Getting to Production with Few-shot Natural Language Generation Models

**Peyman Heidari, Arash Einolghozati, Shashank Jain, Soumya Batra,
Lee Callender, Ankit Arun, Shawn Mei, Sonal Gupta,
Pinar Donmez, Vikas Bhardwaj, Anuj Kumar, Michael White**[*]
Facebook
{peymanheidari, arashe, shajain, sbatra,
leefc, ankitarun, smei, sonalgupta,
pinared, vikasb, anujk, mwhite14850}@fb.com

Abstract

In this paper, we study the utilization of pre-trained language models to enable few-shot Natural Language Generation (NLG) in task-oriented dialog systems. We introduce a system consisting of iterative self-training and an extensible mini-template framework that textualizes the structured input data into semi-natural text to fully take advantage of pre-trained language models. We compare various representations of NLG models' input and output and show that transforming the input and output to be similar to what the language model has seen before during pre-training improves the model's few-shot performance substantially. We show that neural models can be trained with as few as 300 annotated examples while providing high fidelity, considerably lowering the resource requirements for standing up a new domain or language. This level of data efficiency removes the need for crowd-sourced data collection resulting in higher quality data annotated by expert linguists. In addition, model maintenance and debugging processes will improve in this few-shot setting. Finally, we explore distillation and using a caching system to satisfy latency requirements of real-world systems.

1 Introduction

Task-oriented dialog systems are commonplace in automated systems such as voice-controlled assistants, customer service agents, and website navigation helpers. Natural Language generation (NLG) is an essential part of task-oriented dialog systems, which converts data into natural language output to be subsequently served to the users. Since an NLG response directly impacts the user's experience, it should convey all of the information accurately, should be contextualized with respect to the user request, and be fluent and natural.

Commercial NLG systems are typically built on rule- or template-based text generation methods (Reiter and Dale, 2000; Gatt and Krahmer, 2018; Dale, 2020). These systems often consist of a human-authored collection of response templates with slot value placeholders. The placeholders are later filled with the dialog input at the runtime. Template-based NLG modules provide inherent fidelity, strictly controlled style and wording, and low latency, which makes them an appealing choice. However, template-based systems are challenging to scale since new templates need to be authored for different response variations; templates authored for a prior domain are not usually reusable for future domains; and it becomes increasingly arduous to author high-quality templates for complex domains. More importantly, in spite of the high amount of time and resources it usually takes to instill linguistic information into the templates, they are not contextualized on the user query, and the limited set of templates results in bounded naturalness of the system's responses.

Recently, generative models (Wen et al., 2015; Dušek and Jurcıcek, 2016; Rao et al., 2019) have become popular for their data-driven scaling story and superior naturalness over the typical template-based systems (Gatt and Krahmer, 2018; Dale, 2020). However, training reliable and low-latency generative models has typically required tens of thousands of training samples (Balakrishnan et al., 2019; Novikova et al., 2017). Model maintenance with such a large dataset has proven to be challenging, as it is resource-intensive to debug and fix responses, make stylistic changes, and add new capabilities. Therefore, it is of paramount importance to bring up new domains and languages with as few examples as possible while maintaining quality.

Pre-trained models like GPT2 (Radford et al., 2019) have been recently adapted to perform few-shot learning for task-oriented dialog (Peng et al.,

[*] Work done while on leave from Ohio State University.

Proceedings of the 22nd Annual Meeting of the Special Interest Group on Discourse and Dialogue, pages 66–76
July 29–31, 2021. ©2021 Association for Computational Linguistics

2020; Chen et al., 2020). However, these methods have not usually addressed production concerns such as balancing latency and accuracy, which we explore in this paper. Arun et al. (2020) do also consider this trade-off in their data efficiency study, ultimately recommending several sampling and modeling techniques to attain production quality with fast, light-weight neural network models. Since their work is the most similar to ours, we focus our experiments on the most complex domain examined by Arun et al. (2020), the weather dataset, and demonstrate that we can achieve production quality with approximately 8X higher data-efficiency levels by making use of textualized inputs and iterative self-training. In particular, we propose scalable mini-templates to convert structured input into sub-natural text that is more suitable for re-writing by language models. We also utilize knowledge distillation and caching to make our models suitable for production. Finally, we explore model-based acceptability classifiers to ensure fidelity of the generated responses, which is essential for a real-life NLG system. Using this framework, we show that we can bring up a new domain with realistic complexity using only 300 annotated examples.

Our specific contributions are as follows:

1. we introduce a generalizable bottom-up templating strategy to convert structured inputs to semi-natural text;

2. we present results of experiments with different representations of input data and output text including structured vs. textual and lexicalized vs. partially delexicalized;

3. we propose a combination of using pre-trained language models, self-training, knowledge distillation, and caching to train production-grade few-shot NLG models; and

4. we release datasets, model predictions, and human judgements to study the NLG domain stand-up under the few-shot setting.

2 Related Work

Pre-trained language models have shown promising results for generation tasks such as translation, summarization and data-to-text (Lewis et al., 2020; Yang et al., 2020). As noted above, Peng et al. (2020) and Chen et al. (2020) likewise explore pre-trained models for few-shot NLG in task-oriented dialog, but they do not investigate how to achieve acceptable latency while maintaining high quality.

Using templates alongside pre-trained language models for NLG has been recently introduced by Kale and Rastogi (2020), where templates for simple input scenarios are concatenated to form a template for a more complicated scenario. The templated scenario is then fed to a pre-trained language model instead of the structured input. In contrast to this flat approach, which creates a verbose input for the models to re-write, we use an efficient bottom-up approach with simple mini-templates to "textualize" the individual slots and dialog acts to semi-natural and telegraphic text. As such, we don't need to have various templates for simple scenarios and require only one rule for each new slot to be published with the possibility of choosing from several predefined rules. Moreover, the rules can be reused across domains which helps with efficiency and generalization. Also related is the approach of Kasner and Dušek (2020), who use templates extracted from the training data in part, though their approach is then followed by automatic editing and reranking steps.

Self-training has been previously investigated for NLG by Kedzie and McKeown (2019) and Qader et al. (2019), though they do not explore using pre-trained models with self-training. Also related are earlier approaches that use cycle consistency between parsing and generation models for automatic data cleaning (Nie et al., 2019; Chisholm et al., 2017). More recently, Chang et al. (2021) have developed a method for randomly generating new text samples with GPT-2 then automatically pairing them with data samples. By comparison, we take a much more direct and traditional approach to generating new text samples from unpaired inputs in self-training (He et al., 2020), using pre-trained models fine-tuned on the few-shot data for both generation and reconstruction filtering.

3 Task

Our task is to convert a tree-based scenario into natural text, given the original query. An example data item together with its transformations (Section 4) is shown in Table 1.

3.1 Data

Our experiments were conducted using 4 task-oriented datasets. We focused on the most challenging dataset, Conversational Weather, which is

Query	How is the weather over the next weekend?
Structured MR	INFORM_1[temp_low[20] temp_high[45] date_time[colloquial[next weekend]]] CONTRAST_1[INFORM_2[condition[sun] date_time[weekday[Saturday]]] INFORM_3[condition[rain] date_time[weekday[Sunday]]]]
Delexicalized Structured MR	INFORM_1[temp_low[temp_low_1] temp_high[temp_high_1] date_time[colloquial [next weekend]]] CONTRAST_1[INFORM_2[condition[sun] date_time[weekday[weekday_1]]] INFORM_3[condition[rain] date_time[weekday[weekday_2]]]]
Textualized MR	inform low temperature 20, high temperature 45, next weekend. inform sun, on Saturday but inform rain, on Sunday.
Delexicalized Textualized MR	inform low temperature temp_low_1, high temperature temp_high_1, next weekend. inform sun, on weekday_1 but inform rain, on weekday_2.
Structured Reference	INFORM_1[date_time[colloquial[next weekend]]expect a low of temp_low[20] and a high of temp_high[45].] CONTRAST_1[INFORM_2[it will be condition[sunny] date_time[on weekday[Saturday]]] but INFORM_3[it'll condition[rain] date_time[on weekday[Sunday]]] .]
Delexicalized Structured Reference	INFORM_1[date_time[colloquial[next weekend]]expect a low of temp_low[temp_low_1]and a high of temp_high[temp_high_1].] CONTRAST_1[INFORM_2[it will be condition[sunny] date_time[on weekday[weekday_1]]] but INFORM_3[it'll condition[rain] date_time[on weekday[weekday_2]]] .]
Reference	Next weekend expect a low of 20 and a high of 45. It will be sunny on Saturday but it'll rain on Sunday.
Delexicalized Reference	Next weekend expect a low of temp_low_1 and a high of temp_high_1. It will be sunny on weekday_1 but it'll rain on weekday_2.

Table 1: Representations of NLG input and output. **Query**, **Structured MR**, and **Delexicalized Structured MR** are inputs to the NLG task. **Textualized MR** and **Delexicalized Textualized MR** are intermediate model inputs. **Reference** is our desired output, which can be delexicalized in text format as seen in **Delexicalized Reference** or annotated as seen in **Structured Reference** and **Delexicalized Structured MR**.

similar to the one introduced in Balakrishnan et al. (2019). We also used three additional datasets for joint training, namely the Reminder, Time, and Alarm domains released in Arun et al. (2020).

All of the datasets use a tree structure to convey the meaning representation (MR) that has been discussed in Balakrishnan et al. (2019). Discourse relations (CONTRAST and JUSTIFY) were used in some examples to connect a possible list of dialog acts (REQUEST, INFORM, etc.). Many examples contain only a few dialog acts without discourse relations.The dialog acts contain a list of slot key and value pairs. The synthetic user queries and scenarios were generated by engineers, while the annotated responses were created by human annotators following guidelines written by computational linguists. The responses were verified to be grammatical and correct by the linguists to ensure data quality.

We used two test sets for the Weather domain: (1) a challenging version which consists of data from a wider distribution of inputs compared to those we expect to encounter in production, and (2) a real-world version to evaluate the performance realistically. All of our data is simulated and created by expert linguists, who were responsible for adding the annotations illustrated in the references in Table 1. The challenging test set is used to differentiate between models and to measure model robustness in case of possible upstream changes. All reported numbers are against the challenging test set unless otherwise stated. Descriptive statistics of the datasets are shown in Table 2. The new real-world test set for Weather contains 800 samples.[1]

3.2 Metrics

Human evaluation is used to compare the effect of input and output structure and delexicalization on model performance. Judgments were obtained for 493 samples out of the challenging test set. Fol-

[1] The textualized datasets, model outputs, and human evaluation data can be found at https://github.com/facebookresearch/FewShotNLG

Domain	Training	Validation	Test
Weather	25390	3078	3121
Reminder	9716	2794	1397
Time	5530	1529	790
Alarm	7163	2024	1024

Table 2: Number of examples in training, validation, and test sets for all datasets.

lowing Arun et al. (2020), each sample was evaluated by two separate annotators followed by a tie-breaker for correctness and grammaticality:

Correctness Evaluation of semantic correctness of a response. Annotators check for missing slots, hallucinations, and bad slot aggregation.

Grammaticality Checks for grammatical correctness of a sentence, which includes completeness, subject-verb agreement, word order, sentence structure, etc.

In the results, we report the correctness and grammaticality percentage as the proportion of the test items judged to be both correct and grammatical.

We also use *Reconstruction Accuracy* as an offline metric to measure the effect of data reduction and self-training on model performance. We finetune BART large as a reverse model converting responses to input scenarios. After the generation task, the reconstruction model is used to regenerate the scenario. For each sample, if the reconstructed scenario is exactly the same as the original scenario, we count that as a correct generation (Qader et al., 2019). Note that using reconstruction in production is prohibitive due to its high latency.

3.3 Models

The model architectures used in this study are either LSTM-based sequence-to-sequence (S2S) models (Bahdanau et al., 2014) or derivatives of a pretrained large transformer-based S2S model called BART (Lewis et al., 2019). For BART, we use four variants with a total of 6 to 24 encoder and decoder layers (Section 4.3). BART uses byte pair encoding as the tokenization method. For each model fine-tuning, we use the ADAM optimizer with 300 warm-up steps. The initial learning rate of 5e-5 is reduced by a factor of 0.5 if validation loss plateaus for 3 epochs. Each model is trained for 100 epochs with a batch size of 32 (across 8 GPUS) with an early stopping strategy terminating the training if the validation loss stops decreasing for 5 epochs. To decrease latency, all models use a beam size of 1.

In the LSTM-based models, we use trainable $50d$ GloVe embeddings. The tokenization is word based with possibility of out of vocabulary tokens. We use the ADAM optimizer to train the models from random initialization. An initial learning rate of 0.01 is used, which gets reduced by a factor of 0.1 if validation loss plateaus for 2 epochs. The loss function is label smoothed cross entropy, where the beta parameter is between [0.01, 1]. A batch size of 32 is used and all models are trained for 100 epochs with early stopping after 5 epochs.[2]

4 Methodology

4.1 Input and Output Representation

The Meaning Representation (MR) consumed by our NLG model is a tree consisting of discourse relations, dialog acts, and slots (possibly nested). An example of such input is shown in Table 1. We hypothesize that we can utilize the power of pretrained models more effectively by transforming the input to a form closer to what the models have seen during pre-training. As such, we textualize the input trees using mini-templates. We provide templates for the individual nodes in the tree (i.e., dialog acts and slot labels). As such, we traverse the scenario tree and textualize the input iteratively by combining the templates for the nodes we come across (Table 1).

As mentioned earlier, Kale and Rastogi (2020) propose an approach of using templates for simple input scenarios to form input for more complicated flat scenarios, which where subsequently fed to a pre-trained language model. Our approach requires less manual effort since it adopts a bottom-up approach with simpler mini-templates to "textualize" the individual slots (possibly nested) as shown in Figure 1. We recommend several templating schemes which enable us to add new domains to the framework with less resources. As a guideline, one should choose a templating scheme for

[2] Since the model response is conditioned on the user query as well as the meaning representation, there is in principle some risk that BART could generate inappropriate (e.g., profane) outputs in response to specific user queries. While we leave a full investigation of this issue to future work, in practice we have observed that the risk appears to be very low, as the user's query must be recognized as a valid intent before the model is invoked to generate a response, and the model learns to condition the response on the input only in limited ways. Additionally, for task-oriented domains such as weather, it is possible to use a limited vocabulary to further reduce any such risk.

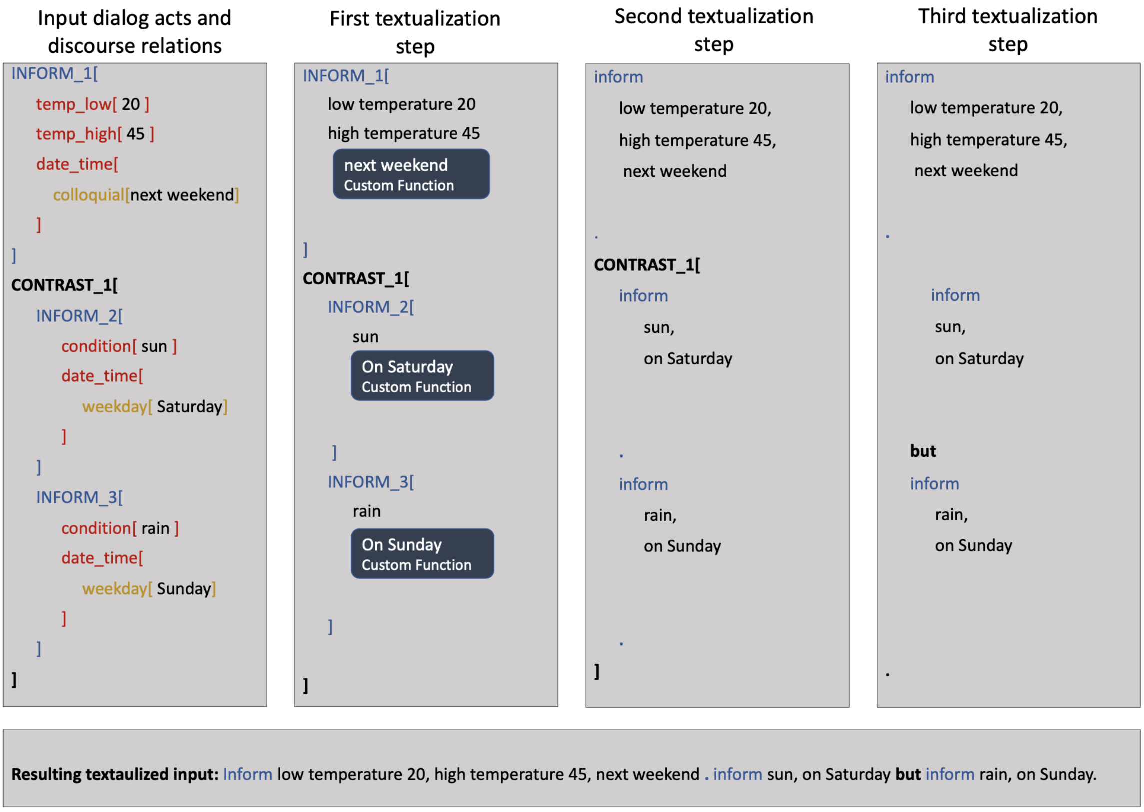

Figure 1: Textualization process using configurable pre-defined templates and custom templates.

new slots that makes the textualized representation understandable for humans. While some slots might require custom templates, our experiments have shown that those are just a small fraction of all slots. Our proposed templating schemes are:

- **Dialog acts:** We prepend the name of the intended dialog act to textualize them after all their slots have been previously textualized.
- **Discourse relations:** Since discourse relations always encompass dialog acts, we use a mapping of them with discourse connectives. For example, dialog acts inside a `Contrast` relation are joined using a *but*, while those inside `Join` are mapped to *and*.
- **Slot values:** A possible behavior for textualizing slots inside dialog acts is just to mention the slot value. For example, we chose to represent weather `condition` using this scheme.
- **Slot name and values:** Slot names are replaced by an engineer-defined string and placed before slot values. For example, we represent slots such as `low_temperature` and `high_temperature` using this method since just using slot values is misleading for the models.
- **Custom:** Writing custom templates for complex slots might be necessary to give the models a better chance to produce high-quality responses. For example, `date_time` and `date_time_range` are textualized using this method in this work.
- **Default:** The default behavior for textualizing any slot which has not been assigned another method is to remove underscores from slot names and prepend it to its slot value. This default behavior enables us to use this system on new domains without any change and expect reasonable performance.

The second technique that we explore is delexicalizing the slot values in order to mitigate model hallucination. During our initial experiments, we observed that in few-shot settings, pre-trained language models can drop some slots or fail to exactly copy their values, which can be catastrophic in a production system. This has been observed in other generation tasks using pre-trained models as well (Einolghozati et al., 2020). Therefore, we explore delexicalization of slots when linguistically permissible. For example, weather `condition` can not be delexicalized since its different values such as `sand storm` or `fog` will change the surface form of the sentence significantly while a slot such as `weekday` can be delexicalized. We also combine the few-shot Weather samples with data for three other domains to provide the model with more task-oriented data.

Balakrishnan et al. (2019) have previously shown that even with delexicalization of slot values, maintaining the tree structure in the output as generated semantic annotations (as shown in Table 1) is useful for rule-based correctness checking of low-capacity LSTM-based NLG models in the full-data setting. Our hypothesis is instead that generating plain (rather than structured) text, together with textualizing the input structure and delexicalization, can help the few-shot NLG task with better utilization of large pre-trained models. In addition, we observe that maintaining the structure in the output increases the sequence length and therefore increases the latency of the models significantly. Therefore, we perform experiments with different variations of the input and output structures as well as various BART sizes.

4.2 Self-Training

Annotating large quantities of high-quality data is time and resource consuming. However, it is often possible to automatically generate a lot of unlabeled data using a synthetic framework. Here, we adapt and extend the semi-supervised self-training strategy introduced by He et al. (2020). As shown in Figure 2, self-training consists of multiple cycles of generation and reconstruction.

We fine-tune BART (Lewis et al., 2020), a pre-trained seq2seq language model, for both steps. For generation, we experiment with various ways of textualizing the scenario tree, concatenated with the input query, before using it as input to the generation model. The reason for the latter is that there could be some subtleties in the original query which would be helpful in the response generation that are not included in the scenario tree. For example, Yes/No-questions are not reflected in the tree: *Is it cold?* and *What's the weather?* have the same scenario tree, though the former would require a Yes/No confirmation in the result. In parallel, the same generation data is used to fine-tune a reconstruction BART large model to obtain the generation input (without the input query), given the responses. After generation in each cycle, we use the reconstruction model to select samples with exact reconstruction match. Finally, the selected samples are added to the training pool for knowledge distillation or the next self-training cycle.[3]

Model	Latency (ms)	Encoder x Decoder (layers)
BART large	935	12 X 12
BART base	525	6 X 6
BART _3_3	253	3 X 3
BART _5_1	114	5 X 1
LSTM	34	1 X 1
Cache	9	-

Table 3: The median inference latency of different models (1000 inferences using 16GB Quadro GP100 GPUs) compared to cache latency.

4.3 Knowledge Distillation

One of the biggest obstacles in real-world application of pre-trained language models such as BART is their prohibitive latency. We explored knowledge distillation to mitigate this issue, here. We perform sequence-level knowledge distillation (Kim and Rush, 2016) from BART large to BART models with various smaller sizes, in addition to a small LSTM model (Table 3).

4.4 Caching

Another solution to mitigate the latency concerns of large models for production systems is to use caching. A median limit of 100ms for productions systems is reasonable in our view. However, as shown in Table 3, the median inference latency even after knowledge distillation into a small BART model is more than 100ms. As such, we can utilize a caching approach that stores model input and output as key-value pairs. Our cache implementation is an RPC call to an indexed datastore, with a median lookup time of **9 ms**. Even with a caching solution, knowledge distillation is essential to limit latency of 90th and 95th percentile of the traffic.

The efficacy of using a cache is largely dependent on the hit rate, which can vary by domain complexity, the inclusion of the user query in the model input, and the amount of delexicalization.

[3] As an alternative to using a reconstruction model to validate the generated responses, we could use our acceptability model (Section 4.5) to filter or rank the responses; we leave these options for future work.

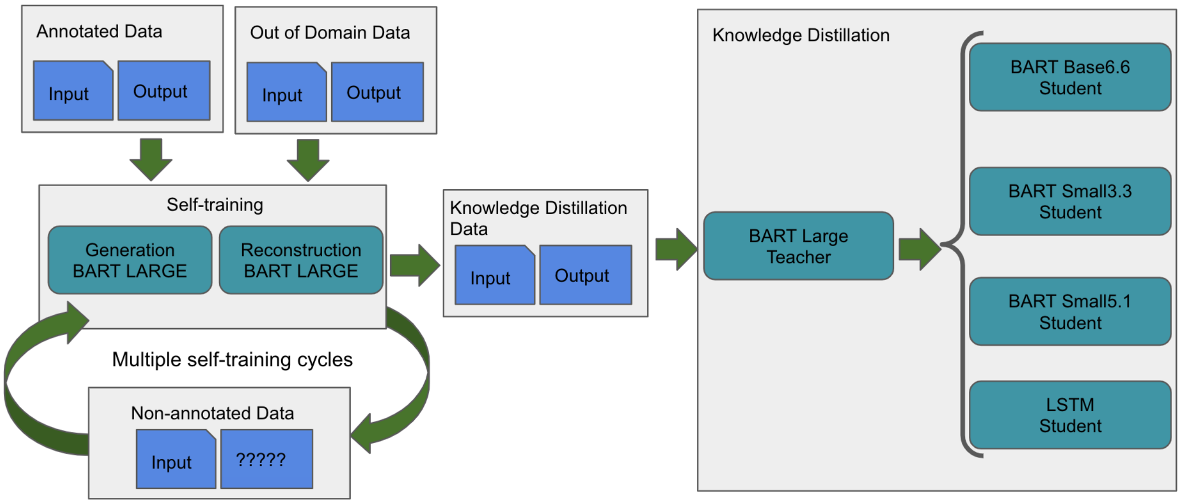

Figure 2: Few-shot NLG process consists of several cycles of self-training followed by knowledge distillation.

4.5 Acceptability Checking

In a production neural NLG system, reliable and low-latency filters are essential to guard against incorrect and ungrammatical model responses. Arun et al. (2020) proposed coupling neural models with fall-back templates to deliver more fluent model responses in a safe manner.[4] Their suggested acceptability checking method, tree accuracy (Balakrishnan et al., 2019), requires retention of the tree-based structure that we are proposing to remove. We explored several recent model-based acceptability checking mechanisms as alternatives (Harkous et al., 2020; Anonymous, 2021). Building an acceptability model requires collecting positive and negative examples. We use the samples that pass the reconstruction step of self-training as the positive ones. The challenge lies in approximating mistakes a model is likely to make in production, and creating a dataset of synthetic negative examples. Anonymous (2021) use mask filling with pre-trained models for creating synthetic incorrect examples, which we adopt using BART.

We train two models, a production-grade convolutional (DocNN) model (Jacovi et al., 2018) with median latency of 8 ms and a high-capacity pre-trained RoBERTa-Base model (Liu et al., 2019) with latency 100 ms. These binary classification models determine whether a sequence of delexicalized textualized input MR concatenated with the delexicalized model output is correct at runtime.

4.6 End-to-End Architecture

To summarize, we first transform and delexicalize the input and output of all samples using the aforementioned input transformation framework. We subsequently annotate several hundred samples from our target domain. The annotated samples are then added to the data from other domains for joint-training. Next, several (usually two) cycles of self-training (generation and reconstruction) are carried out to auto-annotate the remaining target domain input data. Subsequently, sequence-level knowledge distillation from BART large to smaller models is performed. A schematic of the training process can be seen in Figure 2. Finally, a caching system and a few-shot acceptability classifier are trained to cover all production requirements.

5 Results

5.1 Input and Output Representation

Table 4 shows the correctness and grammaticality (c&g) evaluations for various few-shot models in comparison to the full data setting. The results validate our hypothesis that transforming the structured data into a textual form (similar to those used for pre-training BART) increases model performance in few-shot settings. In addition, we observe that delexicalizing some slot values consistently boosts the performance of the NLG models. The correctness and grammaticality score is highly correlated with automatic BLEU scores.Therefore, we recommend adoption of delexed textualized input and delexed text output for training production-quality few-shot NLG models.

In the full data setting, retaining the tree structure

[4] Note that in the case of the Weather domain, the fall-back templates only convey simplified content, as the domain was deemed too complex to develop satisfactory templates for all possible combinations of dialog acts that can appear in the full input MRs.

Input representation	Output representation	BART large	BART base	BART_3_3	BART_5_1	LSTM	Full BART
Lexed Structured	Lexed Structured	73.0	71.2	70.2	69.2	69.6	90.2
Lexed Structured	Delexed Structured	71.4	71.0	67.3	67.5	66.3	**92.5**
Lexed Structured	Lexed Text	79.9	72.4	65.3	66.3	62.1	90.9
Lexed Structured	Delexed Text	81.5	76.1	72.2	68.8	66.5	91.7
Delexed Structured	Delexed Structured	77.3	72.8	67.1	71.2	74.4	90.2
Delexed Structured	Delexed Text	71.8	72.0	66.7	64.7	64.9	90.2
Lexed Textualized	Lexed Text	84.0	78.7	**80.5**	77.1	73.6	88.9
Delexed Textualized	Delexed Text	**85.2**	**80.3**	78.9	**79.5**	**78.5**	88.8

Table 4: Effect of input & output representation on correctness and grammaticality (c&g%) of few-shot model responses (using 250 annotated samples). Full BART uses all annotated training data with a BART base model as the top line. Delexed Textualized input with Delexed Text output achieves the highest performance with most few-shot models. Lexed Structured input with Delexed Structured output reaches the highest full data performance, while performing among the worst combinations in the few-shot setting. Generating delexed text boosts performance consistently compared to lexed text.

helps with more accurate natural language generation (Table 4), which is in line with observations in Balakrishnan et al. (2019). The highest c&g% of 92.5 is achieved when input is lexed structured and output is delexed structured: it is 2.3% higher than performance of the model with the same lexed structured input but with lexed structured output, which is due to the lower possibility of hallucination when the model output is delexed. In addition, this combination has higher performance compared to the one with delexed structured input and delexed structured output, which is possibly due to higher utilization of BART's encoder knowledge while processing the input sequence.

Interestingly, the lexed structured input / delexed structured output combination with the highest full data performance performs poorly in few-shot setting across the board. Indeed, its correctness and grammaticality is more than 10.0% lower than the delexed textualized input / delexed text output combination regardless of the capacity of the model used for knowledge distillation. This is more evidence validating our hypothesis that transforming the structured data into a textual form will result in more utilization of the language knowledge of pre-trained BART models.

5.2 Data Efficiency

We ran experiments at different levels of data-efficiency using BART small5.1 and evaluated their performance using a reconstruction model (trained with full data). Figure 3 shows that the reconstruction accuracy increases with more annotated data, as expected. However, even with 250 annotated samples, we achieve a reconstruction accuracy of 75.0% on the challenging test set, and our low-latency few-shot correctness model improves this to 88.7%. Interestingly, human annotations revealed a performance of 97.8% on the real-world test set for a similar model, and the same correctness model improves this to 98.8%. This observation suggests that even though there remains a substantial gap between few-shot and full-data performance on the challenging set, the few-shot models will perform satisfactorily in a real-world setting.

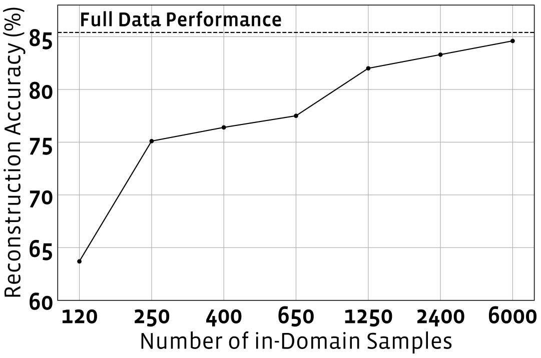

Figure 3: The effect of dataset size on model performance (BART small5.1) with two self-training cycles.

5.3 Self-Training

We also performed experiments to optimize the number of self-training cycles. As shown in Figure 4, even one cycle of self-training increases the performance of the model by 20.0%. From a pool of 31,400 unlabeled samples, more than 13,500 are added during the first self-training cycle, 5,000 more are added in the second cycle followed by just 1,400 in the third cycle. The rate of addition decreases more after the third cycle. We recommend 2-3 self-training cycles considering computational limits. For comparison, we also ran similar experiments without joint training (not using other domains) and self-training, which yields a baseline

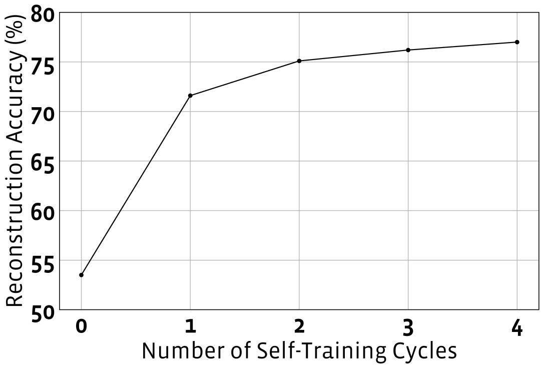

Figure 4: Model performance (BART small5.1 with 250 samples) as a function of the number of self-training cycles.

Model	Macro-F1	Precision (Co)	Recall (InCo)
DoCNN	70.5	88.8	40.9
RoBERTa	75.1	90.9	54.2

Table 5: Correctness model metrics on 493 delexed samples (83 incorrect) from a distilled BART small5.1 model (Co stands for Correct and InCo stands for Incorrect classes). Recall (Co) is kept fixed at 94.9%.

reconstruction accuracy of only 42.7%, more than 10% lower than with joint training.

5.4 Caching

For Weather, we expect a cache rate of about 60% with keys made through concatenation of user query with delexicalized textualized input MR. For BART small5.1, this bring down the median latency to 51 ms, yielding a 64% improvement. We believe that delexicalizing the user input has the potential to improve the hit rate even further. This can be done by replacing the user query words with values that have been delexicalized in the MR.

Using this cache will not reduce the variation of model responses because of how the cache key is constructed. The delexicalized MR used in the cache key will be the same for two requests only if the MRs differ at most in the values of slots that do not affect the model response. For example, if two MRs differ only in the value of `weekday`, the cache will get a hit. However, if anything else such as the weather `condition` is different, there will not be a hit. More importantly, since our models are deterministic, if the model is delexicalized as proposed here and the user query is used in the cache key, the input to the model and the cache key will be exactly the same removing any possibility of reduction in response variation.

5.5 Acceptability Checking

Table 5 shows that it is possible to train correctness models with fully synthetic negative data in a few-shot setting. Complementing high-fidelity generation models with a correctness model similar to the one here makes it possible for few-shot NLG models to meet high production quality bars.

We experimented with using distilled LSTM-based models together with tree accuracy filtering as the correctness checking mechanism, which requires structured output representations, following the recommendations in Arun et al. (2020). Our correctness models with BART small5.1 demonstrated 2.0% higher precision compared to tree accuracy with LSTMs. More importantly, tree accuracy with LSTMs filtered out many more examples (14.4%) compared to the correctness models with BART small5.1 (3.6%), making this combination less suitable at these levels of data efficiency (8X higher).

6 Conclusion

In this paper, we explored for the first time whether few-shot NLG models can be productionized, enabling us to much more effectively scale to new domains and languages. By using a system consisting of a templating approach, pre-trained language models, self-training, and an acceptability classifier, we found that we can stand up domains with a few hundred annotated samples compared to several thousands previously, while also addressing production latency needs via knowledge distillation and caching. At this level of data efficiency, there is no need for crowd-sourced data collection as expert linguists can instead annotate the data used by the system. In addition, model maintenance—including addition of new capabilities, debugging, and changing response style—will become significantly easier using the few-shot system.

Acknowledgments

We thank the anonymous reviewers for their helpful comments. Many thanks to our linguistic engineering team (Anoop Sinha, Shiun-Zu Kuo, Catharine Youngs, Kirk LaBuda, Steliana Ivanova, Ceci Pompeo, and Briana Nettie) for their hard work and for being great partners in this effort. We would also like to thank Naman Goyal for training smaller versions of BART for this paper.

References

Anonymous. 2021. Building adaptive acceptability classifiers for neural NLG. Under review.

Ankit Arun, Soumya Batra, Vikas Bhardwaj, Ashwini Challa, Pinar Donmez, Peyman Heidari, Hakan Inan, Shashank Jain, Anuj Kumar, Shawn Mei, Karthik Mohan, and Michael White. 2020. Best practices for data-efficient modeling in NLG:how to train production-ready neural models with less data. In *Proceedings of the 28th International Conference on Computational Linguistics: Industry Track*, pages 64–77, Online. International Committee on Computational Linguistics.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Anusha Balakrishnan, Jinfeng Rao, Kartikeya Upasani, Michael White, and Rajen Subba. 2019. Constrained decoding for neural NLG from compositional representations in task-oriented dialogue. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. To appear.

Ernie Chang, Xiaoyu Shen, Dawei Zhu, Vera Demberg, and Hui Su. 2021. Neural data-to-text generation with lm-based text augmentation.

Zhiyu Chen, Harini Eavani, Wenhu Chen, Yinyin Liu, and William Yang Wang. 2020. Few-shot NLG with pre-trained language model. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 183–190, Online. Association for Computational Linguistics.

Andrew Chisholm, Will Radford, and Ben Hachey. 2017. Learning to generate one-sentence biographies from Wikidata. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 633–642, Valencia, Spain. Association for Computational Linguistics.

Robert Dale. 2020. Natural language generation: The commercial state of the art in 2020. *Natural Language Engineering*. To appear.

Ondrej Dušek and Filip Jurcıcek. 2016. Sequence-to-sequence generation for spoken dialogue via deep syntax trees and strings. In *The 54th Annual Meeting of the Association for Computational Linguistics*, page 45.

Arash Einolghozati, Anchit Gupta, Keith Diedrick, and Sonal Gupta. 2020. Sound natural: Content rephrasing in dialog systems. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5101–5108, Online. Association for Computational Linguistics.

Albert Gatt and Emiel Krahmer. 2018. Survey of the state of the art in natural language generation: Core tasks, applications and evaluation. *Journal of Artificial Intelligence Research*, 61:65–170.

Hamza Harkous, Isabel Groves, and Amir Saffari. 2020. Have your text and use it too! end-to-end neural data-to-text generation with semantic fidelity. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 2410–2424, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Junxian He, Jiatao Gu, Jiajun Shen, and Marc'Aurelio Ranzato. 2020. Revisiting self-training for neural sequence generation.

Alon Jacovi, Oren Sar Shalom, and Yoav Goldberg. 2018. Understanding convolutional neural networks for text classification. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 56–65, Brussels, Belgium. Association for Computational Linguistics.

Mihir Kale and Abhinav Rastogi. 2020. Few-shot natural language generation by rewriting templates.

Zdeněk Kasner and Ondřej Dušek. 2020. Data-to-text generation with iterative text editing. In *Proceedings of the 13th International Conference on Natural Language Generation*, pages 60–67, Dublin, Ireland. Association for Computational Linguistics.

Chris Kedzie and Kathleen McKeown. 2019. A good sample is hard to find: Noise injection sampling and self-training for neural language generation models. In *Proceedings of the 12th International Conference on Natural Language Generation*, pages 584–593, Tokyo, Japan. Association for Computational Linguistics.

Yoon Kim and Alexander M. Rush. 2016. Sequence-level knowledge distillation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1317–1327, Austin, Texas. Association for Computational Linguistics.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, 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.

Feng Nie, Jin-Ge Yao, Jinpeng Wang, Rong Pan, and Chin-Yew Lin. 2019. A simple recipe towards reducing hallucination in neural surface realisation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2673–2679, Florence, Italy. Association for Computational Linguistics.

Jekaterina Novikova, Ondřej Dušek, and Verena Rieser. 2017. The e2e dataset: New challenges for end-to-end generation. *arXiv preprint arXiv:1706.09254*.

Baolin Peng, Chenguang Zhu, Chunyuan Li, Xiujun Li, Jinchao Li, Michael Zeng, and Jianfeng Gao. 2020. Few-shot natural language generation for task-oriented dialog. *Empirical Methods in Natural Language Processing (EMNLP)*.

Raheel Qader, François Portet, and Cyril Labbé. 2019. Semi-supervised neural text generation by joint learning of natural language generation and natural language understanding models. *arXiv preprint arXiv:1910.03484*.

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

Jinfeng Rao, Kartikeya Upasani, Anusha Balakrishnan, Michael White, Anuj Kumar, and Rajen Subba. 2019. A tree-to-sequence model for neural nlg in task-oriented dialog. In *Proceedings of the 12th International Conference on Natural Language Generation*, pages 95–100.

Ehud Reiter and Robert Dale. 2000. *Building Natural-Language Generation Systems*. Cambridge University Press.

Tsung-Hsien Wen, Milica Gasic, Nikola Mrkšić, Pei-Hao Su, David Vandyke, and Steve Young. 2015. Semantically conditioned LSTM-based natural language generation for spoken dialogue systems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1711–1721. Association for Computational Linguistics.

Zixiaofan Yang, Arash Einolghozati, Hakan Inan, Keith Diedrick, Angela Fan, Pinar Donmez, and Sonal Gupta. 2020. Improving text-to-text pretrained models for the graph-to-text task. In *Proceedings of the 3rd International Workshop on Natural Language Generation from the Semantic Web (WebNLG+)*, pages 107–116, Dublin, Ireland (Virtual). Association for Computational Linguistics.