Knowledge-Driven Slot Constraints for Goal-Oriented Dialogue Systems

Piyawat Lertvittayakumjorn∗  Daniele Bonadiman  Saab Mansour
Imperial College London  Amazon AI  Amazon AI
pl1515@ic.ac.uk  dbonadim@amazon.com  saabm@amazon.com

Abstract
In goal-oriented dialogue systems, users provide information through slot values to achieve specific goals. Practically, some combinations of slot values can be invalid according to external knowledge. For example, a combination of “cheese pizza” (a menu item) and “oreo cookies” (a topping) from an input utterance “Can I order a cheese pizza with oreo cookies on top?” exemplifies such invalid combinations according to the menu of a restaurant business. Traditional dialogue systems allow execution of validation rules as a post-processing step after slots have been filled which can lead to error accumulation. In this paper, we formalize knowledge-driven slot constraints and present a new task of constraint violation detection accompanied with benchmarking data. Then, we propose methods to integrate the external knowledge into the system and model constraint violation detection as an end-to-end classification task and compare it to the traditional rule-based pipeline approach. Experiments on two domains of the MultiDoGO dataset reveal challenges of constraint violation detection and sets the stage for future work and improvements.

1 Introduction
Natural language understanding (NLU) is an important component of goal-oriented dialogue systems. The function of NLU is to construct a semantic frame for a user utterance by performing two tasks – intent classification (IC) and slot labelling (SL) (Chen et al., 2017). The former task aims to identify the intent of the user (i.e., an activity or a transaction that the user wants to accomplish), while the latter task extracts attributes of the intent. For example, given an input utterance “Please add one XL fries to my order” in Figure 1(A), IC classifies that the user intent is “AddToOrder” (Adding a new menu item to the order), while SL detects “one”, “XL”, and “fries” as Quantity, MenuItemSize, and MenuItem, respectively. These two tasks, IC & SL, could be performed either independently (Zhao and Wu, 2016; Haffner et al., 2003; Kurata et al., 2016) or jointly (Xu and Sarikaya, 2013; Li et al., 2018; Gupta et al., 2019) although recent research shows that training jointly generally leads to better results (Hakkani-Tür et al., 2016; Goo et al., 2018).

To make the recognition of intents and slots more reliable, NLU models require the list of all possible intents and the slots associated to each intent. For instance, the intent show_flights has airline, departure_city, arrival_city, departure_date, and departure_time as its associated slots. Practically, each slot has its own type. Some types are domain-agnostic such as DATE for the departure_date, while other types are domain-specific, such as AIRLINE for the slot airline. We also refer to the latter category as custom slot types, for which custom lists of valid entities are provided. Moreover, slots could be marked as either required (such as departure_city and arrival_city) or optional (such as airline and departure_time). All of these details are usually defined structurally in a single document called a bot schema which guides the conversational flow of the dialogue system (Peskov et al., 2019; Rastogi et al., 2019).

Besides the above details, the dialogue domain may have conditions permitting or forbidding some combinations of slot values. For example, for a book_flight intent which has “Singapore airlines” as the airline slot, not all cities are valid destinations where the airlines operate. The NLU may deal with invalid combinations of slot values by just ignoring them, i.e., not detecting them in the SL task. This approach will result in a deteriorated user experience as the users would not know why their attempts to provide slot values are not successful. Therefore, we envision these conditions as constraints between slots, and the system should be able to detect constraint violations and
request new slot combinations from the users when the violations happen. However, to the best of our knowledge, we have not found any existing work formalizing the constraints between slots nor modeling detection of constraint violations.

In this paper, we formally represent the slot constraints which could be integrated into a bot schema and present a new task of constraint violation detection: given a bot schema with constraints, a current utterance, and a conversation history, predict whether the current state of conversation violates any constraints or not and which constraints are violated. After that, we propose three approaches to solve this problem (based on a pipeline approach and an end-to-end approach) and conduct experiments with two domains of the MultiDoGO dataset (Peskov et al., 2019) augmented with constraint violation labels. By design, the end-to-end approach does not suffer from error accumulation (whereas the pipeline approach does); however, it is more difficult to inject the constraint information into the end-to-end approach. The experimental results reveal challenges of the violation detection task together with room for improvement.

Overall, the main contributions of this paper are as follows.

• We experiment with several unsupervised methods for open entity linking (based on string similarity, natural language inference, and combinations of them) as a part of the pipeline approach.

The remainder of this paper is organized as follows. Section 2 explains related work about natural language understanding in dialogue systems as well as entity linking. Section 3 presents formal representations of the constraints. Section 4 proposes the three approaches we use to detect constraint violations. Section 5 explains the created datasets and the experimental results. Finally, section 6 concludes the paper.

2 Related Work

2.1 Goal-Oriented Dialogue Systems

Goal-oriented dialogue systems allow the usage of natural language to achieve specific goals such as food ordering or travel booking. Traditionally, these systems are built using a pipeline approach including user intent and slots detection (NLU), dialog management and knowledge base querying (Levin et al., 2000; Williams and Young, 2007; Young et al., 2013). The ability to interface with external knowledge is essential as it constraints possible entities and their relations per application (e.g., different restaurants can have different menus) and guides the system responses. Constraints detection is usually handled by a post-processing step, for example in the DSTC2 dataset (Henderson et al., 2014), the canthelp act is inferred if the database returns zero results. In addition, previous work integrated knowledge base information or lists of potential slot entities into goal-oriented dialogue systems but did not model constraint violation detection (Madotto et al., 2018; Liu et al., 2018; Rastogi et al., 2019; Zhang et al., 2020). In this
work, we fill the gap by first formalizing the task of constraint violation detection for dialogue systems and modeling it using supervised machine learning.

2.2 Entity Linking

Entity linking aims to link entity mentions (i.e., slot values) \( v \) in user utterances with their corresponding entities \( e \in E \) defined in the bot schema (where \( E \) is a list of all possible entities of the associated slot type). According to Shen et al. (2015), an entity linking system generally consists of three modules. First, candidate generation filters out irrelevant entities from \( E \) to reduce the search space. Second, candidate ranking ranks the candidates to find the entity which the mention most likely refers to. Third, unlinkable mention prediction predicts whether the correct entity is really in \( E \) or not. In this paper, we assume that the first module is not needed because the set \( E \) for goal-oriented dialogue systems is usually in a manageable size. So, our focus is on the last two tasks.

Candidate ranking could be done in either a supervised way (Chen and Ji, 2011; Gupta et al., 2017; Kolitsas et al., 2018) or an unsupervised way (Cucerzan, 2007; Chen et al., 2010; Xu et al., 2018). Potential features for ranking include surface names, popularity, types of the entities, and the context surrounding the mention and the entities (Shen et al., 2015). Usually, it is not easy to find a large annotated dataset to train a candidate ranking model for goal-oriented dialogue systems. Hence, in our approaches, we conduct unsupervised entity linking based on surface names and types of the entities. Due to the same limitation, we use unsupervised methods to perform unlinkable mention prediction which are using a threshold (Ferragina and Sciaiella, 2010; Gottipati and Jiang, 2011), discussed in section 4.

3 Constraint Representation

As constraint violation check must be applied to every state in the conversation, we first define dialogue states as follows.

**Definition 1** A dialogue state \( d \) is a tuple \((d_i, d_s)\) where \( d_i \) is an intent and \( d_s \) is a list of slot-value pairs (Rastogi et al., 2019).

Figure 1(A) shows a dialogue state \( d \) as an example. Next, to represent a constraint, we define atomic formula – the smallest logical condition in constraint statements.

**Definition 2** An atomic formula \( f \) can be written as \((s, o, v)\) where \( s \) is a slot variable, \( v \) is a list of values, and \( o \in \{=, \geq, >, \leq, <, \neq\} \) is an operator. A dialogue state \( d \) satisfies \( f \) if and only if the corresponding slot value \( s \) in \( d_s \) satisfies \( f \).

For instance, the dialogue state \( d \) in Figure 1(A) satisfies an atomic formula \( f = \{(\text{Menu} \text{Item} \text{Size}, \text{in}, \{\text{‘medium’}, \text{‘large’}, \text{‘extra} \text{large’}\})\} \).

**Definition 3** A constraint \( c \) is a triple \((c_i, c_s, c_l)\) where (1) \( c_i \) is a list of intents where the constraint applies, (2) \( c_s \) is a list of associated slots \((s_1, s_2, \ldots, s_n)\), and (3) \( c_l \) is a constraint statement defined on \( c_s \) – a logical formula in disjunctive normal form where each conjunction consists of \( n \) atomic formulas that correspond to \( n \) slot variables in \( c_s \).

Figure 1(B) shows an example of constraints between MenuItem and MenuItemSize, applying to the AddToOrder intent. Basically, it specifies valid sizes of each menu item.

**Definition 4** A constraint \( c \) is applicable to a dialogue state \( d \) if and only if \( d_i \in c_i \) and \( c_s \subseteq d_s \).

In other words, a constraint applies to a dialogue state when the dialogue state has an applicable intent and contains all the relevant slot variables. In Figure 1, the constraint \( c \) is applicable to the dialogue state \( d \) but not applicable to, for instance, \( d' = (\text{AddToOrder}, \{\text{Quantity}: \text{‘Fries’}\}) \).

**Definition 5** A dialogue state \( d \) violates a constraint \( c \) if and only if \( c \) is applicable to \( d \) but \( d \) does not satisfy \( c_l \).

For the running example, \( d \) does not violate \( c \) because the slot-value pairs \{Menu: ‘Fries’, MenuItemSize: ‘extra large’\} of \( d \) satisfies \( c_l \). Note that, in Figure 1(A), the dialogue state is a result of a single utterance. However, a dialogue state in practice contains the information of the current user turn fused with the dialogue state of the previous turn. So, the objective of the constraint violation detection task is checking whether any constraints defined in the bot schema are violated after the dialogue state is updated with the information of the current turn.

4 Constraint Violation Detection

We propose three approaches to tackle this problem. The overview is shown in Figure 2.
4.1 Deterministic Pipeline Approach

To detect constraint violations, the deterministic pipeline approach (DP) performs three steps. First, it runs **intent classification and slot labelling** on the input utterance. Since the detected slot values may have different surface forms from the entities defined in the bot schema and the constraints, DP conducts **entity linking** and updates the dialogue state using the predicted intent and the linked entities, as the second step. In the third step, DP runs a **deterministic satisfiability check** simply on the dialogue state to detect violations.

To implement DP, we use JointBERT (Chen et al., 2019), with default hyper-parameters, to perform IC/SL in the first step. JointBERT utilizes BERT-base (Devlin et al., 2019) as an encoder to jointly predict the intent and the slot values. Following Chen et al. (2019), we add Conditional Random Fields (CRF) on top of the BERT model to leverage dependencies between slot labels.

The second step, entity linking, is challenging because goal-oriented dialogue systems are usually domain-specific and no training data for entity linking is provided. Furthermore, a detected slot value may not correspond to any entity defined in the bot schema. So, this step should predict None as an answer when the value cannot be linked. These two conditions make this step become unsupervised open entity linking. In this paper, we use the following methods to perform this step.

1. **String similarity**: We link a slot value to the most similar defined entity. Three methods to measure similarity are used – exact match, Jaccard Index on character bigrams (so called Bijaccard metric for short) (Jaccard, 1901), and Levenshtein edit distance (Levenshtein, 1966). For the exact match method, we link a slot value to an entity only if their surface forms exactly match (case-insensitive). Otherwise, we return None. In contrast, for Bijaccard and Levenshtein, we always answer the most similar entity. So, they cannot detect unlinkable slot values.

2. **Natural language inference (NLI)**: NLI aims to predict if a hypothesis is true (entailment), false (contradiction), or undetermined (neutral) given a premise. To predict if a slot value \(v\) corresponds to an entity \(e\), we apply a pre-trained NLI model, in particular RoBERTa (Liu et al., 2019) pre-trained on MNLI (Williams et al., 2018), to predict if \(v\) (premise) entails \(e\) (hypothesis) and return the entity that gets the highest entailment score. Also, we set a threshold of 0.8 for predicting unlinkable values. That means we predict None if the highest entailment probability is less than 0.8.

3. **Average scores of methods**: We average the scores returned from the three methods (Bijaccard, Levenshtein, and NLI) to be the final entity score. Bijaccard and NLI scores already stay between 0 and 1 where 1 is the best score. To combine the Levenshtein edit distance with these two methods, we transform the edit distance \(x\) to be \(1 - \frac{x}{a}\) where \(a\) is the length of the slot value \(v\). Then we return the entity with the highest average score. We also have an option of returning None when the highest average score is less than a threshold of 0.5.

4.2 Probabilistic Pipeline Approach

The probabilistic pipeline approach (PP) has the same three steps as the deterministic one. The difference is that instead of linking one slot value to one entity, PP uses the probability distribution (i.e., the entity linking scores normalized using softmax) over the candidate entities (including None) to represent the slot value. To predict whether the dia-
logue state violates a constraint $c$, we calculate the probability of each valid entity combination $\alpha$ according to the constraint statement $c_l$ and define the violation score as $1 - \sum_{\alpha|\alpha=c_l} P(\alpha)$. If the violation score is larger than a threshold of 0.5, PP predicts that the dialogue state violates the constraint $c$.

We use four entity linking methods to generate the raw linking scores (before softmax) including Bijaccard, Levenshtein edit distance (normalized by the length of the slot value), NLI, and average scores of the three methods. The raw score of None is set at the threshold, i.e., 0.8 and 0.5 for NLI and the average method, respectively.

4.3 End-to-End Approach

The end-to-end approach (EE) aims to predict violations without performing intermediate steps like IC/SL or entity linking. This task can be seen as multilabel classification – predicting all the violations that the current dialogue state causes. Hence, the number of classes equals the number of constraints defined in the bot schema. We use BERT as a text encoder and apply a linear layer (with sigmoid function) on top of the embedding of the CLS token to predict violations\(^2\). Then binary cross-entropy loss is used for optimization on the training data that maps conversations to violations. This is different from the pipeline approaches which use the training data at the IC/SL step, not the violation detection step.

Because EE does not construct the dialogue state along the way, it needs to consider both the current turn and all the previous turns to predict violations. Therefore, all the user utterances till the current turn are concatenated to be an input of the BERT model. If the input length is longer than the maximum input length of BERT, we trim off the older turns to make the input meet the length limit.

5 Experiments

5.1 Datasets

As constraint violation detection is a novel problem, there had not been an existing dataset for this task. So, we modified two domains, insurance (sentence-level annotation) and fast food (turn-level annotation), of the MultiDoGO dataset (Peskov et al., 2019), which is an English multi-domain goal-oriented IC/SL dataset, to support violation detection as follows.

- We created a list of possible entities for each custom slot type by manually investigating and grouping slot values annotated in the dataset\(^3\).
- We mapped distinct surface forms of slot values to the corresponding entities we just defined. These mappings would be used as ground truths for entity linking testing.
- We analyzed the co-occurrences of the entities and then manually wrote constraints for each intent.
- We constructed a dialogue state for each turn in the dataset semi-automatically using the mapped entities and meaningful rules. For example, entities found in the ‘ContentOnly’\(^4\) turn were associated to the dialogue state of the most recent domain intent.
- We ran deterministic satisfiability check on the dialogue states and added the constraint violation results to the dataset. The check here is the same as the last step of the DP approach, so we can expect that the last step of DP works perfectly if the input, obtained from the previous step (entity linking), is correct.

Table 1 summarizes the statistics of the augmented MultiDoGO dataset. Both domains share the same set of general intents including OpeningGreeting, ClosingGreeting, Confirmation, ContentOnly, OutOfDomain, ThankYou, and Rejection. The three domain intents of the insurance domain are CheckClaimStatus, GetProofOfInsurance, and ReportBrokenPhone, while the domain intents of the fast food domain concern different types of food such as OrderBreakfastIntent, OrderBurgerIntent, and OrderDessertIntent.

The insurance and the fast food domains have three out of nine and six out of ten custom slot types, respectively. For each custom slot type, we create a closed type constraint indicating that a linked entity must be in the set of possible entities recognized by the slot type. In addition, we

\(^2\)We use the default hyper-parameters of JointBERT.

\(^3\)SL annotations in the public MultiDoGO dataset do not include boundaries of slot values. This is problematic especially for the fast food domain where utterances usually contain multiple slot values consecutively. Hence, we requested the raw fast food data from Peskov et al. and imported the slot boundaries into our modified version using the BIO schema.

\(^4\)ContentOnly intent is used when the user is providing details in response to a question from the agent (Peskov et al., 2019).
have domain-specific constraints enforcing the domain knowledge. The insurance domain has only the car_model_brand constraint specifying valid car models for each car brand. Among the twelve constraints of the fast food domain, eight of them specify valid menu items for each domain intent, two of them specify valid sizes for each menu item, and the other two specify valid ingredients for each menu item.

Concerning conversation statistics, on average, the fast food domain has more slot values per turn than the insurance domain (because a user can mention several ingredients and menu items in one turn). Besides, it has more unlinkable slot values (None), resulting in more closed type constraint violations than the insurance domain. Since the fast food domain has so many constraints, only 32.8% of the conversations and 48.7% of the user turns do not have any violations. The average violations per turn of 1.38 results from some turns having many violations. For instance, when a user orders an unrecognized pizza menu with some unrecognized ingredients, the detected intent is ‘OrderPizzaIntent’ whereas the slots are mapped to ‘None’ entities causing closed type constraint violations for the food item (pizza) slot and the ingredient slot. Moreover, they violate the constraint of valid food items for the ‘OrderPizzaIntent’ intent and another constraint of valid combinations between food items and ingredients.

### Table 1: Dataset statistics of the two MultiDoGO domains augmented with constraint violation data.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Insurance</th>
<th>Fast food</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of general intents</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Number of domain intents</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>Number of slot types</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>Number of custom slot types</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td># of closed type constraints</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td># of domain-specific constraints</td>
<td>1</td>
<td>12</td>
</tr>
</tbody>
</table>

### Conversation statistics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Insurance</th>
<th>Fast food</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of dialogues</td>
<td>2,332</td>
<td>2,076</td>
</tr>
<tr>
<td>Number of user turns</td>
<td>19,675</td>
<td>18,673</td>
</tr>
<tr>
<td>Avg. turns per dialogue</td>
<td>8.44</td>
<td>8.99</td>
</tr>
<tr>
<td>Number of tokens</td>
<td>66,957</td>
<td>71,415</td>
</tr>
<tr>
<td>Avg. tokens per turn</td>
<td>3.40</td>
<td>3.82</td>
</tr>
<tr>
<td>Total unique tokens</td>
<td>6,945</td>
<td>5,747</td>
</tr>
<tr>
<td>Number of slot values</td>
<td>9,504</td>
<td>11,618</td>
</tr>
<tr>
<td>Avg. slot values per turn</td>
<td>0.48</td>
<td>0.62</td>
</tr>
<tr>
<td>% of unlinkable slot values</td>
<td>7.58</td>
<td>14.21</td>
</tr>
<tr>
<td>% of non-violating dialogues</td>
<td>74.79</td>
<td>32.80</td>
</tr>
<tr>
<td>% of non-violating turns</td>
<td>85.62</td>
<td>48.66</td>
</tr>
<tr>
<td>Avg. violations per turn</td>
<td>0.26</td>
<td>1.38</td>
</tr>
</tbody>
</table>

Table 2: Intent classification (IC) and slot labeling (SL) results (in %) of JointBERT used in the pipeline approaches.

### 5.2 Implementation

We used PyTorch as a core framework for the three approaches. External packages we used include JointBERT for IC/SL, edit-distance for string similarity, and transformers for the BERT-base (for all the three approaches) and RoBERTa (for NLI). In addition, we used the softmax temperature of 0.1 to convert raw entity linking scores to probability in the probabilistic pipeline approach.

### 5.3 IC/SL and Entity Linking Results

We first consider the performance of individual components in the pipeline approaches. Table 2 shows the performance of JointBERT for intent classification and slot labelling. It can be seen that JointBERT performed better on the insurance domain for both IC and SL and this trend is consistent with the results of the original MultiDoGO paper. For entity linking, we used several evaluation metrics, all of which were only computed when the intents were correctly classified. These include (1) **Link accuracy**: Given that the SL module detects the value of the correct slot type, link accuracy shows how likely the value is linked to the correct entity (including None). (2) **None recall**: The recall of None being predicted. This metric shows how often it can detect when entity mentions cannot be linked. It is also related to the ability of detecting closed type constraint violations. (3) **Precision, Recall, F1**: Considering all the turns in the test data, compare the predicted entities to the ground truth entities. (These metrics are affected by the performance of IC/SL. If the SL module incorrectly detects the slot type, this could cause low precision, recall, and F1 at the entity level here. In contrast, if the SL module does not detect the slot value, no text will be fed to the entity linker and the entity will not be predicted. This could cause low recall but would not affect the precision.)

---

5. [https://github.com/monologg/JointBERT](https://github.com/monologg/JointBERT)
6. [https://pypi.org/project/edit-distance/](https://pypi.org/project/edit-distance/)
7. [https://huggingface.co/transformers/](https://huggingface.co/transformers/)
8. BERT-base makes our models have ~ 110M parameters.
Table 3: Entity linking results (in %). '-' means that the entity linking method does not support ‘open’ entity linking. In other words, it cannot predict None for unlinkable entity mentions.

Table 3 shows the results of entity linking on two MultiDoGO domains. The simplest method, exact match, yielded acceptable results for the fast food domain and surprisingly good results for the insurance domain. This is because possible entities in the insurance domain (with the types car_brand, car_model, and car_year) usually have only one surface form. For example, we can only say “Honda” to refer to the “Honda” car brand entity. Meanwhile, the slot types of the fast food domain are much more flexible such as food_item and ingredient. A user may say only “meatball” or “meatballs” to refer to the “italian meatballs” entity in the bot schema. Besides, the difference between the two domains is partly because the IC/SL model worked better on the insurance domain and provided more accurate slot values to the entity linking step.

Because exact match is a very strict condition, it predicted None more often than other methods and got the highest None recall, while some other methods do not support open entity linking (including Bijaccard, Levenshtein, NLI, and Average) and got zero None recall. However, applying reasonable None thresholds to NLI and Average boosted up the results for all the metrics. The Average method with the threshold of 0.5 achieved the best link accuracy and F1 for both the insurance domain and the fast food domain. Overall, the results highlight that using a combination of methods results in better entity linking performance.

5.4 Violation Detection Results

This section discusses the overall constraint violation detection results with respect to the following metrics. (1) Turn correct: The proportion of the turns where the violation prediction is exactly correct for all constraints. (2) Turn IoU: The IoU score showing how much overlapping the predicted and the ground truth violations of a given turn are, on average. Let $P$ and $G$ be sets of predicted and ground truth violations of a given turn, respectively. IoU (Intersection over Union) of this turn equals $\frac{|P \cap G|}{|P \cup G|}$. (3) Conversation correct: The proportion of conversations where the violation predictions are correct for all the turns. (4) Precision, Recall, and F1: Consider each violation of a constraint as a positive instance, calculate precision, recall, and F1 of the violations being predicted.

As shown in Table 4, the deterministic pipeline approach (DP) obtained the highest turn correct and conversation correct, and also had the best performance in terms of Precision, Recall, and F1. The probabilistic pipeline approach (PP) had a lower performance than the deterministic pipeline approach, but still achieved acceptable results. The end-to-end approach (EE) had the lowest performance among the three approaches, but still achieved some positive results. Overall, the results highlight that using a combination of methods results in better violation detection performance.
<table>
<thead>
<tr>
<th>Method (Threshold)</th>
<th>Insurance</th>
<th>Fast food</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Conver. correct</td>
<td>Turn correct</td>
</tr>
<tr>
<td>Exact match</td>
<td>81.6 89.9 92.4 71.7 62.1</td>
<td></td>
</tr>
<tr>
<td>Bijaccard</td>
<td>74.9 85.6 88.4 39.2 70.6</td>
<td>27.1</td>
</tr>
<tr>
<td>Levenshtein</td>
<td>73.4 84.6 87.8 40.9 63.3</td>
<td>30.2</td>
</tr>
<tr>
<td>NLI</td>
<td>72.8 84.3 87.8 43.6 63.1</td>
<td>33.4</td>
</tr>
<tr>
<td>Average</td>
<td>80.5 89.6 91.9 70.1 62.6</td>
<td>79.6</td>
</tr>
<tr>
<td>Average (0.5)</td>
<td>74.3 85.0 88.2 42.3 67.3</td>
<td>30.8</td>
</tr>
<tr>
<td>Bijaccard</td>
<td>74.1 84.8 88.4 44.6 66.9</td>
<td>33.5</td>
</tr>
<tr>
<td>Levenshtein</td>
<td>73.7 84.6 88.0 44.3 63.8</td>
<td>33.9</td>
</tr>
<tr>
<td>NLI</td>
<td>70.7 83.1 86.8 44.0 58.7</td>
<td>35.2</td>
</tr>
<tr>
<td>NLI (0.8)</td>
<td>70.2 83.8 86.4 60.9 52.6</td>
<td>72.3</td>
</tr>
<tr>
<td>Average</td>
<td>73.7 84.7 88.2 45.1 64.4</td>
<td>34.6</td>
</tr>
<tr>
<td>Average (0.5)</td>
<td>75.4 85.8 89.3 52.5 57.8</td>
<td>48.1</td>
</tr>
<tr>
<td>End-to-End BERT</td>
<td>83.9 92.1 93.4 75.1 76.2</td>
<td>74.1</td>
</tr>
</tbody>
</table>

Table 4: Constraint violation detection results (in %) of the three approaches. For the pipeline approaches, we compare using different entity linking methods and their impact on the constraint violation detection task.

Comparing the deterministic pipeline (DP) with exact match as the entity linking method got the highest violation recall. This is because the exact match is good at detecting un-linkable slot values (see None recall in Table 3), so it got high recall concerning violation detection of closed type constraints. Conversely, entity linking methods which could not predict None (i.e., Bijaccard, Levenshtein, NLI, Average) got significantly lower violation recall and, hence, F1.

Furthermore, the difference between the two domains in Table 4 are more prominent than what we see for individual steps in Table 2-3. There are several reasons for this. First, the fast food domain has more custom slot types and more constraints. So, it is more difficult to predict violations of all the constraints correctly for each turn – resulting in lower conversation correct and turn correct. Second, for the pipeline approaches, the errors of individual steps of the fast food domain were higher than the errors of the insurance domain; therefore, the gap became larger when the errors were accumulated in the last step. An example in Figure 3(A) illustrates this case. The slot labelling part of Joint BERT identified “white top” and “pizza” as two separate food items. The entity linker, “Average (0.5)”, could not map “white top” to any of the defined entity. The system then understood that the user ordered an unknown food item and returned the closed_type_food_item violation which is incorrect. However, we did not see this particular error with the end-to-end approach.

Comparing the probabilistic pipeline (PP) approaches, we can see that DP outperformed PP in most settings, especially in the insurance domain. We believe that when the entity linking module works accurately (as in the insurance domain), switching from DP to PP probably harms the overall performance since PP adds unnecessary uncertainty to the correct entity predictions. Conversely, when entity linking is a challenging step, PP with an appropriate softmax temperature could yield better results.

According to Table 4, the end-to-end approach (EE) clearly outperformed DP and PP in the insurance domain while being competitive to DP and PP in the fast food domain. This might be because the insurance domain has only one domain-specific (binary) constraint and three closed type (unary) constraints that are easier to learn from the training data. Meanwhile, the fast food domain has twelve binary and six unary constraints, respectively. Without access to the constraint statements, the existing training examples may not be sufficient to teach the end-to-end model all possible cases of the constraints. An example in Figure 3(B) shows that EE falsely returned the food_item-ingredient-invalid violation in response to the input “Hai, I need bbq chicken pizza with cheese” although this sentence in fact did not violate the constraint. This error might be because the model had not seen the combination of bbq chicken pizza and cheese during training and it did not have access to the constraints defined in the bot schema.
6 Conclusions and Future Work

Focusing on goal-oriented dialogue systems, we proposed a novel task – slot constraint violation detection – in NLU, together with constraint representation and three approaches to tackle this problem. While the pipeline approaches apply constraints as a post-processing step after IC/SL, the end-to-end approach attempts to model constraints inside the NLU. This sets the stage for future research and modeling of slot constraints and knowledge within NLU. In particular, there are several ways to enhance the end-to-end approach. For example, we could perform joint learning of IC, SL, and constraint violation detection to share the learned knowledge among tasks. Also, injecting logical constraints into BERT is an interesting direction. One way to do so is to translate constraints into violating and non-violating examples (by generating conversations with templates derived from existing training examples) and use them to train BERT together with other training examples. In addition, using constraints information, one can control the training data generation and the percentage of data with constraint violations depending on expected user behavior.

Acknowledgements

We would like to thank Jason Krone, Yi Zhang, Arshit Gupta, and anonymous reviewers for their helpful comments.

References


A Additional Dataset Statistics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Insurance</th>
<th>Fast food</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Train</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of dialogues</td>
<td>1,632</td>
<td>1,448</td>
</tr>
<tr>
<td>Number of user turns</td>
<td>13,821</td>
<td>13,038</td>
</tr>
<tr>
<td>Avg. turns per dialogue</td>
<td>8.47</td>
<td>9.00</td>
</tr>
<tr>
<td># of non-violating dialogues</td>
<td>1,214</td>
<td>457</td>
</tr>
<tr>
<td># of non-violating turns</td>
<td>11,800</td>
<td>6,303</td>
</tr>
<tr>
<td>% of non-violating dialogues</td>
<td>74.39</td>
<td>31.56</td>
</tr>
<tr>
<td>% of non-violating turns</td>
<td>85.38</td>
<td>48.34</td>
</tr>
<tr>
<td><strong>Dev</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of dialogues</td>
<td>233</td>
<td>214</td>
</tr>
<tr>
<td>Number of user turns</td>
<td>1,958</td>
<td>1,868</td>
</tr>
<tr>
<td>Avg. turns per dialogue</td>
<td>8.40</td>
<td>8.73</td>
</tr>
<tr>
<td># of non-violating dialogues</td>
<td>172</td>
<td>79</td>
</tr>
<tr>
<td># of non-violating turns</td>
<td>1,679</td>
<td>961</td>
</tr>
<tr>
<td>% of non-violating dialogues</td>
<td>73.82</td>
<td>36.92</td>
</tr>
<tr>
<td>% of non-violating turns</td>
<td>85.75</td>
<td>51.45</td>
</tr>
<tr>
<td><strong>Test</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of dialogues</td>
<td>467</td>
<td>414</td>
</tr>
<tr>
<td>Number of user turns</td>
<td>3,896</td>
<td>3,767</td>
</tr>
<tr>
<td>Avg. turns per dialogue</td>
<td>8.34</td>
<td>9.10</td>
</tr>
<tr>
<td># of non-violating dialogues</td>
<td>358</td>
<td>145</td>
</tr>
<tr>
<td># of non-violating turns</td>
<td>3,366</td>
<td>1,822</td>
</tr>
<tr>
<td>% of non-violating dialogues</td>
<td>76.66</td>
<td>35.02</td>
</tr>
<tr>
<td>% of non-violating turns</td>
<td>86.40</td>
<td>48.37</td>
</tr>
</tbody>
</table>

Table 5: Additional dataset statistics by data splits (training, development, and test).

B Computing Infrastructure

All experiments was performed on one NVidia V100 GPU having 16 GB of VRAM.

C A Full Conversation Example

Table 6-7 show the violation detection results of a full conversation predicted by the three baseline approaches, together with the intermediate results from the deterministic pipeline approach (including intents, slots, entities, and dialogue states).
<table>
<thead>
<tr>
<th>Turn</th>
<th>Input</th>
<th>Ground truth IC/SL/Entity linking</th>
<th>Pipeline IC/SL/Entity linking</th>
<th>Violations</th>
</tr>
</thead>
</table>
| 0    | hello | **Intent:** OpeningGreeting  
|      |       | Slots: {}  
|      |       | Entities: {}  
|      |       | DS: {} ✓ | GT: {}  
|      |       | DP: {} ✓  
|      |       | PP: {} ✓  
|      |       | EE: {} ✓ | |
| 2    | i would like to order 2 veg out pizza can you help me with that | **Intent:** OrderPizzaIntent  
|      |       | Slots: [quantity: [2], food_item: [veg out pizza]]  
|      |       | Entities: [quantity: [2], food_item: [pizza]]  
|      |       | DS: [closed_drink_item, closed_food_item, food_item-orderdrink] ✓  
|      |       | | GT: {}  
|      |       | DP: {} ✓  
|      |       | PP: {} ✓  
|      |       | EE: {} ✓ | |
| 4    | my address is 1834 eden drive richmond va 23228 and my contact number is 804 913 4348 | **Intent:** ContentOnly  
|      |       | Slots: {}  
|      |       | Entities: {}  
|      |       | DS: [closed_drink_item, food_item-orderdrink] ✓  
|      |       | | GT: {}  
|      |       | DP: {} ✓  
|      |       | PP: {} ✓  
|      |       | EE: {} ✓ | |
| 6    | i would like high rise | **Intent:** OrderPizzaIntent  
|      |       | Slots: [food_item: [high rise]]  
|      |       | Entities: [food_item: [None]]  
|      |       | DS: [closed_drink_item, food_item-orderpizza, closed_drink_item] ✓  
|      |       | | GT: {}  
|      |       | DP: {} ✓  
|      |       | PP: {} ✓  
|      |       | EE: {} ✓ | |
| 8    | yes i want 2 wine | **Intent:** OrderDrinkIntent  
|      |       | Slots: [drunk_item: [red wine]]  
|      |       | Entities: [drunk_item: [red wine]]  
|      |       | DS: [closed_drink_item, closed_drink_item] ✓  
|      |       | | GT: {}  
|      |       | DP: {} ✓  
|      |       | PP: {} ✓  
|      |       | EE: {} ✓ | |
| 10   | i would prefer red wine | **Intent:** OrderDrinkIntent  
|      |       | Slots: [drink_item: [red wine]] ✓  
|      |       | Entities: [drink_item: [red wine]]  
|      |       | DS: [closed_drink_item, closed_drink_item] ✓  
|      |       | | GT: {}  
|      |       | DP: {} ✓  
|      |       | PP: {} ✓  
|      |       | EE: {} ✓ | |

Table 6: A full conversation example for constraint violation detection (1/2). The results in the pipeline IC/SL/Entity linking column are obtained from JointBERT and Average (0.8) entity linking method. DS stands for dialogue states. GT, DP, PP, and EE are violation results of ground truth, the deterministic pipeline approach, the probabilistic pipeline approach, and the end-to-end approach, respectively.
<table>
<thead>
<tr>
<th>Turn</th>
<th>Input</th>
<th>Ground truth IC/SL/Entity linking</th>
<th>Pipeline IC/SL/Entity linking</th>
<th>Violations</th>
</tr>
</thead>
</table>
| 12   | i want 2 s more pie | **Intent**: OrderDessertIntent  
**Slots**: {quantity: [2], food_item: [s more pie]}  
**Entities**: {quantity: [2], food_item: [s more pie]}  
**DS**: {orderpizzaintent: {quantity: [2], food_item: [None]}, orderdrinkintent: {quantity: [2], drink_item: [red wine]}, orderdessertintent: {quantity: [2], food_item: [smore pie]} } | **GT**: [closed_food_item, food_item-orderpizza]  
**DP**: [closed_food_item, food_item-orderpizza, food_item-orderdrink, closed_drink_item]  
**PP**: [closed_food_item, food_item-orderpizza, food_item-orderdrink, closed_drink_item]  
**EE**: [closed_food_item, food_item-orderpizza, food_item-orderdrink] | |
| 14   | nothing else | **Intent**: Rejection  
**Slots**: {}  
**Entities**: {}  
**DS**: {orderpizzaintent: {quantity: [2], food_item: [None]}, orderdrinkintent: {quantity: [2], drink_item: [None], food_item: [None]}, orderdessertintent: {quantity: [2], food_item: [smore pie]} } | **GT**: [closed_food_item, food_item-orderpizza]  
**DP**: [closed_food_item, food_item-orderpizza, food_item-orderdrink, closed_drink_item]  
**PP**: [closed_food_item, food_item-orderpizza, food_item-orderdrink, closed_drink_item]  
**EE**: [closed_food_item, food_item-orderpizza, food_item-orderdrink] | |
| 16   | yes please confirm the order | **Intent**: Confirmation  
**Slots**: {}  
**Entities**: {}  
**DS**: {orderpizzaintent: {quantity: [2], food_item: [None]}, orderdrinkintent: {quantity: [2], drink_item: [red wine], food_item: [None]}, orderdessertintent: {quantity: [2], food_item: [smore pie]} } | **GT**: [closed_food_item, food_item-orderpizza]  
**DP**: [closed_food_item, food_item-orderpizza, food_item-orderdrink, closed_drink_item]  
**PP**: [closed_food_item, food_item-orderpizza, food_item-orderdrink, closed_drink_item]  
**EE**: [closed_food_item, food_item-orderpizza, food_item-orderdrink] | |
| 18   | thank you for your help | **Intent**: ThankYou  
**Slots**: {}  
**Entities**: {}  
**DS**: {orderpizzaintent: {quantity: [2], food_item: [None]}, orderdrinkintent: {quantity: [2], drink_item: [red wine], food_item: [None]}, orderdessertintent: {quantity: [2], food_item: [smore pie]} } | **GT**: [closed_food_item, food_item-orderpizza]  
**DP**: [closed_food_item, food_item-orderpizza, food_item-orderdrink, closed_drink_item]  
**PP**: [closed_food_item, food_item-orderpizza, food_item-orderdrink, closed_drink_item]  
**EE**: [closed_food_item, food_item-orderpizza, food_item-orderdrink] | |

Table 7: A full conversation example for constraint violation detection (2/2). The results in the pipeline IC/SL/Entity linking column are obtained from JointBERT and Average (0.8) entity linking method. DS stands for dialogue states. GT, DP, PP, and EE are violation results of ground truth, the deterministic pipeline approach, the probabilistic pipeline approach, and the end-to-end approach, respectively.