SQLWOZ: A Realistic Task-Oriented Dialogue Dataset with SQL-Based Dialogue State Representation for Complex User Requirements

Heng-Da Xu[1,2], **Xian-Ling Mao**[1]*, **Fanshu Sun**[1], **Tian-Yi Che**[1], **Cheng-Xin Xin**[1], **Heyan Huang**[1]
[1]Beijing Institute of Technology [2]Amap, Alibaba Group
{xuhengda, maoxl, sunfs, ccty, xin.chengxin, hhy63}@bit.edu.cn

Abstract

High-quality datasets are essential for building effective task-oriented dialogue (TOD) systems. The existing TOD datasets often present overly simplified interactions, where users incrementally express straightforward requests that can be managed with basic slot-value style dialogue states, such as "hotel-area = east." However, this approach does not reflect real-life scenarios in which users may express complex constraints and preferences. To address this gap, in this paper, we propose SQLWOZ, a novel TOD dataset designed to capture complex, real-world user requirements. The user requirements in SQLWOZ include the four categories: 1) multiple values for a slot, 2) excluded values within a slot, 3) preferred or prioritized values, and 4) conditional values based on other conditions. We utilize SQL statements as a formalized and expressive representation of dialogue states within SQLWOZ. To evaluate the dataset, we adapt large language models as dialogue agents and conduct extensive experiments on the SQL-based dialogue state tracking, dialogue response generation, and end-to-end TOD tasks. The experimental results demonstrate the complexity and quality of SQLWOZ, establishing it as a new benchmark for advancing TOD research. [1]

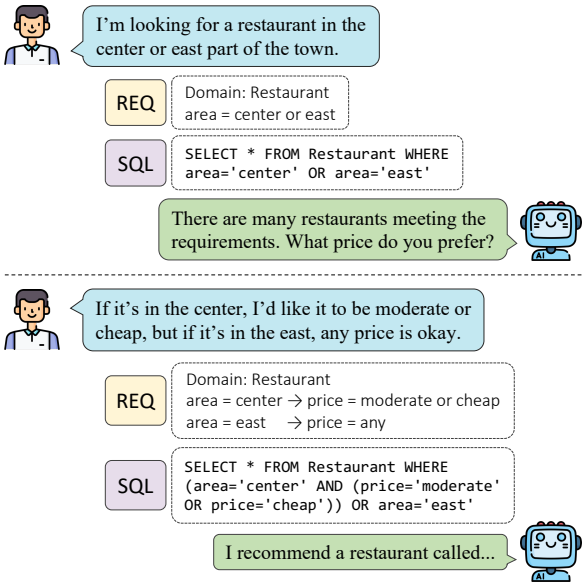

Figure 1: A dialogue example to illustrate the complex user requirements and the SQL-style dialogue state. The user specifies multiple values and conditional requirements in the first and second turns, respectively. REQ is an intuitive representation of user requirements.

1 Introduction

Task-oriented dialogue (TOD) systems play a pivotal role in enabling seamless interactions between users and automated agents (Zhang et al., 2024), facilitating goal-oriented tasks across a wide spectrum of real-world applications, such as travel guidance, restaurant accommodation, and customer support (Budzianowski et al., 2018; Quan et al., 2020). These systems have garnered significant attention from both the research community and industry due to their potential to streamline user interactions, improve accessibility, and enhance user satisfaction in service-oriented applications (Stacey et al., 2024). Recently, rapid progress in this field has been driven by the remarkable success of pre-trained large language models (OpenAI et al., 2024; Touvron et al., 2023), which have introduced new capabilities for understanding and generating natural language in TOD contexts (Xu et al., 2024b).

As the core of traditional TOD systems, the dialogue state tracking (DST) module is responsible for extracting and managing user requirements (Wu et al., 2019). Typically, DST models represent user requirements as a set of slot-value pairs (e.g., "hotel-area = east"), known as the dialogue state (Bebensee and Lee, 2023). This dialogue state then serves as a query parameter for retrieving relevant information from backend

*Corresponding author.
[1]Code: https://github.com/DaDaMrX/SQLWOZ

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 7538–7563
November 4-9, 2025

databases (Kim et al., 2023). Then the dialogue context, state, and retrieved data collectively inform the dialogue policy module, which determines the system's structured response (Peng et al., 2018). Finally, the natural language generation module transforms this structured response into a user-friendly, natural language output (Wang et al., 2020).

However, the conventional slot-value dialogue state representations fall short in capturing the intricacies of real-world user requirements. Real-life conversations often involve complex expressions of intent and constraints that cannot be adequately represented by a simple slot-value structure (Xu et al., 2024a). For instance, users may specify multiple desired values for a single slot, exclude certain values, or express preferences or priorities among some values. Users may also conditionally adjust their requirements based on the satisfaction of previous demands. As shown in Figure 1, these complex requirements frequently emerge in real-life dialogues. In the first turn, the user specifies multiple acceptable values within the same slot, indicating a requirement for a restaurant in either the "center" or "east" part of town. In the second turn, the user introduces a more complex conditional requirement that the desired price range depends on the location of the restaurant. Such complex real-life requirements cannot be adequately captured by traditional slot-value dialogue state tracking approaches, underscoring the urgent need for more robust task-oriented dialogue datasets and modeling approaches.

To address these limitations, we propose SQLWOZ, a novel TOD dataset specifically designed to capture the complex user requirements observed in real-world interactions. SQLWOZ represents user requirements using SQL-based dialogue state representations, offering a more expressive, structured format capable of accommodating various sophisticated user constraints. The user requirements in SQLWOZ are categorized into four types: (1) multiple values for a slot, (2) excluded values within a slot, (3) prioritized or preferred values, and (4) conditional values depending on the satisfaction of other conditions. These slot types are theoretically complete to represent arbitrary user intent (Boole, 1854). For dataset construction, SQLWOZ builds upon the domain ontology of the widely used TOD dataset MultiWOZ (Eric et al., 2020), while realistic dialogue interactions are primarily generated using powerful large language models (e.g., GPT-4o (OpenAI et al., 2024)). Specifically, a heuristic user goal generator is employed to create randomized yet sufficiently complex user goals, which are then used by two LLM agents to generate the full dialogue flow between user and system, along with SQL-style dialogue states. Finally, a validation step ensures quality control by filtering out unsuccessful or low-quality dialogues, ensuring the dataset's overall robustness and reliability.

Given that traditional task-oriented dialogue systems struggle to handle the complex user requirements presented in SQLWOZ, we implement strong baselines by fine-tuning various large language models, such as FlanT5 (Chung et al., 2024) and Llama (Meta, 2024)), to serve as capable dialogue agents. We establish various SQL-based TOD benchmarks, including dialogue state tracking, dialogue response generation, and end-to-end TOD tasks. Extensive experimental results highlight the complexity and high quality of SQLWOZ, positioning it as a challenging new benchmark for advancing the capabilities of task-oriented dialogue systems in managing sophisticated user requirements across diverse application domains.

2 Related Work

2.1 Task-Oriented Dialogue Datasets

Traditional task-oriented dialogue (TOD) datasets are generally categorized into two types based on their data generation methods: Machine-to-Machine (M2M) and Wizard-of-Oz (WOZ). In the M2M paradigm, dialogue skeletons are first generated using rule-based user and system simulators, and then the dialogue skeletons are expanded to natural languages by crowd workers (Shah et al., 2018). A notable example is the Schema-Guided Dialogue (SGD) dataset (Rastogi et al., 2020), which includes comprehensive schemas tailored for diverse task domains.

The WOZ approach, on the other hand, involves human crowd workers directly generating dialogues by role-playing as users and systems (Kelley, 1984). This framework was first applied in TOD research by Wen et al. (2017) with the WOZ dataset and has since inspired the creation of numerous datasets, including FRAMES (Asri et al., 2017), KVRET (Eric et al., 2017), MultiWOZ series (Budzianowski et al., 2018; Eric et al., 2020), ABCD (Chen et al., 2021), STAR (Mosig et al., 2020), CrossWOZ (Zhu et al., 2020), and RiSAWOZ (Quan et al., 2020). These datasets have

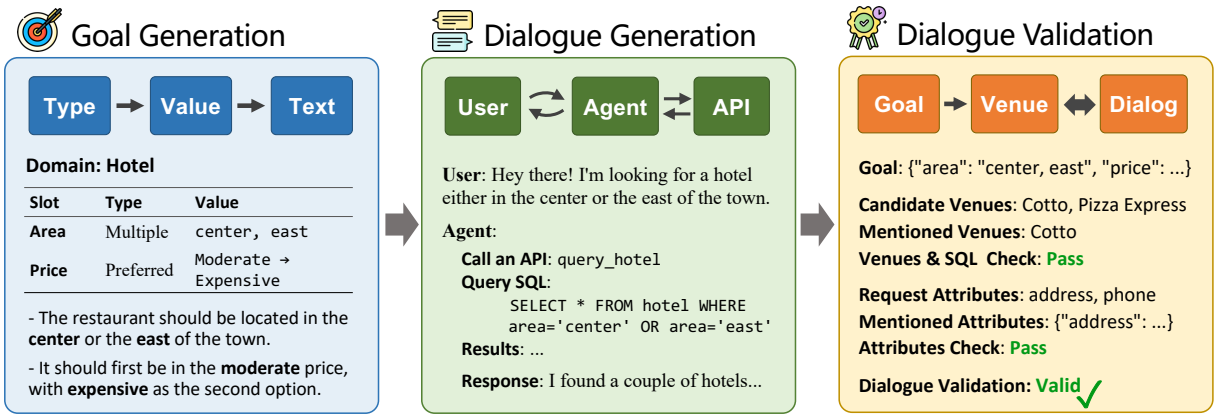

Figure 2: The overall construction pipeline for the SQLWOZ dataset.

significantly advanced TOD research by providing rich and varied conversational data.

2.2 Data Generation with LLMs

The rapid advancements in large language models (LLMs) have introduced new methodologies for dataset construction. For instance, Jin et al. (2024) utilized GPT-4 to construct DailyPersuasion, a multi-domain persuasive dialogue dataset. Yang et al. (2024) demonstrated the potential of combining outputs from strong and weak LLMs to generate diverse text-to-SQL datasets. Similarly, Abolghasemi et al. (2024) employed LLMs to create satisfaction-focused counterfactual dialogues, enhancing the balance of user satisfaction datasets. In the context of TOD, datasets like LUAS (Niu et al., 2024) leverage GPT-4 for dialogue state tracking tasks, while Ulmer et al. (2024) utilized self-play techniques with LLMs simulating different roles in dialogues to develop more dynamic data collection frameworks. Liu et al. (2024) proposed an automated pipeline to construct TOAD, a TOD dataset with diverse response styles. These approaches illustrate the growing reliance on LLMs to address limitations in traditional data collection methods, offering scalable and adaptable solutions for generating complex and nuanced dialogue datasets.

3 Dataset Construction

The construction of the SQLWOZ dataset follows a systematic pipeline comprising three key stages: Goal Generation, Dialogue Generation, and Dialogue Validation. The overall construction process is illustrated in Figure 2 and is detailed below.

3.1 Goal Generation

The first stage in constructing SQLWOZ involves designing sufficiently complex user goals. These goals are based on the ontology of the MultiWOZ dataset (Eric et al., 2020), which encompasses five domains and 13 slots within a travel guidance scenario. MultiWOZ primarily addresses tasks such as assisting users in finding and booking specific venues, such as restaurants or hotels. Each domain includes multiple slots, such as *area* and *food* in the restaurant domain or *price* and *stars* in the hotel domain. However, the user goals in MultiWOZ are relatively simple, with each slot being assigned at most one value.

In contrast, SQLWOZ introduces realistic and complex user goals, categorizing slots into four advanced types according the values: **Multiple**, **Excluded**, **Preferred**, and **Conditional**, alongside the original **Single** type from MultiWOZ. These complex slot types better reflect real-world conversational needs and significantly enhance the dataset's expressiveness and challenge level. In propositional logic, these slot types are functional complete to represent any possible user intent (Post, 1921). The detailed illustration is in the Appendix. The slot types are further detailed as followsl

- **Multiple**: Allowing multiple acceptable values for a slot, e.g., a user requesting a restaurant in either the center or east part of town.
- **Excluded**: Specifying values that are unacceptable, e.g., a user seeking a hotel that is not expensive.
- **Preferred**: Indicating a preference or priority among values, e.g., a user preferring Thai food but accepting Vietnamese food if Thai is

unavailable.

- **Conditional**: Representing dependencies between slots, either within the same domain or across domains, e.g., "If the hotel is in the center, it should be moderately priced; if in the north, it should be expensive."

To generate these complex user goals, the Goal Generation stage involves three main steps: slot & type generation, value generation, and textual goal generation.

Slot and Type Generation. The first step in generating a user goal involves randomly selecting the relevant domains and associated slots. A goal can involve between one and three domains. For multi-domain goals, the selected domains are typically related, with examples including combinations like restaurant and attraction, or hotel, restaurant, and taxi. Each slot in a user goal is assigned a type - Single, Multiple, Excluded, Preferred, or Conditional - according to predefined probabilities specific to each slot. For slots assigned a type other than Single, sub-types are defined and further probabilistically determined. For example, slots categorized under the Multiple or Excluded types have variations in the number of candidate or excluded values that can be specified. Additionally, the Preferred and Conditional types are higher-order categories that may encompass Single, Multiple, or Excluded types as sub-categories.

Value Generation. Once the slot types are determined, specific values are sampled for each slot. For slots that have inherent adjacency or linear relationships, such as *area*, *price*, or *stars*, the sampled values are selected in a way that preserves these natural relationships. For instance, the values for the *area* slot might include "center and east," but not "north and south," as the latter pair would likely contradict a user's geographical preferences. The result of the value generation step is a structured goal representation where each slot is associated with a type and its corresponding values.

Textual Goal Generation. To enhance the accessibility of user goals for both human and language models, textual representations of each goal are generated. For each sub-type of every slot, three distinct language templates are created, ensuring a diverse set of linguistic expressions in the dataset. This variation in phrasing not only enriches the dataset but also ensures that the goals reflect natural, real-world conversational patterns. Any duplicate goals are filtered out to maintain uniqueness, resulting in a robust and varied set of complex user goals for SQLWOZ. Example structural and textual representations of these user goals are provided in the Appendix.

3.2 Dialogue Generation

After defining user goals, realistic dialogues are generated using large language models (e.g., GPT-4o (OpenAI et al., 2024)). Two separate models simulate the roles of the user and the dialogue agent, respectively. The dialogue agent interacts with backend APIs to retrieve information or make reservations. The dialogue generation process consists of three main components: API interfaces, the dialogue agent, and the user simulator.

API Interfaces. SQLWOZ introduces complex user goals where slots may have diverse value formats, making traditional slot-value-based dialogue state tracking methods insufficient. Instead, SQLWOZ employs SQL statements for dialogue state tracking due to their expressive power. In our experiments, all query APIs accept SQL statements as input and return results in the form of Markdown tables. When an invalid SQL statement is provided or an error occurs during the query, the system returns a human-readable error message instead of halting the dialogue process. This enables the language model to handle the error gracefully and continue the interaction effectively.

Dialogue Agent. The dialogue agent is implemented using the OpenAI GPT-4o API, equipped with a carefully crafted prompt. This prompt includes three key components: the task scenario introduction, API descriptions, and dialogue policy instructions. The task scenario introduction outlines the overarching task and domain information, establishing the context for the dialogue. The API descriptions, formatted following OpenAI's function definition conventions, detail the names, parameters, and value constraints of the available APIs. Finally, the dialogue policy instructions guide the agent's behavior, specifying when to invoke APIs and how to respond to the user. For complex user requirements, the agent can autonomously construct intricate SQL statements or make multiple simpler queries as needed. For straightforward user inputs, such as greetings or confirmations, the agent generates responses directly without calling APIs. The prompt for the

dialogue agent is presented in the Appendix.

User Simulator. The user simulator, also based on the GPT-4o language model, operates with the textual user goal embedded in its prompt. Its primary objective is to achieve the specified goal through interactions with the dialogue agent. The simulator is encouraged to articulate complex requirements dynamically and adaptively. It generates the initial utterance to start the dialogue and, once it believes all defined goals have been met, it outputs a predefined signal to indicate the end of the conversation. Additionally, if the agent misunderstands the user's intent or makes errors, the simulator is instructed to inform the agent to make corrections. The generation temperature for both the dialogue agent and the user simulator is set to 0.6, fostering a balance between diversity and coherence in the generated dialogue. The prompt for the user simulator is also presented in the Appendix.

3.3 Dialogue Validation

To ensure the quality and reliability of the collected dialogues, we implement a rigorous validation procedure to filter out unsuccessful or unqualified samples. The first step is to confirm that each dialogue successfully fulfills its corresponding user goal. Since a given user goal can be satisfied by various possible SQL representations, success is determined based on the entities retrieved by the user goal. Specifically, for each user goal, we identify all the candidate venues that meet the specified requirements and check that at least one of these venues is explicitly mentioned within the dialogue utterances. This ensures that the dialogue ultimately fulfills the user's intent, even if some constraints were not fully articulated by the user throughout the conversation. The SQL statements in the API calling log are also inspected to ensure that they do not contain uncorrected mistakes or misunderstandings. Moreover, the requested attributes of the chosen venue and the reservation needs must also be satisfied. Dialogues failing these criteria are discarded. Additionally, we exclude dialogues containing fewer than 3 pairs of user–agent utterances, as well as those containing more than 3 consecutive repetitions of the same API call or identical utterances. Notably, dialogues where incorrect API calls are made but are followed by valid error messages are retained. These instances are valuable for improving the robustness of downstream models, as they reflect real-world scenarios where errors may occur, but the system responds gracefully. A complete validation example is presented in the Appendix.

4 Dataset Analysis

4.1 Overall Statistics

The SQLWOZ dataset contains 22,955 dialogues with a total of 294,214 turns, averaging 12.8 turns per dialogue. It spans 5 domains, includes 30 slots, and supports 8 backend APIs (4 SQL query APIs and 4 booking APIs). SQLWOZ includes 3,606 single-domain dialogues (15.7%) and 19,349 multi-domain dialogues (84.3%). Each dialogue covers 2.0 domains and 5.3 slots on average. These dialogues are divided into training, validation, and test sets with a ratio of 8:1:1, resulting in 18,365, 2,295, and 2,295 dialogues for each set, respectively. Notably, SQLWOZ is generated entirely through an automatic pipeline, enabling scalability in both dataset size and domain coverage. A detailed comparison of SQLWOZ and previous TOD datasets are presented in the Appendix.

Unlike earlier TOD datasets, SQLWOZ does not require a backend interaction for every turn. In total, SQLWOZ includes 96,048 API calls, averaging 4.2 calls per dialogue (2.9 SQL queries and 1.3 booking requests). Among all dialogue turns, 39.7% involve no backend API calls, 55.6% involve a single call, and 4.7% involve two or more calls. This API calling design greatly improves the efficiency of backend interaction in TOD systems. At the same time, SQLWOZ incorporates sufficiently complex task goals that cannot be completed with a single SQL query, enhancing its challenge and realism.

4.2 Slot Type Distribution

Table 1 illustrates the distribution of slots and dialogue turns on different slot types in SQLWOZ. At the user goal slot level, the overall proportion of complex slots is 47.9% in all the dialogue goals. It's worth noting that the ratio can be controlled in the data generation process and we set this ratio to mirror realistic scenarios, as ratio of complex intents in real life would not be extremely high. At the dialogue turn level, we can see that 74.3% of the dialogue turns with SQL statements involve complex slot types, indicating the high proportion of complex user requirements in the dialogues and the high challenge of SQLWOZ. At the dialogue

Slot Type	#Slot	Ratio	#Turn	Ratio
Single	63,027	52.1%	35,880	58.4%
Multiple	15,536	12.8%	17,262	28.1%
Excluded	12,444	10.3%	13,646	22.2%
Preferred	14,986	12.4%	14,227	23.2%
Conditional	15,012	12.4%	11,214	18.3%
- single domain	*11,730*	*9.7%*	*8,031*	*13.1%*
- cross domain	*3,282*	*2.7%*	*3,183*	*5.2%*
Complex Total	57,978	47.9%	45,600	74.3%

Table 1: The distribution of slots and dialogue turns on different slot types in SQLWOZ. Only the turns with SQL statements are considered. "Complex" represents all slot types except "Single". The "Conditional" type is further divided into single-domain and cross-domain sub-types.

level, there are 93.5% (21,467) complex dialogues that are associated with at least one complex slot.

4.3 Quality of User Utterances and SQLs

Beyond the validation of the agent responses during data construction to ensure that the recommended venues satisfy user goals, we also assess the quality of user utterances and SQL statements, as both are pivotal to express complex user requirements and issuing accurate queries. The validation statistics are presented in Table 2.

For **user utterances**, we first measure the ratio of user goals that are actually expressed by the user. Concretely, we extract all the slot value words from each dialogue's user goal, and compute the proportion that appear in the user's utterances. Nearly 70% (67.8%) of the goal values are explicitly expressed in user utterances. Note that full coverage is not mandatory because the user simulator terminates the dialogue once the system offers a satisfactory venue. To examine the articulation of complex goals, we focus on slots with composite values (e.g., Multiple and Preferred types) and compute the fraction whose complete set of values is mentioned within a single utterance. Roughly half of these slots (48.0%) are fully expressed at once, reflecting the inherent complexity of the dialogues. The language diversity validation is presented in the Appendix.

For **SQL statements**, we perform a similar analysis. The ratio of goal values captured in the SQL queries is 57.5%, comparable to the coverage in user utterances. When we restrict attention to goal values that the user explicitly mentions, their realization in SQL rises sharply to 89.1%. Finally, to further measure the consistency between user utterances and SQLs, we calculate, for each turn, the overlap between goal values indicated in the utterance and those encoded in the corresponding SQLs. The resulting overlap of 91.7% demonstrating the strong alignment between user expressions and SQL queries within SQLWOZ.

Statistics about User Utterances & SQLs	Ratio
Goal Values Expressed in User Utterances	67.8%
Composite Values Expressed Simultaneously	48.0%
Goal Values Expressed in SQLs	66.5%
Values in User Utterances Expressed in SQLs	89.1%
Value Overlap of User Utter. and SQLs Pairs	91.7%

Table 2: Statistics about quality of user utterances and SQLs, measuring the coverage of goal values in user utterances and the corresponding SQL statements.

5 Experiments

Given that SQLWOZ adopts a realistic and complex dialogue state representation, traditional slot-value-based dialogue state tracking and task-oriented dialogue models are insufficient to effectively capture the intricacies of user requirements. Consequently, we utilize the SQLWOZ dataset to train novel SQL-based task-oriented dialogue agents capable of autonomously generating SQL queries and response utterances according to user inputs. The dialogue agents are based on LLMs of varying sizes, including smaller models such as GPT-2 (Radford et al., 2018) and FlanT5 (Chung et al., 2024), as well as larger models like Llama 3.2 and 3.3 (Meta, 2025), Qwen 3 (Yang et al., 2025), and DeepSeek-V3 (DeepSeek-AI et al., 2025).

5.1 SQL-based Dialogue State Tracking

We establish a benchmark for SQL-based dialogue state tracking, where dialogue agents are provided with dialogue history and are asked to perform API calls according to the current user input. The agents must determine autonomously whether and how to invoke external APIs, with the flexibility to make zero, one, or multiple API calls. For evaluation, we compare the generated API calls with the ground truth in SQLWOZ, which includes the number of API calls, the API names, and the parameters. For APIs that involve SQL statements as query parameters, the evaluation is based on the retrieved result sets. Two SQL statements are considered identical

Model (Size)	Full-Shot			Few-Shot (10%)			Few-Shot (1%)		
	All API	SQL	Others	All API	SQL	Others	All API	SQL	Others
GPT-2 (124M)	51.16	44.82	70.41	30.25	25.86	62.48	16.15	12.75	38.56
FlanT5 (770M)	53.00	45.50	71.98	39.25	34.36	68.84	28.37	20.77	53.46
Llama 3.2 (1B)	55.16	47.33	74.00	48.25	37.21	71.19	40.59	22.59	56.71
Llama 3.2 (3B)	57.14	50.65	77.35	54.93	43.71	**72.75**	45.92	27.52	59.55
Qwen 3 (8B)	**59.08**	**52.57**	**78.04**	**57.11**	**50.01**	71.31	**48.23**	**31.81**	**60.00**

Table 3: Evaluation results for the SQL-based dialogue-state-tracking task. Metrics reported for all API calls (All API) are further broken down into SQL API calls (SQL) and all other calls (Others). Two SQL statements are treated as equivalent when they return identical result sets.

Model (Size)	Similarity		Diversity	
	BLEU	ROUGH	SE	CE
GPT-2 (124M)	25.12	32.65	7.13	2.71
FlanT5 (770M)	28.47	43.80	7.60	2.90
Llama (1B)	31.82	54.94	8.06	3.09
Llama (3B)	37.05	60.19	8.26	3.15
Qwen 3 (8B)	**39.71**	**63.35**	**8.40**	**3.25**

Table 4: Response generation quality evaluation results, with similarity measured by BLEU and ROUGE-L, and diversity assessed using Shannon Entropy (SE) and Conditional bigram Entropy (CE).

Model	Backbone	M 2.1	M 2.2	SQLWOZ
SimpleTOD	DistilGPT-2	56.45	-	5.13
SPACE-3	UniLM	**57.50**	57.50	6.35
TOATOD	T5-base	54.97	63.79	6.08
SQLWOZ	Llama 1B	56.07	**64.30**	**55.16**

Table 5: Dialogue state tracking performance comparison on traditional TOD datasets (MultiWOZ 2.1, 2.2) and our proposed SQLWOZ. SQLWOZ is converted into the single slot-value style for adapting the traditional DST methods.

if their resulting datasets match.

The results are presented in Table 3. Besides full-shot training, we also evaluate the agents under few-shot settings using 10% and 1% of the training data. To clearly investigate the ability of the models to generate correct SQLs, we further reported the performance solely on the SQL APIs and the other APIs. The ratio of SQL and other APIs is about 7:3. As the results show, for all the evaluated agents, the performance on SQL APIs is much lower than the others, indicating the importance and challenge of SQL statements in SQLWOZ.

5.2 Dialogue Response Generation

We evaluate the ability of the dialogue agents to generate proper response utterances. The agents are provided with the dialogue history and the current user utterance and API calls to inform the generation of their responses. The metrics are chosen from two perspectives: for sentence similarity, we utilize the BLEU (Papineni et al., 2002) and ROUGE-L (Lin and Hovy, 2002) metrics, and for linguistic diversity, we employ Shannon Entropy (SE) and Conditional bigram Entropy (CE) (Xu et al., 2024b). The evaluation results are depicted in Table 4. We can see that larger models consistently outperform smaller ones. GPT-2 achieves a BLEU score of 25.12 while the largest Qwen 3 obtains 39.71. In terms of linguistic diversity, smaller models like GPT-2 show lower Shannon Entropy and Conditional Bigram Entropy, while larger models, particularly Qwen 3, demonstrate higher entropy values, indicating greater diversity in their responses. These results underline the advantages of larger models in both generating accurate and diverse responses.

5.3 Comparison with Traditional TOD Methods and Datasets

To illustrate both the value and the challenge of SQLWOZ, we compare it with the widely used MultiWOZ 2.1 and 2.2 benchmarks (Eric et al., 2020; Zang et al., 2020). We evaluate three representative dialogue-state-tracking (DST) models—SimpleTOD (Hosseini-Asl et al., 2020), SPACE-3 (He et al., 2022), and TOATOD (Bang et al., 2023). Although they rely on different architectures, all three assume the traditional slot-value dialogue-state representation. Accordingly, we convert SQLWOZ to the slot-value style for these models by discarding the additional values contained in its complex slots. The results, shown in Table 5, reveal a clear pattern: while the three models achieve competitive accuracy on MultiWOZ 2.1/2.2, their

Model (Size)	Full-Shot				Zero-Shot (Hotel)			
	Inform	Success	Book	Comb	Inform	Success	Book	Comb
GPT-2 (124M)	31.79	24.45	25.64	28.42	9.19	6.81	4.09	7.32
FlanT5 (124M)	43.30	39.20	40.12	41.48	21.80	18.25	13.75	18.90
Llama 3.2 (1B)	54.97	52.83	53.01	53.94	32.20	28.29	22.42	28.78
Llama 3.2 (3B)	57.58	55.56	**56.71**	56.86	36.50	32.45	25.50	32.74
Qwen 3 (8B)	**59.28**	**58.05**	55.72	**58.08**	**38.11**	**33.95**	**26.79**	**34.24**
Qwen 3 (235B-A22B)†	-	-	-	-	**34.17**	28.33	**16.19**	**28.22**
Llama 3.3 (70B)†	-	-	-	-	32.66	**28.41**	14.85	27.15
DeepSeek-V3 (671B-A37B)†	-	-	-	-	33.87	27.19	14.55	27.37

Table 6: End-to-end task-oriented dialogue evaluation results. In the zero-shot setting, the first 5 models are trained on the other domains and evaluated on the leaved hotel domain dialogues. The last three models with † are directly evalauted on the hotel dialogues without any training.

performance drops sharply on SQLWOZ, highlighting their inability to reason over the richer slot semantics in our dataset. By contrast, our SQLWOZ dialogue agent, achieves the best or comparable performance across all datasets thanks to its explicit modeling of complex user intents.

5.4 End-to-end Task-Oriented Dialogue

To assess the overall performance of dialogue agents on the SQLWOZ dataset, we establish a benchmark for the end-to-end task-oriented dialogue task. Dialogue evaluation needs to be as similar as possible to real dialogue scenarios, thus, we employ a dynamic simulator-based evaluation framework, where the dialogue agent interacts with a user simulator to fulfill user requirements, rather than being provided with fixed user utterances from the test set, which could disrupt the coherence of the dialogue. The simulator-based framework enables the dialogue agent to dynamically adjust its strategies in response to user requests, and also allows it to recover from potential errors made by either the agent or the external APIs (Xu et al., 2024b; Terragni et al., 2023).

Given the high cost and limited availability of the OpenAI API, we train a user simulator to interact with dialogue agents offline, which is based on the Llama 3.2 3B model. We compare the performance of this trained user simulator with the OpenAI API-based user simulator used during the creation of SQLWOZ, and the trained user simulator achieves a comparable dialogue validity rate and language diversity with the OpenAI version. The comparison details are presented in the Appendix.

The trained user simulator is used to evaluate the end-to-end dialogue capabilities of the dialogue agents. Besides the full-shot setting, we also conduct a zero-shot experiment on the hotel domain, where the smaller models ($\leq$ 8B) are adopted in a domain transfer setting with training on the other domains, while the larger models are fully zero-shot testing. For evaluation metrics, we extend the dialogue validation process from the dataset construction phase and adopt metrics from MultiWOZ (Eric et al., 2020) and AutoTOD (Xu et al., 2024b). Specifically, we use four metrics: **Inform**, **Success**, **Book**, and **Combine**. **Inform** measures whether the agent successfully provides the desired venues according to the generated SQL statements. **Success** evaluates whether the agent delivers all the required venue attributes by inspecting the dialogue utterances. **Book** assesses whether the agent correctly makes the reservation. **Combine** is a composite metric defined as $\text{Combine} = 0.5 \cdot \text{Inform} + 0.25 \cdot (\text{Success} + \text{Book})$. The results are presented in Table 6. We can see that the agent performance improves as model size increases, both in the full-shot and zero-shot settings. However, even for the largest model, Qwen 3 8B, the overall Combine performance in the full-shot setting remains below 60, underscoring the challenges posed by the SQLWOZ dataset and highlighting the significant potential for advancing complex, realistic task-oriented dialogue systems.

6 Conclusion

In the paper, we propose SQLWOZ, a realistic task-oriented dialogue dataset that addresses the limitations of traditional slot-value based dialogue state by using SQL-based dialogue state representation. The experiments demonstrate SQLWOZ a robust benchmark for advancing TOD research.

7 Limitations

While SQLWOZ provides a significant step forward in capturing complex, real-life user requirements for task-oriented dialogue (TOD) systems, it has some limitations. Firstly, SQLWOZ introduces a level of complexity in dialogue state representation that may present challenges for traditional TOD models, limiting their ability to perform effectively without extensive modifications. Secondly, our SQL-based approach, while flexible and expressive, requires models with advanced parsing capabilities to handle SQL structures, potentially raising the barrier for deployment in low-resource environments where such model capacities may be limited. Additionally, SQLWOZ primarily focuses on English-language interactions, which may restrict its applicability to multilingual or culturally diverse dialogue scenarios. Future work could explore adapting SQLWOZ to other languages or constructing language-agnostic representations that could capture the same level of complexity across diverse linguistic contexts. Lastly, although we establish strong baselines using large language models, further exploration is needed to assess SQLWOZ's generalizability across a wider range of model architectures and settings.

Acknowledgements

The work is supported by National Natural Science Foundation of China (No. 62402043, 62172039, 62302040, U21B2009 and 62276110), China Postdoctoral Science Foundation (No. 2022TQ0033), Beijing Institute of Technology Research Fund Program for Young Scholars.

References

Amin Abolghasemi, Zhaochun Ren, Arian Askari, Mohammad Aliannejadi, Maarten Rijke, and Suzan Verberne. 2024. CAUSE: Counterfactual assessment of user satisfaction estimation in task-oriented dialogue systems. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 14623–14635, Bangkok, Thailand. Association for Computational Linguistics.

Layla El Asri, Hannes Schulz, Shikhar Sharma, Jeremie Zumer, Justin Harris, Emery Fine, Rahul Mehrotra, and Kaheer Suleman. 2017. Frames: a corpus for adding memory to goal-oriented dialogue systems. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue, Saarbrücken, Germany, August 15-17, 2017*, pages 207–219. Association for Computational Linguistics.

Namo Bang, Jeehyun Lee, and Myoung-Wan Koo. 2023. Task-optimized adapters for an end-to-end task-oriented dialogue system. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 7355–7369, Toronto, Canada. Association for Computational Linguistics.

Björn Bebensee and Haejun Lee. 2023. Span-selective linear attention transformers for effective and robust schema-guided dialogue state tracking. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 78–91, Toronto, Canada. Association for Computational Linguistics.

George Boole. 1854. *An Investigation of the Laws of Thought on Which Are Founded the Mathematical Theories of Logic and Probabilities*. Macmillan, London. First edition.

Paweł Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Iñigo Casanueva, Stefan Ultes, Osman Ramadan, 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*, pages 5016–5026, Brussels, Belgium. Association for Computational Linguistics.

Derek Chen, Howard Chen, Yi Yang, Alexander Lin, and Zhou Yu. 2021. Action-based conversations dataset: A corpus for building more in-depth task-oriented dialogue systems. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3002–3017, Online. Association for Computational Linguistics.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Alex Castro-Ros, Marie Pellat, Kevin Robinson, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Y. Zhao, Yanping Huang, Andrew M. Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. 2024. Scaling instruction-finetuned language models. *J. Mach. Learn. Res.*, 25:70:1–70:53.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean

Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wanjia Zhao, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaokang Zhang, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xinnan Song, Xinxia Shan, Xinyi Zhou, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Yang Zhang, Yanhong Xu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Yu, Yi Zheng, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Ying Tang, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yu Wu, Yuan Ou, Yuchen Zhu, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan. 2025. Deepseek-v3 technical report.

Mihail Eric, Rahul Goel, Shachi Paul, Abhishek Sethi, Sanchit Agarwal, Shuyang Gao, Adarsh Kumar, Anuj Goyal, Peter Ku, and Dilek Hakkani-Tur. 2020. MultiWOZ 2.1: A consolidated multi-domain dialogue dataset with state corrections and state tracking baselines. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 422–428, Marseille, France. European Language Resources Association.

Mihail Eric, Lakshmi Krishnan, François Charette, and Christopher D. Manning. 2017. Key-value retrieval networks for task-oriented dialogue. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue, Saarbrücken, Germany, August 15-17, 2017*, pages 37–49. Association for Computational Linguistics.

Wanwei He, Yinpei Dai, Min Yang, Jian Sun, Fei Huang, Luo Si, and Yongbin Li. 2022. Space-3: Unified dialog model pre-training for task-oriented dialog understanding and generation.

Ehsan Hosseini-Asl, Bryan McCann, Chien-Sheng Wu, Semih Yavuz, and Richard Socher. 2020. A simple language model for task-oriented dialogue. In *Advances in Neural Information Processing Systems*, volume 33, pages 20179–20191. Curran Associates, Inc.

Chuhao Jin, Kening Ren, Lingzhen Kong, Xiting Wang, Ruihua Song, and Huan Chen. 2024. Persuading across diverse domains: a dataset and persuasion large language model. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1678–1706, Bangkok, Thailand. Association for Computational Linguistics.

J. F. Kelley. 1984. An iterative design methodology for user-friendly natural language office information applications. *ACM Trans. Inf. Syst.*, 2(1):26–41.

Yongil Kim, Yerin Hwang, Joongbo Shin, Hyunkyung Bae, and Kyomin Jung. 2023. Injecting comparison skills in task-oriented dialogue systems for database search results disambiguation. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 4047–4065, Toronto, Canada. Association for Computational Linguistics.

Chin-Yew Lin and Eduard Hovy. 2002. Manual and automatic evaluation of summaries. In *Proceedings of the ACL-02 Workshop on Automatic Summarization*, pages 45–51, Phildadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Yinhong Liu, Yimai Fang, David Vandyke, and Nigel Collier. 2024. TOAD: Task-oriented automatic dialogs with diverse response styles. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 8341–8356, Bangkok, Thailand. Association for Computational Linguistics.

Meta. 2024. Introducing meta llama 3: The most capable openly available llm to date. `https://ai.meta.com/blog/meta-llama-3/`. Accessed 2024-05-13.

Meta. 2025. Llama 3.2: Revolutionizing edge ai and vision with open, customizable models. `https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/`.

Johannes E. M. Mosig, Shikib Mehri, and Thomas Kober. 2020. STAR: A schema-guided dialog dataset for transfer learning. *CoRR*, abs/2010.11853.

Cheng Niu, Xingguang Wang, Xuxin Cheng, Juntong Song, and Tong Zhang. 2024. Enhancing dialogue state tracking models through LLM-backed user-agents simulation. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8724–8741, Bangkok, Thailand. Association for Computational Linguistics.

OpenAI, :, Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, Aleksander Mądry, Alex Baker-Whitcomb, Alex Beutel, Alex Borzunov, Alex Carney, Alex

Chow, Alex Kirillov, Alex Nichol, Alex Paino, Alex Renzin, Alex Tachard Passos, Alexander Kirillov, Alexi Christakis, Alexis Conneau, Ali Kamali, Allan Jabri, Allison Moyer, Allison Tam, Amadou Crookes, Amin Tootoochian, Amin Tootoonchian, Ananya Kumar, Andrea Vallone, Andrej Karpathy, Andrew Braunstein, Andrew Cann, Andrew Codispoti, Andrew Galu, Andrew Kondrich, Andrew Tulloch, Andrey Mishchenko, Angela Baek, Angela Jiang, Antoine Pelisse, Antonia Woodford, Anuj Gosalia, Arka Dhar, Ashley Pantuliano, Avi Nayak, Avital Oliver, Barret Zoph, Behrooz Ghorbani, Ben Leimberger, Ben Rossen, Ben Sokolowsky, Ben Wang, Benjamin Zweig, Beth Hoover, Blake Samic, Bob McGrew, Bobby Spero, Bogo Giertler, Bowen Cheng, Brad Lightcap, Brandon Walkin, Brendan Quinn, Brian Guarraci, Brian Hsu, Bright Kellogg, Brydon Eastman, Camillo Lugaresi, Carroll Wainwright, Cary Bassin, Cary Hudson, Casey Chu, Chad Nelson, Chak Li, Chan Jun Shern, Channing Conger, Charlotte Barette, Chelsea Voss, Chen Ding, Cheng Lu, Chong Zhang, Chris Beaumont, Chris Hallacy, Chris Koch, Christian Gibson, Christina Kim, Christine Choi, Christine McLeavey, Christopher Hesse, Claudia Fischer, Clemens Winter, Coley Czarnecki, Colin Jarvis, Colin Wei, Constantin Koumouzelis, Dane Sherburn, Daniel Kappler, Daniel Levin, Daniel Levy, David Carr, David Farhi, David Mely, David Robinson, David Sasaki, Denny Jin, Dev Valladares, Dimitris Tsipras, Doug Li, Duc Phong Nguyen, Duncan Findlay, Edede Oiwoh, Edmund Wong, Ehsan Asdar, Elizabeth Proehl, Elizabeth Yang, Eric Antonow, Eric Kramer, Eric Peterson, Eric Sigler, Eric Wallace, Eugene Brevdo, Evan Mays, Farzad Khorasani, Felipe Petroski Such, Filippo Raso, Francis Zhang, Fred von Lohmann, Freddie Sulit, Gabriel Goh, Gene Oden, Geoff Salmon, Giulio Starace, Greg Brockman, Hadi Salman, Haiming Bao, Haitang Hu, Hannah Wong, Haoyu Wang, Heather Schmidt, Heather Whitney, Heewoo Jun, Hendrik Kirchner, Henrique Ponde de Oliveira Pinto, Hongyu Ren, Huiwen Chang, Hyung Won Chung, Ian Kivlichan, Ian O'Connell, Ian O'Connell, Ian Osband, Ian Silber, Ian Sohl, Ibrahim Okuyucu, Ikai Lan, Ilya Kostrikov, Ilya Sutskever, Ingmar Kanitscheider, Ishaan Gulrajani, Jacob Coxon, Jacob Menick, Jakub Pachocki, James Aung, James Betker, James Crooks, James Lennon, Jamie Kiros, Jan Leike, Jane Park, Jason Kwon, Jason Phang, Jason Teplitz, Jason Wei, Jason Wolfe, Jay Chen, Jeff Harris, Jenia Varavva, Jessica Gan Lee, Jessica Shieh, Ji Lin, Jiahui Yu, Jiayi Weng, Jie Tang, Jieqi Yu, Joanne Jang, Joaquin Quinonero Candela, Joe Beutler, Joe Landers, Joel Parish, Johannes Heidecke, John Schulman, Jonathan Lachman, Jonathan McKay, Jonathan Uesato, Jonathan Ward, Jong Wook Kim, Joost Huizinga, Jordan Sitkin, Jos Kraaijeveld, Josh Gross, Josh Kaplan, Josh Snyder, Joshua Achiam, Joy Jiao, Joyce Lee, Juntang Zhuang, Justyn Harriman, Kai Fricke, Kai Hayashi, Karan Singhal, Katy Shi, Kavin Karthik, Kayla Wood, Kendra Rimbach, Kenny Hsu, Kenny Nguyen, Keren Gu-Lemberg, Kevin Button, Kevin Liu, Kiel Howe, Krithika Muthukumar, Kyle Luther, Lama Ahmad, Larry Kai, Lauren Itow, Lauren Workman, Leher Pathak, Leo Chen, Li Jing, Lia Guy, Liam Fedus, Liang Zhou, Lien Mamitsuka, Lilian Weng, Lindsay McCallum, Lindsey Held, Long Ouyang, Louis Feuvrier, Lu Zhang, Lukas Kondraciuk, Lukasz Kaiser, Luke Hewitt, Luke Metz, Lyric Doshi, Mada Aflak, Maddie Simens, Madelaine Boyd, Madeleine Thompson, Marat Dukhan, Mark Chen, Mark Gray, Mark Hudnall, Marvin Zhang, Marwan Aljubeh, Mateusz Litwin, Matthew Zeng, Max Johnson, Maya Shetty, Mayank Gupta, Meghan Shah, Mehmet Yatbaz, Meng Jia Yang, Mengchao Zhong, Mia Glaese, Mianna Chen, Michael Janner, Michael Lampe, Michael Petrov, Michael Wu, Michele Wang, Michelle Fradin, Michelle Pokrass, Miguel Castro, Miguel Oom Temudo de Castro, Mikhail Pavlov, Miles Brundage, Miles Wang, Minal Khan, Mira Murati, Mo Bavarian, Molly Lin, Murat Yesildal, Nacho Soto, Natalia Gimelshein, Natalie Cone, Natalie Staudacher, Natalie Summers, Natan LaFontaine, Neil Chowdhury, Nick Ryder, Nick Stathas, Nick Turley, Nik Tezak, Niko Felix, Nithanth Kudige, Nitish Keskar, Noah Deutsch, Noel Bundick, Nora Puckett, Ofir Nachum, Ola Okelola, Oleg Boiko, Oleg Murk, Oliver Jaffe, Olivia Watkins, Olivier Godement, Owen Campbell-Moore, Patrick Chao, Paul McMillan, Pavel Belov, Peng Su, Peter Bak, Peter Bakkum, Peter Deng, Peter Dolan, Peter Hoeschele, Peter Welinder, Phil Tillet, Philip Pronin, Philippe Tillet, Prafulla Dhariwal, Qiming Yuan, Rachel Dias, Rachel Lim, Rahul Arora, Rajan Troll, Randall Lin, Rapha Gontijo Lopes, Raul Puri, Reah Miyara, Reimar Leike, Renaud Gaubert, Reza Zamani, Ricky Wang, Rob Donnelly, Rob Honsby, Rocky Smith, Rohan Sahai, Rohit Ramchandani, Romain Huet, Rory Carmichael, Rowan Zellers, Roy Chen, Ruby Chen, Ruslan Nigmatullin, Ryan Cheu, Saachi Jain, Sam Altman, Sam Schoenholz, Sam Toizer, Samuel Miserendino, Sandhini Agarwal, Sara Culver, Scott Ethersmith, Scott Gray, Sean Grove, Sean Metzger, Shamez Hermani, Shantanu Jain, Shengjia Zhao, Sherwin Wu, Shino Jomoto, Shirong Wu, Shuaiqi, Xia, Sonia Phene, Spencer Papay, Srinivas Narayanan, Steve Coffey, Steve Lee, Stewart Hall, Suchir Balaji, Tal Broda, Tal Stramer, Tao Xu, Tarun Gogineni, Taya Christianson, Ted Sanders, Tejal Patwardhan, Thomas Cunninghman, Thomas Degry, Thomas Dimson, Thomas Raoux, Thomas Shadwell, Tianhao Zheng, Todd Underwood, Todor Markov, Toki Sherbakov, Tom Rubin, Tom Stasi, Tomer Kaftan, Tristan Heywood, Troy Peterson, Tyce Walters, Tyna Eloundou, Valerie Qi, Veit Moeller, Vinnie Monaco, Vishal Kuo, Vlad Fomenko, Wayne Chang, Weiyi Zheng, Wenda Zhou, Wesam Manassra, Will Sheu, Wojciech Zaremba, Yash Patil, Yilei Qian, Yongjik Kim, Youlong Cheng, Yu Zhang, Yuchen He, Yuchen Zhang, Yujia Jin, Yunxing Dai, and Yury Malkov. 2024. Gpt-4o system card.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia,

Pennsylvania, USA. Association for Computational Linguistics.

Baolin Peng, Xiujun Li, Jianfeng Gao, Jingjing Liu, and Kam-Fai Wong. 2018. Deep Dyna-Q: Integrating planning for task-completion dialogue policy learning. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2182–2192, Melbourne, Australia. Association for Computational Linguistics.

Emil Leon Post. 1921. Introduction to a general theory of elementary propositions. *American Journal of Mathematics*, 43(3):163–185.

Jun Quan, Shian Zhang, Qian Cao, Zizhong Li, and Deyi Xiong. 2020. Risawoz: A large-scale multi-domain wizard-of-oz dataset with rich semantic annotations for task-oriented dialogue modeling. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 930–940. Association for Computational Linguistics.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2018. Language models are unsupervised multitask learners.

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.

Pararth Shah, Dilek Hakkani-Tür, Gokhan Tür, Abhinav Rastogi, Ankur Bapna, Neha Nayak, and Larry Heck. 2018. Building a conversational agent overnight with dialogue self-play.

Joe Stacey, Jianpeng Cheng, John Torr, Tristan Guigue, Joris Driesen, Alexandru Coca, Mark Gaynor, and Anders Johannsen. 2024. LUCID: LLM-generated utterances for complex and interesting dialogues. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 4: Student Research Workshop)*, pages 56–74, Mexico City, Mexico. Association for Computational Linguistics.

Silvia Terragni, Modestas Filipavicius, Nghia Khau, Bruna Guedes, André Ferreira Manso, and Roland Mathis. 2023. In-context learning user simulators for task-oriented dialog systems. *CoRR*, abs/2306.00774.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models.

Dennis Ulmer, Elman Mansimov, Kaixiang Lin, Lijia Sun, Xibin Gao, and Yi Zhang. 2024. Bootstrapping LLM-based task-oriented dialogue agents via self-talk. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 9500–9522, Bangkok, Thailand. Association for Computational Linguistics.

Kai Wang, Junfeng Tian, Rui Wang, Xiaojun Quan, and Jianxing Yu. 2020. Multi-domain dialogue acts and response co-generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7125–7134, Online. Association for Computational Linguistics.

Tsung-Hsien Wen, David Vandyke, Nikola Mrkšić, Milica Gašić, Lina M. Rojas-Barahona, Pei-Hao Su, Stefan Ultes, and Steve Young. 2017. A network-based end-to-end trainable task-oriented dialogue system. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 438–449, Valencia, Spain. Association for Computational Linguistics.

Chien-Sheng Wu, Andrea Madotto, Ehsan Hosseini-Asl, Caiming Xiong, Richard Socher, and Pascale Fung. 2019. Transferable multi-domain state generator for task-oriented dialogue systems. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 808–819, Florence, Italy. Association for Computational Linguistics.

Heng-Da Xu, Xian-Ling Mao, Puhai Yang, Fanshu Sun, and Heyan Huang. 2024a. Cross-domain coreference modeling in dialogue state tracking with prompt learning. *Knowledge-Based Systems*, 283:111189.

Heng-Da Xu, Xian-Ling Mao, Puhai Yang, Fanshu Sun, and Heyan Huang. 2024b. Rethinking task-oriented dialogue systems: From complex modularity to zero-shot autonomous agent. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2748–2763, Bangkok, Thailand. Association for Computational Linguistics.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge,

Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. 2025. Qwen3 technical report.

Jiaxi Yang, Binyuan Hui, Min Yang, Jian Yang, Junyang Lin, and Chang Zhou. 2024. Synthesizing text-to-SQL data from weak and strong LLMs. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7864–7875, Bangkok, Thailand. 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.

Ming Zhang, Caishuang Huang, Yilong Wu, Shichun Liu, Huiyuan Zheng, Yurui Dong, Yujiong Shen, Shihan Dou, Jun Zhao, Junjie Ye, Qi Zhang, Tao Gui, and Xuanjing Huang. 2024. TransferTOD: A generalizable Chinese multi-domain task-oriented dialogue system with transfer capabilities. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 12750–12771, Miami, Florida, USA. Association for Computational Linguistics.

Qi Zhu, Kaili Huang, Zheng Zhang, Xiaoyan Zhu, and Minlie Huang. 2020. Crosswoz: A large-scale chinese cross-domain task-oriented dialogue dataset. *Trans. Assoc. Comput. Linguistics*, 8:281–295.

A Dataset Example

An example dialogue and the corresponding user goal are provided in Table 10. The user goal is represented in two forms: structural and textual. The textual goal is used within the prompt of the user simulator, while the structural goal is employed for statistical and evaluation purposes. The table illustrates dialogue from the restaurant domain, involving two slots: "food" and "area." These slots are assigned as "preferred" and "multiple" types, respectively. In the first turn of the dialogue, the user specifies their preference for the "food" slot, indicating that "German" is the preferred food type, with "British" as an acceptable alternative. The dialogue agent responds by conducting two retrievals, querying restaurants that serve German or British cuisines. This example highlights the complexity and dynamic nature of the dialogue, which traditional TOD systems are unable to handle due to the sophisticated constraints and multi-faceted interactions.

B Completeness of Slot Types

The 4 complex slot types defined in SQLWOZ (Multiple, Excluded, Preferred, and Conditional), as well as the classical Single type, are functionally complete to cover and express all the possible user intents. The theoretical support behind this is the completeness of Boolean functions in logical propositions (Boole, 1854).

Formally, understanding user intents in TOD systems means expressing user intents in the form of formalized query conditions, which can be treated as logical propositions. In the theory of logical propositions, common logical connectives include AND, OR, NOT, Implication, etc. Correspondingly, in TOD scenarios, 1) the user's requirements for all the mentioned slots must be satisfied, which is the AND relation; 2) the candidate values in Multiple type slots form the OR relation; 3) the Excluded type slots mean the NOT relation; and 4) the Preferred and Conditional type slots mean the Implication relation.

In propositional logic, the set of logical connectives {AND, OR, NOT} is functionally complete, meaning that any propositional formula can be expressed using only these three operations (Post, 1921). More complex connectives, such as Implication, can be defined in terms of AND, OR, and NOT. In task-oriented dialogues, any simple or complex user intents can be expressed with the three basic types: Single, Multiple, and Excluded. SQLWOZ explicitly adds the Preferred and Conditional types to encode priority and condition constraints, thereby increasing both the realism and the difficulty of the benchmark while preserving logical completeness. In summary, the slot-type inventory of SQLWOZ is both theoretically sound and practically expressive, ensuring full coverage of user requirements encountered in real-world task-oriented dialogues.

C Comparsion of Different TOD Datasets

A detailed comparison of SQLWOZ and the other 7 existing task-oriented dialogue datasets is shown in Table 7. The comparison covers multiple dimensions, including the basic numbers and data diversity. SQLWOZ becomes the largest TOD

Dataset	WOZ	MultiWOZ	SGD	STAR	CrossWOZ	RiSAWOZ	TOAD	SQLWOZ
#Domains	1	5	16	13	5	12	11	**5**
#Dialogues	600	8,438	16,142	5,820	5,012	10,000	8,087	**18,365**
#Turns	4,472	115,424	329,964	127,833	84,674	134,580	37,678	**235,354**
Avg. Turns/Dialog	7.5	13.7	20.4	21.71	16.9	13.5	11.4	**12.8**
Avg. Tokens/Turn	11.2	13.1	9.8	11.2	16.2	10.9	10.6	**31.1**
#Uni-grams	1,297	16,336	18,665	12,602	12,502	11,486	10,921	**28,854**
#Bi-grams	7,435	101,440	126,546	92,168	82,491	94,865	90,538	**152,655**
State Style	SV	SV	SV	SV	SV	SV	SV	**SQL**

Table 7: Comparison of our SQLWOZ dataset to other existing task-oriented dialogue datasets. The numbers are based on the respective training sets. In the last line, SV means Slot-Value style.

dataset with 18,365 dialogues, surpassing the previous largest SGD. And SQLWOZ is the only dataset with SQL-style dialogue state expression.

D Distribution of Slot Type

The detailed distribution of slot types on each slot is provided in Table 12. The assignment of specific types to slots is determined based on real-world usage and domain-specific considerations. For certain slots, such as "area" or "price," all five types are applicable. In contrast, for slots with only two possible values, such as "hotel-type" (with "hotel" and "guesthouse"), or "parking" and "internet" (with boolean "yes" and "no" values), multi-valued or exclusive expressions are rarely used, so only the "preferred" and "conditional" types are assigned. Additionally, for slots like "name," "departure," or "leave" (departure time), complex expressions are typically unnecessary, so the "single" type is retained for these slots, avoiding the imposition of more complex types. Overall, the distribution of slot types across the slots is varied, ensuring a high level of diversity and complexity of the dataset.

E Dialogue Validation Details

Table 11 presents a representative example of a complete dialogue validation procedure adopted in SQLWOZ. Firstly, a ground truth reference SQL statement is derived solely from the user goal. It is used to retrieve the full set of candidate venues that satisfy all stipulated constraints. Then, all the generated SQL queries are extracted from the API calling log of the dialogue history. The SQL queries are analyzed, and the queried results of the last most specific SQL statement are used to compare with the candidate venues. Venue validation passes when these two sets intersect, and the name of at least one shared venue is explicitly mentioned in the dialogue. Secondly, if the user's goal also requests specific venue attributes, the corresponding attributes of the hit venues are checked to see whether they appear in the dialogue text. Attribute validation passes if and only if there is at least one venue where all the requested attributes appear in the dialogue. If there is a booking task in the user goal, the booking parameters are checked for the booking validation. Thirdly, an additional validation is conducted to ensure the dialogue is successful and quality, where 1) there are at least 3 pairs of user-agent utterances in the dialogue and 2) there are no more than 3 consecutive repetitions of the same API call. Overall, a dialogue is considered valid and retained in SQLWOZ only when it passes all the venue, attribute (or booking), and additional validations.

F SQL-based Dialogue State Tracking Details

Table 8 presents the scores of a futher SQL-based dialogue state tracking evaluation. Two types of input dialogue history representations are considered: "Utter+API" which includes both dialogue utterances and API call history, and "Utter Only" which only includes utterances to reduce context length. As shown in the table, performance improves significantly with increasing model size. Additionally, the inclusion of API call history ("Utter+API") consistently outperforms the "Utter Only" condition across all experimental settings, highlighting the importance of incorporating historical API calls. Although the few-shot settings show lower performance compared to full-shot training, they still demonstrate considerable potential.

Model (Size)	Full-Shot		Few-Shot (10%)		Few-Shot (1%)	
	Utter+API	Utter Only	Utter+API	Utter Only	Utter+API	Utter Only
GPT-2 (124M)	52.72	51.16	34.41	30.25	22.43	16.15
FlanT5 (770M)	58.80	53.00	46.60	39.25	37.65	28.37
Llama 3.2 (1B)	65.59	55.16	58.82	48.25	52.86	40.59
Llama 3.2 (3B)	68.50	57.14	65.96	54.93	56.41	45.92
Qwen 3 (8B)	69.97	59.08	68.35	57.11	60.21	48.23

Table 8: Evaluation results of the SQL-based dialogue state tracking task. "Utter+API" refers to the full dialogue context, where both history utterances and API calls are provided as input to the agent. "Utter Only" refers to a setting where only the history utterances are provided. For query API calls, SQL statements are considered identical if their resulting datasets match.

User	Agent	Valid Rate	SE	CE
GPT-4o	GPT-4o	92.4%	7.61	3.13
Llama 3B	GPT-4o	88.7%	7.11	2.89

Table 9: Comparison between the two user simulators. GPT-4o means zero-shot user simulator/agent based on the OpenAI GPT-4o model to create the SQLWOZ dataset. Llama 3B means the trained user simulator by the training set of SQLWOZ based on the Llama 3.2 3B model. The scores are reported on the test set of SQLWOZ.

G The Trained User Simulator

In the end-to-end dialogue evaluation, we train a user simulator to interact with dialogue agents offline, in order to avoid the high cost and limited availability of the OpenAI API implementation. The simulator is based on the Llama 3.2 3B model and is trained using dialogues from the SQLWOZ training set. We compare the performance of this trained user simulator with the OpenAI API-based user simulator used during the construction of SQLWOZ. The results, summarized in Table 9, evaluate two key aspects: the validity of the generated dialogues and the linguistic diversity of the generated utterances. As shown in the table, the trained user simulator achieves a comparable dialogue validity rate (88.7% vs. 92.4%) and language diversity to the OpenAI API-based simulator, demonstrating that the trained user simulator is an effective offline alternative to the OpenAI API-based approach.

H The Prompt for the Dialogue Agent

In this section, we present the prompts used by the dialogue agent to construct the SQLWOZ dataset. The dialogue agent is based on the OpenAI GPT-4o model and operates in function-calling mode. The prompt is detailed in Table 13. It begins by outlining the task scenario and the basic action principles, followed by a description of the five domains and a brief overview of the corresponding APIs. Lastly, we provide the generation guidelines for the agent, specifying key instructions regarding the required form of the generated SQL statements and the desired response style.

The APIs used in SQLWOZ are described in detail in separate sections. There are a total of 8 APIs, consisting of 4 querying APIs and 4 booking APIs. An example of a querying restaurant API is shown in Table 14. The API definitions follow OpenAI's format requirements. Since SQL statements serve as the dialogue state representation, the query API has a single parameter, "sql", which accepts a SQL statement string. The description of the query API is slightly longer to provide more details on the available slots, which correspond to the database table columns, for the SQL statements. Moreover, Table 15 presents an example of a booking restaurant API, which outlines the slot and value requirements for making a restaurant booking.

I The Prompt for the User Simulator

The prompt for the user simulator is presented in Table 16. It begins by describing the overall task and the basic principles behind the user simulator. The user goal across all involved domains, is then outlined. This section will be substituted with specific textual goals during the task execution. Finally, the prompt provides guidelines for the user simulator, including the definition of starting and ending signals, the requirement to express all given goals clearly, and the specification of the language style. In our experiments, we encourage

the user simulator to employ slightly more complex expressions to ensure the dataset's difficulty is appropriate.

J Contribution Points

The main contributions of the paper are as follows:

- We point out the problem that the traditional simple slot-value dialogue state representation is not powerful enough to modeling the complex real world user requirements.
- We construct a new TOD dataset SQLWOZ that modeling complex user requirements via SQL statements.
- We model the complex user requirements with four slot types: Multiple, Excluded, Preferred and Conditional, which has been demonstrated to effectively capture the majority of realistic dialogue scenarios.
- We establish comprehensive benchmark experiments on the SQLWOZ dataset by adapting strong LLMs for key TOD tasks, demonstrating the complexity of SQLWOZ and its effectiveness as a challenging benchmark for future TOD research.

Goal

You are planning your trip in Cambridge.
You are looking for a restaurant.
• The restaurant should provide German food primarily, but British food is also acceptable if German food is not offered.
• The restaurant should be in the centre or west of the town.
Once you find the restaurant you want to book a table for 3 people at 16:15 on Wednesday.
Make sure you get the reference number.

```
{
    'restaurant': {
        'info': {
            'food': {'type': 'preferred', 'value': 'german >> british'},
            'area': {'type': 'multiple', 'value': 'center || west'},
        }
        'book': {'day': 'wednesday', 'people': 3, 'time': '16:15'},
    }
}
```

User

Hi there! I'm on the hunt for a restaurant in Cambridge that serves German cuisine. If there are no German options, I'm open to British food too. Can you help me out?

Agent

Call Function: `find_restaurant`
SQL Statement: **`SELECT * FROM restaurant WHERE food = 'german'`**
Result: `No results found.`

Call Function: `find_restaurant`
SQL Statement: **`SELECT * FROM restaurant WHERE food = 'british'`**
Result:

```
| name | area | food | price | address | phone | postcode |
| midsummer house restaurant | centre | british | expensive | Midsummer Common | 01223369...
| the cambridge chop house | centre | british | expensive | 1 Kings Parade | 01223359506 ...
| graffiti | west | british | expensive | Hotel Felix Whitehouse Lane Huntingdon Road | 0...
Total 11 items found.
```

Response: There are several British restaurants in Cambridge. To narrow it down, how about considering the area or price range you're interested in?

User

I'm looking for a place that's either in the center or the west part of town. Can you find something in those areas?

Agent

Call Function: `find_restaurant`
SQL Statement: **`SELECT * FROM restaurant WHERE food = 'british' AND (area = 'centre' OR area = 'west')`**
Result:

```
| name | area | food | price | address | phone | postcode |
| midsummer house restaurant | centre | british | expensive | Midsummer Common | 01223369299...
| the cambridge chop house | centre | british | expensive | 1 Kings Parade | 01223359506 |...
| graffiti | west | british | expensive | Hotel Felix Whitehouse Lane Huntingdon Road | 01...
Total 10 items found.
```

Response: In the center of Cambridge, you have the Midsummer House Restaurant and The Cambridge Chop House, both known for their British cuisine and considered expensive. In the west, there's Graffiti, also offering British food at an expensive price. Would you like to make a reservation at any of these places?

User

Let's go with The Cambridge Chop House in the center. Could you book a table for three people on Wednesday at 16:15? Please make sure to get the reference number for me.

Agent

Call Function: `book_restaurant`
Arguments: `{'day': 'wednesday', 'name': 'the cambridge chop house', 'people': 3, 'time': '16:15'}`
Result: `Booking successful. The reference number is GPF3ZOAL.`

Response: Your table for three at The Cambridge Chop House on Wednesday at 16:15 is successfully booked. The reference number for your reservation is GPF3ZOAL. Enjoy your meal!

User

Great, thank you so much for your help!

Agent

Response: You're welcome! If you need any more assistance, feel free to ask. Enjoy your dining experience!

Table 10: An example dialogue and user goal from the SQLWOZ dataset. The user goal is presented in two forms: structural and textual. The textual goal is employed within the user simulator's prompt, while the structural goal is used for evaluation purposes. The dialogue agent does not have access to the user goal and can only respond to user inputs. The SQL statements generated by the dialogue agent are highlighted in bold.

User Goal	`'restaurant': {`
	`  'info': {`
	`  'food': {'type': 'excluded', 'value': 'NOT gastropub'},`
	`  'price': {'type': 'excluded', 'value': 'NOT cheap'},`
	`  'area': {'type': 'multiple', 'value': 'centre || east'},`
	`  },`
	`  'request': ['postcode', 'phone', 'address'],`
	`}`

Dialogue History	**User**: I want to find a restaurant that's either in the centre or east of Cambridge. Can you help with that? **Call Function**: `find_restaurant` **SQL Statement**: `SELECT * FROM restaurant WHERE area = 'centre' OR area = 'east'` **Result**: ... *(Total 78 items found.)* **Agent**: There are many restaurants in the centre and east of Cambridge. Could you please provide more specific preferences, such as the type of cuisine or price range, to help narrow down the options? **User**: Sure! I'm looking for a restaurant that offers any type of cuisine except gastropub food. **Call Function**: `find_restaurant` **SQL Statement**: `SELECT * FROM restaurant WHERE (area = 'centre' OR area = 'east') AND food != 'gastropub'` **Result**: ... *(Total 74 items found.)* **Agent**: There are still quite a few options. Could you specify a price range or a particular type of cuisine you prefer? That would help narrow down the choices further. **User**: The restaurant should be not in the cheap price range, but any other price range is fine. **Call Function**: `find_restaurant` **SQL Statement**: `SELECT * FROM restaurant WHERE (area = 'centre' OR area = 'east') AND food != 'gastropub' AND price != 'cheap'` **Result**: ... *(Total 58 items found.)* **Agent**: There are still many restaurants that meet your criteria. To narrow it down further, could you specify a particular cuisine you're interested in, or any additional preferences you might have? **User**: No specific cuisine preference, just anything except gastropub food. Could you share the postcode, phone number, and address for one of the options? **Agent**: Certainly! I recommend the Midsummer House Restaurant, which offers British cuisine in an expensive price range. **User**: Could you tell me the postcode, phone number, and address of it? **Agent**: The phone number of the Midsummer House Restaurant is 01223369299 and it's located at Midsummer Common, CB41HA.

Reference SQL	`SELECT * FROM restaurant`
	`WHERE food != 'gastropub' AND food IS NOT NULL`
	`  AND price != 'cheap' AND price IS NOT NULL`
	`  AND area IN ('centre', 'east')`

Candidate Venues	1) midsummer house restaurant, 2) de luca cucina and bar, 3) hotel du vin and bistro, 4) galleria, ... (total 58 venues)

Generated SQLs	`SELECT * FROM restaurant WHERE area = 'centre' OR area = 'east'`
	`SELECT * FROM restaurant WHERE (area = 'centre' OR area = 'east') AND food != 'gastropub'`
	`SELECT * FROM restaurant WHERE (area = 'centre' OR area = 'east') AND food != 'gastropub' AND price != 'cheap'`

Venue Validation	midsummer house restaurant (**Venue Validition Pass!**)
Requested Attributes	`{'postcode': 'cb41ha', 'phone': '01223369299', 'address': 'Midsummer Common'}`
Attribute Validation	postcode: cb41ha, phone: 01223369299, address: Midsummer Common. (**Attribute Validition Pass!**)
Additional Validation	1) At least 3 pairs of user-agent utterances 2) No more than 3 consecutive repetitions of the same API call (**Additional Validition Pass!**)
Validation Conclusion	**Valid Dialogue!**

Table 11: A complete example of validating a dialogue in the SQLWOZ dataset. The validation procedure includes venue validation, attribute/booking validation and additional validation.

Domain	Slot	Single	Multiple	Excluded	Preferred	Conditional
Restaurant	area	12.4%	22.3%	19.0%	13.6%	32.7%
	price	14.4%	23.6%	20.8%	15.1%	26.0%
	food	10.7%	17.9%	15.7%	34.7%	21.0%
	name	100.0%	-	-	-	-
Hotel	area	14.6%	26.1%	21.6%	8.1%	29.7%
	price	12.4%	22.1%	18.6%	23.2%	23.8%
	stars	16.9%	29.2%	22.7%	12.1%	19.1%
	type	46.9%	-	-	35.1%	18.0%
	parking	68.5%	-	-	20.5%	11.0%
	internet	68.8%	-	-	20.0%	11.2%
	name	100.0%	-	-	-	-
Attraction	area	14.1%	22.5%	18.8%	12.6%	32.0%
	type	13.1%	22.7%	17.4%	30.3%	16.5%
	name	100.0%	-	-	-	-
Train	departure	100.0%	-	-	-	-
	destination	100.0%	-	-	-	-
	leave	100.0%	-	-	-	-
	arrive	100.0%	-	-	-	-
	day	34.0%	32.5%	22.3%	11.2%	-
Taxi	departure	100.0%	-	-	-	-
	destination	100.0%	-	-	-	-
	leave	100.0%	-	-	-	-
	arrive	100.0%	-	-	-	-

Table 12: The distribution of slot types for each slot across the five domains. The assignment of slot types is based on real-world usage and domain-specific considerations. Some slots, such as 'area' and 'price,' support all five types, while others are assigned only specific types.

Dialogue Agent Prompt

You are an intelligent task-oriented dialogue agent that can help users to find restaurants, hotels, attractions, trains, and taxis in Cambridge. You can also make reservations or book tickets for the user. You can provide information about the venues and transportations and help the user to make reservations or book tickets. You can also provide recommendations based on the user's constraints. You can also ask the user for more constraints if needed.

When finding information of venues or transportations, you should call the provided functions to query the database via SQL statements. The user requirements may be complex and the SQL statements need to accurately reflect user demands. When many items are retrieved, you can ask the user to provide more constraints. When few items are returned, you need to introduce names of the retrieved items to the user with fluent language.

After the user has the wanted venue, you can ask whether the user wants to make a reservation or book tickets. When making reservations or booking tickets, you should call the provided functions to make reservations or book tickets. Make sure to provide all the required parameters to the booking functions. After booking you should provide the user with the unique reference number for the reservation or purchase.

Domains

Domain 1: Restaurant
The agent helps the user find a restaurant and/or make a reservation.
The user provides the constraints of the restaurant for searching, and then provides the reservation constraints.

Use the ‘find_restaurant‘ function to retrieve the restaurants from the database via a SQL statement.
Use the ‘book_restaurant‘ function to book a restaurant with certain requirements.

Domain 2: Hotel
The agent helps the user find a restaurant and/or make a reservation.
The user provides the constraints of the restaurant for searching, and then provides the reservation constraints.

Use the ‘find_hotel‘ function to retrieve the hotels from the database via a SQL statement.
Use the ‘book_hotel‘ function to book a hotel with certain requirements.

Domain 3: Attraction
The agent helps the user to find an attraction to visit. The user provides the constraints of the attraction for searching.

Use the ‘find_attraction‘ function to retrieve the attractions from the database via a SQL statement.

Domain 4: Train

Continued from previous page

Dialogue Agent Prompt

The agent helps the user wants to find a train to take and/or buy train tickets.
The user provides the constraints of the train for searching and then specify the number of tickets to buy.

Use the ‘find_train‘ function to retrieve the trains from the database via a SQL statement.
Use the ‘buy_train_tickets‘ function to buy train tickets with certain requirements.

Domain 5: Taxi
The agent helps the user wants to find a taxi to take.
The user provides the constraints for the taxi.

Use the ‘book_taxi‘ function to book a taxi with certain requirements.

Generation Guidelines

- The query SQL should accurately and strictly correspond to user requirements. When users express preferences or priority constraints, query conditions should strictly follow the priority order. You should use the highest priority condition first or display the results that meet the highest priority first. If there is no result that meets the highest priority, then display the results of the lower priorities. You should not simply connect multiple priority conditions, because this does not guarantee that the higher priority results will be displayed first.

- When the user’s requirements require multiple SQL queries, you should not generate multiple SQL queries at the same time, but perform them sequentially and decide whether to perform subsequent queries based on the results of the previous queries.

- When the user want to find venue by names, the venue name may be inaccurate. When the name provided by the user cannot be found in the database, you should use some approximate search techniques, such as the ‘LIKE‘ operator in SQL. You should reminder the leading "the" or the trailing "restaurant" or "hotel" in venue names.

- For the response to the user, you should generate only one short paragraph of plain text. You should not generate multiple paragraphs. You should not generate Markdown content with emphasis, bold, numbered or bulleted lists.

- If the markdown table returned by the api cannot show all the retrieved items, the total number of items are noted below the table as "Total xx items found." Your following dialogue strategy should based one the number of items retrieved.
- If there are many items retrieved, you MUST ask the user for more constraints to narrow down the search results. If there are few (less than 10) items retrieved, you can recommend the retrieved items to the user with fluent language.
- When there are a few items retrieved, you can recommending venues or transportations. You should use short and fluent spoken language to recommend the names to the user. You can introduce one or two similarities or differences of the venues but you should not tell much details of them. You must not simply list the retrieved data by bulleted or numbered list. The response should not be long.

Continued from previous page

Dialogue Agent Prompt

```
- If the user updates the requirements and there are folded items in previous api
results (noted by "Total xx items found."), you should conduct a new query with the
latest constraints to get the updated results.  You should not simply filter the
previous results with the new constraints.
```

Table 13: The Prompt of the dialogue agent.

Query Function Example

```
RESTAURANT_QUERY_DESCRIPTION = '''Retrieve the restaurants from the database via
a SQL statement.

The table name is `restaurant` and there are only 4 columns that can be
constrained in the WHERE clause:
- area: the location of the restaurant. only allowed values: north, south, east,
west, centre.
- price: the price range of the restaurant. only allowed values: cheap,
moderate, expensive.
- food: the food type or cuisine of the restaurant. example values: chinese,
indian, italian, japanese, mexican, thai, vietnamese, french, spanish, turkish,
american, british, middle eastern, asian, european...
- name: the name of the restaurant.

The SQL statement should select all the columns from the `restaurant` table with
the beginning of `SELECT * FROM restaurant`.
'''

RESTAURANT_QUERY_FUNCTION = {
    'name': 'find_restaurant',
    'description': RESTAURANT_QUERY_DESCRIPTION,
    'parameters': {
        'type': 'object',
        'properties': {
            'sql': {
                'type': 'string',
                'description': 'The SQL statement to retrieve the restaurants.',
            },
        },
        'required': ['sql'],
        'additionalProperties': False,
    },
}
```

Table 14: The example of a query function for the dialogue agent.

Booking Function Example

```
RESTAURANT_BOOK_FUNCTION = {
    'name': 'book_restaurant',
    'description': 'Book a restaurant with certain requirements',
    'parameters': {
        'type': 'object',
        'properties': {
            'name': {
                'type': 'string',
                'description': 'The name of the restaurant to book.',
            },
            'people': {
                'type': 'integer',
                'description': 'The number of people.',
            },
            'day': {
                'type': 'string',
                'enum': ['Monday', 'Tuesday', 'Wednesday', 'Thursday',
                         'Friday', 'Saturday', 'Sunday'],
                'description': 'The day when the people go in a week.',
            },
            'time': {
                'type': 'string',
                'description': 'The time of the reservation. The time should be in
                               24-hour format HH:MM.',
            },
        },
        'required': ['name', 'people', 'day', 'time'],
        'additionalProperties': False,
    }
}
```

Table 15: The example of a booking function for the dialogue agent.

User Simulator Prompt

You play the user role and talk to a task-oriented dialogue agent to complete some tasks.

You goal in your mind is below. The goal consists of several sub-goals in different domains, such as restaurant, hotel, attraction, train, and taxi. You should carefully understand the goal, then talk with the dialogue agent and gradually express the intents in the goal turn by turn. Your purpose is to achieve the goal as much as possible.

You tend to express complex utterances to validate the ability of the agent. You like to express complex constraints that the agent is hard to query with one simple query. You like to express requirements containing preferences, priorities or conditions. You tend to express all the candidates values in a complex constraint in one turn. But you should not express multiple constraints in one turn.

You should imagine the scenario that you are a tourist in Cambridge. You are happy and excited to this city and want to find some venues. You can imagine more details based on the goal, such as your personality, hobbies, and why you are looking for these venues. Besides proposing the constraints, you can also add some small talk or tell more details in the utterances to express your emotions or feelings.

Goal

You are looking for information in Cambridge.

Domain: Train
You are looking for a train.
- The train should go to ely and leave on monday.
- The train should leave after 21:45 and depart from cambridge.
Once you find the train you want to make a booking for 8 people.
Make sure you get the reference number.

Domain: Restaurant
You are also looking for a place to dine.
- If the restaurant is in the centre of the town, it should be in the expensive price range. If it is in in other areas, it should be in any price range except expensive.
- The restaurant should offer any cuisine other than chinese food.
Make sure you get phone number, address, and postcode.

Guidelines

- In the beginning, you will receive a fix message that "Dialogue Begins." to indicate the start of the dialogue. Then you should generate your first message to the dialogue agent based on the goal.

- The utterances generate by the dialogue agent are given to you. The dialogue agent utterances are marked as the role "user". You should generate your response based on the dialogue agent utterances. Your generated utterances are marked as the role "assistant".

User Simulator Prompt

- The preferences, priorities or conditional constraints MUST be expressed in the separate turn. You tend to express complex utterances to validate the ability of the agent. You should express the **entire** constraint in one turn and let the agent handle it by itself, rather thank breaking it to multiple sub-constraints and expressing them separately. Example utterances:

    - "I'm looking for a hotel in the east, and if these is no thus hotel you want a hotel in the north."

    - "The restaurant should serve Beijing food as the first priority, and if not available, either Indian or British food is fine."

    - "The hotel should first be of type guesthouse, but if it is unavailable, any type is acceptable."

    - "I would like to find a moderate price hotel if it is a guesthouse, but if it is a hotel, the price should be cheap."

- Your language should be **short**. Your output should be in a **spoken** style, not a written style. You language should also be diverse and natural.

- You expression should be diverse and natural. You should not simply repeat the sentences in the goal. You should not use the same language pattern in multiple turns. You should not always use the pattern "I'm looking for a...".

- In on turn, you can only express **one** constraint in the bulleted list as most. You should not express multiple constraints in one turn. For example, you should only specify the area in one turn, and specify price in later turns. You should not specify multiple aspects in one turn. You should first find the venue or train you want, then try to ask for some information or book a table.

- Your subsequent requirements should be based on the venues provided to you. You can further propose more constraints if you are not sure whether the recommend venues meets the goal. When the venues provided by the agent already satisfy some of the constraints that have not been proposed, these constraints do not need to be proposed again.

- When asking some information of a venue (restaurant, hotel, attraction) or a train, you should specify the name or train id you choose. When you choose a venue or a train, you should first ensure it meets all the constraints in the goal. Remember that the dialogue agent may not understand well your previously proposed requirements, so it's important to ensure you choose the right venue or train that meets all the constraints in the goal. When the dialogue agent does not provide the right venues or trains, you should remind the dialogue agent to correct.

- Note that the dialogue agent is not perfect. It may make various mistakes. You should talk to the dialogue agent as patiently as possible, remind it to correct when you find the dialogue agent makes mistakes.

Continued from previous page

User Simulator Prompt

- When the goal is completed, you should directly finish the dialogue by outputting a single sentence "Dialogue Ends." in a separate turn without any other contents to indicate the end of the dialogue. You should not engage in unnecessary small talk with the agent.

- When one of the goal in a domain cannot be completed, even after trying all priority constraints, you should directly skip to the goal in the next domain. You should not repeat multiple meaningless attempts.

Table 16: The prompt for the user simulator.