Few-Shot Adaptation for Parsing Contextual Utterances with LLMs

Kevin Lin [†*] **Patrick Xia** [‡] **Hao Fang** [‡]
[†] UC Berkeley [‡] Microsoft Semantic Machines
k-lin@berkeley.edu {patrickxia, hao.fang}@microsoft.com

Abstract

We evaluate the ability of semantic parsers based on large language models (LLMs) to handle contextual utterances. In real-world settings, there typically exists only a limited number of annotated contextual utterances due to annotation cost, resulting in an imbalance compared to non-contextual utterances. Therefore, parsers must adapt to contextual utterances with a few training examples. We examine four major paradigms for doing so in conversational semantic parsing *i.e.,* Parse-with-Utterance-History, Parse-with-Reference-Program, Parse-then-Resolve, and Rewrite-then-Parse. To facilitate such cross-paradigm comparisons, we construct *SMCalFlow-EventQueries*, a subset of contextual examples from SMCalFlow with additional annotations. Experiments with in-context learning and fine-tuning suggest that Rewrite-then-Parse is the most promising paradigm when holistically considering parsing accuracy, annotation cost, and error types.

1 Introduction

A key challenge in conversational semantic parsing (CSP) is handling *contextual* utterances (*i.e.,* utterances that can only be understood with its context) by mapping them to *non-contextual* programs that can be fulfilled by an executor without relying on the dialogue state. Many approaches have been proposed, *e.g.,* directly mapping the contextual utterance with utterance history to a non-contextual program (Suhr et al., 2018), or mapping to an intermediate contextual program which is then resolved (usually in a deterministic manner) to a non-contextual program (Semantic Machines et al., 2020; Cheng et al., 2020). In these prior works, there is often an assumption of having a substantial corpus of annotated data encompassing both non-contextual utterances and contextual utterances for training a parser. However, in practice, it is more expensive to collect and annotate contextual utterances compared to non-contextual utterances, due to the dependency on the conversation history. Furthermore, annotating non-contextual utterances usually precedes annotating contextual utterances. To reflect such real-world settings, we study few-shot adaptation for parsing contextual utterances, where we first build a parser using a large number of annotated non-contextual utterances, and then adapt it for parsing contextual utterances using a few (or even zero) annotated contextual utterances.

Recent work has shown that large language models (LLMs) are capable of semantic parsing using a few examples (Shin et al., 2021; Shin and Van Durme, 2022). Hence, in this work, we conduct a focused study on few-shot adaptation using LLMs for CSP. Specifically, we consider four major paradigms: Parse-with-Utterance-History, Parse-with-Reference-Program, Parse-then-Resolve, and Rewrite-then-Parse. One challenge of carrying out a comparative study on these paradigms is the lack of annotated data, since existing CSP datasets such as SMCalFlow (Semantic Machines et al., 2020) and CoSQL (Yu et al., 2019) are often annotated based on a single paradigm. Therefore, we construct a new dataset, *SMCalFlow-EQ*, derived from a subset of SMCalFlow dialogues with annotations for all four paradigms.

Our experiments consider both in-context learning (ICL) using GPT-3.5 and fine-tuning (FT) using T5-base 220M (Raffel et al., 2020) for building and adapting parsers. ICL typically has lower accuracy compared to FT, although the two are not strictly comparable as they use different models. The only exception is Parse-with-Reference-Program, suggesting that GPT-3.5 is effective at editing programs using natural language. Overall, we find Rewrite-then-Parse to be the most promising approach, as it achieves similar accuracy to other paradigms in both ICL and FT experiments, while requiring only a few annotated examples for to de-

[*] Work done at Microsoft Semantic Machines.

Findings of the Association for Computational Linguistics: IJCNLP-AACL 2023, pages 348–360
November 1–4, 2023.

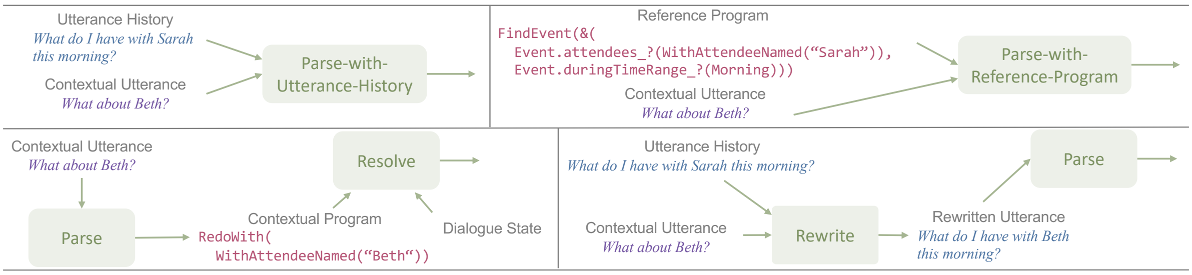

Figure 1: Four canonical paradigms of conversational semantic parsing for contextual utterances.

velop a query rewriter and no additional program annotations. We release code and data to facilitate future work on parsing contextual utterances.[1]

2 Background: LLM-Based Parsing

Following Shin et al. (2021) and Roy et al. (2022), we formulate parsing as a constrained decoding problem, where an LLM is used to predict the next token and a context-free grammar (CFG) is used to validate the predicted token. A program is represented as a sequence of S-expression tokens $y_1 y_2 \dots y_L$. The space of all valid S-expressions is governed by a CFG denoted by $\mathcal{G}$, which can be automatically derived from function definitions and types used in the domain (see Appendix A).

To generate the program for a user utterance, we first feed the LLM with the user utterance and necessary context information as a sequence of tokens. Then the S-expression of the program is generated incrementally. At each decoding step l, we only keep the partial prefix sequence $y_1 y_2 \dots y_l$ if it is allowed by $\mathcal{G}$. This validation can be efficiently performed via Earley's parsing algorithm (Earley, 1970) using the parsing state of the partial sequence $y_1 y_2 \cdots y_{l-1}$.

In this paper, we consider both ICL and FT for constructing LLM-based parsers. For ICL, we prompt the pre-trained LLM with K_{ICL} demonstration examples retrieved via BM25 (Robertson and Walker, 1994; Robertson and Zaragoza, 2009), following Rubin et al. (2022) and Roy et al. (2022). For FT, we continue training the LLM on K_{FT} demonstration examples, producing a new model to be used during constrained decoding.

3 Few-Shot Adaptation

In this paper, we assume there are a large number (M) of annotated non-contextual utterances, $\mathcal{D} = \{(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}), \dots, (\boldsymbol{x}^{(M)}, \boldsymbol{y}^{(M)})\}$, where $\boldsymbol{x}^{(i)}$ denotes the i-th non-contextual utterance in the dataset, $\boldsymbol{y}^{(i)}$ is the corresponding non-contextual program, and M is the number of annotated examples. These examples are used to derive a grammar $\mathcal{G}_1$ and build the parser $\mathcal{P}_1$ for non-contextual utterances via either ICL or FT.

For a contextual utterance $\boldsymbol{u}_t$ at the t-th turn of a dialogue, the goal is to obtain the non-contextual program $\boldsymbol{y}_t$ using the *utterance history* $\boldsymbol{h}_t = [\boldsymbol{u}_{<t}]$, the corresponding programs $\boldsymbol{y}_{<t}$, and/or other information recorded in the dialogue state. Figure 1 illustrates four canonical paradigms for parsing contextual utterances. For each of these paradigms, we would like to obtain a new parser by adapting from the base parser $\mathcal{P}_1$ using N demonstration examples, where $N \ll M$.

3.1 Parsing Paradigms

Parse-with-Utterance-History: In this paradigm, the parser directly predicts $\boldsymbol{y}_t$ by conditioning on the contextual utterance $\boldsymbol{u}_t$ and its history $\boldsymbol{h}_t$. This paradigm has been used in contextual semantic parsing (Zettlemoyer and Collins, 2009; Suhr et al., 2018) and belief state tracking (Mrkšić et al., 2017).

Parse-with-Reference-Program: This paradigm assumes that the salient additional context to parse $\boldsymbol{u}_t$ is captured by a *reference program*, which is a non-contextual program to be revised and typically that from the preceding turn, $\boldsymbol{y}_{t-1}$. The parsing process can be viewed as editing the reference program based on the contextual utterance which directly yields $\boldsymbol{y}_t$. Zhang et al. (2019) employs a similar strategy by using a copy operation during parsing to copy tokens from the reference program for text-to-SQL.

Parse-then-Resolve: This paradigm divides the task into two steps, leading to a modularized system with a parser followed by a resolver. $\boldsymbol{u}_t$ is first mapped to an intermediate program $\tilde{\boldsymbol{y}}_t$ which contains specialized contextual symbols. These contex-

[1] https://github.com/microsoft/few_shot_adaptation_for_parsing_contextual_utterances_with_llms

tual symbols (marking ellipsis or coreference) are resolved deterministically using the dialogue state determined from $\boldsymbol{y}_{<t}$, resulting in the final non-contextual prediction $\boldsymbol{y}_t$. Several recent datasets for CSP have adopted this paradigm (Semantic Machines et al., 2020; Cheng et al., 2020).

Rewrite-then-Parse: This paradigm modularizes the system using a rewriter followed by a parser. The history $\boldsymbol{h}_t$ and contextual utterance $\boldsymbol{u}_t$ are first rewritten into a single non-contextual utterance $\boldsymbol{u}'_t$ Then, $\boldsymbol{u}'_t$ is parsed to $\boldsymbol{y}_t$ by a single-turn semantic parser. This paradigm is closely related to incomplete utterance rewriting (Liu et al., 2020) and conversational query rewriting (Rastogi et al., 2019; Yu et al., 2020; Chen et al., 2020; Song et al., 2020; Inoue et al., 2022; Mao et al., 2023) though the parsing step is usually unnecessary or overlooked in these related studies. Using this paradigm, the rewriter and the parser can be independently developed and maintained.

3.2 Adaptation via ICL

For ICL, we use GPT-3.5 and the following prompt template provided by Shin et al. (2021) and Roy et al. (2022), where placeholders {X1}, {X2}, ... are demonstrations input, {Y1}, {Y2}, ... are demonstrations output, and {X′} is the test input.

```
Let's translate what a human user says into what
a computer might say.

Human: {X1}
Computer: {Y1}

Human: {X2}
Computer: {Y2}

...

Human: {X'}
Computer:
```

For Parse-with-Utterance-History, Parse-with-Reference-Program, and Parse-then-Resolve, the input placeholders are respectively instantiated as $\boldsymbol{h} \mid \boldsymbol{u}$, $\boldsymbol{r} \mid \boldsymbol{u}$, and $\boldsymbol{u}$, where the character $\mid$ is used as the separator. The output placeholders are all instantiated by non-contextual programs $\boldsymbol{y}$, except for Parse-then-Resolve which uses $\tilde{\boldsymbol{y}}$ instead. The test input placeholder follows the same form as demonstration input placeholders. New CFG rules are derived from the program annotations of contextual utterances, *i.e.,* $\tilde{\boldsymbol{y}}$ and $\boldsymbol{y}$, yielding two new grammars $\mathcal{G}_\alpha$ and $\mathcal{G}_\beta$, respectively. During constrained decoding, the joint grammar $\mathcal{G}_1 \cup \mathcal{G}_\alpha$ is used for Parse-then-Resolve, whereas $\mathcal{G}_1 \cup \mathcal{G}_\beta$ is used for the other three paradigms. In other words, the adaptation only changes the set of demonstration examples used during prompt instantiation and augments the CFG used during constrained decoding.

For Rewrite-then-Parse, we can re-use the same grammar $\mathcal{G}_1$ and parser $\mathcal{P}_1$ used for non-contextual utterances, without any annotated programs for contextual utterances.

3.3 Adaptation via FT

For FT, the parser $\mathcal{P}_1$ for non-contextual utterances uses an LLM $\mathcal{M}_1$ fine-tuned from T5-base 220M (Raffel et al., 2020). To adapt this parser for contextual utterances, we continue fine-tuning $\mathcal{M}_1$ on annotated contextual utterances, except for Rewrite-then-Parse which uses $\mathcal{P}_1$ itself. Similar to ICL, different forms of token sequences are used for different paradigms, *i.e.,* $\boldsymbol{h} \mid \boldsymbol{u} \mid \boldsymbol{y}$ for Parse-with-Utterance-History, $\boldsymbol{r} \mid \boldsymbol{u} \mid \boldsymbol{y}$ for Parse-with-Utterance-History, and $\boldsymbol{u} \mid \tilde{\boldsymbol{y}}$ for Parse-then-Resolve. The new grammar is constructed identically to ICL as well.

3.4 Data Annotation Effort

An important axis when comparing different parsing paradigms is the data annotation effort. For Parse-with-Utterance-History, annotating the non-contextual program for a contextual utterance can be a cognitively demanding task, as it needs to account for the full utterance history. Data annotation for Parse-with-Reference paradigm is similar to the Parse-with-Utterance-History, though it may be less cognitively intensive because the human annotator only needs to make a a few edits as opposed to performing a full parse. Compared with Parse-with-Utterance-History, annotations of intermediate programs in the Parse-then-Resolve paradigm are much less context-dependent and more concise, which potentially makes the parser more data efficient. However, this comes at a cost of placing a greater burden on the resolver, which uses custom-designed contextual symbols based on the domain; their expressiveness can greatly affect the quality of the annotations and the complexity of the resolver. Finally, collecting annotations for the the utterance rewriting task is relatively easy and domain independent compared to collecting annotations for parsers which often requires learning a domain-specific language.

4 Experiments

4.1 Data

Existing CSP datasets are often annotated based on only one or two paradigms, making it difficult to compare across different paradigms comprehensively. To address this challenge, we construct a dataset *SMCalFlow-EventQueries* (*SMCalFlow-EQ*) derived from a subset of SMCalFlow (Semantic Machines et al., 2020). It contains 31 training and 100 test instances in total. Each instance consists of a contextual user utterance $\boldsymbol{u}$ during an event-related query (*e.g.,* "*what about Tuesday?*"), the corresponding contextual/intermediate program $\tilde{\boldsymbol{y}}$ and non-contextual program $\boldsymbol{y}$, the utterance history $\boldsymbol{h}$, the reference program $\boldsymbol{r}$, and the rewritten non-contextual utterance $\boldsymbol{u}'$. The programs ($\boldsymbol{y}$, $\tilde{\boldsymbol{y}}$, $\boldsymbol{r}$) are semi-automatically derived from the original SMCalFlow annotations. The rewritten non-contextual utterances $\boldsymbol{u}'$ are manually annotated by domain experts. See Appendix B for details of the dataset construction and examples.

We additionally use 8892 training and 100 test instances of non-contextual utterances (*e.g.,* "*do I have any meetings scheduled after Thursday?*"), each paired with their corresponding non-contextual programs, semi-automatically derived from SMCalFlow as well. These instances are used to construct and evaluate the base parser $\mathcal{P}_1$ for non-contextual utterances.

4.2 Experimental Results

For Parse-with-Reference-Program, we use the oracle reference program, which is the non-contextual program of the preceding turn.[2] For Parse-then-Resolve, we assume an oracle resolver is available, which in practice can be implemented as a rule-based system. The rewriter used for Rewrite-then-Parse is implemented via GPT-3.5, and details are provided in Appendix D. We also consider using the oracle rewritten utterances annotated in the contextual subset of *SMCalFlow-EQ*.

We evaluate the program exact match accuracy on the *SMCalFlow-EQ* test set for all paradigms. Table 1 presents the experimental results. Across all paradigms, FT achieves higher exact match than ICL by 7.9% to 29.4% absolute gain. For FT, Rewrite-then-Parse with oracle rewritten utterances performs the best. There is no significant difference among other approaches, including Rewrite-then-Parse using the GPT-3.5 rewriter which does not require additional fine-tuning. For ICL, Parse-with-Reference-Program performs the best, suggesting it is easier for GPT-3.5 to softly edit a program than parsing directly from natural language. Rewrite-then-Parse using oracle rewritten utterances is still better than the remaining approaches. By comparing the results of Rewrite-then-Parse, it is clear that improving the rewriter can lead to a corresponding improvement in parsing accuracy.

Paradigm	**ICL**	**FT**
Parse-with-Utterance-History	51.8	81.2
Parse-with-Reference-Program	86.1*	78.2
Parse-then-Resolve	70.5*	82.4
Rewrite-then-Parse	65.3*	75.2
Rewrite-then-Parse (oracle)	76.2*	94.0*

Table 1: Exact match accuracy on *SMCalFlow-EQ* test set. For both ICL and FT, we test each paradigm against the corresponding Parse-with-Utterance-History predictions using McNemar's test and show statistically significant ($p < 0.05$) results with *.

We manually examine incorrect predictions made by parsers for contextual utterances and identify common error categories: incorrect top-level program types, alternative parses for the input, extra constraints, missing constraints, and constraints with incorrect arguments/functions (see Table A5 for examples).

For ICL, the most common error type is incorrect function calls. 30% of the errors made by Parse-with-Reference-Program are due to incorrect function use. In particular, the model struggles with predicting rare functions such as negations, potentially because the only knowledge of the target language is from the contextual subset of *SMCalFlow-EQ*.

For FT, 33% of the errors in Parse-then-Resolve are from incorrect top-level program types. Introducing new symbols increases the program space, especially different intermediate programs that have similar functions, suggesting that the design of these specialized contextual symbols is crucial. For Parse-with-Utterance-History, we find that 40% of the errors come from missing constraints, indicating that jointly learn parsing and consolidating constraints from multiple turns is challenging for the parsing model. For Rewrite-then-Parse, 55% of the errors are due to incorrect arguments, and 45% are due to differences in capitalization (*e.g.,* the

[2] It is possible that the reference program is from an earlier turn or does not appear in the history, though the contextual subset does not contain such examples.

Paradigm	ICL	FT
Parse-with-Utterance-History	63.5	84.5
Parse-with-Reference-Program	79.5*	83.0
Parse-then-Resolve	73.5*	85.5
Rewrite-then-Parse	69.5*	82.0
Rewrite-then-Parse (oracle)	75.0*	90.5*

Table 2: Exact match accuracy on *SMCalFlow-EQ* test set combined with non-contextual utterances. For both ICL and FT, we test each paradigm against the corresponding Parse-with-Utterance-History predictions using McNemar's test and show statistically significant ($p < 0.05$) results indicated with $\star$.

rewriter converts a lowercase name to uppercase) which is arguably less critical.

We also examine the overall parsing accuracy on the joint test set of contextual and non-contextual utterances. We use a binary classifier which takes the user utterance as input and determines whether to use the parser for non-contextual utterances or the parser for contextual utterances. The classifier is obtained by fine-tuning the RoBERTa-base (Liu et al., 2019) to on *SMCalFlow-EQ* utterances. The overall classification accuracy is 95.5%. The results are summarized in Table 2. We use exact match accuracy as the evaluation metric, where the prediction is treated as correct only when classification and parsing are both correct.

5 Conclusion

We study a real-world CSP setting, *i.e.,* few-shot adaptation for parsing contextual utterances with LLMs, and compare four different paradigms using both ICL and FT. To facilitate the study, we construct a new dataset, *SMCalFlow-EQ* with annotations for all paradigms. Experiments show that ICL with GPT-3.5 usually underperforms FT with T5-base except for Parse-with-Reference-Program, suggesting GPT-3.5 is good at editing programs via natural language in these data conditions. Overall, Rewrite-then-Parse stands out as a promising approach for future development of LLM-based CSP, as it performs as well as other paradigms but require only a few annotated exampels for the rewriter and no additional program annotation.

6 Limitations

Due to the cost of collecting program annotations for all paradigms, the size of the *SMCalFlow-EQ* test set is relatively small and we only study dialogues from SMCalFlow. While the experiments results are informative under significance test, it would be useful for future work to conduct a similar study on larger and diverse datasets.

The LLMs used in this work are pre-trained primarily on English, and the *SMCalFlow-EQ* also only contains English utterances. It would be interesting to study the few-shot adaptation problem on other languages.

Acknowledgements

We would like to thank Benjamin Van Durme, Matt Gardner, Adam Pauls, and Jason Wolfe for valuable discussions on this paper.

References

Zheng Chen, Xing Fan, and Yuan Ling. 2020. Pre-training for query rewriting in a spoken language understanding system. In *Proceedings of 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7969–7973.

Jianpeng Cheng, Devang Agrawal, Héctor Martínez Alonso, Shruti Bhargava, Joris Driesen, Federico Flego, Dain Kaplan, Dimitri Kartsaklis, Lin Li, Dhivya Piraviperumal, Jason D. Williams, Hong Yu, Diarmuid Ó Séaghdha, and Anders Johannsen. 2020. Conversational semantic parsing for dialog state tracking. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 8107–8117, Online. Association for Computational Linguistics.

Jay Earley. 1970. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102.

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 328–339, Melbourne, Australia. Association for Computational Linguistics.

Shumpei Inoue, Tsungwei Liu, Son Nguyen, and Minh-Tien Nguyen. 2022. Enhance incomplete utterance restoration by joint learning token extraction and text generation. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3149–3158, Seattle, United States. Association for Computational Linguistics.

Qian Liu, Bei Chen, Jian-Guang Lou, Bin Zhou, and Dongmei Zhang. 2020. Incomplete utterance rewriting as semantic segmentation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2846–2857, Online. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv:1907.11692*.

Kelong Mao, Zhicheng Dou, Haonan Chen, Fengran Mo, and Hongjin Qian. 2023. Large language models know your contextual search intent: A prompting framework for conversational search. *arXiv:2303.06573*.

Joram Meron. 2022. Simplifying semantic annotations of SMCalFlow. In *Proceedings of the 18th Joint ACL - ISO Workshop on Interoperable Semantic Annotation within LREC2022*, pages 81–85, Marseille, France. European Language Resources Association.

Nikola Mrkšić, Diarmuid Ó Séaghdha, Tsung-Hsien Wen, Blaise Thomson, and Steve Young. 2017. Neural belief tracker: Data-driven dialogue state tracking. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1777–1788, Vancouver, Canada. Association for Computational Linguistics.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551.

Pushpendre Rastogi, Arpit Gupta, Tongfei Chen, and Mathias Lambert. 2019. Scaling multi-domain dialogue state tracking via query reformulation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Industry Papers)*, pages 97–105, Minneapolis, Minnesota. Association for Computational Linguistics.

Stephen Robertson and Hugo Zaragoza. 2009. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389.

Stephen E. Robertson and Stephen Walker. 1994. Some simple effective approximations to the 2-Poisson model for probabilistic weighted retrieval. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 232–241, Dublin, Ireland.

Subhro Roy, Sam Thomson, Tongfei Chen, Richard Shin, Adam Pauls, Jason Eisner, and Benjamin Van Durme. 2022. BenchCLAMP: A benchmark for evaluating language models on semantic parsing. *arXiv:2206.10668*.

Ohad Rubin, Jonathan Herzig, and Jonathan Berant. 2022. Learning to retrieve prompts for in-context learning. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2655–2671, Seattle, United States. Association for Computational Linguistics.

Semantic Machines, Jacob Andreas, John Bufe, David Burkett, Charles Chen, Josh Clausman, Jean Crawford, Kate Crim, Jordan DeLoach, Leah Dorner, Jason Eisner, Hao Fang, Alan Guo, David Hall, Kristin Hayes, Kellie Hill, Diana Ho, Wendy Iwaszuk, Smriti Jha, Dan Klein, Jayant Krishnamurthy, Theo Lanman, Percy Liang, Christopher H. Lin, Ilya Lintsbakh, Andy McGovern, Aleksandr Nisnevich, Adam Pauls, Dmitrij Petters, Brent Read, Dan Roth, Subhro Roy, Jesse Rusak, Beth Short, Div Slomin, Ben Snyder, Stephon Striplin, Yu Su, Zachary Tellman, Sam Thomson, Andrei Vorobev, Izabela Witoszko, Jason Wolfe, Abby Wray, Yuchen Zhang, and Alexander Zotov. 2020. Task-oriented dialogue as dataflow synthesis. *Transactions of the Association for Computational Linguistics*, 8:556–571.

Noam Shazeer and Mitchell Stern. 2018. Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning*, pages 4596–4604. PMLR.

Richard Shin, Christopher Lin, Sam Thomson, Charles Chen, Subhro Roy, Emmanouil Antonios Platanios, Adam Pauls, Dan Klein, Jason Eisner, and Benjamin Van Durme. 2021. Constrained language models yield few-shot semantic parsers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7699–7715, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Richard Shin and Benjamin Van Durme. 2022. Few-shot semantic parsing with language models trained on code. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5417–5425, Seattle, United States. Association for Computational Linguistics.

Shuangyong Song, Chao Wang, Qianqian Xie, Xinxing Zu, Huan Chen, and Haiqing Chen. 2020. A two-stage conversational query rewriting model with multi-task learning. In *Companion Proceedings of the Web Conference 2020*, WWW '20, page 6–7, New York, NY, USA. Association for Computing Machinery.

Alane Suhr, Srinivasan Iyer, and Yoav Artzi. 2018. Learning to map context-dependent sentences to executable formal queries. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2238–2249, New Orleans, Louisiana. Association for Computational Linguistics.

Shi Yu, Jiahua Liu, Jingqin Yang, Chenyan Xiong, Paul Bennett, Jianfeng Gao, and Zhiyuan Liu. 2020. Few-shot generative conversational query rewriting. In

Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '20, page 1933–1936, New York, NY, USA.

Tao Yu, Rui Zhang, Heyang Er, Suyi Li, Eric Xue, Bo Pang, Xi Victoria Lin, Yi Chern Tan, Tianze Shi, Zihan Li, Youxuan Jiang, Michihiro Yasunaga, Sungrok Shim, Tao Chen, Alexander Fabbri, Zifan Li, Luyao Chen, Yuwen Zhang, Shreya Dixit, Vincent Zhang, Caiming Xiong, Richard Socher, Walter Lasecki, and Dragomir Radev. 2019. CoSQL: A conversational text-to-SQL challenge towards cross-domain natural language interfaces to databases. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1962–1979, Hong Kong, China. Association for Computational Linguistics.

Luke Zettlemoyer and Michael Collins. 2009. Learning context-dependent mappings from sentences to logical form. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 976–984, Suntec, Singapore. Association for Computational Linguistics.

Rui Zhang, Tao Yu, Heyang Er, Sungrok Shim, Eric Xue, Xi Victoria Lin, Tianze Shi, Caiming Xiong, Richard Socher, and Dragomir Radev. 2019. Editing-based SQL query generation for cross-domain context-dependent questions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5338–5349, Hong Kong, China. Association for Computational Linguistics.

A CFG for Constrained Decoding

The CFG used for constrained decoding can be automatically derived from function definitions and types used in the domain. For example, given a function $\texttt{FN}(\arg_1, \cdots, \arg_N)$ with corresponding argument types $\tau_1, \cdots, \tau_N$ and output type τ_O, we can automatically derive a CFG rule $\mathrm{NT}_{\tau_O} \rightarrow$ (FN ($\mathrm{NT}_{\tau_1} \cdots \mathrm{NT}_{\tau_N}$)) where NT_{τ_i} denotes the non-terminal symbol for the type τ_i, and the function name FN and the parentheses are terminal symbols in $\mathcal{G}$. For each primitive type (*e.g.,* "string", "number"), we additionally define CFG rules to expand the non-terminal of the primitive type to terminals representing acceptable values of the type (sometimes using regular expressions).

B Dataset Construction and Examples

The original SMCalFlow data only provide annotations of *contextual programs* for individual utterances. We develop a heuristic-based implementation of `NewClobber` and `ReviseConstraint` to propose candidates of the corresponding non-contextual programs. Specifically, given the non-contextual program

```
(NewClobber (
    (intension)
    (slotConstraint)
    (value)))
```

we modify the non-contextual program of the preceding turn by replacing its fragment satisfying the `slotConstraint` with the new fragment `value`. Similarly, given the non-contextual program

```
(ReviseConstraint (
  (rootLocation)
  (oldLocation)
  (newConstraint)))
```

we modify the non-contextual program of the preceding turn by replacing a fragment `oldConstraint` which satisfies the `oldLocation` and is governed by a bigger fragment satisfying the `rootLocation` with a new fragment

```
(& ((oldConstraint) (newConstraint)))
```

i.e., conjoining the two constraints regardless whether they conflicts with each other. For both cases, when there are multiple possible replacements, all resulting candidates are proposed. These candidates are manually reviewed and edited by the authors to finalize non-contextual program annotations. For example, if `newConstraint` contradicts with a part of `oldConstraint`, we drop the such conflicting parts in the `oldConstraint`.

Furthermore, as noted by Meron (2022), the original annotations of SMCalFlow can be complex and contain many boilerplate segments. Therefore, we use heuristics to simplify the original annotations to obtain programs that are shorter and potentially easier to read and predict. Similar to Meron (2022), the simplification was implemented via a set of tree transformation rules, which convert specific sub-trees of the original program into simplified sub-trees. The list of sub-tree transformations are provided in Table A2–Table A4.

Two data specialists are asked to produce the annotations for the rewritten non-contextual utterances in the contextual subset. They are provided with instructions and training materials, which explains how to rewrite a contextual user utterance with its preceding utterance into a single non-contextual utterance. Each example takes 10 to 30 seconds to annotate. Additionally, annotators were asked to provide a confidence from 0 (least confident) to 3 (most confident) in the rewritten utterance. The average confidence was 2.9. Then they are asked to review the each other's annotations and answer whether they agree with each other. In our pilot data collection, the agreement rate between the two data specialists was 93.3%.

Table A1 provides some examples from in *SMCalFlow-EQ*.

C Fine-tuning Experiment Hyperparameters

For fine-tuning, we employ the Adafactor optimizer (Shazeer and Stern, 2018) and set the batch size to 32. The slanted triangular learning rate scheduler (Howard and Ruder, 2018) is used with a maximum learning rate of 10^{-5} and 1000 warmup steps. We fine-tune $\mathcal{M}_0$ for 10000 steps on the non-contextual subset to obtain $\mathcal{M}_1$, and another 10000 steps on the corresponding data to obtain the models for individual paradigms. For constrained decoding, the maximum output sequence length is 1000.

Utterance	Last Utterance	Oracle Rewritten Utterance	Non-Contextual Program	Contextual Program

What about later next week?	Did I have any meetings early next week?	Did I have any meetings later next week?	`(QueryEventResponseIsNonEmpty`
			`(FindEventWrapperWithDefaults`
			
			`(Event.duringDateRangeConstraint_?`
			`(LateDateRange`
			`(NextWeekList)))))`
				`(Execute (ReviseConstraint`
				`(DefaultRootLocation)`
				`(^(Event)`
				`ConstraintTypeIntension)`
				
				`(Event.duringDateRangeConstraint_?`
				`(LateDateRange`
				`(NextWeekList)))))`

Actual I meant the day after tomorrow.	Is there any appointments tomorrow?	Is there any appointments the day after tomorrow?	`(QueryEventResponseIsNonEmpty`
			`(FindEventWrapperWithDefaults`
			`(Event.onDate_?`
			`(adjustByPeriod (Tomorrow)`
			`(toDays 1)))))`
				`(Execute (ReviseConstraint`
				`(DefaultRootLocation)`
				`(^(Event)`
				`ConstraintTypeIntension)`
				`(Event.onDate_? (adjustByPeriod`
				`(Tomorrow) (toDays 1)))))`

What about training?	Is there a vacation scheduled for me?	Is there a training scheduled for me?	`(QueryEventResponseIsNonEmpty`
			`(FindEventWrapperWithDefaults`
			`(Event.subject_? (?~=`
			`\"training\"))))`
				`(Execute (ReviseConstraint`
				`(DefaultRootLocation)`
				`(^(Event)`
				`ConstraintTypeIntension)`
				`(Event.subject_? (?~=`
				`\"training\"))))`

Table A1: Dataset examples.

D Rewriter Implementation

The rewriter used for Rewrite-then-Parse is implemented via GPT-3.5 (`text-davinci-003`). The prompt template is shown below, where placeholders {H1}, {H2}, … are for the utterance history (*i.e.,* the preceding utterances), {X1}, {X2}, … are for contextual user utterances, {Z1}, {Z2}, … are for rewritten non-contextual utterances, and {H'} and {X'} are for test input.

```
Combine the utterances into a single utterance
with the meaning of the last utterance.


Last Utterance: {H1}
Current Utterance: {X1}
Rewritten Utterance: {Z1}

Last Utterance: {H2}
Current Utterance: {X2}
Rewritten Utterance: {Z2}

...

Last Utterance: {H'}
Current Utterance: {X'}
Rewritten Utterance:
```

We sample 8 demonstration examples are sampled uniformly from the contextual subset training instances. Greedy decoding is used with 50 maximum tokens and no frequency or presence penalty. The BLEU score using the oracle rewritten utterances as reference is 93.6.

Original	**Simplified**
`(& (^($type) EmptyStructConstraint) ($c))`	`($c)`
`(& ($c) (^($type) EmptyStructConstraint))`	`($c)`
`(> (size (QueryResponse.results ($response))), 0L)`	`(QueryEventResponseIsNonEmpty ($response))`
`(AttendeeListHasRecipientConstraint (RecipientWithNameLike (^(Recipient) EmptyStructConstraint) (PersonName.apply $name)))`	`(WithAttendeeNamed ($name))`
`(AttendeeListHasRecipient (Execute (refer (extensionConstraint (RecipientWithNameLike (^(Recipient) EmptyStructConstraint) (PersonName.apply $name))))))`	`(WithAttendeeNamed ($name))`
`(AttendeeListExcludeRecipient (Execute (refer (extensionConstraint (RecipientWithNameLike (^(Recipient) EmptyStructConstraint) (PersonName.apply $name))))))`	`(WithoutAttendeeNamed ($name))`

Table A2: List of sub-tree transformations for simplifying SMCalFlow programs (part 1).

Original	Simplified
`(EventAtTime ($event) ($time))`	`(& ($event) (Event.atTime_? ($time)))`
`(EventDuringRangeTime ($event) ($timeRange))))`	`(& ($event) (Event.duringTimeRangeConstraint_? ($timeRange)))`
`(EventOnDate ($date) ($event))`	`(& ($event) (Event.onDate_? ($date)))`
`(EventDuringDateRange ($event) ($dateRange))))`	`(& ($event) (Event.duringDateRangeConstraint_? ($dateRange)))`
`(EventOnDateTime (DateAtTimeWithDefaults (($date) ($time)) ($event)))`	`(& ($event) (& (Event.onDate_? ($date)) (Event.atTime_? ($time))))`
`(EventOnDateAfterTime (($date) ($event) ($time)))`	`(& ($event) (& (Event.onDate_? ($date)) (Event.afterTime_? ($time))))`
`(EventOnDateBeforeTime (($date) ($event) ($time)))`	`(& ($event) (& (Event.onDate_? ($date)) (Event.beforeTime_? ($time))))`
`(EventOnDateFromTimeToTime (($date) ($event) ($time1) ($time2)))`	`(& ($event) (& (Event.onDate_? ($date)) (Event.betweenTimeAndTime_? ($time1) ($time2))))`

Table A3: List of sub-tree transformations for simplifying SMCalFlow programs (part 2).

Original	Simplified
`(EventAfterDateTime (($event) ($dateTime)))`	`(& ($event) (Event.afterDateTime_? ($dateTime)))`
`(EventBeforeDateTime (($event) ($dateTime)))`	`(& ($event) (Event.beforeDateTime_? ($dateTime)))`
`(EventOnDateWithTimeRange (EventOnDate ($date) ($event)) ($timeRange))`	`(& ($event) (& (Event.onDate_? ($date)) (Event.duringTimeRangeConstraint_? ($timeRange))))`
`(EventOnDateWithTimeRange (EventDuringRange ($event) ($dateRange) ($timeRange)))`	`(& ($event) (& (Event.duringDateRangeConstraint_? ($dateRange)) (Event.duringTimeRangeConstraint_? ($timeRange))))`
`(EventDuringRangeDateTime ($event) ($dateTimeRange))`	`(& ($event) (Event.duringDateTimeRangeConstraint_? ($dateTimeRange)))`

Table A4: List of sub-tree transformations for simplifying SMCalFlow programs (part 3).

<table>
<tr><th>Error Type</th><th>Gold</th><th>Predicted</th></tr>
<tr><td>Top-level Incorrect</td><td><pre>(Execute (ReviseConstraint
    (DefaultRootLocation) (^(Event)
    ConstraintTypeIntension) (&
    (Event.attendees_?
    (WithAttendeeNamed "kim")) (&
    (Event.onDate_? (Tomorrow))
    (Event.subject_? (?~= "lunch
    meeting"))))))</pre></td><td><pre>(Execute (NewClobber (DefaultIntension)
    (^(Recipient)
    ConstraintTypeIntension) (intension
    (RecipientWithNameLike (^(Recipient)
    EmptyStructConstraint)
    (PersonName.apply "kim")))))</pre></td></tr>
<tr><td>Alternate parse</td><td><pre>(FindEventWrapperWithDefaults (&
    (Event.attendees_?
    (WithAttendeeNamed "Barry"))
    (Event.start_? (DateTime.date_? (?=
    (Tomorrow))))))</pre></td><td><pre>(FindEventWrapperWithDefaults (&
    (Event.attendees_?
    (WithAttendeeNamed "Barry"))
    (Event.onDate_? (Tomorrow))))</pre></td></tr>
<tr><td>Extra Constraint</td><td><pre>(QueryEventResponseIsNonEmpty
    (FindEventWrapperWithDefaults
    (Event.attendees_? (&
    (WithAttendeeNamed "Marco")
    (WithAttendeeNamed "Peyton")))))</pre></td><td><pre>(QueryEventResponseIsNonEmpty
    (FindEventWrapperWithDefaults (&
    (Event.attendees_?
    (WithAttendeeNamed "Peyton")) (&
    (Event.attendees_?
    (WithAttendeeNamed "Marco"))
    (Event.duringDateRangeConstraint_?
    (FullMonthofMonth (Date.month
    (Today))))))))</pre></td></tr>
<tr><td>Missing Constraint</td><td><pre>(QueryEventResponseIsNonEmpty
    (FindEventWrapperWithDefaults (&
    (Event.attendees_?
    (WithAttendeeNamed "Bob")) (&
    (Event.duringTimeRangeConstraint_?
    (Afternoon)) (Event.onDate_?
    (Tomorrow))))))</pre></td><td><pre>(QueryEventResponseIsNonEmpty
    (FindEventWrapperWithDefaults (&
    (Event.attendees_?
    (WithAttendeeNamed "Bob"))
    (Event.duringTimeRangeConstraint_?
    (Afternoon)))))</pre></td></tr>
<tr><td>Constraint With Incorrect Function</td><td><pre>(Execute (NewClobber (DefaultIntension)
    (extensionConstraint
    (^(LocationKeyphrase)
    AlwaysTrueConstraint)) (intension
    (LocationKeyphrase.apply "EVO"))))</pre></td><td><pre>(Execute (NewClobber (DefaultIntension)
    (^(Recipient)
    ConstraintTypeIntension) (intension
    (RecipientWithNameLike (^(Recipient)
    EmptyStructConstraint)
    (PersonName.apply "EVO")))))</pre></td></tr>
<tr><td>Constraint With Incorrect Argument</td><td><pre>(QueryEventResponseIsNonEmpty
    (FindEventWrapperWithDefaults
    (Event.onDate_? (adjustByPeriod
    (Tomorrow) (toDays 1)))))</pre></td><td><pre>(QueryEventResponseIsNonEmpty
    (FindEventWrapperWithDefaults
    (Event.onDate_? (adjustByPeriod
    (Tomorrow) (toDays 2)))))</pre></td></tr>
</table>

Table A5: Error examples with gold parses.