. Consequently, we rely on agenda-based user simulation [13], similarly to the state-of-the-art RL chatbot engines [16], [7], [17]. We use an agenda-based user simulator to pre-train our model, which teaches our RL agent a basic dialog policy that does not necessarily take user characteristics into account. We train and evaluate our RL agent using a more realistic user simulator which takes into account that users may drop out of conversation. Our adaptive simulation mechanism enables studying the adaptivity aspect of a chatbot.

In agenda-based user simulation, a conversation is described by a user goal and a user agenda. A user goal consists of inform act and slot-value pairs that serve as user constraints, and request slots whose values the user wishes to know. For example, a pizza ordering user goal has `type=pepperoni`, `size=medium`, and `crust=thin` as constraints, and slots `toppings` and `wait-time` as requests. A user agenda is a stack of user actions built and updated from the user goal and the bot utterances to determine the next simulated user utterance.

A user simulator should resemble the real user behaviour: Users may get irritated by the chatbot utterances and choose to drop out, i.e., leave the conversation; they may provide irrelevant or uninterpretable utterances; or, ideally, they may

provide the requested information, e.g., slot values. Existing agenda-based user simulators fail to account for the non-ideal scenarios and assume that users continue to converse with a chatbot until their goals are achieved. We extend the agenda-based user simulation paradigm by modeling the non-ideal scenarios. The probability of a certain user responding according to the three scenarios is a function of her characteristics; estimating these probabilities is non-trivial due to their reliance on user characteristics. We denote the probability of user dropout, providing uninterpretable utterances, and providing relevant utterances by p_d, p_e, and p_s, respectively.

Our simulator determines the probability of each user response using a *user profile*, which is a set of factors that describe a user. Inspired by the research in online customer behavior [14] and the work in [16], we choose the following factors to define a user profile: $security$, $privacy$, $uncertainty$, $guarantee$, $price$, and $promotion$. Each factor is a numeric value between 0 and 1. $Security$ and $privacy$ quantify the level to which a user is concerned with her transaction security and data privacy, respectively. For users with high values, the chance of dropping out if the chatbot does not explicitly provide security and privacy guarantees is high. $Uncertainty$ quantifies a user's need for additional information, such as customer reviews, to clear her confusion. $Guarantee$, $price$, and $promotion$ quantify a user's interest in product guarantees, price, and promotional coupons, respectively. A user u_j is described with a vector of her parameter values:

$$\mathbf{u}_j = [security(u_j), privacy(u_j), uncertainty(u_j), guarantee(u_j), price(u_j), promotion(u_j)]^\intercal.$$

We explain next the process of computing p_d, p_e, and p_s. These probabilities are functions of the interaction between a supernode in a dependency graph (which results in a chatbot utterance or a group of utterances) and a specific user profile. We define $p_d^k(u_j)$ as the drop out rate of user u_j after k turns. The user u_j is first initiated based on a global user drop out rate and her profile is as follows:

$$p_d^0(u_j) = global\_dropout + \tfrac{1}{2} \times \textstyle\sum_{f=1}^{6} \mathbf{u}_j[f]. \quad (1)$$

The dropout increase penalizes the event that the chatbot fails to address a user concern specified by the user profile. Let

$$\mathbf{l}_i = [security(l_i), privacy(l_i), uncertainty(l_i), guarantee(l_i), price(l_i), promotion(l_i)]^\intercal$$

be a vector of *relevance parameters* associated with a slot l_i. Each parameter is a numeric value $\in [-1, 1]$ that indicates the relevance of utterances generated by node l_i to a certain user parameter; negative values indicate an adversary effect, and positive values indicate an engaging effect. Let the *effect* of slot l_i to user u_j be the inner product of $\mathbf{l}_i$ and $\mathbf{u}_j$: $\textit{effect}(l_i, u_j) = \mathbf{l}_i^\intercal \cdot \mathbf{u}_j$.

The user dropout rate after having received k chatbot utterances is updated to reflect slot l_i's effect as follows:

$$p_d^k(u_j) = max(0, min(1, p_d^{k-1}(u_j) - \delta \times \textit{effect}(l_i, u_j))) \quad (2)$$

where δ is a constant factor specifying the maximum dropout change per conversation turn. We further penalize chatbot dependency graph violations, which typically confuse users, using parameter β:

$$p_d^k(u_j) = max(0, min(1, p_d^{k-1}(u_j) + \beta)) \quad (3)$$

For simplicity, we assume that the probability of a user generating an uninterpretable utterance p_e is a constant ϵ bounded by the complement of the drop out probability: $p_e = min(\epsilon, 1 - p_d)$. Finally, the probability of a user generating a valid utterance is: $p_s = 1 - p_d - p_e$.

V. Evaluation

Domains. We evaluated our framework in two domains: movie ticket booking and pizza ordering. Each domain dataset consists of a database of items (movie tickets, pizza configurations), a random set of user goals, and a list of domain slots. For the movie ticket booking domain, we used the dataset created by Li et al. [7], which contains 280 goal-oriented dialogues. In each of these dialogues, a user attempts to book a ticket for one of 991 movies. We used the 133 goals sampled in [7] to initiate the user simulations. A goal in this context is a combination of slot values describing the user's desired movie booking, and a list of slots whose values are queried by the user (Section IV). For the pizza ordering domain, we built a database of pizza orders where we based our pizza specifications and slot list on the options Dominos offers. For example, a "small" pizza can have two crust types: "hand tossed" and "gluten free". We randomly sampled 10,000 pizza configurations as user goals and used them to initiate our simulations.

Implementation details. For the pizza ordering domain, we used an extended version of the dependency graph in Figure 3, where we added optional supernodes to address user concerns related to privacy, security, and final price details. For the movie booking domain, we created a dependency graph similar to that of the pizza ordering, but with supernodes containing movie related slots. We built our RL model as described in Section III, and used Deep Q-Network [18](DQN) to train our RL agent. We used Adam optimizer with a learning rate 10^{-3}, binary cross entropy as a loss function, and a batch size of 16 in our training. Our DQN has two layers with relu and softmax activations, and hidden size of 80. The reward value for successful and failed conversations are $+30$ and -30, respectively.

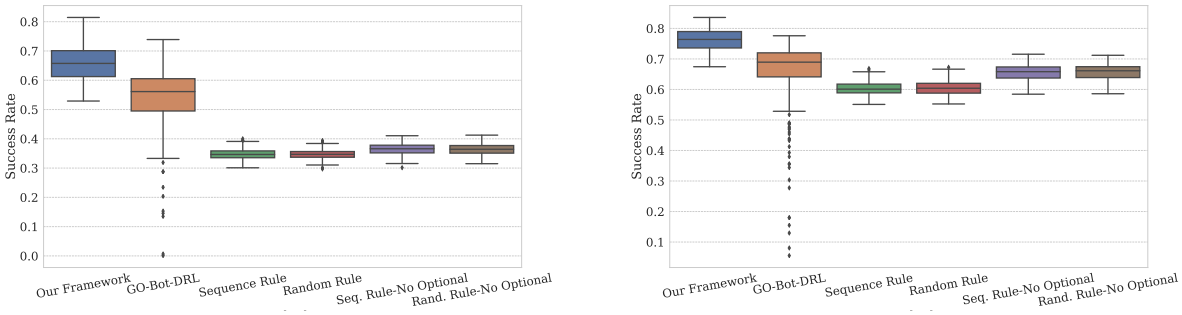
Competing methods. We compare our framework against the following methods:

GO-Bot-DRL [7]: State-of-the-art RL dialog policy generation model that learns dependency violations from negative user reactions, which are manifested through an increase in user dropout rate.

Sequence Rule: A diagram-based dialog policy that selects the next action according to a predefined dialog flow and includes all optional slots. We also compare against a variant of this baseline that does not present optional node.

Parameter	Value	Description
Global initial drop out	3%	The base drop out rate (Equation 1). Consider a conversation that lasts for 20 turns: 3% global drop out rate results in a probability of $(1-0.03)^{20} \approx 0.543$, or 54.3% that a user had not dropped out by the end of turn 20 (without considering other factors such as slot error rate p_e).
Drop out delta (δ)	1%, 0.5%	The maximum drop out change per conversation turn in Equation 2.
Drop out increase due to violation (β)	3%, 4%	Increase in user drop out when she is presented with a dependency violating utterance. (Equation 3).
Slot error rate (p_e)	5%	The rate of uninterpretable user utterances and NLU errors; we set the value of this parameter similarly to other studies [7], [17].

TABLE I: User simulator parameters. We chose the values of the parameters in a way that maintains a reasonable dropout probability of around 50% within 20 dialog turns. Higher values would result in unrealistically high dropout probabilities; thus, not enabling the RL models to make progress, and lower values would result in unrealistically easy dialog tasks (user rarely drops out) that do not challenge the RL models enough.



(a) Movie domain

(b) Pizza domain

Fig. 4: Success rates of all competing methods.

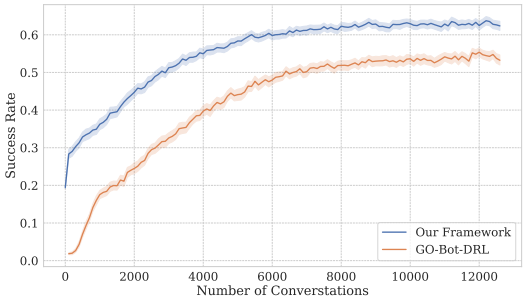
Random Rule: A diagram-based dialog policy where the next action is picked at random to fill the next empty slot and includes all optional slots. We also compare against a variant of this baseline that does not present optional slots.

Evaluation metrics. Similarly to other works in goal-oriented dialog systems [7], [17], we use the success rate as the main quantitative measure of success for a dialog policy, and we use simulated conversations in our evaluation. The success rate of a dialog policy is the ratio of conversations where the user reaches the end of the conversation, e.g., confirms a pizza order. For the RL based models, we report success rates during the training (i.e., adaptation) and testing phases; we used different user goals in the training and the testing phases. For the training phase, we show how the success rates evolve as an RL agent does more conversations to highlight the difference in the amount of training conversations needed for each model. For the testing phase, we simulated 2,500 conversations from unseen user goals for all the competing methods. Although it is desirable to evaluate dialog systems by having them chat with real users, (i) the large number of conversations; (ii) the need for diverse user profiles; and (iii) the nature of our dialogs, the semantic level, make it unrealistic to perform evaluation using real user studies.

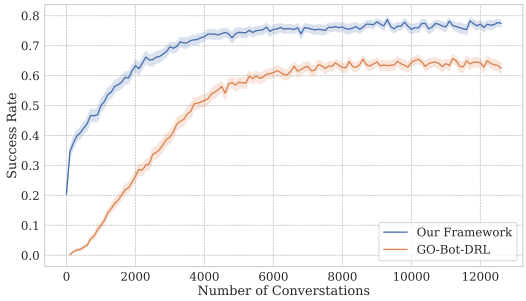
Experimental setup. We did our experiments on a set of different user profiles (explained in Section IV). Specifically, there are six user parameters, each can take a value between 0 and 1. In our experiments, we set the value of each parameter to either 0 or 1; therefore, we consider $2^6 = 64$ different user profiles. Note that the user profiles are not visible to any of the models, however, RL models adapt to different user profiles based on user behaviour during the training phase. We report the average results across all user profiles. Since we have a fixed number of user goals to initiate simulated conversations, we randomly partition these goals into train/test sets with a ratio of 9 to 1 (i.e., we do a 10-fold cross validation). We pre-trained our framework and GO-Bot-DRL using a traditional agenda-based user simulator to teach them a basic dialog policy. We trained and evaluated both models using our user simulator (Section IV) to study their adaptivity. The values of the user simulator parameters are described in Table I.

A. Success rate comparison

Figure 4 shows the success rates for all competing methods averaged over 2,500 conversations. Our framework and GO-Bot-DRL were trained using 12,000 simulated conversations. This experiment shows that our framework outperforms other methods in the more realistic setting where users may drop out of a conversation based on their characteristics. Compared to GO-Bot-DRL, our framework's success rate is 22.69% and 23.77% higher on average for the movie and pizza domains, respectively. Diagram-based methods' performances are lower than both our method and the GO-Bot-DRL, especially in the movie ticket booking domain. Since the pizza ordering



(a) Movie domain



(b) Pizza domain

Fig. 5: Evolution of success rate for our framework and GO-Bot-DRL. The average success rates over intervals of 100 conversations are shown for both methods.

task is shorter and simpler than the movie booking one, all methods perform reasonably well. Diagram-based methods achieve lower success rates because they fail to recover when they receive an erroneous utterance; they get stuck and cause users to drop out, unlike RL models that ultimately learn to avoid conversational patterns that lead to dropouts.

B. Adaptation comparison

Figure 5 shows how the success rate evolves with the number of simulated conversation during the training (i.e., adapting to latent user profiles) phase for our framework and GO-Bot-DRL. Note that both models were pre-trained using the same simulated conversations (generated using a traditional agenda based simulator) before staring this experiment. In the training phase, our framework starts with a higher success rate because it learned to avoid violating conversations during pre-training more effectively, thanks to the dependency graph restrictions. For both domains, our framework adapts to user characteristics faster than GO-Bot-DRL. Specifically, our framework achieves a success rate of 30% after tens of conversations in the movie domain, while GO-Bot-DRL reaches the same success rate after thousands of conversations. This highlights the advantage of coupling dependency graphs with RL: dependency graphs restrict RL's reliance on conversations to learn violating conversational patterns. For the movie domain, our framework reaches a success rate of around 65% after 8,000 conversations. while GO-Bot-DRL reaches a success rate of around 55% after 12,000 conversations. Similar patterns can be observed in the pizza domain as well.

C. Analysis

In this section, we study some aspects of the performance of the competing dialog policy generation methods to provide some insight into why our model achieves a higher success rate than other methods. Specifically, we study the effect of dependency graphs on conversation length and ratio of violating conversations.

Violation rate comparison. In this experiment, we validate the benefit of coupling dependency graphs with RL by showing that our framework is able to learn avoiding conversations

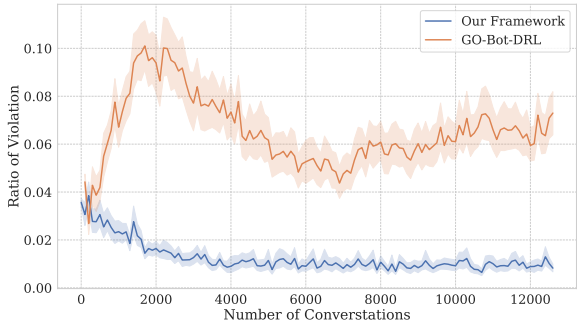


Fig. 6: Rate of violating conversations in the movie domain (lower is better)

that violate dependency graph constraints earlier than the GO-Bot-DRL. Note that both models eventually learn avoiding violating conversations. In our framework, the reward function gives the RL agent a negative reward as soon as it generates a violating utterance based on the respective dependency graph constraints, which results in the model learning to avoid violating actions as early as possible. In GO-Bot-DRL, the agent learns to avoid violating actions eventually due the dropout rate increase (parameter β in Equation 3). Figure 6 shows that our method quickly achieves a significantly lower rate of violating conversations than GO-Bot-DRL.

Conversation length comparison. Figure 7 shows the average conversation length for all methods in the pizza domain. Note that in goal-oriented chatbots, it is desirable to keep conversation length to a minimum, while achieving high success rate, to avoid losing user interest. The variants of rule-based methods that never present optional slots achieve a small average conversation length, and their counterparts that always show optional slots have high conversation length; however, these methods have a low success rate due to their rigid dialog flow. Both our framework and GO-Bot-DRL provide dynamic conversation flows and their average conversation length is very similar. This indicates that both methods are not always presenting or skipping optional slots, but they rather

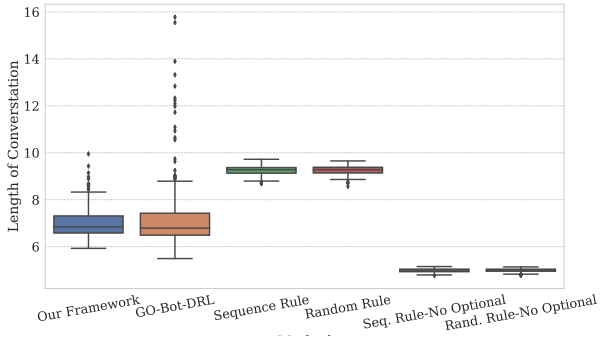


Fig. 7: Average conversation length for all competing methods in the pizza domain.

learn whether to present optional slots or not based on the specific user characteristics. Yet our framework achieves higher success rate compared to GO-Bot-DRL. This confirms that our framework learns which optional slots to present, based on user characteristics, more effectively.

VI. Related Work

Chit-chat dialog system. Many dialog systems are designed to engage in open-ended dialog with users, referred to in the literature as chit-chat systems [1]. Early chit-chat systems used rule-based techniques. A prominent example of an early chit-chat system is ELIZA [2], which uses pattern matching and transformation rules to generate system utterances. Modern chit-chat systems are either retrieval-based or generative [19]. The authors in [20] propose a retrieval-based dialog system that combines recurrent neural networks (RNNs) with latent topic models to learn representations of user utterances and responses, and retrieves relevant responses using the learned representations. These systems are generally meant to entertain users without necessarily having any goal, and are outside the scope of this paper.

Goal-oriented dialog systems. We focus in this paper on goal-oriented dialog systems, where the system is meant is to help a user achieve a specific task. The authors in [15] proposed the frame-based dialog management paradigm, which is the basis for most work in goal-oriented dialog systems [1]. In this framework, the main elements of a conversation are frames and dialog acts. Each frame is a group of semantically related slots, which collectively constitute the information the system needs to know. Dialog acts are the actions that the system can make to interact with a user and fill slots. Most recent works on goal-oriented dialog systems use either an end-to-end or a modular pipeline consisting of: natural language understanding (NLU) module, dialog state tracking (DST) module, dialog policy (DP) module, and natural language generation (NLG) module [21]. End-to-end goal-oriented dialogue systems such as the ones described in [22] and [23] are trained using the user-system utterances in an end-to-end fashion, and they try to capitalize on the success of the seq2seq-based open-domain chit-chat systems. Despite all such encouraging results, the scarcity of domain specific dialogue data [24] makes it difficult to train robust end-to-end dialog systems. Thus, the modular pipeline remains more practical. While progress in any of the mentioned modules is likely to improve the overall quality of goal-oriented dialog systems, we concentrate on dialog policy design and management.

Dialog policy management. An intuitive way to manage dialog policies is using diagrams [25] (or rules), where each slot corresponds to a node, and the transitions between slots are decided based on the slot value provided by the user. While such dialog policy is rigid, it is predictable and fairly easy to design. Thus, commercial dialog system platforms such as Google's DialogFlow [9] and Amazon Lex [10] embrace it. Many research systems model dialog policy generation as a Markov decision process (MDP) [26], [27] or a partially observed MDP (POMDP) [28], [29], [30], [31] and utilize reinforcement learning to optimize for better dialog policies. The authors in [32] propose a comprehensive framework for dialog management. In this framework, the authors propose a construct similar in spirit to our chatbot dependency graphs; however, dependency graphs are more flexible in the sense that they encabsulate richer sets of dialog flows. ConvLab [33], which is a suite of tools for training and evaluating goal-oriented dialog systems offers implementations of state-of-the-art RL-based dialog policy generation methods. Unlike diagrams, our framework offers a convenient way to design flexible dialog policies, and unlike pure RL methods, our framework does not heavily rely on training data to adapt to user characteristics.

VII. Conclusions

We have presented a flexible dialog policy generation framework for goal-oriented chatbots. First, we proposed chatbot dependency graphs as an intuitive structure that captures conversation constraints imposed by the respective domain. Dependency graphs are easy to design and they mitigate the rigidity of chatbot diagrams by encapsulating all valid conversation flows. Then, we proposed coupling dependency graphs with RL models, and we proposed training and evaluating these models using a user simulator that takes into account different user characteristics. Our RL agent is able to dynamically select the best valid conversation flow based on the dependency graph constraints and the user characteristics. We evaluated our framework using an agenda-based user simulator augmented with the notion of user profiles and user dropouts for more realistic user modeling. Our experimental evaluation in the movie ticket booking and pizza ordering domains show that our method achieves success rates up to 23.77% higher than a state-of-the-art RL-based method.

Acknowledgment

This work was supported by NSF grants IIS-1619463, IIS-1838222, and IIS-1901379.

REFERENCES

[1] D. Jurafsky and J. H. Martin, *Speech and Language Processing*. preprint, 2017, ch. Dialog Systems and Chatbots.

[2] J. Weizenbaum *et al.*, "Eliza—a computer program for the study of natural language communication between man and machine," *Communications of the ACM*, vol. 9, no. 1, pp. 36–45, 1966.

[3] L. Zhou, J. Gao, D. Li, and H.-Y. Shum, "The design and implementation of xiaoice, an empathetic social chatbot," *arXiv preprint arXiv:1812.08989*, 2018.

[4] J. Williams, A. Raux, and M. Henderson, "The dialog state tracking challenge series: A review," *Dialogue & Discourse*, vol. 7, no. 3, pp. 4–33, 2016.

[5] A. Coucke, A. Saade, A. Ball, T. Bluche, A. Caulier, D. Leroy, C. Doumouro, T. Gisselbrecht, F. Caltagirone, T. Lavril *et al.*, "Snips voice platform: an embedded spoken language understanding system for private-by-design voice interfaces," *arXiv preprint arXiv:1805.10190*, 2018.

[6] "Dialog system technology challenge 8," https://sites.google.com/dstc.community/dstc8/, 2019.

[7] X. Li, Y.-N. Chen, L. Li, J. Gao, and A. Celikyilmaz, "End-to-end task-completion neural dialogue systems," in *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 2017, pp. 733–743.

[8] A. B. Siddique, S. Oymak, and V. Hristidis, "Unsupervised paraphrasing via deep reinforcement learning," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1800–1809. [Online]. Available: https://doi.org/10.1145/3394486.3403231

[9] Google, "Google dialogflow," https://dialogflow.com/, 2019.

[10] Amazon, "Amazon lex," https://aws.amazon.com/lex/, 2019.

[11] J. Gao, M. Galley, L. Li *et al.*, "Neural approaches to conversational ai," *Foundations and Trends® in Information Retrieval*, vol. 13, no. 2-3, pp. 127–298, 2019.

[12] B. Liu and I. Lane, "Iterative policy learning in end-to-end trainable task-oriented neural dialog models," in *2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. IEEE, 2017, pp. 482–489.

[13] J. Schatzmann, B. Thomson, K. Weilhammer, H. Ye, and S. Young, "Agenda-based user simulation for bootstrapping a pomdp dialogue system," in *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers*. Association for Computational Linguistics, 2007, pp. 149–152.

[14] E. Constantinides, "Influencing the online consumer's behavior: the web experience," *Internet research*, vol. 14, no. 2, pp. 111–126, 2004.

[15] D. G. Bobrow, R. M. Kaplan, M. Kay, D. A. Norman, H. Thompson, and T. Winograd, "Gus, a frame-driven dialog system," *Artificial intelligence*, vol. 8, no. 2, pp. 155–173, 1977.

[16] P. Shah, D. Hakkani-Tür, G. Tür, A. Rastogi, A. Bapna, N. Nayak, and L. Heck, "Building a conversational agent overnight with dialogue self-play," *arXiv preprint arXiv:1801.04871*, 2018.

[17] X. Li, Z. C. Lipton, B. Dhingra, L. Li, J. Gao, and Y.-N. Chen, "A user simulator for task-completion dialogues," *arXiv preprint arXiv:1612.05688*, 2016.

[18] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[19] R. Yan, "" chitty-chitty-chat bot": Deep learning for conversational ai." in *IJCAI*, vol. 18, 2018, pp. 5520–5526.

[20] Y. Wu, Z. Li, W. Wu, and M. Zhou, "Response selection with topic clues for retrieval-based chatbots," *Neurocomputing*, vol. 316, pp. 251–261, 2018.

[21] U. Farooq, A. B. Siddique, F. Jamour, Z. Zhao, and V. Hristidis, "App-aware response synthesis for user reviews," in *2020 IEEE International Conference on Big Data (Big Data)*, 2020, pp. 699–708.

[22] A. Bordes, Y.-L. Boureau, and J. Weston, "Learning end-to-end goal-oriented dialog," *arXiv preprint arXiv:1605.07683*, 2016.

[23] M. Eric and C. D. Manning, "A copy-augmented sequence-to-sequence architecture gives good performance on task-oriented dialogue," *arXiv preprint arXiv:1701.04024*, 2017.

[24] H. Chen, X. Liu, D. Yin, and J. Tang, "A survey on dialogue systems: Recent advances and new frontiers," *Acm Sigkdd Explorations Newsletter*, vol. 19, no. 2, pp. 25–35, 2017.

[25] M. F. McTear, "Modelling spoken dialogues with state transition diagrams: experiences with the cslu toolkit," in *Fifth International Conference on Spoken Language Processing*, 1998.

[26] E. Levin, R. Pieraccini, and W. Eckert, "A stochastic model of human-machine interaction for learning dialog strategies," *IEEE Transactions on speech and audio processing*, vol. 8, no. 1, pp. 11–23, 2000.

[27] S. Singh, D. Litman, M. Kearns, and M. Walker, "Optimizing dialogue management with reinforcement learning: Experiments with the njfun system," *Journal of Artificial Intelligence Research*, vol. 16, pp. 105–133, 2002.

[28] V. Rieser and O. Lemon, *Reinforcement learning for adaptive dialogue systems: a data-driven methodology for dialogue management and natural language generation*. Springer Science & Business Media, 2011.

[29] J. D. Williams and S. Young, "Partially observable markov decision processes for spoken dialog systems," *Computer Speech & Language*, vol. 21, no. 2, pp. 393–422, 2007.

[30] S. Young, M. Gašić, S. Keizer, F. Mairesse, J. Schatzmann, B. Thomson, and K. Yu, "The hidden information state model: A practical framework for pomdp-based spoken dialogue management," *Computer Speech & Language*, vol. 24, no. 2, pp. 150–174, 2010.

[31] B. Thomson and S. Young, "Bayesian update of dialogue state: A pomdp framework for spoken dialogue systems," *Computer Speech & Language*, vol. 24, no. 4, pp. 562–588, 2010.

[32] D. Bohus and A. I. Rudnicky, "The ravenclaw dialog management framework: Architecture and systems," *Computer Speech & Language*, vol. 23, no. 3, pp. 332–361, 2009.

[33] S. Lee, Q. Zhu, R. Takanobu, X. Li, Y. Zhang, Z. Zhang, J. Li, B. Peng, X. Li, M. Huang *et al.*, "Convlab: Multi-domain end-to-end dialog system platform," *arXiv preprint arXiv:1904.08637*, 2019.