Show, Don’t Tell: Demonstrations Outperform Descriptions for Schema-Guided Task-Oriented Dialogue

Anonymous ACL submission

Abstract

Building universal dialogue systems that can seamlessly operate across multiple domains/APIs and can generalize to new ones with minimal supervision and low maintenance is a critical challenge. Recent works have leveraged natural language descriptions for schema elements to build such systems. However, descriptions only provide indirect supervision for downstream tasks, while still requiring effort to construct. In this work, we propose Show, Don’t Tell, which uses a short labeled example dialogue to show the semantics of a schema rather than telling the model about the schema elements via descriptions. While requiring similar effort from service developers, we show that using short examples as schema representations with large language models results in stronger performance and better generalization on two popular dialogue state tracking benchmarks: the Schema-Guided Dialogue (SGD) dataset and the MultiWoZ leave-one-out benchmark.

1 Introduction

With the widespread adoption of task-oriented dialogue (TOD) systems, these need to support an ever-increasing variety of services/APIs. Since many service developers lack the resources to collect labeled data or the requisite ML expertise, zero/few-shot transfer to unseen services becomes critical to the democratization of dialogue agents.

New approaches to TOD that can generalize to new services mainly rely on combining two techniques: large language models like BERT (Devlin et al., 2019) and T5 (Raffel et al., 2020), and schema-guided modeling i.e. using natural language descriptions of schema elements (intents and slots) as model inputs to enable inference on unseen services (Rastogi et al., 2020a,b). Models combining the two currently show state-of-the-art results on dialogue state tracking (DST) (Heck et al., 2020; Lee et al., 2021a; Anon, 2021).

However, description-based schema representations have drawbacks: precise natural language descriptions still take manual effort and can be tricky to write, while only constituting indirect supervision for unseen services compared to an example dialogue. Furthermore, Lee et al. (2021b) showed that state-of-the-art schema-guided DST models may not be robust to variation in schema descriptions, causing significant accuracy drops.

Alternatively, we propose using a single dialogue example (with final state annotations) in place of the service schema representation, similar to one-shot priming (Brown et al., 2020). Rather than telling the model about schema element semantics in natural language, we aim to show the schema through a demonstration, as in Figure 1. Our approach, “Show, Don’t Tell (SDT),” when applied to two SoTA DST models, offers consistently superior accuracy and generalizes better to new APIs across both the SGD (Rastogi et al., 2020b) and MultiWoZ-Leave-One-Out (Budzianowski et al., 2018; Lin et al., 2021b) benchmarks, while being more data-efficient and robust to schema variations.

2 Show, Don’t Tell

Following SoTA models, we pose DST as a seq2seq task (Wu et al., 2019; Zhao et al., 2021a), where the input language model (in our case, T5) is finetuned on the training set for a DST dataset. During fine-tuning and evaluation, the model input consists of a prompt and context, and the target contains ground truth belief states. We consider two models/prompt formats as our baselines:

- **T5-ind** (Lee et al., 2021a): Model input comprises of the dialogue history as context concatenated with one slot description as the prompt. The target is the value of that slot in the dialogue state. Inference must be done per slot i.e. values for different slots are independently decoded.

- **T5-seq** (Anon, 2021): Model input comprises
the descriptions of all slots as the prompt, followed by the dialogue history as the context. The target is the sequence of slot-value pairs in the dialogue state. In other words, the dialogue state is decoded sequentially in a single pass.

We modify the above prompt formats to include demonstrations instead of descriptions as follows. The new example-based prompt formats are described below and illustrated in Figure 1.

- **SDT-ind**: A prompt $P_{i}^{\text{ind}}$ comprises a single labeled slot value pair for slot $i$ formatted as

  $$P_{i}^{\text{ind}} = [\text{ex}]; d_{i}^{\text{ind}}; [\text{slot}]; sv_{i}$$

  where $d_{i}^{\text{ind}}$ is a single user utterance indicating a value for slot $i$, and $sv_{i}$ is the slot value pair. $[\text{ex}]$, $[\text{slot}]$ are special delimiter tokens.

- **SDT-seq**: A prompt $p_{i}^{\text{seq}}$ comprises a single labeled dialogue turn formatted as:

  $$p_{i}^{\text{seq}} = [\text{ex}]; d_{1}; ..., d_{n}; [\text{slot}]; sv_{1}; ..., sv_{m}$$

  i.e. the prompt is constructed by concatenating all utterances in the example dialogue followed by all slot-value pairs in the final dialogue state.

For all prompt formats (T5-* and SDT-*), we format the values for categorical slots (taking one of a fixed set of values) as a multiple-choice question.

The context in both prompt formats is a concatenation of the dialogue history for the current training example. The final model input is formed by concatenating the prompt and the context strings. The target string is unchanged, containing only the value for the specific slot for independent decoding and the turn belief state for sequential decoding. More details on the prompt design and its impact on performance are provided in Appendix C.

**Formulating prompt examples**: Given neither SDT prompt format contains slot descriptions, it is imperative that the prompt(s) contain enough semantic information to infer values for all slots in the schema. This is easy for SDT-ind, which uses a separate prompt for each slot. However, for SDT-seq, we ensure that the chosen example dialogue contains annotations for all slots in that schema.

### 3 Experimental Setup

**Datasets**: We conduct experiments on two DST benchmarks: Schema-guided Dialogue (SGD) (Rastogi et al., 2020b) and MultiWOZ 2.1 (Budzianowski et al., 2018; Eric et al., 2019). For MultiWOZ, we evaluate on the cross-domain transfer setup from Wu et al. (2019); Lin et al. (2021a), where models are trained on all domains but one and evaluated on the holdout domain. For SGD, we created prompt dialogues manually, with 5.9 turns on average, compared to 15.3 average turns in the SGD single-domain dataset. For MultiWoZ, we selected a short dialogue containing all slots from each holdout domain’s training set for the prompt.

**Implementation**: We train SDT models by fine-tuning pretrained T5 1.1 checkpoints of various sizes. For both datasets, we select one example prompt per service schema (for SDT-seq) or slot (for SDT-ind), and use the same prompt for all examples for that service/slot across training and evaluation. Unless otherwise noted, all T5-based models (T5/SDT-seq/ind) are finetuned on T5-XXL (11B parameters). Appendices A and B have more details on training and baselines respectively.
Table 1: SGD test set JGA for SDT versus other approaches. *Data augmentation/special rules applied.

<table>
<thead>
<tr>
<th>Model</th>
<th>All</th>
<th>Seen</th>
<th>Unseen</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRC+WD-DST*</td>
<td>86.5</td>
<td>92.4</td>
<td>84.6</td>
</tr>
<tr>
<td>T5-seq</td>
<td>86.4</td>
<td>95.8</td>
<td>83.3</td>
</tr>
<tr>
<td>T5-ind</td>
<td>87.7</td>
<td>95.3</td>
<td>85.2</td>
</tr>
<tr>
<td>SDT-ind</td>
<td>87.5±0.6</td>
<td>95.1±0.5</td>
<td>85.0±0.9</td>
</tr>
<tr>
<td>SDT-seq</td>
<td>88.8±0.5</td>
<td>95.8±0.2</td>
<td>86.4±0.7</td>
</tr>
</tbody>
</table>

Table 2: Cross-domain (leave-one-out) JGA on MultiWOZ 2.1. Results for TRADE, SUMBT, and TransferQA from (Kumar et al., 2020), (Campagna et al., 2020), and (Lin et al., 2021a), respectively.

<table>
<thead>
<tr>
<th>Model</th>
<th>Attraction</th>
<th>Hotel</th>
<th>Restaurant</th>
<th>Taxi</th>
<th>Train</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRADE</td>
<td>20.1</td>
<td>14.2</td>
<td>12.6</td>
<td>59.2</td>
<td>22.4</td>
<td>25.7</td>
</tr>
<tr>
<td>SUMBT</td>
<td>22.6</td>
<td>19.8</td>
<td>16.5</td>
<td>59.5</td>
<td>22.5</td>
<td>28.2</td>
</tr>
<tr>
<td>TransferQA</td>
<td>31.3</td>
<td>22.7</td>
<td>26.3</td>
<td>61.9</td>
<td>36.7</td>
<td>35.8</td>
</tr>
<tr>
<td>T5-seq</td>
<td>76.4</td>
<td>26.1</td>
<td>74.9</td>
<td>85.9</td>
<td>64.6</td>
<td>65.6</td>
</tr>
<tr>
<td>SDT-seq</td>
<td>75.0</td>
<td>32.0</td>
<td>73.1</td>
<td>86.6</td>
<td>77.5</td>
<td>68.8</td>
</tr>
</tbody>
</table>

Table 3: SGD test JGA across T5 model sizes.

<table>
<thead>
<tr>
<th>Model</th>
<th>10-shot</th>
<th>1%</th>
<th>10%</th>
</tr>
</thead>
<tbody>
<tr>
<td>T5-seq</td>
<td>79.2</td>
<td>80.0</td>
<td>86.4</td>
</tr>
<tr>
<td>T5-ind</td>
<td>76.3</td>
<td>82.2</td>
<td>87.7</td>
</tr>
<tr>
<td>SDT-seq</td>
<td>76.3±1.1</td>
<td>83.2±0.4</td>
<td>88.8±0.5</td>
</tr>
</tbody>
</table>

Table 4: Data efficiency experiments on SGD test set.

<table>
<thead>
<tr>
<th>Model</th>
<th>10-shot</th>
<th>1%</th>
<th>10%</th>
</tr>
</thead>
<tbody>
<tr>
<td>T5-seq</td>
<td>51.0</td>
<td>79.4</td>
<td>83.0</td>
</tr>
<tr>
<td>SDT-seq</td>
<td>70.7</td>
<td>84.5</td>
<td>87.4</td>
</tr>
</tbody>
</table>

Table 5 shows SDT-seq achieves the highest average JGA ($JGA_{v1-5}$) and lowest schema sensitivity ($SS_{JGA}$), indicating it performs best for the remaining 2 domains.
Example Dialogue Segment

<table>
<thead>
<tr>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. T5-seq confused between similar slots</td>
</tr>
<tr>
<td>I need to find train tickets to Anaheim, CA. When would you like to travel, and where are you going to? Traveling to Sacramento on the 4th.</td>
</tr>
<tr>
<td>2. Slot values appearing as in context</td>
</tr>
<tr>
<td>Can you please add an alarm called Grocery run.</td>
</tr>
<tr>
<td>3. Categorical values not seen in context</td>
</tr>
<tr>
<td>I like Broadway shows and want to see one on Tuesday next week.</td>
</tr>
</tbody>
</table>

Figure 2: Examples of common error patterns made by SDT-seq compared to T5-seq.

is the most robust of the compared models. Note, however, that the JGA drop indicates SDT-seq is still sensitive to slot name variations.

<table>
<thead>
<tr>
<th>Model</th>
<th>JGA_{Orig}</th>
<th>JGA_{V_w,v_s}</th>
<th>Diff_{rel}</th>
<th>SS_{JGA}</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGP-DST*</td>
<td>60.5</td>
<td>60.5</td>
<td>0.0</td>
<td>51.9</td>
</tr>
<tr>
<td>T5-ind_{base}*</td>
<td>72.6</td>
<td>72.6</td>
<td>0.0</td>
<td>40.4</td>
</tr>
<tr>
<td>T5-seq</td>
<td>86.4</td>
<td>86.4</td>
<td>0.0</td>
<td>27.0</td>
</tr>
<tr>
<td>SDT-seq</td>
<td>88.8</td>
<td>88.8</td>
<td>0.0</td>
<td>24.1</td>
</tr>
</tbody>
</table>

Table 5: Robustness evaluation on the SGD-X test sets. *Results from Lee et al. (2021b).

5 Discussion

5.1 Writing descriptions vs. demonstrations

We note that the information provided to SDT is not identical to what is provided to usual schema-guided models, as SDT trades out natural language descriptions in exchange for a demonstration of how to identify slots in a dialogue. However, we argue that from a developer’s point of view, creating a single example is a similar amount of effort as writing descriptions, so we consider the methods to be comparable. For creating the SDT-seq prompts for each service in SGD, an experienced annotator took ~2 hours, compared to ~90 minutes for generating descriptions for all slots in all services. SDT-ind prompts are arguably even simpler to write.

Descriptions, however, have their advantages: they are agnostic to the model architecture and writing them does not require knowledge of dialogue systems, which SDT-seq prompts does. Given the performance gain, however, example-based prompts may be a better choice for many settings, especially for smaller model sizes.

5.2 Error analysis

Figure 2 contains some common error patterns in predictions from T5-seq and SDT-seq. These indicate that SDT benefits from having a better understanding of the context around unseen slots, or when slot descriptions are too similar to one another (#1). However, SDT can be limited by its prompt: for instance, in #3 it has only seen context for the other categorical value for slot event_type.

6 Related Work

Prior approaches focused on framing DST as question answering (Ruan et al., 2020; Ma et al., 2020; Zhang et al., 2021). Many MultiWoZ cross-domain models leverage slot names/descriptions (Wu et al., 2019; Lee et al., 2019; Lin et al., 2021a).

Pretrained generative LLMs (Raffel et al., 2020; Brown et al., 2020) have enable framing NLP tasks as seq2seq problems. Some DST papers (Zhao et al., 2021a; Feng et al., 2021) look at settings with no train-test discrepancy. Many studies explore the efficacy of task-specific prompts (Jiang et al., 2020; Liu et al., 2021). Madotto et al. (2020) prime LMs with examples for dialogue tasks, but without finetuning. Wei et al. (2021) fine-tune language models to understand prompts for a different task.

7 Conclusion

We study the use of demonstrations as LM prompts to convey the semantics of APIs in lieu of natural language descriptions for TOD. Even though they take similar effort to construct, they outperform description-based prompts in our experiments across DST datasets (SGD and MultiWOZ), model sizes, and training data sizes, while being more robust to changes in schemata. This work provides developers of TOD systems with more options for API representations to enable transfer to unseen services. In future work, we would like to explore this representation for other TOD tasks (e.g. dialogue management and response generation).
8 Ethical Considerations

We proposed a more efficient way of building TOD systems by leveraging demonstrations in place of descriptions, leading to increased accuracy with minimal/no data preparation overhead. We conduct our experiments on publicly-available TOD datasets in English, covering domains which are popular for building conversational agents for. We are hopeful our work leads to building more accurate TOD systems with similar or less overhead, and encourages further research in the area.

References


Zhengbao Jiang, Frank F. Xu, Jun Araki, and Graham Neubig. 2020. How can we know what language models know?


Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhihui Yang, and Jie Tang. 2021. Gpt understands, too.

Yue Ma, Zengfeng Zeng, Dawei Zhu, Xuan Li, Yiyang Yang, Xiaoyuan Yao, Kajjie Zhou, and Jianping Shen. 2020. An end-to-end dialogue state tracking system with machine reading comprehension and wide deep classification.


Laria Reynolds and Kyle McDonell. 2021. Prompt programming for large language models: Beyond the few-shot paradigm.


A SDT Model Details

All T5 checkpoints used are available publicly\(^1\). For all experiments, we use a sequence length of 2048, dropout of 10\% and a batch size of 16. We used a constant learning rate of $1e^{-3}$ or $1e^{-4}$. All models were trained for 50k steps or until convergence, and each experiment was conducted on either 64 or 128 TPU v3 chips (Jouppi et al., 2017).

B Baseline Models

For SGD, we compare against SGP-DST (Ruan et al., 2020), MRC+WD-DST (Ma et al., 2020), T5-seq (Anon, 2021) and T5-ind (Lee et al., 2021a).

For MultiWOZ, we compare against TRADE (Wu et al., 2019), SUMBT (Lee et al., 2019), TransferQA (Lin et al., 2021a), and T5-seq.

Transfer QA is based on T5-large, and T5-ind and T5-seq are based on T5-XXL in this paper unless otherwise noted.

C Prompt Design

We experimented with various formats for the SDT prompt before arriving at the final format. Below, we list alternative designs that we tried and their impact on JGA, as evaluated on the SGD test set.

C.1 Categorical value strings vs. multiple choice answers

We found that JGA dropped -2\% when we tasked the model with decoding categorical values instead of multiple choice answers - e.g. payment\_method=debit card instead of payment\_method=b (where b is linked to the value debit\_card in the prompt as described in Section 2). We found that when tasking the model to decode categorical values, it would often decode related yet invalid values, which we counted as false in our evaluation. For example, instead

\footnote{\url{https://github.com/google-research/text-to-text-transfer-transformer/blob/main/released_checkpoints.md}}
of debit card, the model might decode bank balance.

C.2 Slot IDs vs. slot names

When we delexicalized slot names with slot IDs, JGA dropped -5%. One downside of this approach is that the model lost access to valuable semantic information conveyed by the slot name. Another downside is that the model could not distinguish two slots that had the same value in the prompt. For example, if the prompt was "I would like a pet-friendly hotel room with wifi" and the corresponding slots were 1=True (has_wifi) and 2=True (Pets_allowed), it is ambiguous which ID refers to which slot.

The potential upside of using slot IDs was to remove dependence on the choice of slot name, but this did not succeed for the reasons above.

C.3 Decoding active slots vs. all slots

We experimented with training the model to only decode active slots rather than all slots with none values when they were inactive. JGA dropped -0.4%, which we hypothesized could be a result of greater dissimilarity between the slot-value string in the prompt (which contained all slots by construction) and the target, which only contained a subset of slots.

C.4 In-line annotations vs. dialogue+slots concatenated

We hypothesized that bringing the slot annotation in the prompt closer to where it was mentioned in the dialogue might help the model better understand the slot’s semantic meaning. We changed the format as follows:

- Original: [example] [user] I would like a pet-friendly hotel room with wifi [system] I found ... [slot] has_wifi=True
- In-line: [example] [user] I would like a pet-friendly hotel room with wifi [has_wifi=True] [system] I found ...

However, this decreased JGA by more than -20%. We hypothesized that this was likely due to a mismatch between the prompt’s annotations and the target string format, which remained the same.