Neural Pipeline for Zero-Shot Data-to-Text Generation

Anonymous ACL submission

Abstract

In data-to-text (D2T) generation, training on in-domain data leads to overfitting to the data representation and repeating training data noise. We examine how to avoid finetuning the pretrained language models (PLMs) on D2T generation datasets while still taking advantage of surface realization capabilities of PLMs. Inspired by pipeline approaches, we propose to generate text by rephrasing single-item templates using a sequence of modules trained on general-domain text-based operations—ordering, aggregation, and paragraph compression. We train PLMs for performing these operations on a synthetic corpus WIKIFLUENT which we build from English Wikipedia. Our experiments on two major triple-to-text datasets—WebNLG and E2E—show that our approach enables D2T generation from RDF triples in zero-shot settings.[1]

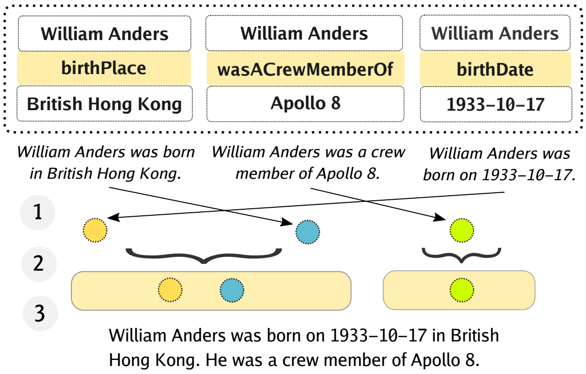

Figure 1: A scheme of our pipeline for zero-shot data-to-text generation from RDF triples: (1) ordering, (2) aggregation, (3) paragraph compression. Individual pipeline modules are trained on a large general-domain text corpus and operate over text in natural language. In-domain knowledge is included only in the simple hand-crafted templates for each predicate.

1 Introduction

The aim of data-to-text (D2T) generation is to produce natural language descriptions of structured data (Gatt and Krahmer, 2018; Reiter and Dale, 1997). Although pipelines of rule-based D2T generation modules are still used in practice (Dale, 2020), end-to-end approaches based on PLMs recently showed superior benchmark performance (Ke et al., 2021; Chen et al., 2020a; Ferreira et al., 2020; Kale and Rastogi, 2020b; Ribeiro et al., 2020), surpassing pipeline systems (Ferreira et al., 2019) in both automatic and human evaluation metrics.

Finetuning PLMs on human-written references is widely accepted as a standard approach for adapting PLMs to the D2T generation objective and achieving good performance on a given benchmark (Agarwal et al., 2021; Ke et al., 2021). Nevertheless, this approach brings issues: Most obviously, finetuning the model for the domain-specific data distribution leads to overfitting on the particular benchmark, decreasing performance on out-of-distribution data (Laha et al., 2020). Moreover, collecting a large set of references for a particular domain is costly and time-consuming, as the data are usually collected using crowdsourcing (Dušek et al., 2020). Few-shot approaches are an alternative, requring only several tens or hundreds of annotated examples (Chen et al., 2020c; Ke et al., 2021; Su et al., 2021a). However, robustness of these approaches is questionable—selecting a representative set of examples which would improve performance is difficult (Chang et al., 2021a), and the limited sample is often noisy, increasing the chance of hallucinations and omissions (Dušek et al., 2019; Harkous et al., 2020; Rebuffel et al., 2021).

In this paper, we provide an alternative to this traditional paradigm by formulating the D2T generation from RDF triples as a sequence of general-domain operations over text in natural language. We start by transforming individual triples to text using trivial templates, which we subsequently or-

[1] The anonymized version of our code and data is available at https://anonymous.4open.science/r/zeroshot-d2t-pipeline/.

der, aggregate, and compress on the paragraph level to produce the resulting description of the data. All the pipeline modules operate over natural language text and are built upon PLMs trained on our WIKIFLUENT corpus. WIKIFLUENT contains 934k examples of first paragraphs from the English Wikipedia, each supplied with a synthesized set of simple template-like sentences conveying the same meaning. Our approach allows generating natural language descriptions from triples with a minimum amount of domain-specific rules or knowledge and without using training data from the D2T datasets. We show that our approach can yield large improvements upon simple baselines and match older supervised systems in terms of fluency, while bringing potential for further improvements and advantages with respect to controllability.

Our contributions are the following:

(1) We propose an alternative D2T generation approach based on general-domain text-to-text operations (ordering, aggregation, and paragraph compression).
(2) We introduce a synthetic WIKIFLUENT corpus containing 934k sentences based on English Wikipedia, providing training data for the operations in (1).
(3) We apply our system on two D2T datasets and evaluate its performance both automatically and manually, including the contribution of individual pipeline modules.
(4) We release our code, data, pretrained models, and system outputs to ease future research.

2 Related Work

D2T Generation with PLMs Large neural language models pretrained on self-supervised tasks (Lewis et al., 2020; Liu et al., 2019; Devlin et al., 2019) have recently gained a lot of traction in D2T generation research (Ferreira et al., 2020). Following Chen et al. (2020c), other works adopt PLMs for few-shot D2T generation (Chang et al., 2021b; Su et al., 2021a). Kale and Rastogi (2020b) and Ribeiro et al. (2020) showed that PLMs using linearized representations of data can outperform graph neural networks on graph-to-text datasets, recently surpassed again by graph-based models (Ke et al., 2021; Chen et al., 2020a). Although the models make use of general-domain pretraining tasks, all of them are eventually finetuned on domain-specific data.

Templates in Data-Driven D2T Generation Using simple handcrafted templates for individual keys or predicates is an efficient way of introducing domain knowledge while preventing text-to-text models from overfitting to a specific data format (Heidari et al., 2021; Kale and Rastogi, 2020a; Kasner and Dušek, 2020). Transforming individual triples to text is also used in Laha et al. (2020) whose work is the most similar to ours. They also build a three-step pipeline for zero-shot D2T generation, but they use handcrafted rules for producing the output text and do not address content planning.

Content Planning in D2T Generation Content planning, i.e. ordering input facts and aggregating them into individual sentences, is a traditional part of the D2T generation pipeline (Ferreira et al., 2019; Gatt and Krahmer, 2018; Reiter and Dale, 1997). As previously demonstrated, using a content plan in neural D2T generation has important impact on the overall text quality (Moryossef et al., 2019a,b; Puduppully et al., 2019; Zhao et al., 2020; Trisedya et al., 2020). Recently, Su et al. (2021b) have shown that using a content plan leads to improved quality of PLM outputs. All the aforementioned models plan directly using predicates or keys in the D2T datasets representing the corresponding data item. Unlike these works, our planner is trained on ordering sentences in natural language.

Sentence Ordering Sentence ordering is the task of organizing a set of natural language sentences to increase the coherence of a text (Barzilay et al., 2001; Lapata, 2003). Several neural methods for this task were proposed, using either interactions between pairs of sentences (Chen et al., 2016; Li and Jurafsky, 2017), global interactions (Gong et al., 2016; Wang and Wan, 2019), or combination of both (Cui et al., 2020). We base our ordering module (§5.1) on the recent work of Calizzano et al. (2021), who use a pointer network (Wang and Wan, 2019; Vinyals et al., 2015) on top of a PLM.

Fact Aggregation The compact nature of the target text description results in aggregating multiple facts in a single sentence. Previous works (Wiseman et al., 2018; Shao et al., 2019; Shen et al., 2020; Xu et al., 2021) capture the segments which correspond to individual parts of the input as latent variables. Unlike these works, we adopt a simpler scenario using an already ordered sequence of facts, in which we selectively insert delimiters marking sentence boundaries.

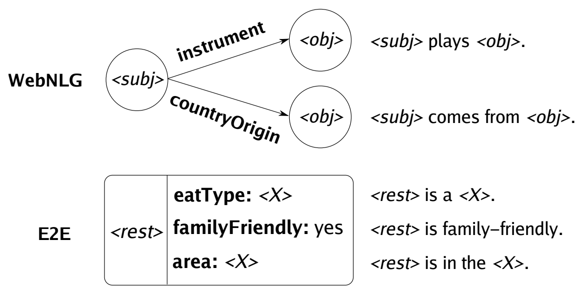

Figure 2: Examples of templates for the predicates in the WebNLG and E2E datasets.

Paragraph Compression We introduce *paragraph compression* as a new task in our D2T generation pipeline. As the last step in the pipeline, it is closely related to linguistic realisation, however—since we already work with natural language in this step—the focus of our task is on sentence fusion, rephrasing, and coreference resolution. Unlike text summarization or simplification (Zhang et al., 2020; Jiang et al., 2020), we aim to convey the complete semantics of the text without omitting any facts. In contrast to sentence fusion (Geva et al., 2019; Barzilay and McKeown, 2005) or sentence compression (Filippova and Altun, 2013), we operate in the context of multiple sentences in a paragraph. The task is the central focus of our WikiFluent corpus (§4), which we synthesize using a model for the reverse task, split-and-rephrase, i.e. splitting a complex sentence into simpler ones while preserving semantics (Botha et al., 2018; Narayan et al., 2017).

3 Method

We first give an overview of our neural D2T generation pipeline (§3.1). Next, we describe the individual steps, starting by applying simple templates to transform data to text (§3.2), followed by individual modules for ordering (§3.3), aggregation (§3.4), and paragraph compression (§3.5).

3.1 Method Overview

We focus on the task of producing a natural language description Y for a set of n RDF triples $X = \{x_1, \dots, x_n\}$. Each triple $x_i = \{s_i, p_i, o_i\}$ consists of subject s_i, predicate p_i, and object o_i. We assume that we can transform each triple x_i to a *fact* f_i (where f_i is a sentence in natural language describing x_i) by filling the single-triple template $t_{p_i} \in T$ for the predicate p_i: $t_{p_i}(s_i, o_i) \rightarrow f_i$.

We proceed as follows – given an input X, we:

(1) apply the templates to transform the set of triples X to the set of facts: $F = T(X) = \{f_1, \dots, f_n\}$ (§3.2),

(2) sort the facts F using an ordering module which outputs an ordered sequence of facts $F_o = O(F) = \{f_{o_1}, \dots, f_{o_n}\}$ (§3.3),

(3) obtain sentence delimiters by inputting the ordered facts F_o into an aggregation module $A(F_o) = \{\delta_{o_1}, \delta_{o_2}, \dots, \delta_{o_{n-1}}\}; \delta_i \in \{0, 1\}$, where $\delta_{o_i} = 1$ indicates the presence of a delimiter, i.e., that the sentences with facts f_{o_i} and $f_{o_{i+1}}$ should *not* be fused (§3.4),

(4) input the ordered sequence with delimiters $F_a = \{f_{o_1}, \delta_{o_1}, f_{o_2}, \dots, \delta_{o_{n-1}}, f_{o_n}\}$ into the paragraph compression module which generates the final description $P(F_a) = Y$ (§3.5).

3.2 Templates

The first step in our pipeline involves transforming each of the input triples X into a set of facts F in natural language by using a template t_{p_i} for each predicate p_i. We need at least one template for each predicate. Typically, the template will include placeholders which are filled with s_i and o_i.

The transformation serves two purposes: (a) preparing the data for the subsequent text-to-text operations, (b) introducing in-domain knowledge about the semantics of individual predicates. Note that the filled templates are allowed to contain minor disfluencies since the text will be rephrased in the final step of the pipeline. See §5.5 for our approach to gathering the templates and Figure 2 for examples of the templates we use in our datasets.

We acknowledge that this step may be a bottleneck on datasets with an unconstrained (or very large) set of predicates, which is why we also discuss possibilities for automating this step in §7.

3.3 Ordering

We assume that the default order of triples X (and the respective facts F) is random. To maximize the coherency of the resulting description, we apply an ordering model O to get an ordered sequence of facts: $F_o = \{f_{o_1}, \dots, f_{o_n}\}$. The coherence of the final text will also depend on the paragraph compression step, but grouping related facts together (e.g. facts mentioning *birth date* and *birth place*) helps the paragraph compression model to focus only on fusing and rephrasing the neighboring sentences. We describe our ordering model in §5.1.

3.4 Aggregation

The aggregation model takes a sequence of ordered facts F_o as input and produces a sequence of sentence delimiters $A(F_o) = \{\delta_{o_1}, \delta_{o_2}, \ldots, \delta_{o_{n-1}}\}$; $\delta_i \in \{0, 1\}$. The output $\delta_i = 1$ means that the neighboring facts are should be mentioned separately, serving as a hint for the paragraph compression model *not* to fuse the neighboring sentences. Conversely, $\delta_i = 0$ means that the facts should be aggregated and their corresponding sentences should be fused (see §5.2 and §5.3).

3.5 Paragraph Compression

The paragraph compression model (see §5.3 for simplified variants) takes as input the ordered sequence of facts with delimiters $F_a = \{f_{o_1}, \delta_{o_1}, f_{o_2}, \ldots, \delta_{o_{n-1}}, f_{o_n}\}$ and produces a resulting text Y. The objectives of the model are two-fold: (a) *fusing* related sentences, i.e., sentences i and j in between which $\delta_i = 0$, and (b) *rephrasing* the text to improve its fluency, e.g. fixing minor disfluencies in the templates, replacing noun phrases with refering expressions, etc. The focus is on minor rephrasing since the goal is to preserve the semantics of the original text.

4 WikiFluent Corpus

A key to our approach is building a large-scale synthetic corpus providing training data for the text operations in our pipeline. Our corpus needs to cover a broad range of domains while capturing the sentence style in D2T generation, both regarding the input templates and the target descriptions. In other words, we aim to build a corpus in which:

- the input is a set of simple, template-like sentences,
- the output is a fluent text in natural language preserving the semantics of the input.

As we describe below in detail, we achieve that by applying a split-and-rephrase model and a coreference resolution model on a set of human-written paragraphs in English Wikipedia. We consider the processed text as a source and the original text as the target. The process is illustrated in Figure 3; corpus statistics are included in Appendix A.

4.1 Data Source

For building the WikiFluent corpus, we extracted 934k first paragraphs of articles from a Wikipedia dump[2] using WikiExtractor (Attardi, 2015). The paragraphs contain mostly concise, fact-based descriptions from a wide range of domains. We selected paragraphs with length between 30-430 characters, filtering out lists, disambiguations, repeated and malformed paragraphs. To further ensure that the length of inputs is balanced, we selected 250k examples each from 4 equidistant length ranges (30-130 characters, etc.).

[2] enwiki-20210401-pages-articles-multistream

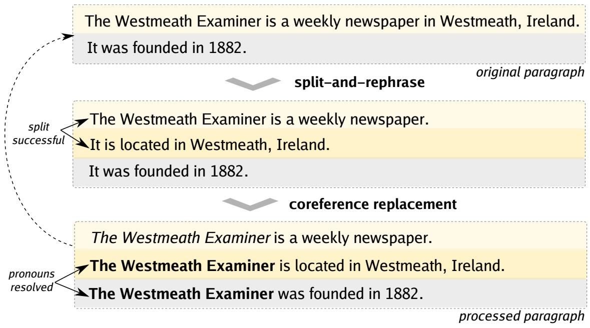

Figure 3: The building process of the WikiFluent corpus. We apply a split-and-rephrase model on each sentence in the paragraph and resolve coreferences in the split sentences.

4.2 Split-and-Rephrase

For generating the target set of sentences, we divide each paragraph into sentences using NLTK (Bird, 2006) and apply a *split-and-rephrase* model on each sentence. Split-and-rephrase is a task of splitting a complex sentence into a meaning preserving sequence of shorter sentences (Narayan et al., 2017). We train our model on the large-scale WikiSplit corpus by Botha et al. (2018), containing human-made sentence splits from Wikipedia edit history. Following the setup in the rest of our experiments, we train the encoder-decoder PLM BART-base (Lewis et al., 2020) on the WikiSplit dataset in a sequence-to-sequence setting. We apply the trained split-and-rephrase model on each sentence, uniformly randomly choosing between 0-2 recursive calls to ensure that the splits are not deterministic. If the sentence cannot be meaningfully split, the model tends to duplicate the sentence on the output; in that case, we use only the original sentence and do not proceed with the splitting.

4.3 Coreference Replacement

Next, we concatenate the split sentences and apply a coreference resolution model (Gardner et al., 2018) in order to replace referring expressions with their antencendents (e.g., pronouns with noun phrases). This allows to better follow the style of the templates in which the entities are always fully verbalized. Since we keep the referring expressions in the original human-written text, we can train the paragraph compression module to generate them in the final text description.

4.4 Filtering

To assert that the generated sentences convey the same semantics as the original paragraph, we use a pretrained RoBERTa model[3] (Liu et al., 2019) trained on the MultiNLI dataset (Williams et al., 2018) for checking the semantic accuracy of the generated text. Following Dušek and Kasner (2020), we test if the original paragraph entails each of the synthesized sentences (checking for omissions), and if the set of concatenated synthesized sentences entails the original paragraph (checking for hallucinations). In a filtered version of the WIKIFLUENT corpus, we include only the examples without omissions or hallucinations (as computed by the model), reducing it to approximately 3/4 of the original size.

5 Experiments

We show how we build our pipeline (§5.1-5.4) and discuss the D2T generation datasets which we use for our experiments (§5.5). The details of our training setup are included in Appendix B.

5.1 Ordering Model

For our ordering model (see §3.3), we use the *Simple Pointer* model from Calizzano et al. (2021). The model is based on a pretrained BART-base extended with a pointer network from Wang and Wan (2019). We provide a short description of the model here; for details see Calizzano et al. (2021).

In the encoding phase, facts F are concatenated and tokenized. Each fact is surrounded by special tokens denoting the beginning (<s>) and the end (</s>) of the fact. The sequence is processed by the BART encoder, generating a sequence of encoder states E for each end token </s> representing the preceding fact.

The decoding proceeds autoregressively. To bootstrap the decoding process, the pair of tokens <s></s> is fed into the decoder, producing the decoder state d_1. The pointer network (attending to d_1 and E), selects the first ordered fact f_{o_1}, which is fed into the decoder in the next step. The process is repeated until the all the facts are decoded in a particular order.

The pointer network computes the probability of a fact to be on the j-th position, using the encoder output E and the decoder output d_j. The network is based on the scaled dot product attention, where d_j is the query and encoder outputs E_i are the keys:

$$Q = d_j W_Q$$

$$K = E W_K$$

$$P_j = \text{softmax}\left(\frac{QK^T}{\sqrt{b}}\right).$$

Here W_Q and $W_K \in \mathbb{R}^{b \times b}$, b is the dimension of BART hidden states, and $P_j \in \mathbb{R}^{n+1}$ is the probability distribution for the j-th position (i.e., P_{ji} is the probability that fact f_i is on the j-th position).

We train the model using the split sentences in the WIKIFLUENT corpus, randomly shuffling the order of the sentences and training the model to restore their original order.

5.2 Aggregation Model

We base our aggregation model (cf. §3.4) on RoBERTa-large (Liu et al., 2019) with a token classification head.[4] Similarly to the ordering model (§5.1), we input the sequence of facts F_o into the model, separating each pair of facts f_{o_i} with a special token </s> (used by the model as a separator). Subsequently, the token classification layer classifies each separator </s>$_i$ position into two classes $\{0, 1\}$ corresponding to the delimiter δ_i. We ignore the outputs for the non-separator tokens while computing the cross-entropy loss.

We create the training examples using the split sentences in the WIKIFLUENT corpus, in which we set $\delta_i = 0$ for the sentences $i, i+1$ which were originally aggregated (i.e., are the result of splitting a single sentence) and $\delta_i = 1$ otherwise.

5.3 Paragraph Compression Model

We adopt BART-base for our paragraph compression model. We train the model in a sequence-to-sequence setting on the WIKIFLUENT corpus, concatenating the split sentences on the input. We add delimiters between sentences i and $i+1$ where $\delta_i = 1$ using a special token <sep>, which we add to the model vocabulary. As shown in Keskar et al. (2019), including control codes for training

[3] https://huggingface.co/roberta-large-mnli

[4] https://huggingface.co/transformers/model_doc/roberta.html#robertafortokenclassification

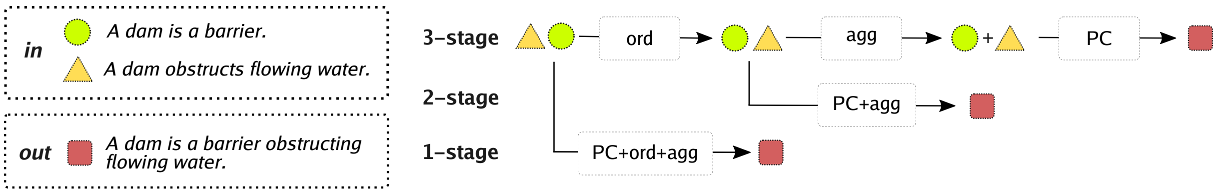

Figure 4: An example illustrating how the individual modules are trained and subsequently applied as the parts of the pipeline. See §5.1 for description of the ordering model (ORD), §5.2 for the aggregation model (AGG), and §5.3 for the versions of the paragraph compression model (PC, PC+AGG, PC+ORD+AGG).

the model can steer the model towards producing certain outputs. We evaluate our model's behavior with respect to ordering and aggregation in §6.3.

5.4 Ablation Study

In order to evaluate individual components of our pipeline, we train three versions of the PC model (see §5.3). The models share the same architecture and targets, but differ in their inputs:

- PC – the model takes as an input ordered facts with delimiters (as described in §3.5),
- PC+AGG – the model takes as an input ordered facts *without* delimiters (i.e., the aggregation is left implicitly to the model),
- PC+ORD+AGG – the model takes as an input facts in *random* order and *without* delimiters (i.e., both ordering and aggregation are left implicitly to the model).

Subsequently, we test three versions of the pipeline (see Figure 4):

- 3-STAGE – a full version of the pipeline consisting of the ordering model, the aggregation model and the PC model (following the full pipeline from §3),
- 2-STAGE – a pipeline consisting of the ordering model and the PC+AGG model,
- 1-STAGE – a single stage consisting of the PC+ORD+AGG model.

We evaluate all versions of the pipeline with PC models trained on the *full* and *filtered* versions of the WIKIFLUENT dataset (see §4).

5.5 D2T Datasets

We test our approach on two English D2T datasets, WebNLG and E2E. They differ in domain, size, textual style, and number of predicates (see Appendix A for details).

WebNLG The WebNLG dataset (Gardent et al., 2017) contains RDF triples from DBPedia (Auer et al., 2007) and their crowdsourced descriptions. The dataset was extended for the WebNLG+ Challenge (Ferreira et al., 2020), but we use the version 1.4 for comparability to prior work. Templates for WebNLG could be extracted from the training data by delexicalizing single-triple examples. However, the examples are noisy and such data would not be available in a zero-shot setup. Therefore, we handcrafted templates for all 354 predicates, including unseen predicates in the test set.[5]

E2E The E2E dataset (Novikova et al., 2017; Dušek et al., 2020) contains restaurant recommendations in the form of attribute-value pairs. We use the cleaned version of the dataset (Dušek et al., 2019). Following previous work, we transformed the attribute-value pairs into RDF triples (using the restaurant name as a subject) and then applied the same setup as for WebNLG. We created a template for each of the 8 attributes manually.

6 Evaluation

We evaluate outputs from the {1,2,3}-STAGE variants of our pipeline automatically (§6.1) and manually (§6.2). Further, we evaluate the performance of the content planning modules and the ability of the PC module to follow the content plan (§6.3).

6.1 Automatic Metrics

Following prior work, we use BLEU (Papineni et al., 2002) and METEOR (Banerjee and Lavie, 2005) to evaluate the outputs against the human references.[6] We also evaluate the number of omission and hallucination errors (i.e., facts missing or added, respectively) using a metric from Dušek and Kasner (2020) based on a RoBERTa model (Liu et al., 2019) pretrained on natural language inference (NLI).[7]

[5] The templates are single-sentence and mostly clear-cut verbalizations of the predicates. We did not use human references from the dataset when creating the templates.

[6] We use the implementation from `https://github.com/tuetschek/e2e-metrics`.

[7] We additionally evaluated the outputs on the E2E dataset using the provided pattern-based slot error script. See Ap-

Input	*(Allen Forrest; background; solo singer), (Allen Forrest; genre; Pop music), (Allen Forrest; birthPlace; Dothan, Alabama)*
Templ.	Allen Forrest is a solo singer. Allen Forrest performs Pop music. Allen Forrest was born in Dothan, Alabama.
Model	Allen Forrest is a solo singer who performs Pop music. He was born in Dothan, Alabama.
Human	Born in Dothan, Alabama, Allen Forrest has a background as a solo singer and was a pop artist.
Input	*name[Wildwood], eatType[restaurant], food[French], area[riverside], near[Raja Indian Cuisine]*
Templ.	Wildwood is a restaurant. Wildwood serves French food. Wildwood is in the riverside. Wildwood is near Raja Indian Cuisine.
Model	Wildwood is a restaurant serving French food. It is in the riverside near Raja Indian Cuisine.
Human	A amazing French restaurant is called the Wildwood. The restaurant is near the Raja Indian Cuisine in riverside. They love kids.

Table 1: Example outputs of our model (3-STAGE, filtered). See Appendix D for more examples.

		B	M	O	H
UPF-FORGe*		38.65	39.00	0.075	0.101
MELBOURNE*		45.13	37.00	0.237	0.202
Ke et al. (2021)†*		66.14	47.25	-	-
Laha et al. (2020)†		24.80	34.90	-	-
COPY		37.18	38.77	0.000	0.000
full	3-STAGE	42.92	39.07	0.051	0.148
	2-STAGE	42.90	39.28	**0.043**	0.125
	1-STAGE	39.08	38.94	0.071	0.204
filtered	3-STAGE	43.19	39.13	0.152	**0.073**
	2-STAGE	**43.49**	**39.32**	0.146	0.096
	1-STAGE	42.99	38.81	0.202	0.093

Table 2: Automatic metrics on WebNLG. B = BLEU, M = METEOR, O = omissions / # facts, H = hallucinations / # examples. The systems marked with asterisk (*) are trained on the WebNLG dataset. Results for the systems marked with † are copied from the respective papers.

		B	M	O	H
TGEN*		40.73	37.76	0.016	0.083
Harkous et al. (2020)*		43.60	39.00	-	-
COPY		24.19	34.89	0.000	0.000
full	3-STAGE	**36.04**	36.95	**0.001**	**0.001**
	2-STAGE	35.84	36.91	**0.001**	**0.001**
	1-STAGE	30.81	36.01	0.009	0.122
filtered	3-STAGE	35.88	36.95	**0.001**	**0.001**
	2-STAGE	36.01	**36.99**	**0.001**	**0.001**
	1-STAGE	34.08	36.32	0.012	0.050

Table 3: Automatic metrics on E2E. B = BLEU, M = METEOR, O = omissions / # facts, H = hallucinations / # examples. The systems marked with asterisk (*) are trained on the E2E dataset. The results for Harkous et al. (2020) are copied from the paper.

We include a diverse set of baselines for comparison. For WebNLG (see Table 2), we include the results of UPF-FORGe and MELBOURNE systems from the first run of WebNLG Challenge (Gardent et al., 2017) which are comparable in terms of automatic metrics and semantic errors, and the results of Ke et al. (2021), which is a state-of-the-art system using structure-aware encoder and task-specific pretraining. Laha et al. (2020) is (to our knowledge) the only other zero-shot D2T generation system applied on WebNLG. TGEN (Dušek and Jurčíček, 2015) is the baseline system for the E2E Challenge (Dušek et al., 2020) and Harkous et al. (2020) is a state-of-the art supervised system applied on the cleaned E2E (see Table 3). For both datasets, COPY is the baseline of copying the templates verbatim.

The automatic evaluation suggests that while our system lags behind state-of-the-art supervised systems, it shows considerable improvements compared to the COPY baseline (e.g., ~12 BLEU points for E2E) and matches performance of some older supervised systems. The COPY baseline is substantially better than the zero-shot system of Laha et al. (2020), suggesting that quality of the templates plays an important role. The 2-STAGE system is generally on par with the 3-STAGE system (or better), which indicates that implicit aggregation using the PC-AGG model may be sufficient. However, an advantage of having a separate aggregation module is the possibility to control the aggregation step explicitly. The filtered version of the dataset generally brings better results, although it brings also an increase in the number of omissions.

6.2 Manual Evaluation

We manually evaluated 100 outputs of the models regarding factual errors (hallucinations, omissions, incorrect fact merging, redundancies) as well as grammatical errors. The results are listed in Table 4. The 1-STAGE model (which has to order the facts implicitly) tends to repeat the facts in the text (especially in E2E) and produces frequent hallucinations. These problems are only slightly reduced in the *filtered* version, but they are largely elim-

pendix C for the details.

		WebNLG					E2E				
		H	I	O	R	G	H	I	O	R	G
full	3-STAGE	3	39	2	2	16	0	1	0	0	17
	2-STAGE	8	36	1	5	16	1	1	0	1	23
	1-STAGE	28	27	6	10	20	17	0	1	79	45
filtered	3-STAGE	2	37	2	1	15	0	0	0	0	17
	2-STAGE	5	32	1	2	14	0	0	0	0	11
	1-STAGE	8	40	6	6	16	11	2	1	41	22

Table 4: Number of manually annotated errors on 100 examples: H = hallucinations, I = incorrect fact merging, O = omissions, R = redundancies, G = grammar errors or disfluencies.

	B-2	Acc
Transformer (Ferreira et al., 2019)†	52.20	0.35
Step-by-step (Moryossef et al., 2019b)†	70.80	0.47
PLANENC (Zhao et al., 2020)†	80.10	0.62
Plan-then-generate (Su et al., 2021b)†	84.97	0.72
RANDOM	47.00	0.29
BART+ptr (Calizzano et al., 2021)	59.10	0.48

Table 5: Evaluation of our zero-shot ordering model based on Calizzano et al. (2021). The results marked with † are copied from the respective papers.

inated with 2-STAGE and 3-STAGE models. We note these models create almost no hallucinations or omissions. However, the outputs on WebNLG for all systems suffer from semantic errors resulting from merging of unrelated facts. This mostly happens with unrelated predicates connected to the same subject/object (e.g. “X was born in Y”, “X worked as Z” expressed as “X worked as Z in Y”; see Appendix D for examples). On the E2E data, which has a simpler triple structure (all predicates share the same subject), the outputs are generally consistent and the 2-STAGE and 3-STAGE models exhibit almost no semantic errors. As we discuss in §7, more research is needed for ensuring the final consistency of the text. The grammar errors and disfluencies stem mainly from over-eager paragraph compression or from artifacts in our templates; they are relatively minor (e.g., missing “is” in “serves French food and family-friendly”).

6.3 Content Planning

Following Su et al. (2021b) and Zhao et al. (2020), we report the accuracy (Acc) and BLEU-2 score (B-2) of our **ordering model** on WebNLG against the human-generated plans from Ferreira et al. (2018). The results are listed in Table 5. RANDOM is the baseline of generating a random order. The results show that although our approach lacks behind state-of-the-art supervised approaches, it can outperform both the random baseline and the Transformer-based approach from Ferreira et al. (2019) while not using any training examples from WebNLG.

We also evaluate the accuracy of our **aggregation model**, using triples ordered according to the plans from Ferreira et al. (2018) as input. The accuracy is 0.33 per example and 0.62 per sentence boundary (random baseline is 0.23 and 0.50, respectively). The results show that although our approach is better than the random baseline, further investigation regarding plausible fact aggregation schemes is needed.

Finally, we manually evaluate how the **PC model** follows the content plan using 100 randomly chosen examples with more than 1 triple on WebNLG and E2E. We find that the model follows the content plan in 95% and 100% of cases, respectively. The incorrect cases include a fact not properly mentioned and an extra boundary between the sentences without a separator. We can thus conclude that the pretraining task successfully teaches the PC model to follow a given content plan.

7 Discussion and Future Work

In the current form, our pipeline can be directly applied to generating text from RDF triples (or similarly structured data) which require no extra processing. Further extensions are needed for more complex D2T scenarios, e.g. datasets requiring content selection or common-sense and logical reasoning (Wiseman et al., 2017; Lin et al., 2019; Chen et al., 2020b).

Our approach regarding handcrafting a single template for each predicate is quite basic. Generating simple statements from the triples automatically, e.g., using the approach of Laha et al. (2020), could reduce the manual workload and allow applying our approach on datasets with a less constrained set of data attributes such as ToTTo (Parikh et al., 2020) or DART (Nan et al., 2021). Moreover, explicitly including a denoising task for the paragraph compression model could help to tackle the disfluencies in the templates.

More research is also needed on semantic errors stemming from merging of facts in improper ways. We suggest that explicitly controlling the semantics of sentence fusion (Ben-David et al., 2020) could help to mitigate this issue, while still keeping the advantages of a zero-shot approach.

References

Oshin Agarwal, Heming Ge, Siamak Shakeri, and Rami Al-Rfou. 2021. Knowledge graph based synthetic corpus generation for knowledge-enhanced language model pre-training. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3554–3565.

Giuseppe Attardi. 2015. Wikiextractor. `https://github.com/attardi/wikiextractor`.

Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. Dbpedia: A nucleus for a web of open data. In *The semantic web*, pages 722–735. Springer.

Satanjeev Banerjee and Alon Lavie. 2005. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 65–72, Ann Arbor, Michigan. Association for Computational Linguistics.

Regina Barzilay, Noemie Elhadad, and Kathleen McKeown. 2001. Sentence ordering in multidocument summarization. In *Proceedings of the first international conference on Human language technology research*.

Regina Barzilay and Kathleen R McKeown. 2005. Sentence fusion for multidocument news summarization. *Computational Linguistics*, 31(3):297–328.

Eyal Ben-David, Orgad Keller, Eric Malmi, Idan Szpektor, and Roi Reichart. 2020. Semantically driven sentence fusion: Modeling and evaluation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*, pages 1491–1505.

Steven Bird. 2006. Nltk: the natural language toolkit. In *Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions*, pages 69–72.

Jan A. Botha, Manaal Faruqui, John Alex, Jason Baldridge, and Dipanjan Das. 2018. Learning to split and rephrase from Wikipedia edit history. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 732–737, Brussels, Belgium. Association for Computational Linguistics.

Rémi Calizzano, Malte Ostendorff, and Georg Rehm. 2021. Ordering sentences and paragraphs with pre-trained encoder-decoder transformers and pointer ensembles. In *Proceedings of the 21st ACM Symposium on Document Engineering*, pages 1–9.

Ernie Chang, Xiaoyu Shen, Hui-Syuan Yeh, and Vera Demberg. 2021a. On training instance selection for few-shot neural text generation. *arXiv preprint arXiv:2107.03176*.

Ernie Chang, Xiaoyu Shen, Dawei Zhu, Vera Demberg, and Hui Su. 2021b. Neural data-to-text generation with lm-based text augmentation. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 758–768.

Wenhu Chen, Yu Su, Xifeng Yan, and William Yang Wang. 2020a. KGPT: Knowledge-grounded pre-training for data-to-text generation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 8635–8648, Online. Association for Computational Linguistics.

Xinchi Chen, Xipeng Qiu, and Xuanjing Huang. 2016. Neural sentence ordering. *arXiv preprint arXiv:1607.06952*.

Zhiyu Chen, Wenhu Chen, Hanwen Zha, Xiyou Zhou, Yunkai Zhang, Sairam Sundaresan, and William Yang Wang. 2020b. Logic2text: High-fidelity natural language generation from logical forms. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*, pages 2096–2111.

Zhiyu Chen, Harini Eavani, Wenhu Chen, Yinyin Liu, and William Yang Wang. 2020c. 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.

Baiyun Cui, Yingming Li, and Zhongfei Zhang. 2020. Bert-enhanced relational sentence ordering network. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6310–6320.

Robert Dale. 2020. Natural language generation: The commercial state of the art in 2020. *Natural Language Engineering*, 26(4):481–487.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186.

Ondřej Dušek, David M Howcroft, and Verena Rieser. 2019. Semantic noise matters for neural natural language generation. In *Proceedings of the 12th International Conference on Natural Language Generation*, pages 421–426.

Ondřej Dušek and Filip Jurčíček. 2015. Training a natural language generator from unaligned data. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 451–461.

Ondřej Dušek and Zdeněk Kasner. 2020. Evaluating semantic accuracy of data-to-text generation with natural language inference. In *Proceedings of the 13th International Conference on Natural Language Generation*, pages 131–137.

Ondřej Dušek, Jekaterina Novikova, and Verena Rieser. 2020. Evaluating the state-of-the-art of end-to-end natural language generation: The e2e nlg challenge. *Computer Speech & Language*, 59:123–156.

et al. Falcon, WA. 2019. Pytorch lightning. *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning*, 3.

Thiago Ferreira, Claire Gardent, Nikolai Ilinykh, Chris van der Lee, Simon Mille, Diego Moussallem, and Anastasia Shimorina. 2020. The 2020 bilingual, bi-directional webnlg+ shared task overview and evaluation results (webnlg+ 2020). In *Proceedings of the 3rd International Workshop on Natural Language Generation from the Semantic Web (WebNLG+)*.

Thiago Castro Ferreira, Diego Moussallem, Emiel Krahmer, and Sander Wubben. 2018. Enriching the webnlg corpus. In *Proceedings of the 11th International Conference on Natural Language Generation*, pages 171–176.

Thiago Castro Ferreira, Chris van der Lee, Emiel van Miltenburg, and Emiel Krahmer. 2019. Neural data-to-text generation: A comparison between pipeline and end-to-end architectures. 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 552–562.

Katja Filippova and Yasemin Altun. 2013. Overcoming the lack of parallel data in sentence compression. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1481–1491.

Claire Gardent, Anastasia Shimorina, Shashi Narayan, and Laura Perez-Beltrachini. 2017. The WebNLG challenge: Generating text from RDF data. In *Proceedings of the 10th International Conference on Natural Language Generation*, pages 124–133, Santiago de Compostela, Spain.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F Liu, Matthew Peters, Michael Schmitz, and Luke Zettlemoyer. 2018. Allennlp: A deep semantic natural language processing platform. In *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*, pages 1–6.

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.

Mor Geva, Eric Malmi, Idan Szpektor, and Jonathan Berant. 2019. Discofuse: A large-scale dataset for discourse-based sentence fusion. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3443–3455.

Jingjing Gong, Xinchi Chen, Xipeng Qiu, and Xuanjing Huang. 2016. End-to-end neural sentence ordering using pointer network. *arXiv preprint arXiv:1611.04953*.

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.

Peyman Heidari, Arash Einolghozati, Shashank Jain, Soumya Batra, Lee Callender, Ankit Arun, Shawn Mei, Sonal Gupta, Pinar Donmez, Vikas Bhardwaj, et al. 2021. Getting to production with few-shot natural language generation models. In *Proceedings of the 22nd Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 66–76.

Chao Jiang, Mounica Maddela, Wuwei Lan, Yang Zhong, and Wei Xu. 2020. Neural crf model for sentence alignment in text simplification. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7943–7960.

Mihir Kale and Abhinav Rastogi. 2020a. Template guided text generation for task-oriented dialogue. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6505–6520, Online. Association for Computational Linguistics.

Mihir Kale and Abhinav Rastogi. 2020b. Text-to-text pre-training for data-to-text tasks. In *Proceedings of the 13th International Conference on Natural Language Generation*, pages 97–102, Dublin, Ireland. Association for Computational Linguistics.

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.

Pei Ke, Haozhe Ji, Yu Ran, Xin Cui, Liwei Wang, Linfeng Song, Xiaoyan Zhu, and Minlie Huang. 2021. Jointgt: Graph-text joint representation learning for text generation from knowledge graphs. *arXiv preprint arXiv:2106.10502*.

Nitish Shirish Keskar, Bryan McCann, Lav R Varshney, Caiming Xiong, and Richard Socher. 2019. Ctrl: A conditional transformer language model for controllable generation. *arXiv preprint arXiv:1909.05858*.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Anirban Laha, Parag Jain, Abhijit Mishra, and Karthik Sankaranarayanan. 2020. Scalable micro-planned generation of discourse from structured data. *Computational Linguistics*, 45(4):737–763.

Mirella Lapata. 2003. Probabilistic text structuring: Experiments with sentence ordering. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 545–552.

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.

Jiwei Li and Dan Jurafsky. 2017. Neural net models of open-domain discourse coherence. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 198–209.

Bill Yuchen Lin, Ming Shen, Yu Xing, Pei Zhou, and Xiang Ren. 2019. Commongen: A constrained text generation dataset towards generative commonsense reasoning.

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 preprint arXiv:1907.11692*.

Amit Moryossef, Yoav Goldberg, and Ido Dagan. 2019a. Improving quality and efficiency in plan-based neural data-to-text generation. In *Proceedings of the 12th International Conference on Natural Language Generation*, pages 377–382.

Amit Moryossef, Yoav Goldberg, and Ido Dagan. 2019b. Step-by-step: Separating planning from realization in neural data-to-text generation. *arXiv preprint arXiv:1904.03396*.

Linyong Nan, Dragomir Radev, Rui Zhang, Amrit Rau, Abhinand Sivaprasad, Chiachun Hsieh, Xiangru Tang, Aadit Vyas, Neha Verma, Pranav Krishna, et al. 2021. Dart: Open-domain structured data record to text generation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 432–447.

Shashi Narayan, Claire Gardent, Shay B. Cohen, and Anastasia Shimorina. 2017. Split and rephrase. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 606–616, Copenhagen, Denmark. Association for Computational Linguistics.

Jekaterina Novikova, Ondrej Dušek, and Verena Rieser. 2017. The E2E Dataset: New Challenges for End-to-End Generation. In *Proceedings of the 18th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 201–206, Saarbrücken, Germany.

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

Ankur Parikh, Xuezhi Wang, Sebastian Gehrmann, Manaal Faruqui, Bhuwan Dhingra, Diyi Yang, and Dipanjan Das. 2020. Totto: A controlled table-to-text generation dataset. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1173–1186.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32:8026–8037.

Ratish Puduppully, Li Dong, and Mirella Lapata. 2019. Data-to-text generation with content selection and planning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 6908–6915.

Clément Rebuffel, Marco Roberti, Laure Soulier, Geoffrey Scoutheeten, Rossella Cancelliere, and Patrick Gallinari. 2021. Controlling hallucinations at word level in data-to-text generation. *arXiv preprint arXiv:2102.02810*.

Ehud Reiter and Robert Dale. 1997. Building applied natural language generation systems. *Natural Language Engineering*, 3(1):57–87.

Leonardo FR Ribeiro, Martin Schmitt, Hinrich Schütze, and Iryna Gurevych. 2020. Investigating pretrained language models for graph-to-text generation. *arXiv preprint arXiv:2007.08426*.

Zhihong Shao, Minlie Huang, Jiangtao Wen, Wenfei Xu, and Xiaoyan Zhu. 2019. Long and diverse text generation with planning-based hierarchical variational model. 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 3257–3268, Hong Kong, China. Association for Computational Linguistics.

Xiaoyu Shen, Ernie Chang, Hui Su, Cheng Niu, and Dietrich Klakow. 2020. Neural data-to-text generation via jointly learning the segmentation and correspondence. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7155–7165, Online. Association for Computational Linguistics.

Yixuan Su, Zaiqiao Meng, Simon Baker, and Nigel Collier. 2021a. Few-shot table-to-text generation with prototype memory. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 910–917.

Yixuan Su, David Vandyke, Sihui Wang, Yimai Fang, and Nigel Collier. 2021b. Plan-then-generate: Controlled data-to-text generation via planning. *arXiv preprint arXiv:2108.13740*.

Bayu Trisedya, Jianzhong Qi, and Rui Zhang. 2020. Sentence generation for entity description with content-plan attention. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 9057–9064.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. *Advances in Neural Information Processing Systems*, 28:2692–2700.

Tianming Wang and Xiaojun Wan. 2019. Hierarchical attention networks for sentence ordering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7184–7191.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. 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 1112–1122.

Sam Wiseman, Stuart M Shieber, and Alexander M Rush. 2017. Challenges in data-to-document generation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2253–2263.

Sam Wiseman, Stuart M Shieber, and Alexander M Rush. 2018. Learning neural templates for text generation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3174–3187.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

Xinnuo Xu, Ondřej Dušek, Verena Rieser, and Ioannis Konstas. 2021. Agggen: Ordering and aggregating while generating. *arXiv preprint arXiv:2106.05580*.

Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter Liu. 2020. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization. In *International Conference on Machine Learning*, pages 11328–11339. PMLR.

Chao Zhao, Marilyn Walker, and Snigdha Chaturvedi. 2020. Bridging the structural gap between encoding and decoding for data-to-text generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2481–2491.

A Dataset Statistics

Statistics for the datasets described in the paper are listed in Table 7.

B Experimental Setup

B.1 Our Models

We implemented the models for split-and-rephrase, aggregation, and paragraph compression in PyTorch Lightning (Paszke et al., 2019), using the PyTorch (Falcon, 2019) version of the BART and RoBERTa models from the Huggingface library (Wolf et al., 2019).

We use the Adam (Kingma and Ba, 2015) optimizer ($\beta_1 = 0.9, \beta_2 = 0.997, \varepsilon = 1^{-9}$) with learning rate 2^{-5}, linear scheduling and 0.1 warmup proportion; batches of size 8 and accumulating gradients with factor 4. We train the models for 1 epoch on a single GeForce RTX 3090 GPU with 24 GB RAM. We use greedy decoding in all our experiments.

B.2 Ordering

For training the ordering model, we used the implementation from Calizzano et al. (2021) [8] including their training parameters. We plan to fully integrate the ordering model into our framework in the future.

C Additional Results

We provide evaluation of semantic accuracy on the E2E dataset as evaluated with the slot-error script based on matching regular expressions in Table 6.[9]

Note that our manual investigation of a sample of the data shows that the majority of the errors identified in our model outputs are false. For example, the following regular expression used in the slot-error script:

```
prices?(?:  range)?(?:w+)0,3 high
```

matches *"(...) price range and high customer rating (...)"*, incorrectly classifying the presence of the extra slot *priceRange[high]*. This importance of this problem is exacerbated by the consistent

[8] https://github.com/airKlizz/passage-ordering

[9] https://github.com/tuetschek/e2e-cleaning/blob/master/slot_error.py

		miss	add	miss+add
TGEN		0.0060	0.0433	0.0016
COPY		0.0000	0.0000	0.0000
full	3-STAGE	0.0238	0.0698	0.0060
	2-STAGE	0.0054	0.0363	0.0000
	1-STAGE	0.0043	0.0330	0.0000
filtered	3-STAGE	0.0444	0.0487	0.0076
	2-STAGE	0.0043	0.0368	0.0000
	1-STAGE	0.0043	0.0347	0.0000

Table 6: Proportion of output examples with missed only, added only, and both missed and added facts, according to the regular-expression-based E2E slot error script.

outputs of our models, which tend to repeat certain patterns. However, we also manually identified several cases in which an error was found correctly, e.g. the model hallucinating *"3 out of 4 customer rating"* instead of *"3 out of 5 customer rating"*.

D Example Outputs

Tables 8, 9, 10, and 11 show examples of behavior of our models on the **WebNLG dataset**. Tables 12 and 13 show examples of behavior of our models on the **E2E dataset**.

The green color marks the model outputs which are completely correct, the red color marks the errors. For better readability of the input format, we add numeric order identifiers for the individual facts (bold, in squared brackets). These are subsequently used as references in the Order and Aggregation rows. Note that zero-th input in E2E is used as a subject in the RDF triples.

	# train	# dev	# test	tok/src	tok/tgt	sent/src	sent/tgt
WebNLG	18,102	870	1,862	26.8	22.6	3.0	1.4
Clean E2E	33,236	4,299	1,847	29.2	22.3	4.2	1.5
WIKIFLUENT-*full*	915,855	9,346	9,346	52.9	41.1	3.9	2.0
WIKIFLUENT-*filtered*	700,517	7,149	7,149	45.6	35.4	3.4	1.8

Table 7: Number of examples (train / dev / test), average number of tokens per source and target, average number of sentences per source and target (after filling the templates for the D2T datasets).

Input	***[1]*** *(Andrews County Airport; elevationAboveTheSeaLevel (in metres); 973.0)* ***[2]*** *(Andrews County Airport; runwayLength; 896.0)* ***[3]*** *(Andrews County Airport; location; Texas)*
Templates	Andrews County Airport is 973.0 metres above the sea level. Andrews County Airport runway is 896.0 m long. Andrews County Airport is located in Texas.
Order	3 2 1
Aggregation	3 <sep> 2 1
Models (all)	Andrews County Airport is located in Andrews County, Texas. Its runway is 896.0 m long and 973.0 metres above the sea level.
Human	The runway length of Andrews County Airport (located in Texas and 973.0 above sea level) is 896.

Table 8: Correct behavior of the models on WebNLG. All the models ({1,2,3}-STAGE *full/filtered*) produce the same output.

Input	***[1]*** *(Akron, Ohio; country; United States)* ***[2]*** *(United States; language; English language)*
Templates	Akron, Ohio is from United States. English language is spoken in United States.
Order	1 2
Aggregation	1 <sep> 2
3-stage full	Akron, Ohio is from United States. English language is spoken in Akron, Ohio, United States, which is located in the U.S. state of Ohio.
2-stage full	Akron, Ohio is from United States. English language is spoken in Akron, Ohio.
1-stage full	English language is spoken in the United States. Akron, Ohio is from the state of Ohio in the U.S. state of Illinois.
3-stage filtered	Akron, Ohio is from United States. English language is spoken in the United States of America.
2-stage filtered	Akron, Ohio is from United States. English language is spoken in Akron, Ohio.
1-stage filtered	English language is spoken in the United States. Akron, Ohio is from the United Kingdom.
Human	The country of Akron, Ohio, is United States, where English is the language spoken.

Table 9: Incorrect behavior on WebNLG: besides the minor disfluencies caused by the templates (*"Akron, Ohio is from..."*), the models (except for 3-STAGE *filtered*) tend to hallucinate and merge the facts incorrectly.

Input	*[1] (Juan Perón; party; Labour Party (Argentina))* *[2] (Alberto Teisaire; inOfficeWhilePresident; Juan Perón)* *[3] (Alberto Teisaire; nationality; Argentina)* *[4] (Argentina; language; Spanish language)*
Templates	Juan Perón belongs to the Labour Party Argentina. Alberto Teisaire was in office while Juan Perón was a president. Alberto Teisaire is from Argentina. Spanish language is spoken in Argentina.
Order	3 2 1 4
Aggregation	3 2 <sep> 1 <sep> 4
3-stage full	Alberto Teisaire is from Argentina, who was in office while Juan Perón was a president. He belongs to the Labour Party Argentina. Spanish language is spoken in Argentina.
2-stage full	Alberto Teisaire is from Argentina. He was in office while Juan Perón was a president. He belongs to the Labour Party Argentina. Spanish language is spoken in Argentina.
1-stage full	Alberto Teisaire is from Argentina. He was in office while Juan Perón was a president. Spanish language is spoken in Argentina. Perón belongs to the Labour Party Argentina.
3-stage filtered	Alberto Teisaire is from Argentina, in office while Juan Perón was a president. He belongs to the Labour Party Argentina. Spanish language is spoken in Argentina.
2-stage filtered	Alberto Teisaire is from Argentina. He was in office while Juan Perón was a president. He belongs to the Labour Party Argentina. Spanish language is spoken in Argentina.
1-stage filtered	Alberto Teisaire is from Argentina. He was in office while Juan Perón was a president. Spanish language is spoken in Argentina. Juan belongs to the Labour Party Argentina.
Human	Alberto Teisaire, is a national of Argentina, where Spanish is spoken. Alberto Teisaire, was in office while Juan Perón was president. Juan Perón was a member of the Labour Party in Argentina.

Table 10: Incorrect behavior on WebNLG: by using a pronoun "He", the models tend to imply that *Alberto Teisaire* (instead of *Juan Perón)* belongs to the Labour Party.

Input	*[1] (Alfa Romeo 164; relatedMeanOfTransportation; Fiat Croma)* *[2] (Alfa Romeo 164; assembly; Italy)* *[3] (Italy; capital; Rome)*
Templates	Alfa Romeo 164 is related to Fiat Croma. Alfa Romeo 164 was assembled in Italy. Italy's capital is Rome.
Order	2 3 1
Aggregation	2 3 <sep> 1
3-stage full	Alfa Romeo 164 was assembled in Italy's capital, Rome. It is related to Fiat Croma.
2-stage full	Alfa Romeo 164 was assembled in Italy's capital, Rome. It is related to Fiat Croma.
1-stage full	Alfa Romeo 164 was assembled in Italy and related to Fiat Croma. Italy's capital is Rome.
3-stage filtered	Alfa Romeo 164 was assembled in Italy's capital, Rome. It is related to Fiat Croma.
2-stage filtered	Alfa Romeo 164 was assembled in Italy's capital, Rome. It is related to Fiat Croma.
1-stage filtered	Alfa Romeo 164 was assembled in Italy. It is related to Fiat Croma. It was assembled at the Italian capital is Rome.
Human	The Fiat Croma and the Alfa Romeo 164 are both cars. The latter was built in Italy where the capital is Rome.

Table 11: Incorrect behavior on WebNLG: the models tend to merge the country of vehicle's assembly and its capital.

Input	*[0] name[The Cricketers]* *[1] eatType[restaurant]* *[2] near[All Bar One]* *[3] priceRange[cheap]* *[4] food[Chinese]* *[5] customerRating[average]* *[6] familyFriendly[yes]*
Templates	The Cricketers is a restaurant. The Cricketers is near All Bar One. The Cricketers has cheap price range. The Cricketers serves Chinese food. The Cricketers has average customer rating. The Cricketers is family-friendly.
Order	1 4 2 6 3 5
Aggregation	1 4 2 <sep> 6 3 5
3-stage full	The Cricketers is a restaurant serving Chinese food near All Bar One. It is family-friendly, has cheap price range and average customer rating.
2-stage full	The Cricketers is a restaurant serving Chinese food near All Bar One. It is family-friendly with cheap price range and average customer rating.
1-stage full	The Cricketers is a restaurant serving Chinese food. It is family-friendly and has cheap price range. It serves Chinese food near All Bar One. It has average customer rating.
3-stage filtered	The Cricketers is a restaurant serving Chinese food near All Bar One. It is family-friendly with cheap price range and average customer rating.
2-stage filtered	The Cricketers is a restaurant serving Chinese food near All Bar One. It is family-friendly with cheap price range and average customer rating.
1-stage filtered	The Cricketers is a restaurant serving Chinese food. It is family-friendly and has average customer rating. It serves Chinese food near All Bar One. It has cheap price range.
Human	For a central family friendly venue serving cheap Chinese food, try The Cricketers restaurant. It has average customer ratings and is near All Bar One.

Table 12: Correct behavior of the models on E2E: the models can deal even with a higher amount of input values.

Input	*[0] name[The Phoenix]* *[1] area[riverside]* *[2] priceRange[cheap]* *[3] familyFriendly[yes]* *[4] near[Raja Indian Cuisine]* *[5] eatType[restaurant]*
Templates	The Phoenix is in the riverside. The Phoenix has cheap price range. The Phoenix is family-friendly. The Phoenix is near Raja Indian Cuisine. The Phoenix is a restaurant.
Order	5 1 4 <sep> 3 2
Aggregation	5 1 4 3 2
3-stage full	The Phoenix is a restaurant in the riverside near Raja Indian Cuisine. It is family-friendly with cheap price range.
2-stage full	The Phoenix is a restaurant in the riverside near Raja Indian Cuisine. It is family-friendly with cheap price range.
1-stage full	The Phoenix is a restaurant near Raja Indian Cuisine in the riverside. It is family-friendly and has cheap price range. It has cheap food range.
3-stage filtered	The Phoenix is a restaurant in the riverside near Raja Indian Cuisine. It is family-friendly with cheap price range.
2-stage filtered	The Phoenix is a restaurant in the riverside near Raja Indian Cuisine. It is family-friendly with cheap price range.
1-stage filtered	The Phoenix is a restaurant near Raja Indian Cuisine in the riverside. It is family-friendly and has cheap price range. It has cheap food.
Human	Cheap food and a family friendly atmosphere at The Phoenix restaurant. Situated riverside near the Raja Indian Cuisine.

Table 13: Incorrect behavior on E2E: the 1-STAGE models add redundant information to the output.