Unsupervised End-to-End Task-Oriented Dialogue with LLMs: The Power of the Noisy Channel

Brendan King and **Jeffrey Flanigan**
University of California, Santa Cruz
{bking2,jmflanig}@ucsc.edu

Abstract

Training task-oriented dialogue systems typically requires turn-level annotations for interacting with their APIs: e.g. a dialogue state and the system actions taken at each step. These annotations can be costly to produce, error-prone, and require both domain and annotation expertise. With advances in LLMs, we hypothesize that unlabeled data and a schema definition are sufficient for building a working task-oriented dialogue system, completely unsupervised. We consider a novel unsupervised setting of only (1) a well-defined API schema (2) a set of unlabeled dialogues between a user and agent. We propose an innovative approach using expectation-maximization (EM) that infers turn-level annotations as latent variables using a noisy channel model to build an end-to-end dialogue agent. Evaluating our approach on the MultiWOZ benchmark, our method more than doubles the dialogue success rate of a strong GPT-3.5 baseline.[1]

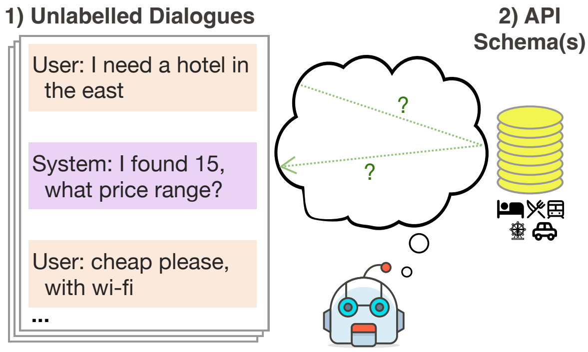

Figure 1: An overview of our unsupervised dialogue problem. We assume 1) unlabeled goal-oriented dialogues between a user and agent and 2) a well-defined schema $\mathcal{S}$ with APIs suitable for fulfilling goals. We infer the unseen interactions between the agent and API, and use this to produce an end-to-end dialogue agent.

1 Introduction

Task-oriented dialogue systems, which use APIs to complete tasks on behalf of users, have been a longstanding challenge within conversational AI. Recent advances in large language models (LLMs) have further stimulated interest in task-oriented systems and LLMs which can use APIs as tools. To facilitate API use, successful task-oriented dialogue systems usually employ a modular approach: predicting a dialogue state which includes arguments to API calls, and dialogue acts for planning an appropriate response, before finally producing a natural language reply. Training such systems typically requires expert annotation of these structured intermediates for every dialogue turn. Even in settings where human-human dialogues are abundantly available, the high cost and expertise required to annotate the dialogues poses a significant hurdle to system development.

Recent work has shown that LLMs can accomplish a broad set of useful tasks without any structured labels for a task (Brown et al., 2020). These include 'zero-shot' approaches to task-oriented dialogue sub-tasks such as Dialogue State Tracking (DST) (Hu et al., 2022; King and Flanigan, 2023; Heck et al., 2023), intent detection (Pan et al., 2023), grounded response generation (Li et al., 2023b), and even zero-shot end-to-end dialogue systems (Hudeček and Dusek, 2023). Still, existing approaches generally do not perform well enough for real-world use, and none are able to make effective use of in-domain unlabeled dialogues.

In this work, we ask whether we can use existing unlabeled dialogues (without any labels or API calls annotated) along with an API specification, to build a working dialogue agent, without needing an expert to annotate data. This addresses a common real-world scenario. Many high value dialogue tasks are currently carried out by human agents, who interface a user with some software system. These conversations can be recorded and

[1] Our code is available at https://github.com/jlab-nlp/nc_latent_tod

Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pages 8283–8300
November 12-16, 2024

transcribed, and the API(s) supporting the agent typically have well-formed specifications. However, annotating the API calls and system acts needed for aligning the two is time consuming and requires annotation expertise. In lieu of this, 'zero-shot' systems have been proposed, but these still require an expert to annotate a 'formatting example' (Hu et al., 2022; King and Flanigan, 2023), or a more detailed 'policy skeleton' (Zhang et al., 2023).

We instead propose the following setting: we assume an API schema definition $\mathcal{S}$, and plenty of available human-human dialogues in natural language, but no annotations on these dialogues (Figure 1). We demonstrate that one can develop a conversational agent for the API schema in this setting without any assistance from an expert annotator. Our contributions are as follows:

- We construct an end-to-end task-oriented dialogue agent with an LLM solely from unlabeled dialogues and an API definition, without any turn-level labels or supervision from de-lexicalized utterances. To the best of our knowledge, we are the first to consider this setting.

- We train our dialogue system by inferring all the pseudo-labels necessary (API calls, system actions) to train a traditional end-to-end dialogue system from unlabeled dialogues, using prompts which are automatically generated from the API schema.

- To improve the inferred labels, we devise a novel Hard-EM (Dempster et al., 1977) approach which uses predictions as in-context examples for the LLM and as data for iteratively fine-tuning a final model.

- We propose a noisy-channel 'code-to-text' reranking approach, which is instrumental to our method and greatly improves our pseudo-label quality and final system.

2 Preliminaries

A task-oriented dialogue consists of turns of utterances between a user and an agent which interfaces the user with a programmable system or API to accomplish a task. Typically the system response utterance follows the user's utterance. We denote u_t as the user's utterance at turn t, and r_t as the system's response. We assume the APIs supported by the system are defined in a schema $\mathcal{S}$, which gives names and descriptions for all arguments supported in each API, as well as the possible values any categorical arguments may take (Rastogi et al., 2020). This is analogous to standardized formats for API documentation, many of which could be easily converted to a schema definition.

Task-oriented systems require some method for interacting with the APIs in $\mathcal{S}$. Modular approaches use a Dialogue State Tracking (DST) module, which predicts a belief state b_t: a collection of arguments to API call(s) needed to satisfy the user's goal. A belief state is commonly represented with a set of slot-value pairs:

$$b_t = \{(s_1, v_1), (s_2, v_2), ...(s_n, v_n)\}$$

For example, if a user says 'I'm looking for a restaurant south of town', a DST system might produce the belief state {(restaurant-area, south)}, which can be used to query a restaurant API. We assume zero labeled belief states and infer them from unlabeled dialogues using the space of possible states supported by the schema definition $\mathcal{S}$.

We also make use of system dialogue acts to structure our agent's communicative intents with a policy module. Given a dialogue state and context for a turn t, the policy predicts set of dialogue acts to be communicated in the system response r_t. For instance, the policy might determine that we should ask the user to narrow their search to a price range: A_t = {Request(restaurant-pricerange=?)}. An appropriate system response might be: "Sure, are you looking for a particular price range?" Like belief states, we assume zero supervised examples of A_t and infer them from unlabeled dialogues.

3 Method Overview

We treat the turn-level labels needed for training an end-to-end dialogue system as latent variables, and infer them from unlabeled dialogues. We assume only the fully-lexicalized sequence of user and system utterances $u_1, r_1, ...u_T, r_T$, and the schema $\mathcal{S}$ defining the system's capabilities, which defines the space of valid dialogue state and act labels. Importantly, our prompts are automatically generated from the API schema.

In §4, we outline our noisy-channel prompting method for inferring the turn-level labels necessary for training our dialogue agent. We give an overview of the latent variables we infer in Figure 2. We assume we cannot query the APIs or observe

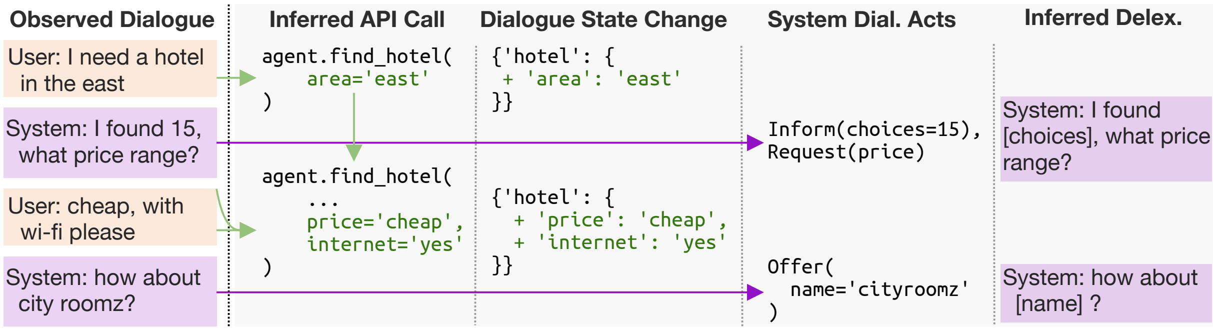

Figure 2: An overview of the latent variables annotated in our unsupervised labeling process which are used to train the dialogue model. Our DST Module (§4.1) infers the API call(s) with arguments at each turn, from which we can derive the dialogue state change. Our DAT or Act Tagging module (§4.2) predicts the dialogue acts communicated in the observed system response, which can be used to infer de-lexicalized responses for training a response generator.

results while labeling dialogues offline, as the obtained API results may have changed. In § 5, we train a complete dialogue agent by fine-tuning on prompts derived from our inferred pseudo-labels.

4 Inferring Latents via Noisy Channel

In this section, we present our method for inferring latent annotations for the dialogue states $b_1...b_T$ and dialogue acts $A_1...A_T$ for each dialogue turn t given only the unlabeled user and system utterances $(u_1, r_1, u_2, r_2, ...u_T, r_T)$. To do this, we devise a noisy-channel prompting approach for DST and dialogue act tagging (DAT) using StarCoder (Li et al., 2023a), a code-based LLM. First, we use a text-to-code prompt to infer the API call(s) made by the system in each dialogue, and build the dialogue state from inferred API call arguments (§ 4.1). We use a similar text-to-code prompt to infer the latent act(s) communicated in each agent response, so that we can reverse-engineer an agent's policy (§ 4.2). For both tasks, we find much better performance when re-ranking latent predictions according to a noisy-channel model, in which we condition the observed utterance on a predicted latent in a code-to-text prompt (§ 4.3). Finally, we leverage the in-context learning ability of LLMs by re-using our predictions as exemplars (§ 4.4). Given these initial pseudo-labels, we iteratively improve their quality using Hard-EM (Dempster et al., 1977) (§ 4.5).

4.1 Inferring API Calls and Dialogue State

We prompt the LLM with a text-to-code prompt for inferring the latent dialogue state as an API call. Figure 6a in Appendix A gives an example of our prompt. We generate a prompt enumerating the intents available in the schema $\mathcal{S}$ as APIs callable by our agent. Following Hu et al. (2022), we predict the appropriate function call conditioned on the prior system response r_{t-1}, the current user utterance u_t, and the previous belief state prediction $\hat{b}_{t-1}$. We then extract a dialogue state *change* $\Delta\hat{b}_t$ from the arguments to the call, and compute the next dialogue state as $\hat{b}_t = \Delta\hat{b}_t + \hat{b}_{t-1}$. While used offline here, this DST method is causal with respect to dialogue inputs and is the same as our method in online inference.

4.2 Inferring System Acts

For inferring system acts, we use a similar text-to-code prompt for predicting the set of dialogue acts A_t communicated in a given system response r_t. See Figure 6b in Appendix A for an example of our prompt. We define each act our system could take in the prompt instructions. For input from each turn, we find best performance when conditioning only on the response to tag, r_t. For our set of supported acts, we use a subset of the universal dialogue acts proposed in Paul et al. (2019), where some acts such as "Inform" or "Offer" may use slots defined in $\mathcal{S}$. For example, an agent choosing to offer to book a user at a hotel named 'acorn guest house' might be represented as Offer(hotel_name='acorn guest house'). See Appendix C for our complete dialogue act set. Importantly, we use the schema definition $\mathcal{S}$ and our act set to validate each act prediction, removing predicted keys which do not belong to $\mathcal{S}$, or acts which are not in the set. For example, the 'text' key is not valid for a 'ThankYou' act, so a prediction of "ThankYou(text='thanks, have a good day')" would be normalized to only "ThankYou()". Using the inferred system acts, we use a rule-based method to delexicalize the system responses for training the response generator (Figure 2, right).

4.3 Noisy Channel LLM Prompting

We find that a noisy channel prompting method (Min et al., 2022) significantly improves the quality of our inferred dialogue states and acts. Here we describe noisy channel prompting using a simple example, and then describe its application to dialogue state tracking and system act tagging.

A typical prompt for machine reading comprehension might be:

```
<Optional in-context examples (c)>
Passage: <Passage (z)>
Question: <Question (x)>
Answer:
```

Given this prompt of the in-context examples c, passage z, question x, an answer y completion is found with the language model by maximizing or sampling from $Pr(y|x, z, c)$. We call this the **direct prompt**.

The **"noisy channel" prompt** is:

```
<Optional in-context examples (c)>
Passage: <Passage (z)>
Answer: <Answer (y)>
Question: <Question (x)>
```

where the likelihood of the question now depends on the answer. To use the noisy channel LLM prompt, we first sample k samples from the direct prompt, and then pick the best output answer y according to the noisy channel prompt probability. One can choose to score the joint probability of the answer followed by the question, i.e. $Pr(x|y, z, c)Pr(y|z, c)$, or only the conditional $Pr(x|y, z, c)$, following Min et al. (2022).[2]

To apply this method to inferring dialogue states, we first sample a set of possible belief state changes using top-p sampling (Holtzman et al., 2020) from the direct DST prompt, and then pick the best dialogue state according to the noisy channel prompt (see Figure 3). We use an analogous procedure for inferring system acts. For DST, we find scoring with the joint $Pr(x|y, z, c)Pr(y|z, c)$ to perform best, and scoring with the conditional $Pr(x|y, z, c)$ best for act tagging.

4.4 Retrieval-Augmented In-context Learning

To leverage the in-context learning abilities of LLMs, we retrieve from a pool of examples from our predictions. Because we assume no labeled examples, this pool starts with zero examples and is filled incrementally. We retrieve up to k examples

Direct DST Prompt

```
response = agent.handle_turn(
    belief_state=BeliefState(attraction=dict(
                             name='byard art')),
    last_system_utterance="byard art is at 344 oxford " + \
                          "street, anything else?",
    user_utterance="Yes, I need a taxi to king station",
    user_intent=[agent.book_taxi(destination='king station')]
```

Noisy Channel DST Prompt

```
response = agent.handle_turn(
    belief_state=BeliefState(attraction=dict(
                             name='byard art')),
    last_system_utterance="byard art is at 344 oxford " + \
                          "street, anything else?",
    user_intent=[agent.book_taxi(destination='king station')],
    user_utterance="Yes, I need a taxi to king station",
```

Figure 3: Instances from our 'direct' and 'noisy channel' prompts for DST. Best viewed in color. After sampling a ***DST completion*** from the 'direct' prompt, we score it by the likelihood of the input user utterance conditioned on it in the 'noisy channel' prompt.

for in-context learning from this pool using an unsupervised dense retriever, with examples ranked by embedding cosine distance.[3] We use $k = 8$ and $k = 6$ for DST, DAT respectively. For retriever inputs, we use $(\hat{b}_{t-1} \cdot r_{t-1} \cdot u_t)$ and $(u_t \cdot r_t)$ for DST and DAT respectively, where $\cdot$ indicates concatenation. Applied naively, this in-context learning approach can suffer a majority label bias (Zhao et al., 2021). We adjust for biases introduced in the initially small example pool by 1) not using any in-context examples until we have a minimum of $n = 32$ examples in the pool and 2) using our API schema $\mathcal{S}$ to require at least 4 distinct labels in each set of in-context examples.[4] Our algorithm for producing initial pseudo-labels is in Appendix D.

4.5 Refining the Labels with Hard-EM

While the labels we produce in § 4.1-§ 4.4 can be used directly for training an end-to-end dialogue system, we find their quality can be improved through expectation-maximization (Dempster et al., 1977). For every dialogue turn in our dataset, our initial pseudo-labels provide the expected dialogue state and system dialogue acts according to our zero-shot system. We then jointly fine-tune an LLM as a noisy-channel DST & DAT system to maximize the likelihood of these expected labels. We use smaller version of our prompted LLM, StarCoder 3B (Li et al., 2023a).

For each turn, we derive (prompt, completion) pairs for 'direct' text-to-code and 'channel' code-

[2] In the latter case, the prior $Pr(y|z, c)$ is uniformly $\frac{1}{k}$ for the k samples from the direct prompt.

[3] We use MPNet (Song et al., 2020), available on Huggingface as `sentence-transformers/all-mpnet-base-v2`

[4] We consider two dialogue state change labels to be distinct if they update different *slots*, and two act labels to be distinct if they embody different acts or different slots

to-text DST and DAT modules, as defined in § 4. We then combine and shuffle these pairs into a single training set for joint fine-tuning. For efficient training, we shorten our prompts by removing in-context examples as well as the function definitions used in the in-context learning setting. We find up-sampling the 'channel' prompts so that there is a 2:1 ratio of 'channel' to 'direct' instances for training improves performance.

After fine-tuning, the model can be used to produce improved pseudo-labels by re-labeling each dialogue, using the same noisy-channel inference methods. Following this, we can repeat the fine-tuning process. This train and re-label process can be repeated for any number of iterations, though we find a single re-labeling is sufficient.

5 End-to-End System

Following (Su et al., 2022), we utilize a multi-task fine-tuning method for training a single LLM as a complete dialogue system, consisting of a dialogue state tracker, policy, and response generator.

DST For the DST sub-task, we again use both 'direct' and 'channel' (prompt, completion) pairs. This allows us to use the same noisy-channel inference method presented in §4.

Policy For the Policy sub-task, we use a text-to-code prompt where we simply condition on the k=5 most recent utterances in the dialogue history: $H_t = (u_{t-2}, r_{t-2}, u_{t-1}, r_{t-1}, u_t)$. The completion is the current turn's system acts A_t, which will be used to ground the next response r_t. We do not use a noisy-channel variant for Policy, and greedily decode an act prediction at inference time:

$$\hat{A}_t = \underset{A_t \in \mathcal{V}^*}{\operatorname{argmax}} \; P(f_{\text{prompt}}(H_t)))$$

Response Generation For Response Generation, we condition on the turn's observed system and user utterances (r_{t-1}, u_t) and our policy's act prediction $\hat{A}_t$). The completion is the observed system response r_t. We also do not use a noisy-channel variant for response generation, and greedily decode the response:

$$\hat{r}_t = \underset{A_t \in \mathcal{V}^*}{\operatorname{argmax}} \; P(f_{\text{prompt}}(r_{t-1}, u_t, A_t)))$$

Following prior works, we predict *delexicalized* responses, where values for slots in the system response are replaced with placeholders for the slot name. For example, instead of generating "The phone number for acorn guest house is 555-5309" directly, we would predict "The phone number for the [hotel_name] is [hotel_phone]", where values could be filled in. Importantly, we never presume access to gold delexicalized responses. Instead, we use our predicted acts, e.g. "Inform(name='acorn guest house', phone='555-8309')" to delexicalize the observed response for training.

End-to-end Training Our approach to end-to-end training is as follows. We first derive (prompt, completion) pairs for each module (DST, Policy and Response Generation), including 'channel' prompts for DST. To improve training efficiency, we shorten each prompt by removing in-context examples and function definitions from the in-context learning setting. We then combine and shuffle these pairs into a single training set for joint fine-tuning and up-sample the 'channel' prompts for DST using the same 2:1 ratio. Using this training set, we fine-tune StarCoder 3B using cross-entropy loss and AdamW with default hyperparameters.

6 Experiments

We conduct unsupervised end-to-end dialogue (E2E) and dialogue state tracking (DST) experiments on the MultiWOZ 2.2 dataset (Zang et al., 2020; Budzianowski et al., 2018), containing over ten thousand multi-domain task-oriented dialogues crowd-sourced in a wizard-of-oz setup. We use the fully lexicalized, unlabeled dialogues from the training set to build our system, and evaluate on the test set. First, we demonstrate the value of our approach in an end-to-end dialogue evaluation, following prior works on task-oriented dialogue (§ 6.1). Second, we conduct a dialogue state tracking evaluation to more carefully evaluate the quality of our pseudo-annotations (§6.2).

6.1 End-to-End (E2E) Experiments

In E2E experiments, we use our complete system to both predict API call arguments and generate a next system response in natural language. We evaluate our generated responses with Inform rate, Success rate, and BLEU, as well as a Combined score of $0.5(\text{Inform} + \text{Success}) + BLEU$, following prior works (Budzianowski et al., 2018; Nekvinda and Dušek, 2021). We provide details on these metrics in Appendix B.

We compare our approach to the previous state-of-the-art unsupervised methods, a GPT-3.5 zero-

Model	Schema?	Labels?	Dialogues?	Inform	Success	BLEU	Combined
Supervised Results							
PPTOD (Su et al., 2022)	✓	✓	✓	82.6	72.2	18.2	95.6
DiactTOD (Wu et al., 2023)	✓	✓	✓	89.5	84.2	17.5	104.4
Our (supervised)	✓	✓	✓	67.9	61.7	14.6	79.4
Zero-Shot with Formatting Example(s)							
SGP-TOD-GPT3.5 (Zhang et al., 2023)	✓	Few (‡)	✗	82.0	72.5	9.22	86.5
Fully Unsupervised Results							
Sees gold delexicalized conversation history							
LLaMa†	✓	✗	✗	-	4	1.61	-
GPT 3.5 Turbo†	✓	✗	✗	44.8	31.2	3.3	41.3
Sees only fully-lexicalized dialogues							
GPT 3.5 Turbo (− gold delex.)	✓	✗	✗	40.7	26.7	3.7	37.4
Ours (StarCoder 15B - no EM)	✓	✗	✗	50.0	19.6	3.2	38
Ours (StarCoder 3B - w/ EM)	✓	✗	✓	**78.1**	**68.3**	**13.6**	**86.8**

Table 1: Unsupervised end-to-end results in MultiWOZ 2.2. (†) indicates models from Hudeček and Dusek (2023). Results for LLaMa are from Hudeček and Dusek (2023), which does not report the Inform rate. (‡) SGP-TOD uses a prompt with both a formatting example and a "Policy Skeleton", which contains an additional 10-20 hand-crafted instances of the correct system acts and response for an input user utterance or returned DB result. For fairer comparison in our fully unsupervised setting, we re-run the GPT 3.5 baseline without the supervision of de-lexicalized responses provided in the conversation history (− gold delex.). Despite far fewer parameters, we find substantial improvements in our methods which leverage unlabeled dialogues

shot baseline (Hudeček and Dusek, 2023), and SGP-TOD (Zhang et al., 2023). Where possible, we report results for both the original approach and modifications required to fit our fully unsupervised setting. For reference, we also run our own method in the fully-supervised setting. We train a model using the procedure in § 5 using the annotations sourced from crowd-workers in the MultiWOZ 2.2 corpus (Budzianowski et al., 2018; Zang et al., 2020), rather than the pseudo-labels predicted in § 4. We also compare with existing supervised approaches as a reference point. We include DiactTOD (Wu et al., 2023), which to our knowledge is the supervised state-of-the-art, and PPTOD (Su et al., 2022), which uses a multi-task fine-tuning approach similar to our own in § 5, for T5 encoder-decoder models (Raffel et al., 2020).

6.2 DST Experiments

We conduct multi-domain DST experiments on the MultiWOZ Dataset in order to evaluate the quality of our pseudo-annotations. We use our DST Module to predict and evaluate only latent dialogue states, which collect the arguments required for unseen API calls.

Following prior works, we evaluate DST performance with joint-goal accuracy (JGA), or whether a given dialogue state is completely accurate. More details are available in Appendix B.

We compare to our ChatGPT 3.5 Turbo baseline (Hudeček and Dusek, 2023), as well as prior

With One Formatting Example	
IC-DST (StarCoder 15B)	24.58
RefPyDST (StarCoder 15B)	17.17
IC-DST (Codex)	35.02
RefPyDST (Codex)	40.88
Fully Unsupervised	
IC-DST (StarCoder 15B)	15.66
RefPyDST (StarCoder 15B)	13.88
GPT 3.5 Turbo (Hudeček and Dusek, 2023)	13.05
Ours (StarCoder 15B → 3B)	**39.70**

Table 2: Joint Goal Accuracy (JGA) of our method's dialogue state predictions and zero-shot baselines. 15B → 3B indicates our approach uses StarCoder 15B to compute initial labels (§ 4.1 - § 4.4) and StarCoder 3B when iteratively fine-tuning and re-labelling (§ 4.5), for a final model size of 3B

zero-shot DST methods. These include IC-DST (Hu et al., 2022), which re-frames DST as text-to-SQL, and RefPyDST which re-frames DST as text-to-python (King and Flanigan, 2023). By default, both of these works use OpenAI Codex (Chen et al., 2021) which is now unavailable. We apply their prompting approaches to StarCoder 15B for a clearer comparison.

7 Results

7.1 E2E Performance

We present E2E results for our unsupervised dialogue agent in Table 1. We find that our method achieves state-of-the-art performance in our fully unsupervised setting, more than doubling the Success Rate and Combined score of the GPT 3.5 Turbo baseline of Hudeček and Dusek (2023). When we remove the supervision of delexicalization for fairer comparison (− gold delex.), we find even greater improvement across all end-to-end metrics. As discussed in § 9, SGP-TOD uses both a supervised formatting example and a 'Policy Skeleton', containing additional supervision for Policy and Response Generation. With no implementation publicly available, we were unable to run a modified version of their experiments without this supervision for fair comparison. Despite a less-supervised setting, our method is able to perform comparably, even slightly out-performing SGP-TOD in Combined score. Remarkably, our unsupervised EM approach also outperforms the supervised variant of our model due to improvements in Inform and Success rate, suggesting the Dialogue acts we infer are of high quality.

7.2 DST Performance

Our DST results are shown in Table 2. Where possible, we distinguish between 'zero-shot' results which include a hand-engineered formatting example, and the same method applied without the formatting example.[5] We find that our method significantly outperforms our GPT 3.5 Turbo baseline by 26% joint goal accuracy. Our approach performs nearly as well as the best method using OpenAI Codex with a supervised formatting example, using less than 10% of the parameters at any time (175B vs. 15B). When applying the IC-DST and RefPyDST prompting methods to StarCoder, our method significantly outperforms both, with and without a formatting example.

7.3 Ablations

In Figure 4, we conduct an ablation to evaluate both the impact of our noisy channel modeling and the value of iterative re-labeling in our EM approach. We compare our proposed system to one in which each module is replaced by only greedily sampling from its 'direct' variant, at both labeling and end-to-end inference time. We plot our Combined end-to-end performance across iterations of EM, with '0' indicating our zero-shot system. We find that EM improves our end-to-end performance in both our noisy-channel approach and greedy ablation, and that our noisy-channel inference methods are important to dialogue success, with a 30 and 33 point improvement over our greedy baseline with 1 and 2 EM steps, respectively. We report Inform, Success, BLEU, and joint goal accuracy (JGA) for this ablation in Appendix E.

[5] Due to the deprecation of OpenAI Codex, we were unable to run experiments for IC-DST or RefPyDST without a formatting example on the original Codex model

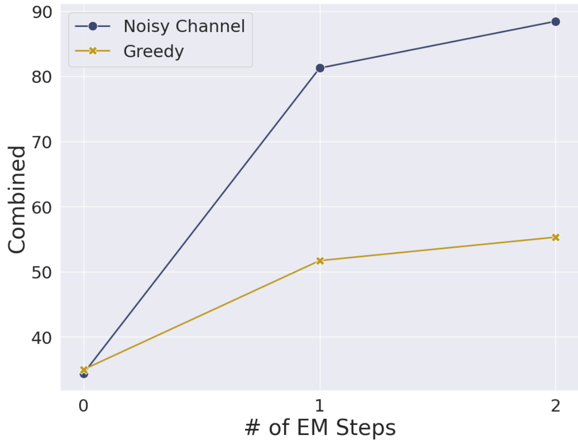

Figure 4: Combined score ($0.5(\text{Inform} + \text{Success}) + BLEU$) vs. the number of steps of expectation-maximization in our Noisy Channel method vs. a Greedy Ablation. '0' is zero-shot inference

Error Type	Noisy Channel	Greedy
Successful (no errors)	5	0
DST Failure: Incorrect State	1	1
Policy Failure: Inappropriate Dialogue Act	0	8
Policy Failure: Suboptimal Dialogue Act	4	2
Response not faithful to act	1	6

Table 3: An analysis of errors occurring in a human evaluation of 10 validation dialogues, comparing our noisy channel model to a greedy ablation. Any one error can make the dialogue unsuccesful, but more than one error can occur. A suboptimal dialog act is acceptable in the context and differs substantially from the gold human dialog act.

7.4 Error Analysis

We conduct an error analysis comparing the greedy model to our best noisy channel model (both with two EM steps). We randomly sampled 10 validation dialogues where at least one model was unsuccessful and analyze each system's behavior. Results of our human evaluation are shown in Table 3, and examples of each error type can be found in Appendix G. A common mistake (8 out of the 10 dialogues) of the greedy model is predicting incorrect dialogue acts and then producing a response that is not grounded in the dialogue. We find the noisy channel model does not make this type of mistake in any of the 10 dialogues. We hypothesize that the greedy model overuses frequently occurring dia-

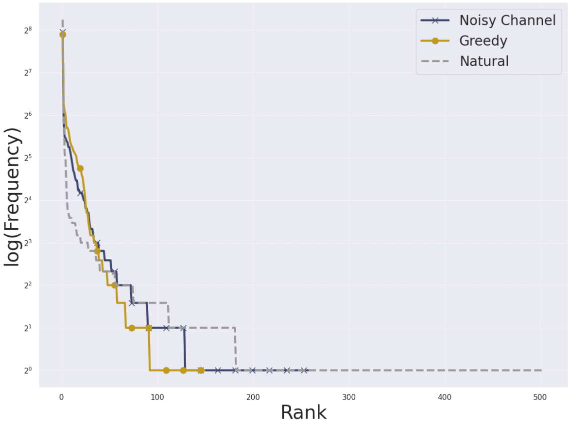

Figure 5: log(Frequency) vs. Rank of dialogue acts used by each model over a 200 dialogue sample of the validation set. 'Natural' refers to human annotations. We find our Noisy Channel approach uses a higher number of unique dialogue acts than the Greedy approach and better matches the characteristics of the distribution used by human annotators

logue acts like 'Request' regardless of context, and evaluate this by plotting the rank vs. frequency of dialogue acts not considering the values[6] for each model in Figure 5. We find that while both policies overuse common dialogue acts relative to a human agent, our noisy-channel method better utilizes the long tail of possible dialogue acts.

8 Contamination Analysis

Evaluation of unsupervised methods that use LLMs has the potential issue of **task contamination**, where supervised examples are seen in pretraining data (Li and Flanigan, 2024). Inclusion of supervised examples of the task in LLM pretraining data would render the model no longer unsupervised and the evaluation potentially biased: tasks for which the training data has been seen may have a higher performance than truly unsupervised tasks.

To address this issue, we quantify the presence of contamination in LLM pre-training data and estimate its potential impact on our results. Fortunately, the complete pre-training corpus for StarCoder is publicly available for analysis.[7]

We conduct an exhaustive search for supervised pairs of our dialogue subtasks in the StarCoder pre-training data using a semi-automated search with manual review. Details of our search procedure are in Appendix F. We find no complete dialogues with supervised labels. We do find 42 turns labeled with act tagging, and 42 turns labeled with DST in the pre-training corpus, categorized in Table 4.[8] We consider a (x, y) pair to be 'Correct' if the state change/dialogue act y is actually correct for the utterance x, and to be 'Authentic' if the (x, y) pair is found verbatim in the MultiWOZ corpus.[9] Astonishingly, we find half of the found Act Tagging pairs are incorrect, and could possibly mislead a pre-trained model if the model learned from them. We also find that less than half of the turns are authentic for either task, and find a number of them derive from Github issues discussing problems with dialogue simulators.

Task	Turns	Correct	Authentic
Act Tagging	42	21	5
DST	42	36	19

Table 4: Number of discovered contaminated turns per task, as well as the number which are correct or verified as being in the MultiWOZ dataset. The MultiWOZ dataset contains $71,522$ labeled turns for each task, indicating fewer than 0.06% are found to be contaminated

Additionally, we estimate the degree to which the contamination we discover could exaggerate expected performance of our method on an unseen schema, by using contaminated (x, y) pairs as in-context examples.[10]

In Table 5, we compare our zero-shot prompt, which receives no examples of any kind, with a 'contaminated' variant which uses k=3 examples derived from contamination in the pre-training corpus. The 'contaminated' model retrieves the most relevant contaminated fragments from a pool using the dense retrieval approach described in §4.4. These are inserted as a triple-quoted string block, so that the prompt remains syntactically valid python. By leaving contaminated examples in their original format, we test whether their inclusion elicits memorized knowledge rather than providing guidance on input/output formatting. Surprisingly, we find including this supervision via contaminated

[6]For the rank vs. frequency analysis, we consider the dialogue act for a turn to be the set of acts and slots (but not values) used by the system. For example the act "Inform(hotel_name='acorn guest house')" becomes {(Inform, hotel_name)}

[7]https://huggingface.co/datasets/bigcode/starcoderdata

[8]The average dialogue length in MultiWOZ is 13.9 turns. Put together, the set of contaminated turns would be roughly the length of 6 dialogues

[9]A 'Correct' pair might arise from printing training data, and an incorrect pair from discussion of a failure case.

[10]Ideally, one would pre-train an identical StarCoder model on a corpus *without* contamination. However, this is computationally impractical. We are also not aware of any available LLM that can be verified as not contaminated for this task.

Method	Inform	Success	BLEU	Combined
Ours (zero-shot)	**49.0**	**15.0**	3.0	**35.0**
Ours (k=3 contam ex.)	44.5	14.0	**3.8**	33.1
Ours (Full EM)	**80.5**	**69.0**	**13.7**	**88.5**

Table 5: Performance comparison when we include contaminated in-context examples. We find *including* this supervision hurts performance, and does not explain the strong performance of our noisy-channel EM approach

fragments *hurts* performance, indicating that these examples do not provide meaningful supervision for our task. Further, the substantial gains in our noisy-channel EM approach suggest our method is doing more than simply eliciting schema-specific knowledge memorized in pre-training.

9 Related Work

Zero-shot Dialogue A few recent works have proposed zero-shot approaches to dialogue problems using LLMs. Hu et al. (2022) and (King and Flanigan, 2023) propose DST methods which prompt code based LLMs in a text-to-SQL or text-to-program format, respectively. Similarly, (Li et al., 2024) propose a prompt-based function-calling approach to DST using LLMs. Each of these methods rely on prompts tailored to the schema and the use of one or more supervised 'formatting' example(s), which requires annotation expertise. Zhang et al. (2023) extends this approach to end-to-end task-oriented dialogue by adding a policy prompter for GPT 3.5. In addition to a formatting example, their policy prompt requires a hand-crafted 'policy-skeleton' consisting of examples of the appropriate system act and reply in response to different user utterances or database results. Our approach differs in that we require zero labeled examples of any kind. Hudeček and Dusek (2023) propose a zero-shot end-to-end method for prompting instruction-tuned LLMs like GPT 3.5. However, this method presumes *delexicalized* system responses $r_1 ... r_{t-1}$ in the conversation history as input, where entities are replaced with placeholders. Producing these inputs requires ground-truth annotations and gives a form of supervision about the entities and their attributes within a dialogue (see Table 1 for a comparison for GPT 3.5 Turbo with and without delex supervision). In contrast, we only assume fully-lexicalized dialogues, which do not provide this supervision and require no human annotation. We adapt the method of Hudeček and Dusek (2023) to use lexicalized dialogues as inputs, and use this approach as our baseline. Chung et al. (2023) propose an end-to-end method which prompts GPT-4 for interactions with a knowledge base before producing a response, however it generalizes poorly to the multi-domain setting.

Semi-supervised TOD Some works propose semi-supervised approaches to end-to-end task-oriented dialogue. Zhang et al. (2020) propose an end-to-end sequence-to-sequence model where the dialogue state is a latent variable. Liu et al. (2021a) adapt this approach for use with pre-trained language models, fine-tuning GPT-2. While successful, these approaches require a non-trivial amount of supervised data. Hudeček and Dušek (2022) learn latent dialogue acts using variational recurrent neural networks without turn level dialogue state labels, but still require observed DB/API calls and responses. Other semi-supervised works also evaluate their method in an unsupervised setting (Jin et al., 2018; Liu et al., 2023). However, these works also assume delexicalized training dialogues, which requires ground-truth annotation and gives a form a supervision to the model.

Noisy channel and re-ranking methods A few previous works have utilized noisy channel methods for task-oriented dialogue or prompting methods. Liu et al. (2021b) pre-train a noisy channel for task-oriented dialogues as a sequence to sequence model, requiring significant labeled training data. Min et al. (2022) propose noisy channel prompting for few-shot classification, which inspires our generalization to the generative setting.

10 Conclusion

We introduce the first (to our knowledge) approach for building a working task-oriented dialogue system with large language models using only unlabled dialogues and an API schema. The approach leverages an LLM and expectation-maximization to infer missing labels as latent variables from unlabeled dialogues. In ablations, we find that a noisy channel approach vastly improves performance, and explore some possible explanations for this in an error analysis. This work advances the state-of-the-art for using LLMs to build task oriented dialogue systems with minimal annotations. Future work can consider applying our method for inferring structured representations from natural text to other NLP tasks.

11 Limitations

Data contamination in LLM pre-training poses a hurdle for accurate benchmarking across NLP, and particularly for unsupervised methods. In an idealized setting, there would be a suitably strong task-oriented dialogue benchmark that could be verified as not belonging to the pre-training corpus of each new and more capable LLM. This is not the case for our setting or for many others, and warrants careful attention from the NLP community. For our setting, we were able to properly define problematic contamination and search for it in our LLM's pre-training corpus, thanks to the open release of the pre-training data. We found limited contamination and demonstrated that the contamination we found was not helpful in eliciting task knowledge that might have been memorized in pre-training.

All experiments in this paper were conducted on pre-existing public dialogue corpora, collected explicitly for training task-oriented dialogue agents with the knowledge of all participants (Budzianowski et al., 2018). Our use of the StarCoder model also falls within the terms of it's Responsible AI License. It is important that subsequent applications of our method also adhere to any fair-use policies governing collected dialogues or transcripts.

Acknowledgements

We thank Nilay Patel, Geetanjali Rakshit, Zekun Zhao, Brian Mak, Changmao Li, Rongwen Zhao, and other JLab members for insightful feedback on preliminary drafts and discussions of this work. We also thank the anonymous reviewers and area chairs for their detailed and helpful feedback. The authors were supported in part by the NSF National AI Institute for Student-AI Teaming (iSAT) under grant DRL 2019805. The opinions expressed are those of the authors and do not represent views of the NSF. We are thankful for the computing resources provided by the Pacific Research Platform's Nautilus cluster, supported by the National Science Foundation under Award Numbers CNS-1730158, ACI-1540112, ACI1541349, OAC-1826967, the University of California Office of the President, and the University of California San Diego's California Institute for Telecommunications and Information Technology/Qualcomm Institute.

References

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Paweł Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Iñigo Casanueva, Ultes Stefan, Ramadan Osman, and Milica Gašić. 2018. Multiwoz - a large-scale multi-domain wizard-of-oz dataset for task-oriented dialogue modelling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. ArXiv:2107.03374 [cs].

Willy Chung, Samuel Cahyawijaya, Bryan Wilie, Holy Lovenia, and Pascale Fung. 2023. InstructTODS: Large language models for end-to-end task-oriented dialogue systems. In *Proceedings of the Second Workshop on Natural Language Interfaces*, pages 1–21, Bali, Indonesia. Association for Computational Linguistics.

A. P. Dempster, N. M. Laird, and D. B. Rubin. 1977. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38.

Michael Heck, Nurul Lubis, Benjamin Ruppik, Renato Vukovic, Shutong Feng, Christian Geishauser, Hsien-chin Lin, Carel van Niekerk, and Milica Gasic. 2023. ChatGPT for zero-shot dialogue state tracking: A solution or an opportunity? In *Proceedings of the*

61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), pages 936–950, Toronto, Canada. Association for Computational Linguistics.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2020. The curious case of neural text degeneration. In *International Conference on Learning Representations*.

Yushi Hu, Chia-Hsuan Lee, Tianbao Xie, Tao Yu, Noah A. Smith, and Mari Ostendorf. 2022. In-context learning for few-shot dialogue state tracking. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 2627–2643, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Vojtěch Hudeček and Ondřej Dušek. 2022. Learning interpretable latent dialogue actions with less supervision. In *Proceedings of the 2nd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 12th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 297–308, Online only. Association for Computational Linguistics.

Vojtěch Hudeček and Ondrej Dusek. 2023. Are large language models all you need for task-oriented dialogue? In *Proceedings of the 24th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 216–228, Prague, Czechia. Association for Computational Linguistics.

Xisen Jin, Wenqiang Lei, Zhaochun Ren, Hongshen Chen, Shangsong Liang, Yihong Zhao, and Dawei Yin. 2018. Explicit State Tracking with Semi-Supervision for Neural Dialogue Generation. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 1403–1412. ArXiv:1808.10596 [cs].

Brendan King and Jeffrey Flanigan. 2023. Diverse retrieval-augmented in-context learning for dialogue state tracking. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 5570–5585, Toronto, Canada. Association for Computational Linguistics.

Changmao Li and Jeffrey Flanigan. 2024. Task Contamination: Language Models May Not Be Few-Shot Anymore. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16):18471–18480.

Raymond Li, Loubna Ben allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia LI, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Joel Lamy-Poirier, Joao Monteiro, Nicolas Gontier, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Ben Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy, Jason T Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Urvashi Bhattacharyya, Wenhao Yu, Sasha Luccioni, Paulo Villegas, Fedor Zhdanov, Tony Lee, Nadav Timor, Jennifer Ding, Claire S Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro Von Werra, and Harm de Vries. 2023a. Starcoder: may the source be with you! *Transactions on Machine Learning Research*. Reproducibility Certification.

Zekun Li, Zhiyu Chen, Mike Ross, Patrick Huber, Seungwhan Moon, Zhaojiang Lin, Xin Dong, Adithya Sagar, Xifeng Yan, and Paul Crook. 2024. Large language models as zero-shot dialogue state tracker through function calling. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8688–8704, Bangkok, Thailand. Association for Computational Linguistics.

Zekun Li, Baolin Peng, Pengcheng He, Michel Galley, Jianfeng Gao, and Xifeng Yan. 2023b. Guiding Large Language Models via Directional Stimulus Prompting. In *Advances in Neural Information Processing Systems*, volume 36, pages 62630–62656. Curran Associates, Inc.

Hong Liu, Yucheng Cai, Zhenru Lin, Zhijian Ou, Yi Huang, and Junlan Feng. 2021a. Variational Latent-State GPT for Semi-Supervised Task-Oriented Dialog Systems. ArXiv:2109.04314 [cs].

Qi Liu, Lei Yu, Laura Rimell, and Phil Blunsom. 2021b. Pretraining the Noisy Channel Model for Task-Oriented Dialogue. *Transactions of the Association for Computational Linguistics*, 9:657–674.

Qing-Bin Liu, Shi-Zhu He, Cao Liu, Kang Liu, and Jun Zhao. 2023. Unsupervised Dialogue State Tracking for End-to-End Task-Oriented Dialogue with a Multi-Span Prediction Network. *Journal of Computer Science and Technology*, 38(4):834–852.

Sewon Min, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2022. Noisy channel language model prompting for few-shot text classification. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5316–5330, Dublin, Ireland. Association for Computational Linguistics.

Tomáš Nekvinda and Ondřej Dušek. 2021. Shades of BLEU, flavours of success: The case of MultiWOZ. In *Proceedings of the 1st Workshop on Natural Language Generation, Evaluation, and Metrics (GEM 2021)*, pages 34–46, Online. Association for Computational Linguistics.

Wenbo Pan, Qiguang Chen, Xiao Xu, Wanxiang Che, and Libo Qin. 2023. A Preliminary Evaluation of ChatGPT for Zero-shot Dialogue Understanding. Publisher: arXiv Version Number: 1.

Shachi Paul, Rahul Goel, and Dilek Hakkani-Tür. 2019. Towards Universal Dialogue Act Tagging for Task-Oriented Dialogues. In *Proc. Interspeech 2019*, pages 1453–1457.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Abhinav Rastogi, Xiaoxue Zang, Srinivas Sunkara, Raghav Gupta, and Pranav Khaitan. 2020. Towards Scalable Multi-Domain Conversational Agents: The Schema-Guided Dialogue Dataset. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):8689–8696.

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2020. MPNet: Masked and Permuted Pre-training for Language Understanding. In *Advances in Neural Information Processing Systems*, volume 33, pages 16857–16867. Curran Associates, Inc.

Yixuan Su, Lei Shu, Elman Mansimov, Arshit Gupta, Deng Cai, Yi-An Lai, and Yi Zhang. 2022. Multi-task pre-training for plug-and-play task-oriented dialogue system. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4661–4676, Dublin, Ireland. Association for Computational Linguistics.

Qingyang Wu, James Gung, Raphael Shu, and Yi Zhang. 2023. DiactTOD: Learning generalizable latent dialogue acts for controllable task-oriented dialogue systems. In *Proceedings of the 24th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 255–267, Prague, Czechia. Association for Computational Linguistics.

Xiaoxue Zang, Abhinav Rastogi, Srinivas Sunkara, Raghav Gupta, Jianguo Zhang, and Jindong Chen. 2020. MultiWOZ 2.2 : A Dialogue Dataset with Additional Annotation Corrections and State Tracking Baselines. In *Proceedings of the 2nd Workshop on Natural Language Processing for Conversational AI*, pages 109–117, Online. Association for Computational Linguistics.

Xiaoying Zhang, Baolin Peng, Kun Li, Jingyan Zhou, and Helen Meng. 2023. SGP-TOD: Building task bots effortlessly via schema-guided LLM prompting. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 13348–13369, Singapore. Association for Computational Linguistics.

Yichi Zhang, Zhijian Ou, Min Hu, and Junlan Feng. 2020. A Probabilistic End-To-End Task-Oriented Dialog Model with Latent Belief States towards Semi-Supervised Learning. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9207–9219, Online. Association for Computational Linguistics.

Zihao Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. 2021. Calibrate before use: Improving few-shot performance of language models. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 12697–12706. PMLR.

A Prompt Examples

Figure 6 provides abridged instances of our direct prompts for DST and for Act Tagging. Figure 6a shows our prompt for inferring API call(s) or changes to the dialogue state from an unlabeled dialogue, as detailed in §4.1. Our prompts use python keyword arguments to provide the input variables for a given sub-task, and to prompt the LLM for the next variable of interest. Using the arbitrary ordering of keyword arguments in Python function calls, our 'channel' prompts simply re-order the arguments in order to score the likelihood of the user's utterance given the predicted state change. Figure 6b provides a similar abridged instance of our direct prompt for tagging dialogue acts in an unlabeled dialogue. Here, we simply condition on the observed system response r_t.

B Metric Details

End-to-End (E2E) Dialogue Metrics We measure end-to-end dialogue performance using the Inform rate, Success rate, and BLEU, following prior works, using the automatic evaluation provided by Nekvinda and Dušek (2021).[11]

A dialogue is considered Informed if the most recently mentioned result for each domain meets the user's goal constraints, and is considered Successful if it is Informed and all values for requested slots are presented to the user. For example, if a user were to ask 'Can you give me the phone number of a cheap hotel in the east part of town?', the dialogue would be Informed if we refer them to a hotel that is actually in the cheap price range and in the east, and Successful if we additionally provide the phone number, as requested. BLEU is computed against a single reference response, and the Combined score is $0.5(\text{Inform} + \text{Success}) + BLEU$.

Dialogue State Tracking Metrics Following prior works, we evaluate DST performance with joint-goal accuracy (JGA): for a turn x_t, a dialogue state prediction $\hat{y}_t$ is considered correct only if all slot names and values match the gold annotation

[11] https://github.com/Tomiinek/MultiWOZ_Evaluation

```
class DialogueAgent:
    <one method per intent in the schema with all informable slots>
    def book_taxi(self, leave_at: str = None, destination: str = None,
                  departure: str = None,
                  arrive_by: str = None) -> Intent:
        """
        book taxis to travel between places

        Parameters:
            leave_at: (str) leaving time of taxi
            destination: (str) destination of taxi
            departure: (str) departure location of taxi
            arrive_by: (str) arrival time of taxi
        """
        pass
   ...


if __name__ == '__main__':
    agent = DialogueAgent()

    # Provide the call matching the user's intent in this context

    <in-context exemplars from self-predictions may go here>

    response = agent.handle_turn(
        belief_state=BeliefState(attraction=dict(
                                     name='byard art',
                                     type="museum",
                                     area="south")),
        last_system_utterance="byard art is at 344 oxford " + \
                              "street, anything else?",
        user_utterance="Yes, I need a taxi to king station",
        user_intent=[agent.book_taxi(destination='king station')]
```

(a) Our 'direct' DST prompt with italicized ***completion***

```
<one Entity per service in schema, with informable + requestable slots>
class Taxi(Entity):
    """
    Parameters:
        leave_at: (str) leaving time of taxi
        destination: (str) destination of taxi
        departure: (str) departure location of taxi
        arrive_by: (str) arrival time of taxi
        type: (str) car type of the taxi
        phone: (str) phone number of the taxi
    """
    ...

< a class for each of the acts supported in our system>
class Inform(Act):
    """Provide information."""
    entity: Entity = None


class Request(Act):
    """Ask for specific information or action."""
    values: List[str] = None
...

if __name__ == '__main__':
    agent = DialogueAgent()

    # Provide the dialogue acts matching the observed system response

    <in-context exemplars from self-predictions may go here>

    response = agent.handle_turn(
        system_response="Ok, where will you be departing from?",
        system_acts=[Request(values=['departure'])]
```

(b) Our 'direct' act tagging prompt, with italicized ***completion***

Figure 6: Abridged prompt and completion examples from our in-context learning approach to initial labelling for DST and DAT (Act Tagging), best viewed in color. Key-word arguments are used to include variables from the turn context and to prefix the completion

state y_t. We again use the evaluation provided in Nekvinda and Dušek (2021). Following their work, we accept fuzzy matches for non-categorical string values, such as the name of a restaurant or hotel, using the `fuzzywuzzy` library and a fuzz ratio of 0.95.[12]

C Dialogue Acts

Following Paul et al. (2019), we use a universal set of dialogue acts for managing our agents communicative intents. We omit some acts for simplicity and to reduce the context length required to enumerate them in a prompt. Table 6 lists each act and a description. Since our dialogue set is not directly comparable to prior works, we do not directly evaluate act tagging or policy accuracy. Instead, acts serve only as an intermediate representation for planning responses in our end-to-end system.

D Offline Labeling Algorithm

Algorithm 1 gives our algorithm for pseudo-labeling of unlabeled dialogues.

E Further results across EM Steps

Here we expand on our ablations in § 7, which evaluates our method with and without our proposed noisy-channel prompting across iterations of expectation-maximization (EM). In Figure 7, we break down the performance gains we observed in our 'Combined' metric into Inform rate, Success rate, and BLEU, where Combined $= 0.5(Inform + Success) + BLEU$. '0' iterations of EM indicates our zero-shot prompting system, without any in-context examples or EM. We find that EM substantially improves performance in all cases, and particularly for our noisy-channel prompting approach. We find the noisy channel prompting approach improves performance on all metrics, with the most substantial gains over the greedy baseline in Inform and Success rates. This suggests that within our algorithm, noisy-channel inference may be particularly important when inferring the system's dialogue acts in order to reverse-engineer an accurate policy.

In Figure 8, we analyze dialogue state tracking performance across iterations of EM using Joint Goal Accuracy (JGA). We find our noisy-channel prompting approach improves the accuracy of our dialogue state tracking predictions across iterations of EM when compared to a greedy, direct prompting approach.

F Contamination Search & Result Details

F.1 Procedure

We detail our method for finding instances of task contamination within the StarCoder pre-training

[12] https://pypi.org/project/fuzzywuzzy/

Act	Description (as used in our prompt)
Inform(x=y)	Provide information.
Offer(x=y)	System provides an offer or suggestion based on results.
Confirm(x=y)	Seek confirmation of something.
Affirm(x=y)	Express agreement or confirmation.
Negate(x=y)	User or System denies or negates.
NotifySuccess(x=y)	Notify of a successful action or result.
NotifyFailure(x=y)	Notify of an error or failure.
Acknowledge	Acknowledge.
Goodbye	Goodbye.
Greeting	Greeting.
ThankYou	ThankYou.
RequestAlternatives	Ask for other options, alternatives, or any additional user goals.
Request(x=?)	Ask for specific information or action.

Table 6: Dialogue acts supported by our system, adapted from the universal dialogue acts proposed in Paul et al. (2019). "x=y" indicates the act can take on arbitrary key-value arguments, and "x=?" indicates the act takes on one or more unpaired arguments. We reduce the number of acts and lengths of descriptions relative to Paul et al. (2019) in order to fit within the LMs context length

Algorithm 1 Our algorithm for initial pseudo-labeling of unlabeled dialogues in $\mathcal{D}_{train}$

procedure INITIALOFFLINELABEL($\mathcal{D}_{train}, \theta_{ret}, \theta$)
  $\mathcal{P} \leftarrow \emptyset$ ▷ Initialize example pool
  $\mathcal{B} \leftarrow []$ ▷ Store predictions by dialogue id and turn index
  for $t = 0$ **to** $\max_{d \in \mathcal{D}_{train}} |d|$ **do** ▷ Loop by increasing turn index
    for all $(d_{id}, u_t, r_{t-1}, r_t)$ **in** $\mathcal{D}_{train}$ **do** ▷ d_{id} is dialogue ID
      $\hat{b}_{t-1} \leftarrow \mathcal{B}[d_{id}][t-1]$ **or** $\emptyset$ ▷ Fetch $\hat{b}_{t-1}$ if known
      $\hat{b}_t \leftarrow$ OFFLINEDST($\mathcal{P}, \theta_{ret}, \hat{b}_{t-1}, r_{t-1}, u_t$)
      $\hat{A}_t \leftarrow$ OFFLINEACTTAG($\mathcal{P}, \theta_{ret}, u_t, r_t$)
      $\mathcal{P} \leftarrow \mathcal{P} \cup \{(r_{t-1}, u_t, r_t, \hat{b}_t, \hat{A}_t)\}$ ▷ Add in-context example for future labeling
    end for
  end for
end procedure
procedure OFFLINEDST($\mathcal{P}, \theta_{ret}, \hat{b}_{t-1}, r_{t-1}, u_t$)
  $\mathcal{E}_k \leftarrow \theta_{ret}(\hat{b}_t \cdot r_{t-1} \cdot u_t, \mathcal{P})$ ▷ Retrieve up to k in-context examples
  $\mathcal{C} \leftarrow \Delta b_t \underset{\text{top-p}}{\sim} P(f_{\text{prompt}}(\mathcal{E}_k, \hat{b}_{t-1}, r_{t-1}, u_t))$ ▷ Sample w/ 'direct' prompt
  $\Delta\hat{b}_t \leftarrow \underset{\Delta b_t \in \mathcal{C}}{\operatorname{argmax}} P(u_t | f_{\text{prompt}}(\mathcal{E}_k, \hat{b}_{t-1}, r_{t-1}, \Delta b_t)$ ▷ Re-rank w/ 'channel' prompt
  return $\hat{b}_{t-1} + \Delta\hat{b}_t$
end procedure
procedure OFFLINEACTTAG($\mathcal{P}, \theta_{ret}, u_t, r_t$)
  $\mathcal{E}_k \leftarrow \theta_{ret}(u_t \cdot r_t, \mathcal{P})$ ▷ Retrieve up to k in-context examples
  $\mathcal{C} \leftarrow A_t \underset{\text{top-p}}{\sim} (P(f_{\text{prompt}}(\mathcal{E}_k, r_t)))$ ▷ Sample w/ 'direct' prompt
  return $\underset{A_t \in \mathcal{C}}{\operatorname{argmax}} P(\mathcal{E}_k, A_t, r_t)$ ▷ Re-rank w/ 'channel' prompt
end procedure

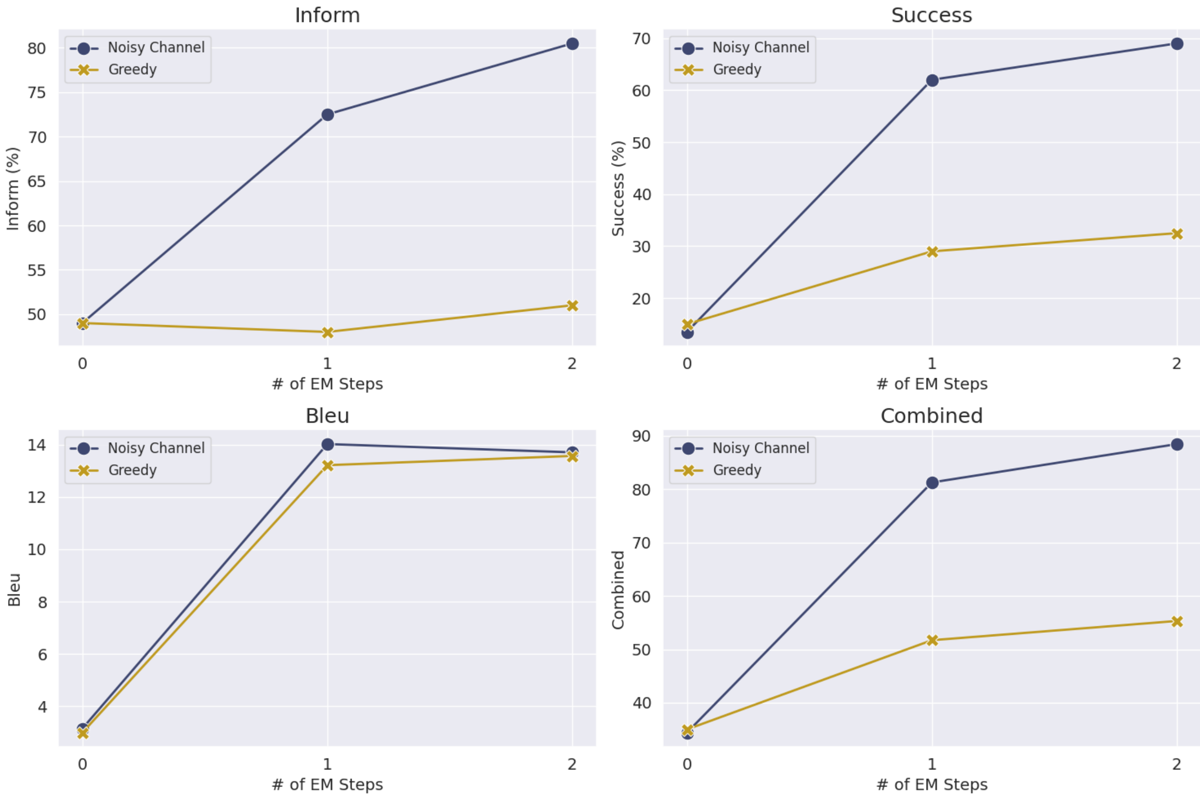

Figure 7: Breaking down Combined $= 0.5(\text{Inform} + \text{Success}) + BLEU$ into components Inform Rate, Success Rate, and BLEU across iterations of EM between our proposed noisy-channel approach and a greedy ablation, which omits noisy-channel prompting at inference time and when labeling dialogue states & system acts in the expectation step. We find improvement across all components, and particularly our Inform and Success Rates

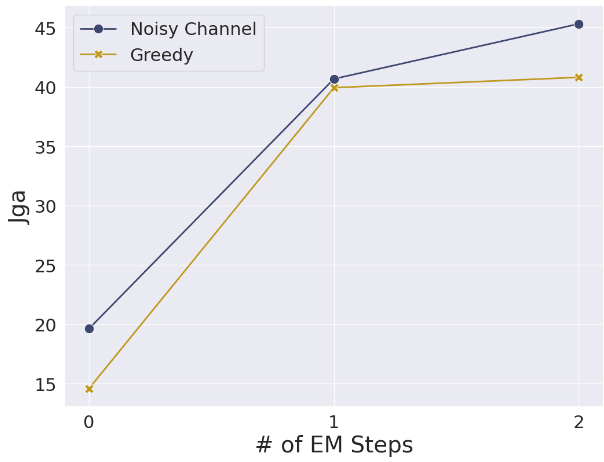

Figure 8: Joint Goal Accuracy (JGA) of our inferred API call(s)/Dialogue states across iterations of EM. We find improved dialogue state tracking performance when using our noisy-channel method at inference time and when labeling dialogue states offline in the expectation step for training, compared to a greedy direct prompting approach

set. We are particularly interested in *supervised pairs* (x, y) where y belongs to our schema of interest $\mathcal{S}$, for any of the dialogue sub-tasks used in our system. We devise a method for searching the complete pre-training corpus for contaminated (x, y) pairs, where x is an utterance we might observe from either the system or user, and y is the latent dialogue state change or dialogue act supporting $\mathcal{S}$. For each utterance x from either the system or user, we collect all documents from the pre-training corpus which contain the complete utterance. We use the elastic search index provided for the StarCoder pre-training data, which accounts for differences in capitalization, punctuation, and interrupting white-space.[13] Following this, we search matching documents for keywords from y (e.g. slot names and values) to determine which of these documents may plausibly contain a supervised label and warrant manual review. For dialogue states, these are the slot names and values, discarding extremely generic keywords like 'name'. For act tags, these are the act names, slots, and values. We then consider a document to need manual review if 40% or more of the keywords are found in the 500 characters before or after a matching x in a document. Finally, we hand-check the remaining documents and extract contaminated (x, y) pairs.

F.2 Examples

Table 7 contains examples of contamination discovered in our search process, and the type of document in which they were found. Notably, none of the examples found closely match our output formatting.

G Error Analysis

We conduct an error analysis comparing the greedy model to our best noisy channel model (both with two EM steps). We randomly sampled 10 validation dialogues where at least one model was unsuccessful and analyze each system's behavior. Here we present examples of each error type from Table 3. Examples can be viewed in Table 8.

The first failure case we present is a dialogue state tracking failure, in which the DST system misses a slot critical to achieving the users goal.

The second example presents a common failure mode in the greedy model, in which commonly occurring dialogue acts are predicted when not appropriate in the context. In this example, the greedy model predicts that it should 'Request' the area of the attraction from the user, even though it is already known. This causes the response generator to hallucinate that there must be no API results to be shared with the user, and that the user should look for attractions in another area (i.e. a 'RequestAlternatives'). Our noisy channel model correctly chooses to 'Inform' the user, and produces a response faithful to this dialogue act.

The third example presents sub-optimal dialogue acts used by both models. The noisy channel and greedy model both predict that they should request information about the attraction 'type' from the user, a seemingly reasonable conclusion. Alternatively, the human agent in both cases proactively makes a suggestion. In an online evaluation, the user could provide this information possibly complete a successful dialogue, though in a few more turns than required of the human agent. In the offline evaluation, these dialogues are considered unsuccessful.

In the final example, we show an instance of the greedy model not being faithful to the dialogue acts predicted by the policy. In this case, the predicted dialogue acts are also inappropriate: the system should not request information from the user that they would not be expected to know. The generated response instead tries to offer a restaurant to the user, but without a value grounded in the API response. This can be considered a hallucination, as the restaurant may or may not exist or match the user's criteria.

[13] https://github.com/bigcode-project/search/blob/main/index.py

Contaminated Input	**Contaminated Output**	**Sub-Task**	**Source**
I need a restaurant to dine at in Cambridge on my upcoming trip . I need info about chiquito restaurant bar restaurant .	restaurant-inform«<name===chiquito restaurant bar	DST	Jupyter Notebook
i would like to book a 5 star , or closest to it , in the east part of town please .	"<SOB> hotel area = east, stars = 5, type = hotel <EOB> <SOB> hotel area = east, stars = 5 restaurant area = east <EOB>"	DST	Python
[Syst] the train id is tr8292 and the price is 16.50 pounds.	[SYS_DA] train-inform-leave-tr8292 [SYS_DA] train-inform-ticket-16.50 pounds	Act Tagging	Github Issue

Table 7: Example inputs and outputs in contaminated documents from each task, discovered in the StarCoder pre-training corpus. We include the source type of each document

Error Type: Incorrect Dialogue State

User: find me a place to stay which has 0 star and preferably a guest house
System: There are two guesthouse-type hotels in Cambridge with 0 stars. Do you have an area of town in mind?
User: North part of town for sure
System: that could be city centre north b and b.it is cheap with both internet and parking. can i reserve for you?
User: Yes, please book it for 6 people and 3 nights starting from tuesday.
System: I booked your hotel reservations for 6 people for 3 days starting Tuesday. Your reference number is : EB5RRD33 . May i help you with something else.
User: I am looking to go to a multiple sports center too.
System: on what area do you need the attraction?
User: I'd like it to be in the centre, please.

DST Prediction (✗): `{..., 'attraction': {'type': 'multiple sports',`
`- 'area': 'centre'`
`}}`

Error Type: Inappropriate Dialogue Act (Greedy)

User: I'm looking for a train that departs from birmingham new street. It should arrive by 18:30.
System: There is one arrives at 17:23 and another at 18:23. What day are you traveling?
User: I am leaving on tuesday for cambridge.
System: I have train TR5630 leaving at 15:40 and arriving at 18:23. Would that work for you?
User: That sounds great. Thanks.
System: Did you need tickets for that train?
User: Not right now, but thanks for asking. I was wondering if there are any parks on the south side of town, though.

Noisy Channel Acts (✓): `[Inform(entity=Attraction(name='[value_name]', area='[value_area]'))]`
Noisy Channel Response (✓): There is the [value_name] in the [value_area] of town.
Greedy Acts (✗): `[Request(service='attraction', values=['area'])]`
Greedy Response (✗): There are no parks on the south side of town. Would you like me to look in another area?
Human Acts: `Offer(entity=Attraction(address='[value_address]', area='[value_area]', price='[value_price]', name='[value_name]')`
Human Response: On the [value_area] we have [value_name] at [value_address]. The admission is [value_price] are you interested?

Error Type: Suboptimal Dialogue Act (Both)

User: i need information about wagamama restaurant.
System: i have their info, what would you like to know?
User: I need to book a table for 2 people please.
System: Okay! For when?
User: Wednesday, 17:45. Reference number, please?
System: Let me book that for you now.
User: After you get around to that reference number, I also want to go somewhere in the Centre of town.

Noisy Channel Acts (✗): `[Request(service='attraction', values=['type'])]`
Greedy Acts (✗): `[Request(service='attraction', values=['type'])]`
Noisy Channel Response (✗): What type of attraction are you looking for?
Greedy Response (✗): What type of attraction are you looking for?
Human Acts: `[Offer(entity=Attraction(name='all saints church'))]`
Human Response: What about [value_name]?

Error Type: Not Faithful to Act (Greedy)

User: Do you have any recommendations for restaurants in Cambridge?
System: There are 110 restaurants in Cambridge, is there a specific cuisine you're looking for?
User: I'm sorry, it is actually a train that I'm looking for. I need to depart from Cambridge on Saturday.
System: Well can you tell me yur destination please?
User: I need to go to leicester and leave from cambridge after 18:00. There are 8 of us, can you make a reservation and send me the reference number?
System: I booked you on the TR6034 that leaves Cambridge at 18:21 and arrives at 20:06. Your reference is NYXZ1SOR and the total fee of 241.92 is payable at the station
User: Perfect. I am also looking for a moderate priced restaurant in the east
System: There are two indian restaurants and one italian. Do either of these interest you?
User: I am interested in Italian.

Noisy Channel Acts (✓): `[Offer(entity=Restaurant(name='[value_name]'))]`
Noisy Channel Response (✓): I would recommend [value_name]. Would you like me to book a table for you?
Greedy Acts (✗): `[Request(service='restaurant', values=['name', 'address', 'phone', 'postcode'])]`
Greedy Response (✗): I have the caffee uno located in the centre. Would you like me to make reservations?
Human Acts: `Inform(entity=Restaurant(address='[value_address]', name='[value_name]'))`
Human Response: The [value_name] is located at [value_address]. Would you like me to book a reservation?

Table 8: Examples of each error type from our analysis in §7.4. In the first error, we show a dialogue state tracking failure which prevents the agent from providing one of the correct attractions. In the second, we show an an inappropriate dialogue act, in which the greedy model requests an area from the user, even though it is already known, leading to a hallucination that there must be no results. In the third, we show a suboptimal dialogue act, used by both systems. Both responses are plausible, but the human agent proactively assists the user in fewer turns. In the final example, we show an instance in which the Greedy model is not faithful to its chosen (incorrect) dialogue act, offering a restaurant that may or may not actually exist or match the user's criteria