Evaluating Sequence-to-Sequence Modelling for Dialogue State Tracking

Mathieu Tuli
Department of Computer Science
University of Toronto
mathieutuli@cs.toronto.edu

Sarvagya Agrawal
Department of Computer and Mathematical Sciences
University of Toronto
sarvagya.agrawal@mail.utoronto.ca

Abstract

Dialogue State Tracking (DST) is a key component of Task Oriented Dialogue. As Task Oriented Dialogue systems become more and more prevalent, research has trended towards making DST more efficient and scalable using sequence-to-sequence (Seq2Seq) modelling. In this work, we survey this trend and evaluate the performance of various Seq2Seq models on DST and analyze some of the claims made about the benefits of Seq2Seq over traditional slot-filling architectures.

1 Introduction

Growlingly popular Task-Oriented Dialogue (TOD) systems such as Siri and Alexa aim to assist users with specific tasks based on their perceived intent through interactive natural dialogue. A fundamental core to TOD systems is Dialogue State Tracking (DST); the task of tracking information (domain bound slots (e.g. time, location) and information values (e.g. 5:00PM, Toronto)) throughout turns in a dialogue [1, 2, 3]. Recent development have shown increasing usage of sequence-to-sequence (Seq2Seq) models for DST and end-to-end modeling [4, 5]. In this work, we propose to critically evaluate the benefits/drawbacks of using Seq2Seq models for the task of DST.

Typical approaches for DST have been largely based on slot-filling architectures (SFA) [6, 7], where user utterances are encoded and classification over predfined slots is performed. Seq2Seq Modelling, where user utterances are directly translated to a string-form of slots and values, has been studied to make DST more scalable (applicable unseen domains say) [8, 9, 10]. On a general level, we aim to draw a comparison between utilizing Seq2Seq vs. SFA models for DST. We implement one SFA and two Seq2Seq architectures, namely, a Transformer and a bidirectional long short term memory (LSTM) recurrent neural network (RNN) and conduct experiments to analyze DST accuracy. Furthermore, evaluate the dialogue level acuracy for the Seq2Seq and SFA models to highlight limitations of current state of TOD systems in terms of dialogue level conversational understanding.

2 Related Works

SFA systems [11, 6, 12] are quite common in current research/industry for the task of DST. There has also been research in applying Few Shot Learning to TOD systems to decrease the amount of training data required and also to increase scalability [3]. As far as Seq2Seq models are concerned, there have been many approaches to improve the task of DST such as [8, 13]. The idea of using RNN's for TOD systems has been explored in [14] where they propose that RNN's help better model temporal dependencies in Language. The work presented here is closely related to [15] which employs an RNN architecture with LSTM units for DST. A lot of contemporary work has also been conducted to explore the efficiency of using ensembled Transformers such as BERT [16] for slot filling in DST [17, 18]. Moreover, causal language models have also been adapted as state-of-the-art for TOD such as SimpleTOD [2] which employs a GPT-2 based Transformer model.

Preprint. Under review.

3 Methodology

3.1 Data

In this work, we consider SMCalFlow [9] and MultiWOZ-2.1 [19] for DST. SMCalFlow represents dialogue state as a computational graph, and MultiWOZ-2.1 represents it as a comma-seperated set of tuples "<*domain-name slot-name value*, ... >" (see Appendix E for some examples). For both SMCalFlow and MultiWOZ-2.1, input and output sequences are tokenized using a default lower-case tokenizer from the OpenNMT-Py library [20] (Spacy English core small). For SMCalFlow, input sequences are composed as $C_i = [\phi_u; U_i; \phi_b; B_{i-1}; \ldots; \phi_u; U_{i-\ell}; \phi_b]$ for turn i of context length ℓ, where $[;]$ is the concatenation of token sequences, $\phi_u =$ *__User* and $\phi_b =$ *__StartOfProgram*. For MultiWOZ-2.1, $C_i = [\phi_u; U_{i-\ell}; \phi_a; A_{i-\ell}; \ldots; \phi_u; U_i; \phi_b]$ where U_i and A_i are user utterances and agent utterances at some turn i, respectively, $\phi_u =$ *__User*, $\phi_a =$ *__Agent*, and $\phi_b =$ *__Belief*. Note each $\phi_{u,a,b}$ are simply delineating tokens. Each dataset has $t_i, i \in [N]$ conversation turns, where multiple indepedent turns might belong to a single dialogue $d_j, j \in [J]$. For SMCalFlow, $N = 155,923$, and MultiWOZ-2.1 has $N = 77,410$. Output sequences are belief state B_i, which is simply a linearized form of the dialogue state (see subsection 3.1). We will denote K and K' as the different length of the input and output sequences, respectively. This data formatting is from [9, 2].

3.2 Sequence to Sequence Modelling

We consider Seq2Seq modelling rather than the traditional method of slot-filling (see Appendix A for details on how SFAs work). Seq2Seq modelling considers some input context C_i, and the goal is to generate the belief state B_i. More so, we aim to estimate the conditional probability $P(B_i|C_i)$. To accomplish this, an encoder-decoder architecture is used, where the encoder encodes the input sequence into some context embedding $Z \in \mathbb{R}^{K \times D}$, which is fed to the decoder to generate the output sequence auto-regressively. Our loss function is then the negative log-likelihood of this probability as

$$P(B_i|C_i) = \prod_{k=1}^{K'} P(B_{i,k}|Z, B_{i,1}, \ldots, B_{i,k-1}), \mathcal{L}_{NLL} = -\sum_{i}^{N} \sum_{k=1}^{K'} log\left[P(B_{i,k}|Z, B_{i,1}, \ldots, B_{i,k-1})\right]$$

where $B_{i,k}$ is the k-th token in the output sequence for the i-th turn. Our Seq2Seq architectures (RNN/Transformer) model this probability $P(B_{i,k}|Z, B_{i,1}, \ldots, B_{i,k-1})$. We train with teacher forcing.

3.2.1 Word embeddings.

We use GloVe [21] pretrained word embeddings, as done in [9]. We use the GloVe-840B-300D embeddings: 300-dimensional embeddings trained on 840B tokens from Common Crawl.

3.2.2 Models

For a more detailed description of the models used, see Appendix B and Appendix C.

Recurrent neural network. For our RNN, we utilize the pointer-generator network [22]. It is composed of an encoder (a single-layer bi-directional LSTM and a decoder (a singler-layer unidirectional LSTM). Attention over the encoder hidden state is utilized to produced the context vector that is fed to the decoder. Further, a copy-mechanism is utilized, and toggles between generating a new token from the attention mechanism distribution or directly copying one of the attended input embeddings.

Transformer. We use a variant of the original Transformer architecture from [23]. The encoder is a block of 6 stacked encoder layers. Each input sequence is embedded using sinusoidal position-embeddings summed with GloVe embeddings into $C_i' \in \mathbb{R}^{K \times D}$. We use $D = 300$. The decoder is a block of 6 stacked decoder layers. We employ 12-headed attention.

3.3 Evaluation

For training, we monitor accuracy and perplexity. Accuracy is the percentage of correctly predicted tokens, and perplexity (PPL) is the measure of how fluently a model can predict future tokens from the current context [24]. Formally, PPL $= e^{\min(\frac{\mathcal{L}_{NLL}}{k}, 100)}$, k being current word count in sequence.

For the purposes of DST, evaluation is done using the Joint Goal Accuracy (JGA). Turn-level JGA is the percentage of turns that our model has correctly identified and filled the proper slots, independent of order. Dialogue-level JGA is the percentage of dialogues where each predicted turn-level belief state in the dialogue is exactly matched to the ground truth. Formally, we denote the order-independent matching of slots between the predicted state B and ground truth B' as $B \overset{*}{=} B'$. Hence,

$$\text{JGA-turn} \ = \frac{1}{N}\sum_{i}^{N} \mathbb{I}[B_i \overset{*}{=} B'_i], \quad \text{JGA-diag} \ = \frac{1}{J}\sum_{j}^{J} \mathbb{I}[B_i \overset{*}{=} B'_i \, \forall t_i \in d_j]$$

where $\mathbb{I}[\cdot] = 1$ if the inner condition holds, otherwise $\mathbb{I}[\cdot] = 0$.

4 Experiments

4.1 Setup

We conduct our study on SMCalFlow and MultiWOZ-2.1. See Appendix D for hyper-parameter tuning details. We use early-stopping, monitoring perplexity with a window of 5 epochs. Unless otherwise specified, we use context length $\ell = 1$ for SMCalFlow and $\ell = max$ for MultiWOZ-2.1. We attempted to do at least 2 trials for each model. All evaluation is performed by decoding using beam search, with 10 beams. Note reference to RNN or SFA inherently means the pointer-generator or TriPpy models, respectively. See footnote for code link [1].

Seq2Seq vs. SFA. We compare Seq2Seq (RNN and Transformer) against a SFA on MultiWOZ-2.1. We compare turn/dialogue JGA, and also evaluate each Seq2Seq's ability to generalize to unseen domains (i.e. evaluate on DST for domain-slot pairs not seen in training). We omit the *taxi* domain from MultiWOZ-2.1 during training (see RNN$^{\text{taxi-omitted}}$ and Transformer $^{\text{taxi-omitted}}$).

RNN vs. Transformer. We compare RNN vs. Transformer using turn/dialogue-level JGA, as well as training metrics, on both SMCalFlow and MultiWOZ-2.1.

Pretrained vs randomly initialized. It is known and has been shown that Transformers perform best when they have been pretrained [25], and recently SimpleTOD [2] showed the power of a pretrained language model on dialogue state tracking. Specifically, they showed how a pretrained GPT-2 model can achieve state-of-the-art on MultiWOZ-2.1. We aim to see what happens when pretraining is done for a Seq2Seq Transformer. We pretrain our Transformer on the CNN-DailyMail summarization dataset as used in [22], and then fine-tune then entire model on MultiWOZ-2.1.

Context length. For the RNN on SMCalFlow, we evaluate context length on JGA using $\ell \in \{1, 3, 9\}$.

4.2 Results

Seq2Seq vs. SFA. Table 1 shows how TriPpy significantly outperforms either the pointer-generator network or the Transformer. Further, we can clearly see that the validation accuracy for TripPy is a much stronger indicator for the actual JGA performance, as compared to the Seq2Seq models, where the high validation accuracy misleads performance. Despite TripPy outperforming both Seq2Seq models, it was shown in [8] that this may not always be the case. Our results here can be attributed to the simple models we used, and with more computational resources, it would be interesting to consider larger models (such as a Bert/GPT-2-based encoder-decoder). We leave this as future work. Note however that TriPpy cannot be applied to SMCalFlow, given the nature of the dataset, which highlights a benefit of using Seq2Seq. Further, despite Seq2Seq models being proposed for their potential application to unseen domains/slots, we see that this is not the case, and they fail to properly predict the *taxi* domain-slots, achieving 0.00 JGA (on both validation and test sets). Output prediction analysis shows that the taxi domain-slots are simply never predicted.

RNN vs. Transformer. Surprisingly, the pointer-generator (RNN) significantly outperforms the Transformer. Even though training accuracy, validation accuracy, and training perplexity converge to similar levels, the Transformer does not generalize as well and has lower validation perplexity. We can see that despite similar training results, this does not translate to turn- or dialogue-level JGA. Further, note that the RNN with ~ 30 epochs, while the Transformer requires ~ 300.

[1] https://github.com/mathieutuli/controllable-text-generation/tree/csc2516

Table 1: Joint goal accuracy (JGA) for turn-level and dialogue-level dialogue state tracking on SMCalFlow and MultiWOZ-2.1 for various models. Standard deviation over trials is show in the subscript. Test JGA is shown for MultiWOZ-2.1, and validation JGA is shown for SMCalFlow since no test set is provided. Transformer$^{\text{pretrained}}$ is the fine-tuned Tranformer from pretrained weights.

Architecture	Dataset	Acc.$^{\text{Train}}$(%)	Acc.$^{\text{Val}}$(%)	PPL$^{\text{Train}}$	PPL$^{\text{Val}}$	JGA-turn	JGA-diag
RNN$^{\ell=1}$	SMCalFlow	$99.12_{0.13}$	$98.95_{0.10}$	$1.03_{0.00}$	$1.04_{0.00}$	$\mathbf{72.60_{0.67}}$	$\mathbf{36.60_{0.18}}$
RNN$^{\ell=3}$	SMCalFlow	$99.18_{0.12}$	$98.90_{0.11}$	$1.03_{0.00}$	$1.04_{0.00}$	$70.72_{0.42}$	$34.84_{0.13}$
RNN$^{\ell=9}$	SMCalFlow	$99.12_{0.11}$	$98.96_{0.15}$	$1.03_{0.00}$	$1.04_{0.00}$	$72.52_{0.69}$	$36.70_{0.18}$
Transformer$^{\ell=1}$	SMCalFlow	$98.99_{0.16}$	$98.38_{0.12}$	$1.03_{0.00}$	$1.16_{0.05}$	$57.60_{0.80}$	$22.23_{0.23}$
TripPy	MultiWOZ-2.1	$99.47_{0.08}$	$77.38_{0.09}$	N/A	N/A	$\mathbf{53.59_{0.02}}$	$\mathbf{33.20_{0.06}}$
RNN	MultiWOZ-2.1	$98.22_{0.08}$	$97.64_{0.11}$	$1.06_{0.00}$	$1.09_{0.01}$	$40.86_{0.59}$	$9.8_{0.24}$
RNN$^{\text{taxi-omitted}}$	MultiWOZ-2.1	$98.38_{0.00}$	$94.5_{0.00}$	$1.06_{0.00}$	$1.30_{0.08}$	$0.00_{0.00}$	$0.00_{0.00}$
Transformer	MultiWOZ-2.1	$99.05_{0.09}$	$95.67_{0.14}$	$1.03_{0.00}$	$1.29_{0.09}$	$27.07_{0.74}$	$2.50_{0.10}$
Transformer$^{\text{pretrained}}$	MultiWOZ-2.1	$98.90_{0.00}$	$95.88_{0.00}$	$1.03_{0.00}$	$1.29_{0.12}$	$00.00_{0.00}$	$00.00_{0.00}$
Transformer$^{\text{taxi-omitted}}$	MultiWOZ-2.1	$95.89_{0.00}$	$69.02_{0.00}$	$1.13_{0.04}$	$7.21_{0.20}$	$00.00_{0.00}$	$00.00_{0.00}$

Pretrained vs randomly initialized. We can see that the transformer with pretrained weights performs significantly worse than the randomly initialized transformer. The zero-valued JGA results is not a mistake, as despite performing very similar in terms of training performance, the resulting predicted output sequences are poorly structured and fail in every case.

Context length. Context length has some effect on the turn/dialogue-level JGA for the RNN model, with $\ell = 3$ performing the worst. It is not immediately clear why this is, but we postulate that for $\ell \in \{1, 9\}$, attention over the context is more focused on the recent user utterance and dialogue state, whereas for $\ell = 3$, the context is still small enough that the model attends to too much unnecessary or wrong information, an effect that might get diluted/removed with a larger/smaller context.

5 Additional Discussion

Effect of initialization. We notice that despite efforts to pretrain the transformer to improve results, this in fact resulted in completely opposite behaviour. Analyzing samples of predicted outputs analysis showed that the pretraining caused disturbances in the structure of the predicted output sequences. Specifically, the pretrained Transformer struggled to properly delineate the domain-slot-value tuples in the dialogue state, often placing commas at random points, resulting in poor performance. It appears as though the pretraining process caused the model to learn normal sentence structure too strongly, and the fine-tuning did not correct this. Despite this, it was shown in [2] that pretraining in the causal language model setting was very beneficial, as so it remains to see if using models such as GPT-2 as the encoder/decoder base would be beneficial. See Appendix F for examples.

Characterizing sources of failure. We were surprised by the performance discrepancy between the RNN and the Transformer (even without pretraining), as we expected both models to at least be competitive. To understand why, we parsed 100 randomly sampled incorrect outputs (see Appendix G for examples) from both the RNN and the Transformer on both datasets and found that for MultiWOZ-2.1, both models are very good at predicting the proper output structure (i.e. delineating by commas), but the Transformer struggles significantly more at predicting the *value* of information. Although the Transformer can identify proper domain-name and slot-name, it struggles at translating the proper input entities. It often mixes up entities from other domain-slot pairs, or mixes up entities said from the user vs. entities said from the agent. It seems the copy-mechanism in the RNN allows it to perform this task more successfully, so we question whether it would be beneficial for Transformers. For SMCalFlow, the Transformer struggles with the ordering of information, which isn't a concern in MutilWOZ-2.1, but is for SMCalFlow's dataflow format. We have no obvious answer as to why, and perhaps this lack of hierarchical reasoning over tokens in its target vocabulary is something that could be resolved with larger, pretrained causal language model [26] or by modifying the input context.

6 Conclusion

In this work, we compared Seq2Seq and SFA for DST, as well as the performance of RNNs vs. Transformers in application to this task. We highlighted that despite the promise of Seq2Seq modelling for DST, SFAs still perform better but are limited to certain types of data. Further, we highlighted that despite the promise of Transformers over RNNs, Transformers struggle significantly more in DST.

Contribution (not part of main draft)

Write Up

Introduction: Sarvagya Agrawal

Related Works: Sarvagya Agrawal

Methodology: Mathieu Tuli

Experiments: Mathieu Tuli + Sarvagya Agrawal

Additional Discussion/Conclusion: Mathieu Tuli

Appendix: Mathieu Tuli + Sarvagya Agrawal

Code & Running Experiments

data preprocessing: Mathieu Tuli + Sarvagya Agrawal

Model training (code + running): Mathieu Tuli

Model evaluatio + Error Analysis: Mathieu Tuli

References

[1] Steve Young, Milica Gašić, Blaise Thomson, and Jason D Williams. Pomdp-based statistical spoken dialog systems: A review. *Proceedings of the IEEE*, 101(5):1160–1179, 2013.

[2] Ehsan Hosseini-Asl, Bryan McCann, Chien-Sheng Wu, Semih Yavuz, and Richard Socher. A simple language model for task-oriented dialogue. *arXiv preprint arXiv:2005.00796*, 2020.

[3] Andrea Madotto. Language models as few-shot learner for task-oriented dialogue systems. *arXiv preprint arXiv:2008.06239*, 2020.

[4] Tsung-Hsien Wen, David Vandyke, Nikola Mrksic, Milica Gasic, Lina M Rojas-Barahona, Pei-Hao Su, Stefan Ultes, and Steve Young. A network-based end-to-end trainable task-oriented dialogue system. *arXiv preprint arXiv:1604.04562*, 2016.

[5] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *arXiv preprint arXiv:1409.3215*, 2014.

[6] Michael Heck, Carel van Niekerk, Nurul Lubis, Christian Geishauser, Hsien-Chin Lin, Marco Moresi, and Milica Gašić. Trippy: A triple copy strategy for value independent neural dialog state tracking. *arXiv preprint arXiv:2005.02877*, 2020.

[7] Samuel Louvan and Bernardo Magnini. Recent neural methods on slot filling and intent classification for task-oriented dialogue systems: A survey. *arXiv preprint arXiv:2011.00564*, 2020.

[8] Yue Feng, Yang Wang, and Hang Li. A sequence-to-sequence approach to dialogue state tracking. *arXiv preprint arXiv:2011.09553*, 2020.

[9] Jacob Andreas, John Bufe, David Burkett, Charles Chen, Josh Clausman, Jean Crawford, Kate Crim, Jordan DeLoach, Leah Dorner, Jason Eisner, et al. Task-oriented dialogue as dataflow synthesis. *Transactions of the Association for Computational Linguistics*, 8:556–571, 2020.

[10] Abhinav Rastogi, Dilek Hakkani-Tür, and Larry Heck. Scalable multi-domain dialogue state tracking. In *2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pages 561–568. IEEE, 2017.

[11] Chien-Sheng Wu, Andrea Madotto, Ehsan Hosseini-Asl, Caiming Xiong, Richard Socher, and Pascale Fung. Transferable multi-domain state generator for task-oriented dialogue systems. *arXiv preprint arXiv:1905.08743*, 2019.

[12] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[13] Haoyang Wen, Yijia Liu, Wanxiang Che, Libo Qin, and Ting Liu. Sequence-to-sequence learning for task-oriented dialogue with dialogue state representation. *arXiv preprint arXiv:1806.04441*, 2018.

[14] Grégoire Mesnil, Yann Dauphin, Kaisheng Yao, Yoshua Bengio, Li Deng, Dilek Hakkani-Tur, Xiaodong He, Larry Heck, Gokhan Tur, Dong Yu, et al. Using recurrent neural networks for slot filling in spoken language understanding. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(3):530–539, 2014.

[15] Bing Liu and Ian Lane. Attention-based recurrent neural network models for joint intent detection and slot filling. *arXiv preprint arXiv:1609.01454*, 2016.

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[17] Qian Chen, Zhu Zhuo, and Wen Wang. Bert for joint intent classification and slot filling. *arXiv preprint arXiv:1902.10909*, 2019.

[18] Yan Zeng and Jian-Yun Nie. Jointly optimizing state operation prediction and value generation for dialogue state tracking, 2021.

[19] Mihail Eric, Rahul Goel, Shachi Paul, Adarsh Kumar, Abhishek Sethi, Peter Ku, Anuj Kumar Goyal, Sanchit Agarwal, Shuyang Gao, and Dilek Hakkani-Tur. Multiwoz 2.1: A consolidated multi-domain dialogue dataset with state corrections and state tracking baselines, 2019.

[20] Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander Rush. OpenNMT: Open-source toolkit for neural machine translation. In *Proceedings of ACL 2017, System Demonstrations*, pages 67–72, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[21] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[22] Abigail See, Peter J Liu, and Christopher D Manning. Get to the point: Summarization with pointer-generator networks. *arXiv preprint arXiv:1704.04368*, 2017.

[23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[24] Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical neural story generation. *arXiv preprint arXiv:1805.04833*, 2018.

[25] Kevin Lu, Aditya Grover, Pieter Abbeel, and Igor Mordatch. Pretrained transformers as universal computation engines, 2021.

[26] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[27] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[28] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Huggingface's transformers: State-of-the-art natural language processing, 2020.

A Slot-Filling Architectures

Traditionally, slot-filling methods can be generalized as follows: (1) each architecture will have a set of slot-gates, one per domain-slot pair, that is responsible for keeping tracking of the different information over a dialogue; (2) an encoder of some kind (often complex Bert-like architectures [6]) will encode a user utterance (or some context of user utterances and/or agent utterances) into some context vector $Z \in \mathbb{R}^D$; (3) using Z, for each slot-gate independently, we perform classification over the set of classes C as

$$P = softmax(WZ + b) \in \mathbb{R}^{|C|} \tag{1}$$

where $W \in \mathbb{R}^{|C| \times D}$ and $b \in \mathbb{R}^{|C|}$. C is simply the status of the slot-gate, and often holds classes such as $\{none, dontcare, span, inform, refer, \ldots\}$ that denote what kind of information flow is going to occur for the current slot-gate (i.e. receive new information, ignore information, etc.).; (4) a final state generator will then extract/fill-in the proper information for each slot (this is often some additional language modelling head on-top of say Bert base).

B Pointer-Generator Architecture

We describe the pointer-generator network architecture as in [22]. At each iteration t, the bi-directional LSTM encoder encodes token x_t from the input sequence, which produces a hidden state h_i. The decoder similar has a decoder hidden state s_t produced from the previous word-embedding (during training, we use teacher forcing, feeding in the embedding from the ground truth output sequence). Attention over both states is thus calculated as

$$a_t = softmax\left(v^\top \tanh(W_h h_i + W_s s_t + b)\right) \tag{2}$$

Note that v, W_h, and W_s are all learnable parameters. This attention distribution tells the decoder what to pay attention to. This distribution is used to produce a context vector $h'_t = \sum_i a_{t,i} h_i$.

The context vector is further concatenated with the decoder hidden state s_t. We then get the distribution over the vocabulary $P_v(w)$ while predicting word w by passing our concatenated vector through two linear layers. This distribution can be calculated as:

$$P_v(w) = softmax(V'(V[s_t, h'_t] + b) + b') \tag{3}$$

Here, V, V', b and b' are learnable parameters.

The copying mechanism of the pointer generator network is established by the generation probability $p \in [0, 1]$ which is the probability of the predicted word being generated from the given vocabulary as defined in [22]:

$$p = \sigma(w_{h'}^\top h'_t + w_s^\top s_t + w_x^\top x_t + b_p) \tag{4}$$

Here, $w_{h'}, w_s, w_x$ and $b \in \mathbb{R}$ are learnable parameters. This probability is the factor which chooses whether the generated word comes from the vocabulary or the input text. Then we obtain a probability distribution of the next word as follows:

$$P(w) = pP_v(w) + (1 - p) \sum_{i:w_i = w} a_{t,i} \tag{5}$$

As we can see, the generation probability p controls whether the next word will be taken from the vocabulary or input.

The pointer generator network in [22] also employs what is referred to as a *coverage mechanism* to solve the common problem of repetition in sequence-to-sequence models. The coverage model requires us to define a coverage vector c_t as follows:

$$c_t = \sum_{t'=0}^{t-1} a_{t'} \tag{6}$$

Then, to instil this coverage mechanism into the pointer-generator network, the attention mechanism defined in equation (2) is modified as follows:

$$a_t = softmax\left(v^\top \tanh(W_h h_i + W_s s_t + w_c c_{i,t} + b)\right) \tag{7}$$

As before, w_c is a learnable parameter.

Now, the probability distribution $P(w)$ given in equation (5) can be calculated with the modified attention as shown in equation (7).

C Transformer Architecture

The Transformer architecture is an encoder-decoder model. It is defined as follow from [23].

The encoder is a stack of N of the same layers, where each layer is composed of two sub-layers: (1) a multi-head self-attention mechanism and (2) a fully connected feed-forward network. Both sub-layers employ a residual connected and layer normalization.

The decoder is a stack of M of the same layers, where each layer is composed of three sub-layers: (1) a multi-head self-attention mechanism, (2) a fully connected feed-forward network, and (3) another multi-head attention mechanism. This third layer is responsible for attending to the output of the encoder. As before, each sub-layer employs layer normalization with a residual connection. Further, the attention in the decoder is masked to prevent attending to future tokens in the sequence.

Note that all embedding layers and sub-layers are of dimension D. We can formulate as follows:

$$\text{Attention}(Q, K, V) = softmax(\frac{QK^\top}{\sqrt{d_k}})V$$

where $Q, K \in \mathbb{R}^{d_k}$ and $V \in \mathbb{R}^{d_v}$. Following, multi-head attention is simply the contenation of multiple attention mechanisms as

$$\begin{aligned} \text{MHAttention}(Q, K, V) &= [head_1; \ldots; head_k]W^O \\ head_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \end{aligned}$$

where $[;]$ is the concatenation operation and $QW_i^Q \in \mathbb{R}^{D \times d_k}, KW_i^K \in \mathbb{R}^{D \times d_k}, VW_i^V \in \mathbb{R}^{D \times d_v}$, and $W^O \in \mathbb{R}^{hd_v \times D}$ are learnable parameter matrices.

The fully connected feed-forward network is constructed as two linear transformations with ReLU activation in between [23].

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

A 2048-dimensional full-connected layer is used.

As mentioned in the draft, we use sinusoidal position embeddings, which are defined as

$$\begin{aligned} PE_{pos,2i} &= sin(pos/100000^{2i/D}) \\ PE_{pos,2i+1} &= cos(pos/100000^{2i/D}) \end{aligned}$$

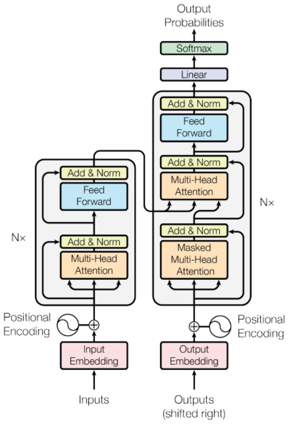

Figure 1: Transformer architecture taken from [23]

The final probability distribution over the vocabulary is achieved by passing the inputs through both the encoder and the decoder as visualized in Figure 1, and taking a softmax over the vocabulary.

D Hyper-Parameter Tuning

For the pointer-generator network, we used a learning rate of 0.001 with the Adam [27] optimizer, a dropout of 0.5 and no attention dropout. This follows the suggested hyper-parameters from [9]. This configuration was applied to both datasets. We additionally tested learning rates of $\{0.1, 0.01, 0.005\}$ and found no improvements. For the Transformer network, we used a learning rate of 2, using the Adam optimizer with Noam decay and dropout of 0.1. This configuration is chosen as it replicates results from [23, 20]. We additionally tested learning rates of $\{1, 0.1, 0.005, 0.00005\}$ based on other common configurations found in HuggingFace [28], but found no improvements. For TripPy, we simply used their suggested default hyper-parameters as no changes to the experiments were made and they performed hyper-parameter optimization. See their paper for additional details. A batch size of 128 is used for the pointer-generator and Transformer, and 48 for TripPy.

For testing, we test greedy search and beam search of widths $\{2, 5, 10\}$ and found beam search of width to be optimal 10.

E Data

Table 2: Examples from the SMCalFlow dataset with $\ell = 1$.

Input Context (C_1)	__User What time is my appointment with Jerri Skinner on Friday? __StartOfProgram
Output Belief State (B_1)	(Yield :output (:start (singleton (:results (FindEventWrapperWithDefaults :constraint (EventOnDate :date (NextDOW :dow # (DayOfWeek " FRIDAY ")) :event (Constraint[Event] :attendees (AttendeeListHasRecipientConstraint :recipientConstraint (RecipientWithNameLike :constraint (Constraint[Recipient]) :name # (PersonName " Jerri Skinner ")))))))))))
Input Context (C_{184})	__User Delete Senior Expo to Boomers __StartOfProgram (Yield :output (DeleteCommitEventWrapper :event (DeletePreflightEventWrapper :id (:id (singleton (:results (FindEventWrapperWithDefaults :constraint (Constraint[Event] :subject (? = # (String " Senior Expo to Boomers ")))))))))) entity@1503303357 __User No, I actually need lunch canceled that day instead. __StartOfProgram
Output Belief State (B_{184})	(Yield :output (Execute :intension (ReviseConstraint :rootLocation (roleConstraint # (Path " output ")) :oldLocation (Constraint[Constraint[Event]]) :new (EventOnDate :date (Execute :intension (refer (extensionConstraint (Constraint[Date])))) :event (Constraint[Event] :subject (? = # (String " lunch ")))))))

Table 3: Examples from the MultiWOZ-2.1 dataset.

Input Context (C_1)	__User am looking for a place to to stay that has cheap price range it should be in a type of hotel __Belief
Output Belief State (B_1)	< hotel name not mentioned , hotel area not mentioned , hotel parking not mentioned , hotel pricerange cheap , hotel stars not mentioned , hotel internet not mentioned , hotel type hotel >
Input Context (C_{160})	__User whoa whoa , easy there tiger , lets narrow the search down first . in the center , 0 stars , cheap and it can be a hotel or guesthouse . __Agent how about el shaddai ? fits your request perfectly . __User is el shaddai a guest house or hotel ? __Belief
Output Belief State (B_{160})	< restaurant food seafood , restaurant pricerange not mentioned , restaurant name not mentioned , restaurant area centre , hotel name el shaddia guesthouse , hotel area centre , hotel parking not mentioned , hotel pricerange cheap , hotel stars 0 , hotel internet not mentioned , hotel type dontcare , train leaveat not mentioned , train destination cambridge , train day wednesday , train arriveby 08:00 , train departure stevenage , train book people 7 >

F Failure of Pretrained Transformer

Table 4: Examples of the pretrained Transformer on MultiWOZ-2.1.

Input Context	__User i need information on a hotel that include -s free parking please . __Belief
Ground Truth Belief State	< hotel name not mentioned , hotel area not mentioned , hotel parking yes , hotel pricerange not mentioned , hotel stars not mentioned , hotel internet not mentioned , hotel type not mentioned >
Predicted Belief State	-s hotel name not mentioned , hotel area not mentioned , hotel parking yes , hotel __Belief not mentioned , hotel stars not mentioned , hotel internet not mentioned , hotel type hotel
Input Context	__User could you help me find a guesthouse on the west side ? __Agent there are 2 guesthouses on the west side , 1 cheap and 1 moderate -ly priced . __User does either of those offer free parking ? __Agent actually , they both offer free wifi and free parking . finches b & b is cheap -er , and the hobsons house is more moderate -ly priced . which do you prefer ? __User let s go with finches . can you book me a room for 2 people on saturday ? we'd like to stat for 4 nights . __Agent i have booked your stay . do you need anything else ? __User just the reference number thanks __Agent sure thing , your reference number is e7r5knp0 . would you like me to help you find anything else today ? __User that is all ! thank you ! __Belief
Ground Truth Belief State	< hotel name finches bed and breakfast , hotel area west , hotel parking yes , hotel pricerange not mentioned , hotel stars not mentioned , hotel internet not mentioned , hotel type guesthouse , hotel book stay 4 , hotel book day saturday , hotel book people 2 >
Predicted Belief State	, hotel name not mentioned , hotel area west , hotel parking not mentioned , hotel , not mentioned , hotel stars not mentioned , hotel internet not mentioned , hotel type -er , hotel book people 2 , hotel book day saturday , hotel book stay 4 ,

G Characterizing Sources of Failure

Table 5: Incorrectly predicted outputs from the MultiWOZ-2.1 dataset using the Transformer. Notice it's struggle to relate the value of information, but yet it correctly gets the domain-slot pairs.

Input Context	__User i need information on a hotel that include -s free parking please . __Belief
Ground Truth Belief State	< hotel name not mentioned , hotel area not mentioned , hotel parking yes , hotel pricerange not mentioned , hotel stars not mentioned , hotel internet not mentioned , hotel type not mentioned >
Predicted Belief State	< hotel name not mentioned , hotel area not mentioned , hotel parking yes , hotel pricerange not mentioned , hotel stars not mentioned , hotel internet not mentioned , hotel type hotel >
Input Context	__User could you help me find a guesthouse on the west side ? __Agent there are 2 guesthouses on the west side , 1 cheap and 1 moderate -ly priced . __User does either of those offer free parking ? __Agent actually , they both offer free wifi and free parking . finches b & b is cheap -er , and the hobsons house is more moderate -ly priced . which do you prefer ? __User let s go with finches . can you book me a room for 2 people on saturday ? we'd like to stat for 4 nights . __Agent i have booked your stay . do you need anything else ? __User just the reference number thanks __Agent sure thing , your reference number is e7r5knp0 . would you like me to help you find anything else today ? __User that is all ! thank you ! __Belief
Ground Truth Belief State	< hotel name finches bed and breakfast , hotel area west , hotel parking yes , hotel pricerange not mentioned , hotel stars not mentioned , hotel internet not mentioned , hotel type guesthouse , hotel book stay 4 , hotel book day saturday , hotel book people 2 >
Predicted Belief State	< hotel name not mentioned , hotel area west , hotel parking yes , hotel pricerange not mentioned , hotel stars not mentioned , hotel internet yes , hotel type guesthouse , hotel book stay 2 , hotel book day saturday , hotel book people 4 >
Input Context	__User What 's my schedule look like next week ? __StartOfProgram (Yield :output (FindEventWrapperWithDefaults :constraint (EventDuringRange :event (Constraint[Event]) :range (NextWeekList)))) __User And where 's that taking place ? __StartOfProgram (Yield :output (:location (Execute :intension (refer (extensionConstraint (Constraint[Event])))))) __User WHere 's the next one taking place ? __StartOfProgram
Ground Truth Belief State	(Yield :output (:location (Execute :intension (refer (cursorNext (extensionConstraint (Constraint[Event])))))))
Predicted Belief State	(Yield :output (Execute :intension (refer (cursorNext (extensionConstraint (Constraint[Event]))))))