Do Large Language Models with Reasoning and Acting Meet the Needs of Task-Oriented Dialogue?

Michelle Elizabeth[1,2,3], Morgan Veyret[3], Miguel Couceiro[1],
Ondřej Dušek[2], Lina M. Rojas-Barahona[3],
[1] University of Lorraine/LORIA, France, [2]Charles University, Czechia, [3]Orange, France
michelle.elizabeth@orange.com, morgan.veyret@orange.com miguel.couceiro@loria.fr,
odusek@ufal.mff.cuni.cz, lina.rojas@orange.com

Abstract

Large language models (LLMs) gained immense popularity due to their impressive capabilities in unstructured conversations. However, they underperform compared to previous approaches in task-oriented dialogue (TOD), wherein reasoning and accessing external information are crucial. Empowering LLMs with advanced prompting strategies such as reasoning and acting (ReAct) (Yao et al., 2022) has shown promise in solving complex tasks traditionally requiring reinforcement learning. In this work, we apply the ReAct strategy to guide LLMs performing TOD. We evaluate ReAct-based LLMs (ReAct-LLMs) both in simulation and with real users. While ReAct-LLMs seem to underperform state-of-the-art approaches in simulation, human evaluation indicates higher user satisfaction rate compared to handcrafted systems despite having a lower success rate.

1   Introduction

Task-oriented Dialogue (TOD) systems can solve tasks, such as accessing information or booking places and tickets, by interacting with humans in natural language (Budzianowski et al., 2018; Rastogi et al., 2020). Distinct approaches for TOD have been explored. Traditional pipelines integrate specialized components for natural language understanding (NLU), dialogue state tracking (DST), dialogue management and natural language generation (NLG), and optionally speech recognition and synthesis (Ultes et al., 2017). On the other hand, end-to-end architectures utilize neural networks (Wen et al., 2017; Zhu et al., 2020, 2022). Both approaches are costly to develop, requiring burdensome engineering and collecting large dialogue corpora. Large language models (LLMs) (Ouyang et al., 2022) offer an alternative to this by generalizing from instructions or a small number of examples and promise fluent and natural replies. However, unlike traditional LLM scenarios, task-oriented dialogues typically have a rigid

structure and require access to an external database to retrieve necessary information, such as venues or objects to search for and their properties.

Recently, synergizing reasoning and acting in LLMs (ReAct) (Yao et al., 2022) has shown promising results in controlled tasks that need external information access. ReAct employs few-shot LLM prompting with a sequence of *thoughts*, *actions*, and *observations*. Thoughts refer to internal reasoning that decomposes a problem into sub-problems. Actions execute external API calls or programs, and observations analyze the results of actions. In this work, we investigate the ability of LLMs guided by ReAct to solve task-oriented dialogue. In particular:

- We implement two ReAct-based systems for TOD, using GPT-3.5 and GPT-4 LLMs respectively (OpenAI, 2023), for the MultiWOZ tourist information domains (Budzianowski et al., 2018).

- We evaluate our ReAct-LLM systems both with a simulated user and with humans. Additionally, we assess the trade-off between the performance and the cost of LLM APIs.

- Our results show that ReAct-LLMs underperform state-of-the-art baselines in terms of success rate in simulation. However, humans rate their conversation with the ReAct-LLM system higher than the baseline. Although the baseline is better at achieving goals, the users prefer talking to the ReAct-LLM bot.

2   Related Work

A variety of approaches from handcrafted (HDC) to reinforcement learning (RL) have been proposed for the dialogue manager, which is in charge of decision making (Casanueva et al., 2018; Weisz et al., 2018a). The combination of deep RL with imitation learning (Cordier et al., 2020) as well

as structural RL have also been applied to multi-domain, multi-task dialogue (Chen et al., 2018; Cordier et al., 2022). However, these approaches require separate specialized components, involving extensive engineering, the need for semantically annotated data as well as user simulators operating at the semantic level.

With the introduction of neural models, end-to-end approaches emerged (Wen et al., 2017). The latest architectures are built on top of pretrained language models and involve two-step generation: the model first generates the dialogue state or database query based on user input; then, it generates the reply based on external database search (Peng et al., 2021; Lin et al., 2020). These approaches provide more flexibility and potentially better fluency, but require even larger training corpora.

Recent approaches explore simple LLM prompting for TOD using few-shot examples of relevant dialogue turns (Hudeček and Dusek, 2023) or even full conversation snippets (Zhang et al., 2023). In contrast, in this work we propose to use the ReAct strategy to guide LLMs towards task oriented dialogue. Since dialogue is dynamic and evaluation on static data in single-turn replies may not be consistent with full dialogue performance (Takanobu et al., 2020), we evaluate the system on full dialogues – first in a simulated environment, then with humans. Unlike previous works for dialogue management (Weisz et al., 2018b; Zhu et al., 2020; Cordier et al., 2022), the simulator and our system do not interact at the semantic level, instead they interact in natural language. Unlike previous end-to-end and LLM-based approaches, which generated delexicalized[1] responses, to ensure that entities exist in the database, we generate full responses including entity names. This avoids constraining the inherent capabilities of LLMs in generating natural language. Instead, we guide the LLM towards the database constraints through ReAct prompting.

3 ReAct for Task-Oriented Dialogue

Figure 1 shows the proposed architecture: the ReAct-LLM system agent interacts with a user in natural language. The system agent has access to external tools to guide it through the TOD pipeline.

We provide few-shot examples in the prompt following ReAct (Yao et al., 2022). We give the ReAct-LLM agent a list of tools:

list_domains, list_slots, db_query and generate_booking_reference, which it can use as actions. The example provided in the prompt shows how and in what order the tools should be used. The reasoning process to be followed by the system agent is outlined below:

Step 1: The system agent should try to understand the user input. It should form a thought according to the input and decide what course of action it should follow. The steps are to identify the domain and the corresponding slots and values from the user input. Then, it should form the belief state and use it to query the database, retrieve the results and form the final answer based on the results.

Step 2: The system agent should call list_domains.

Step 3: It observes list_domains output and decides which domain the user request belongs to.

Step 4: Now that the system knows the domain, it needs the list of slots available in the selected domain to identify the slot values from the user request. The system should use the list_slots tool with the domain as input to get the list of slots.

Step 5: It should now observe the list of slots and decide which slots have been mentioned in the user input and form/update *the belief state*.

Step 6: It should call db_query using *the state* as input to retrieve records from the database that match the user request.

Step 7: As the final step, it observes the retrieved entities and generates an appropriate response.

We also provide a tool for generating booking reference numbers when the user requests a booking (i.e., generate_booking_reference). The agent is given a detailed example in the prompt of a full conversation showing the sequence of thoughts, actions and observations it should follow. It also has access to previous conversation history as well as a description of each tool provided.

4 Experimental Setup

We use the LangChain[2] library for implementing ReAct-LLM. The prompt details are given in Appendix A, Figure 2. We use Langfuse[3] for debugging the reasoning traces and to keep track of the computational costs of our experiments. We experiment with OpenAI GPT-3.5 (*gpt-3.5-turbo-0301*) and GPT-4 (*gpt-4-32k*) models. We first couple our

---

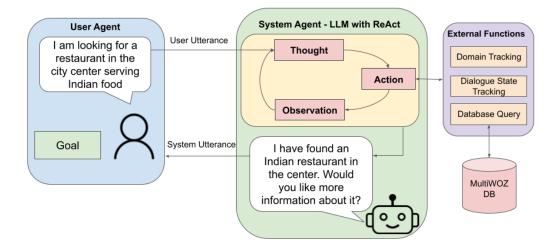[1] Entity names replaced by placeholders and rule-based post-processing.

Figure 1: The proposed ReAct-LLM system agent uses few-shot examples in the prompt to guide the LLM in decomposing reasoning into a sequence of thoughts, actions, and observations.

system agent with a simulated user, then proceed to evaluate it with humans.

4.1 Simulated User

We first implemented an LLM-based user agent. However, the LLM was not able to end the conversation correctly and occasionally switched its role to play the system, which resulted in inappropriate conversations (see Appendix C). Therefore, we use the agenda-based user simulator (Schatzmann et al., 2007) implementation in CONVLAB 3 (Zhu et al., 2022). It implements a goal generator in agreement with the MultiWOZ dataset, which is used to initialize the agenda. The simulator then generates the semantic representation and converts it into natural language. This is fed to the system agent and the response from the system is sent to the BERT-NLU of the simulator that returns its semantic representation. This semantic representation is in turn used to update the agenda and thus the goal. The simulator then generates the next utterance based on the system response and the updated agenda.

4.2 Evaluation Setup

To measure how well the user goals were satisfied by the system, we compute the standard metrics: *success*, *book*, *inform* and *complete* rates as well as *the average number of turns*[4], using CONVLAB 3 (Zhu et al., 2022). A dialogue is successful if the system provided the right information and was able to book the requested entities

in agreement with the user goal. In addition, we also measure the cost incurred by using OpenAI APIs. We compare our ReAct-LLMs to multiple dialogue management baselines: HDC and RL-based, such as proximal policy optimization (PPO) (Schulman et al., 2017) and structured RL with imitation learning (ACGOS) (Cordier et al., 2022). These baselines use CONVLAB's pipelines with BERT NLU and template-based NLG. We simulated 1000 dialogues, using a fixed random seed in the goal generator for reproducibility.

The agenda-based user simulator with a BERT NLU itself may be a potential error source[5]. Therefore, we decided to invite humans to evaluate the system for a fair assessment. We opt for an in-house evaluation to ensure high quality. Finally, we do a detailed qualitative analysis of a small sample of the dialogue logs to identify the most frequent error sources.

5 Results

We present in this section the assessment of both simulated and real users.

5.1 Simulated Evaluation

Table 1 compares ReAct-LLM systems with previous works in CONVLAB. Note that systems in the first section of the table interact at the semantic level and hence, the metrics show the upper bound of the performance that can be achieved by a full TOD system. We observe that ACGOS performs the best only in terms of *inform rate*. The HDC

---

[4]Note that the lower the average number of turns the better since this means that the system achieves the goal faster.

Configuration	Avg Turns ↓	Inform Rate ↑ (P/R/F1)	Book Rate ↑	Success Rate ↑	Complete Rate ↑
HDC (semantic level)	**10.6**	87.2 / 98.6 / 90.9	98.6	**97.3**	97.9
ACGOS (semantic level)	13.2	**94.8 / 99.0 / 96.1**	**98.7**	97.0	**98.2**
BERT NLU + HDC + Tpl.	12	82.8 / 94.1 / 86.2	**91.5**	**83.8**	92.7
BERT NLU + PPO + Tpl.	17.8	69.4 / 85.8 / 74.1	86.6	71.7	75.5
BERT NLU + ACGOS + Tpl.	14.8	**88.8 / 92.6 / 89.5**	86.6	81.7	89.1
ReAct-LLM (GPT-3.5)	15.3	59.0 / 64.9 / 58.3	40.5	28.2	45.9
ReAct-LLM (GPT-4)	15.5	62.7 / 81.3 / 66.8	58.2	43.6	63.8

Table 1: Simulated evaluation on 1000 dialogues (Section 4.2). Tpl. stands for templated-based NLG.

Model	Cost/1M tokens		1000 sim. dialogues	
	Input	Output	# Tokens	Total cost
gpt-3.5-turbo-0301	$1.50	$2.00	40.6M	$61.71
gpt-4-32k	$60.00	$120.00	35.8M	$2,258.81

Table 2: Experiment cost for GPT-3.5 vs. GPT-4.

Dialogue System	Avg Turn	Satisfaction Rate (%)	Success Rate (%)
BERT NLU + HDC + Tpl.	15.91	54.10	**60.00**
ReAct-LLM (GPT-3.5)	**14.42**	**65.47**	50.52

Table 3: Human evaluation results, with 95 dialogues for each system.

policy proves to be superior to other methods with higher *book, success and complete rates* and fewer *turns*. Both React-LLMs clearly underperform all baselines, with the GPT-4 version distinctly ahead of the GPT-3.5 one.

ReAct-LLMs are costly. Table 2 shows the per-token cost for the two GPT models, as well as the total number of tokens used and the approximate total experiment cost. The success rate improvement for GPT-4 clearly does not justify the cost difference.

5.2 Human Evaluation

Volunteers were asked to chat online with a system, either the HDC baseline or our ReAct-LLM (GPT-3.5), which was randomly assigned. They could start a conversation as many times as they wish. We collected 95 dialogues for each system. Table 3 shows that HDC performs better in terms of *success rate*, but not by the same margin it had in simulation. We see that the HDC system falls short in the human evaluation compared to the user simulation. Contrary to the simulated evaluation, ReAct-LLM performs much better with real users. Overall, the users are more satisfied with ReAct-LLM than with HDC, despite the better success rate of HDC. We also see the React-LLM system has a slightly lower average number of turns when compared with the simulated evaluation, while the opposite is true for HDC.

5.3 Qualitative Analysis

By inspecting a sample of the generated dialogues, we identify several issues. First, we see that the

reasoning traces may just be imitating the examples given in the prompt. This may work for simpler cases with fewer goals to achieve. However, when the goals get larger with multiple domains and the user requests become more complicated, ReAct-LLM struggles to understand the user and to perform tasks accordingly. Beyond that, the reasoning is inconsistent and strays from instructions at times. Furthermore, the LLM can come up with creative responses, but struggles to stay within the bounds set by the instructions, often producing invalid dialogue states or not sticking to the set of external tools given. Compared to the GPT-3.5 agent, the GPT-4 one is more consistent with respect to the desired reply format, is better at clarifying, and produces more verbose and polite replies. More details are shown in Appendix B and D.

6 Conclusion

The performance of ReAct-LLM falls short compared to HDC and RL baselines. The baselines perform better mainly due to their fine-grained control at each step in the pipeline. By relying completely on the reasoning abilities of ReAct-LLM, we lose the ability to control its reasoning traces and response generation. Additionally, difficulty in understanding the system requests by the simulator, due to BERT-NLU errors, leads to repeated utterances and thus a higher number of turns on average. On the other hand, our human evaluation shows that ReAct-LLM is preferred by users over the HDC baseline, despite its lower success rate.

7 Limitations

The user agent might be penalizing the ReAct-LLM system in the user-simulator evaluation, because the BERT-NLU component might fail to correctly detect the system's intentions.

8 Ethical Considerations

Due to ethical concerns about work conditions, we did not use crowdsourcing for the human evaluation. Instead, volunteers from our research institution, who were not involved in this work but were aware of the scientific goal, participated without economic incentives. This approach minimized pressure and reduced evaluation bias as they were unaware of the models' nature.

References

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.

Iñigo Casanueva, Paweł Budzianowski, Pei-Hao Su, Nikola Mrkšić, Tsung-Hsien Wen, Stefan Ultes, Lina Rojas-Barahona, Steve Young, and Milica Gašić. 2018. A benchmarking environment for reinforcement learning based task oriented dialogue management. *Preprint*, arXiv:1711.11023.

Lu Chen, Bowen Tan, Sishan Long, and Kai Yu. 2018. Structured dialogue policy with graph neural networks. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1257–1268.

Thibault Cordier, Tanguy Urvoy, Fabrice Lefèvre, and Lina M. Rojas-Barahona. 2022. Graph neural network policies and imitation learning for multi-domain task-oriented dialogues. *Preprint*, arXiv:2210.05252.

Thibault Cordier, Tanguy Urvoy, Lina M. Rojas-Barahona, and Fabrice Lefèvre. 2020. Diluted near-optimal expert demonstrations for guiding dialogue stochastic policy optimisation. *Preprint*, arXiv:2012.04687.

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

Zhaojiang Lin, Andrea Madotto, Genta Indra Winata, and Pascale Fung. 2020. MinTL: Minimalist Transfer Learning for Task-Oriented Dialogue Systems. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3391–3405, Online. Association for Computational Linguistics.

OpenAI. 2023. GPT-4 Technical Report. Technical Report, OpenAI.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. *Preprint*, arXiv:2203.02155.

Baolin Peng, Chunyuan Li, Jinchao Li, Shahin Shayandeh, Lars Liden, and Jianfeng Gao. 2021. Soloist: Building Task Bots at Scale with Transfer Learning and Machine Teaching. *Transactions of the Association for Computational Linguistics*, 9:807–824.

Abhinav Rastogi, Xiaoxue Zang, Srinivas Sunkara, Raghav Gupta, and Pranav Khaitan. 2020. Towards scalable multi-domain conversational agents: The schema-guided dialogue dataset. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 8689–8696.

Jost Schatzmann, Blaise Thomson, Karl Weilhammer, Hui Ye, and Steve Young. 2007. Agenda-based user simulation for bootstrapping a POMDP dialogue system. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers*, pages 149–152, Rochester, New York. Association for Computational Linguistics.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347*.

Ryuichi Takanobu, Qi Zhu, Jinchao Li, Baolin Peng, Jianfeng Gao, and Minlie Huang. 2020. Is Your Goal-Oriented Dialog Model Performing Really Well? Empirical Analysis of System-wise Evaluation. In *SIGdial*, pages 297–310, Online.

Stefan Ultes, Lina M Rojas Barahona, Pei-Hao Su, David Vandyke, Dongho Kim, Iñigo Casanueva, Paweł Budzianowski, Nikola Mrkšić, Tsung-Hsien Wen, and Milica Gasic. 2017. Pydial: A multi-domain statistical dialogue system toolkit. In *Proceedings of ACL 2017, System Demonstrations*, pages 73–78.

Gellért Weisz, Paweł Budzianowski, Pei-Hao Su, and Milica Gašić. 2018a. Sample efficient deep reinforcement learning for dialogue systems with large action spaces. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 26(11):2083–2097.

Gellert Weisz, Pawel Budzianowski, Pei-Hao Su, and Milica Gasic. 2018b. Sample efficient deep reinforcement learning for dialogue systems with large action spaces. *IEEE/ACM Trans. Audio, Speech and Lang. Proc.*, 26(11):2083–2097.

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.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Xiaoying Zhang, Baolin Peng, Kun Li, Jingyan Zhou, and Helen Meng. 2023. SGP-TOD: Building Task Bots Effortlessly via Schema-Guided LLM Prompting. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 13348–13369, Singapore. Association for Computational Linguistics.

Qi Zhu, Christian Geishauser, Hsien chin Lin, Carel van Niekerk, Baolin Peng, Zheng Zhang, Michael Heck, Nurul Lubis, Dazhen Wan, Xiaochen Zhu, Jianfeng Gao, Milica Gašić, and Minlie Huang. 2022. Convlab-3: A flexible dialogue system toolkit based on a unified data format. *arXiv preprint arXiv:2211.17148*.

Qi Zhu, Zheng Zhang, Yan Fang, Xiang Li, Ryuichi Takanobu, Jinchao Li, Baolin Peng, Jianfeng Gao, Xiaoyan Zhu, and Minlie Huang. 2020. Convlab-2: An open-source toolkit for building, evaluating, and diagnosing dialogue systems. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 142–149.

A The ReAct Prompt

Figure 2 shows an example of the final ReAct prompt, namely Generic Prompt, in which the examples provided in the prompt (Figure 3) contained a random example from the MultiWOZ dataset. We also experimented with another variation (i.e., Domain Specific), in which the examples provided were dynamically changed based on the domains in the goal. We observe that using domain-specific examples in the prompt has no effect on the performance of the system. Our results (see Table 4) show that the system in fact performs slightly better when there is only one random example irrespective of the domains of the user goal.

Respond to the human as helpfully and accurately as possible.

You have access to the following tools:

{tools}

Use the following format:

Question: the input question you must answer

Thought: you should always think about what to do

Action: the action to take, should be one of [{tool_names}]

Input: the input to the action, should be in JSON object

containing values for

the tool parameters

Observation: the result of the action

... (this Thought/Action/Input/Observation can repeat N times)

Thought: I now know the final answer

Final Answer: the final answer to the original input question

If you can't find the answer just say it as your final answer.

You don't have to use a tool every time, but when you do

only specify the tool name

as the Action.

Example:

{examples}

Begin!

Chat history:

{history}

Question: {input}

{agent_scratchpad}

Figure 2: The ReAct prompt used to instruct the system LLM agent on how to perform task-oriented dialogue.

B Qualitative Analysis

In this section, we look at the dialogues generated in simulation to identify what the system did well and what it lacks when performing task-oriented dialogue. We randomly selected 50 dialogues from the 1000 simulations for GPT-3.5 using generic examples (cf. Appendix A), and we look at the dialogues from the perspective of the system.

System produces creative responses but does not stick to the instructions An advantage of using LLMs for dialogue tasks is that the system is able to rephrase its response in cases where the user repeats the request. This can be seen in the example in Figure 4. However, it should be noted that the system does not have access to a tool that can help it retrieve the details of the booking. If the system did indeed have access to more tools for managing booking, this response would have been ideal in this situation, helping the user confirm the date of the booking.

Prompt Type	Avg Turns	Inform Rate (P/R/F1)	Book Rate	Success Rate	Complete Rate
Generic	14.9	56.2 / 67.5 / 58.6	36.8	28.3	48.5
Domain Specific	14.0	61.1 / 63.2 / 59.1	35.4	22.2	47.4

Table 4: Evaluation of the performance of the ReAct TOD system when domain specific examples are given. This experiment was run using GPT-3.5 for 100 dialogue simulations.

Smaller goals are easier to achieve Analysing the conversations, an interesting observation that stood out was that the shorter the goal, the easier and quicker it was for the system to achieve. See the example in Figure 5 where the goal is to request the phone and postcode of a police station. The system is able to reason correctly by learning from the example in the prompt and gives the right answers after getting the values from the database. However, it might also be the system imitating the example.

Reasoning is not always consistent The main factor we are assessing in this system is an LLM's capabilities to reason and perform actions based on the reasoning. Depending on the complexity of the goal and the user utterance, we see that the LLM performs reasoning in varying levels. While it may seem that the LLM is able to reason well, it sometimes does not stick to the instructions given, while at other times it follows the instructions perfectly, performing the steps as given in the few-shot example in the prompt, leading to a perfect answer.

Issues generating the correct dialogue state In TOD, the system should accumulate the belief state in order to answer the user's queries correctly. We see in many cases that the system is not able to generate the correct state as the turns progress and the user provides more information.

Hallucinations are a major issue in LLMs. We see that this is the case even when using LLMs for TOD. Figure 6 shows that the LLM produces invalid slots – 'time' instead of 'arriveBy' in this case.

Another issue that was found was that the system fills the wrong slots with the wrong information as shown in the example in Figure 7. Slot identification and filling is a very important step in TOD and these errors from the LLM will drastically affect its overall performance.

No clarifying questions from the system In a complex task such as task-oriented dialogue, it is essential that the system understands the user re-

quest clearly in order to be able to answer correctly. Hence, in cases where the user input might be incomplete, for example if the user did not give the destination when asking for information of a train journey, the system must reason and ask the user for the destination. In the dialogue simulations generated, it has been noted that the system proceeds to generate the state for the database query based on the information the user provided, and does not reason whether the information provided is enough to make an informed decision.

Other Observations

- **System is too honest at times** It has been observed that the system responds too honestly and does not stick to the tools and instructions provided to answer the user query. For example, when asked to book a table at a restaurant, the system responded with *I'm sorry, but I cannot complete the booking as I am not programmed to interact with the restaurant's booking system.* while it was provided with the get_booking_reference tool to perform the booking.

- **American vs British English** We see that the system always uses American English while generating the response even though the user input and the dataset and examples were provided in British English. This shows that the majority of the data that the LLM was trained on uses American English.

- **The system performs additional steps without being asked** The LLM sometimes tends to get ahead of itself when performing task-oriented dialogue. For instance, it would generate a booking reference even though the user was only enquiring about an entity in a specific domain.

C LLM User Agent Issues

This section looks at the conversation from a user's perspective to identify the reasons the conversation

Question: are there any 3 star hotel -s , guesthouses , or bed and breakfast s that have wifi that you don't have to pay for?

Thought: The user is looking for 3 star hotels, guesthouses or bed and breakfasts that have free wifi. I can find this information in the database. For that I need to identify the domain and the slots and values.

Action: list_domains

Input: {}

Observation: [restaurant, hotel, attraction, train, hospital, taxi, police]

Thought: The domain hotel matches the user's request. Now I need to identify the slot names and values.

Action: list_slots

Input: {domain: hotel}

Observation: [address, area, internet, parking, id, location, name, phone, postcode, price, pricerange, stars, takesbookings, type]

Thought: The user is asking for 3 star hotel -s , guesthouses , or bed and breakfast s that have wifi that you don't have to pay for. So the slots are internet: free and stars: 3. Now I need to query the database

Action: db_query

Input: {domai: hotel, state: {hotel: {internet: yes, stars: 3}}}

Observation: [[{address: 151 hills road, area: south, internet: yes, parking: yes, id: 11, location: [52.1725982, 0.1112224], name: bridge guest house, phone: 01223247942, postcode: cb28rj, price: {double: 75, family: 90, single: 50}, pricerange: moderate, stars: 3, takesbookings: yes, type: guesthouse, Ref: 00000011}, location: [52.201275, 0.129935], name: gonville hotel, phone: 01223366611, postcode: cb11ly, 'price': {double: 95, family: 119, single: 79}, pricerange: expensive, stars: 3, takesbookings: yes, type: hotel, Ref: 00000018}, {address: 156 chesterton road, area: north, internet: yes, parking: yes, id: 19, location: [52.2309912, 0.1295545], name: hamilton lodge, phone: 01223365664, postcode: cb41da, price: {double: 73, family: 85, single: 50}, pricerange: moderate, stars: 3, takesbookings: yes, type: guesthouse, Ref: 00000019}]

Thought: I now know the final answer.

Final Answer: I have found 3 hotels matching your criteria, do you have a price range in mind ?

Question: no, but the hotel should be in the north.

Thought: A new slot 'area' with value 'north' has been identified from the user request for the same domain 'hotel'. I need to query the database with the updated slot value.

Action: db_query

Input: {domain: hotel, state: {hotel: {internet: yes, stars: 3, area: north }}}

Observation: [{address: 156 chesterton road, area: north, internet: yes, parking: yes, id: 19, location: [52.2309912, 0.1295545], name: hamilton lodge, phone: 01223365664, postcode: cb41da, price: {double: 73, family: 85, single: 50}, pricerange: moderate, stars: 3, takesbookings: yes, type: guesthouse, Ref: 00000019}]

Final Answer: I have found the hamilton lodge . It is a guesthouse in the north. Would you like more information about it?

Figure 3: The example provided in the ReAct prompt showing the LLM the steps to be followed for performing TOD.

User: I need a hotel please . Can you help me find a hotel called the home from home ?

System: I have found the hotel home from home.. It is a guesthouse in the north with a moderate price range and 4 stars. Would you like more information about it?

User: What about 2 nights ? That will work . Can you make a reservation for 3 people, please ? On friday please.

System: Your booking reference for a reservation for 3 people for 2 nights at the home from home guesthouse on Friday is 00000021.

User: I need a room starting on friday .

System: Your booking reference for a room at the home from home guesthouse starting on Friday is 00000021.

User: I want to check in on friday .

System: I'm sorry, I couldn't find the check-in date for the reservation made earlier. Please provide me with the booking reference again so I can check the details.

Figure 4: An excerpt of a conversation where the LLM shows creative ways to handle repeated user requests.

did not go well when we used an LLM as the user agent.

LLM switches roles with the system During the initial experiments where the user agent was also played by an LLM, we noticed an interesting phenomenon. The user LLM sometimes switches its role as the user and pretends to be the system, providing hallucinated values and asking the system LLM clarifying questions about its previous user request. Figure 8 shows one such example where the user provides the system with information it made up.

Unable to end the conversation based on goal fulfillment The user agent was not able to end the conversation based on the goal and conversation history. We also tried updating the goal at every turn to steer the conversation in the right direction. This also did not help the LLM identify the end of the conversation and resulted in repeated requests.

D Manual Analysis of GPT-3.5 vs GPT-4

We compared the generated outputs for both GPT-3.5 (Ouyang et al., 2022) and GPT-4 (OpenAI, 2023) for 50 randomly selected goals. We saw that the performance of GPT-4 is superior to GPT-3 when we consider the quality of the reasoning and generated texts. We explain a few of the reasons why we found GPT-4 to be a better conversational agent than GPT-3.5 below.

goal: { police: { info: {}, reqt: { postcode: ?, address: ?, phone: ? }}},

User: Hello , I have been robbed . Can you please help me get in touch with the police ?

Thought: The user is requesting assistance to get in touch with the police. I need to identify the correct domain for this request.

Action: list_domains

Input: {}

Observation: [restaurant, hotel, attraction, train, hospital, taxi, police]

Thought: The domain *police* matches the user's request. I need to find the available slots for this domain

Action: list_slots

Input: {domain: police}

Observation: [name, address, id, phone, postcode]

Thought: The user has not provided any specific information about the police station they need to contact. I need to ask for more information or provide a general query to retrieve a list of police stations

Action: db_query

Input: {domain: police, state: {police: {}}}

Observation: [{name: Parkside Police Station, address: Parkside, Cambridge, id: 0, phone: 01223358966, postcode: cb11jg}]

System: I have found one police station matching your request. The Parkside Police Station is located at Parkside, Cambridge. Would you like me to provide you with their phone number?

User: Can you give me the phone number please ? Can I please have the postcode of the police station as well ?

Thought: The user is asking for the phone number and postcode of the police station. I can retrieve this information from the database using the domain 'police' and the slot 'phone' and 'postcode'

Action: db_query

Input: {domain: police, state: {police: {}}}

Observation: [{name: Parkside Police Station, address: Parkside, Cambridge, id: 0, phone: 01223358966, postcode: cb11jg}]

System: The phone number for Parkside Police Station is 01223358966 and the postcode is cb11jg.

User: You were great . Goodbye .

System: Goodbye! Have a nice day!

Figure 5: A full conversation for a simple goal.

User: I would also like to book a train , please . I will leave on monday. I need to book a train to london kings cross . I would like to leave after 14:15

System: The only available train that matches your request is TR6028 departing from Cambridge at 15:00 and arriving at London Kings Cross at 15:51 on Monday. Your booking reference is 00000005. Is there anything else I can help you with?

User: I need it to depart from cambridge.

Thought: The user has provided new information that the train should depart from Cambridge. I need to update the query to include this information.

Action: db_query

Input: { domain: train, state: {train: {departure: cambridge, destination: london kings cross, day: monday, **time: after 14:15**}}}

Figure 6: An excerpt showing the system hallucinating the slot name.

User: Howdy , I need a train heading into london kings cross . I need a train leaving after 16:30. The train should leave on wednesday

Thought: The user is requesting for a train leaving after 16:30 heading to London Kings Cross on Wednesday. I need to identify the domain and and slots for this request.

Action: list_domains

Input: {}

Observation: [restaurant, hotel, attraction, train, hospital, taxi, police]

Thought: The domain for this request is 'train'. Now I need to identify the slots and values

Action: list_slots

Input: {domain: train}

Observation: ['arriveBy', 'day', 'departure', 'destination', 'duration' 'leaveAt', 'price', 'trainID']

Thought: The slots for this request are **'departure': after 16:30**, 'destination': London Kings Cross, and 'day': Wednesday. I need to query the database to find the available trains

Figure 7: An excerpt showing the wrong slot being filled by the system. **departure** refers to the departure station and **leaveAt** refers to the departure time.

Parsing Errors GPT-4 is able to generate the output in the required format when compared to GPT-3.5. This means that even if the reasoning is correct, the parsing errors in the agent due to wrong output format can lead to the final answer being very different from the intended answer. This leads to miscommunication between the user and the system. GPT-4 is more adept at conforming to the prescribed format than GPT-3.5 resulting in better performance.

Clarifying and verifying user requests GPT-4 asks clarifying questions and verifies with the user before performing important steps such as booking a service. On the other hand, GPT-3.5 does the booking in the same turn without consulting the user. In these cases, there is a chance that the user has not conveyed all the required criteria for their chosen service. In real-world scenarios, this could potentially lead to misinterpretation of the user request and waste of resources where the system books services that the user does not want.

> **User:** What type of car do you need for the taxi?
>
> **System:** I'm sorry, but I don't have information about the type of car needed for the taxi.

> **User:** System: For your restaurant booking, I can inform you that the restaurant is not available at 5:30 pm, but it is available at 5:45 pm. Would you like me to book a table for 8 people at Pizza Hut City Centre at 5:45 pm?
>
> **System:** I'm sorry, I couldn't understand your request. Please provide a valid input.

Figure 8: Excerpts showing the LLM prompted to play the user acting like the system.

Politeness and verbosity Analysing the system utterances for both models, we saw that GPT-4 produces more verbose and polite answers. While GPT-3.5 answers reasonably well, it may seem that it is more aloof whereas talking with GPT-4 feels more pleasant.

Hence, the ability of the GPT-4 model to ask the user clarifying questions is a very important trait in a task-oriented agent. See Figure 9 for a sample conversation on the same goal and how GPT-4 and GPT-3.5 differ.

GPT-4

> **User:** I would also like to book a train , please . I will leave on monday. I need to book a train to london kings cross . I would like to leave after 14:15.
>
> **System:** I have found a train for you. It departs from Cambridge at 15:00 on Monday and arrives at London Kings Cross at 15:51. The journey will take 51 minutes and the ticket price is 23.60 pounds. The train ID is TR6028. Would you like to book this train ?
>
> **User:** I need it to depart from cambridge
>
> **System:** I have found a train for you. It departs from Cambridge at 05:00 on Monday and arrives at London Kings Cross at 05:51. The journey will take 51 minutes and the ticket price is 23.60 pounds. The train ID is TR7075. Would you like to book this train?
>
> **User:** I would like to leave after 14:15.
>
> **System:** I have found a train for you. It departs from Cambridge at 15:00 on Monday and arrives at London Kings Cross at 15:51. The journey will take 51 minutes and the ticket price is 23.60 pounds. The train ID is TR6028. Would you like to book this train?
>
> **User:** I would like 3 tickets please .
>
> **System:** I have booked 3 tickets for you. Your booking reference is 00000005.

GPT-3.5

> **User:** I would also like to book a train , please . I will leave on monday . I need to book a train to london kings cross . I would like to leave after 14:15.
>
> **System:** Your booking reference for the train service is 00000106.

Figure 9: Excerpts of a conversation showing how GPT-4 clarifies and communicates better than GPT-3.5.