Step-by-Step: Separating Planning from Realization in Neural Data-to-Text Generation*

Amit Moryossef[†] **Yoav Goldberg**[†‡] **Ido Dagan**[†]
amitmoryossef@gmail.com, {yogo,dagan}@cs.biu.ac.il

[†]Bar Ilan University, Ramat Gan, Israel
[‡]Allen Institute for Artificial Intelligence

Abstract

Data-to-text generation can be conceptually divided into two parts: ordering and structuring the information (planning), and generating fluent language describing the information (realization). Modern neural generation systems conflate these two steps into a single end-to-end differentiable system. We propose to split the generation process into a symbolic text-planning stage that is faithful to the input, followed by a neural generation stage that focuses only on realization. For training a plan-to-text generator, we present a method for matching reference texts to their corresponding text plans. For inference time, we describe a method for selecting high-quality text plans for new inputs. We implement and evaluate our approach on the WebNLG benchmark. Our results demonstrate that decoupling text planning from neural realization indeed improves the system's reliability and adequacy while maintaining fluent output. We observe improvements both in BLEU scores and in manual evaluations. Another benefit of our approach is the ability to output diverse realizations of the same input, paving the way to explicit control over the generated text structure.

1 Introduction

Consider the task of data-to-text generation, as exemplified in the WebNLG corpus (Colin et al., 2016). The system is given a set of RDF triplets describing facts (entities and relations between them) and has to produce a fluent text that is faithful to the facts. An example of such triplets is:

```
John, birthPlace, London
John, employer, IBM
```

With a possible output:

1. John, who was born in London, works for IBM.

Other outputs are also possible:

2. John, who works for IBM, was born in London.

3. London is the birthplace of John, who works for IBM.

4. IBM employs John, who was born in London.

These variations result from different ways of structuring the information: choosing which fact to mention first, and in which direction to express each fact. Another choice is to split the text into two different sentences, e.g.,

5. John works for IBM. John was born in London.

Overall, the choice of fact ordering, entity ordering, and sentence splits for these facts give rise to 12 different structures, each of them putting the focus on somewhat different aspect of the information. Realistic inputs include more than two facts, greatly increasing the number of possibilities.

Another axis of variation is in how to verbalize the information for a given structure. For example, (2) can also be verbalized as

2a. John works for IBM and was born in London.

and (5) as:

5a. John is employed by IBM. He was born in London.

We refer to the first set of choices (how to structure the information) as *text planning* and to the second (how to verbalize a plan) as *plan realization*.[1]

The distinction between planning and realization is at the core of classic natural language generation (NLG) works (Reiter and Dale, 2000; Gatt and Krahmer, 2017). However, a recent wave of *neural NLG systems* ignores this distinction

*This research was supported in part by the German Research Foundation through the German-Israeli Project Cooperation (DIP, grant DA 1600/1-1) and by a grant from Theo Hoffenberg and Reverso.

[1]Note that the variation from 5 to 5a includes the introduction of a pronoun. This is traditionally referred to as *referring expression generation* (REG), and falls between the planning and realization stages. We do not treat REG in this work, but our approach allows natural integration REG systems' outputs.

Proceedings of NAACL-HLT 2019, pages 2267–2277
Minneapolis, Minnesota, June 2 - June 7, 2019.

and treat the problem as a single end-to-end task of learning to map facts from the input to the output text (Gardent et al., 2017; Dušek et al., 2018). These neural systems encode the input facts into an intermediary vector-based representation, which is then decoded into text. While not stated in these terms, the neural system designers hope for the network to take care of both the planning and realization aspect of text generation. A notable exception is the work of Puduppully et al. (2018), who introduce a neural content-planning module in the end-to-end architecture.

While the neural methods achieve impressive levels of output fluency, they also struggle to maintain coherency on longer texts (Wiseman et al., 2017), struggle to produce a coherent order of facts, and are often not faithful to the input facts, either omitting, repeating, hallucinating or changing facts (the NLG community refers to such errors as errors in *adequacy* or *correctness* of the generated text). When compared to template-based methods, the neural systems win in fluency but fall short regarding content selection and faithfulness to the input (Puzikov and Gurevych, 2018). Also, they do not allow control over the output's structure. We speculate that this is due to demanding too much of the network: while the neural system excels at capturing the language details required for fluent realization, they are less well equipped to deal with the higher levels text structuring in a consistent and verifiable manner.

Proposal we propose an explicit, symbolic, text planning stage, whose output is fed into a neural generation system. The text planner determines the information structure and expresses it unambiguously—in our case as a sequence of ordered trees. This stage is performed symbolically and is guaranteed to remain faithful and complete with regards to the input facts. Once the plan is determined,[2] a neural generation system is used to transform it into fluent, natural language text. By being able to follow the plan structure closely, the network is alleviated from the need to determine higher-level structural decisions and can track what was already covered more easily. This allows the network to perform the task it excels in, producing fluent, natural language outputs.

[2]The exact plan can be determined based on a data-driven scoring function that ranks possible suggestions, as in this work, or by other user provided heuristics or a trained ML model. The plans' symbolic nature and precise relation to the input structures allow verification of their correctness.

We demonstrate our approach on the WebNLG corpus and show it results in outputs which are as fluent as neural systems, but more faithful to the input facts. The method also allows explicit control of the output structure and the generation of diverse outputs (some diversity examples are available in the Appendix). We release our code and the corpus extended with matching plans in https://github.com/AmitMY/chimera.

2 Overview of the Approach

Task Description Our method is concerned with the task of generating texts from inputs in the form of RDF sets. Each input can be considered as a graph, where the entities are nodes, and the RDF relations are directed labeled edges. Each input is paired with one or more reference texts describing these triplets. The reference can be either a single sentence or a sequence of sentences. Formally, each input G consists of a set of triplets of the form (s_i, r_i, o_i), where $s_i, o_i \in V$ ("subject" and "object") correspond to entities from DBPedia, and $r_i \in R$ is a labeled DBPedia relation (V and R are the sets of entities and relations, respectively). For example, Figure 1a shows a triplet set G and Figure 1d shows a reference text. We consider the data set as a set of input-output pairs (G, ref), where the same G may appear in several pairs, each time with a different reference.

Method Overview We split the generation process into two parts: text planning and sentence realization. Given an input G, we first generate a text plan $plan(G)$ specifying the division of facts to sentences, the order in which the facts are expressed in each sentence, and the ordering of the sentences. This data-to-plan step is non-neural (Section 3). Then, we generate each sentence according to the plan. This plan-to-sentence step is achieved through an NMT system (Section 4).

Figure 1 demonstrates the entire process.

To facilitate our plan-based architecture, we devise a method to annotate (G, ref) pairs with the corresponding plans (Section 3.1), and use it to construct a dataset which is used to train the plan-to-text translation. The same dataset is also used to devise a plan selection method (Section 3.2).

General Applicability It is worth considering the dataset-specific vs. general applicability aspects of our method. On the low-level details, this

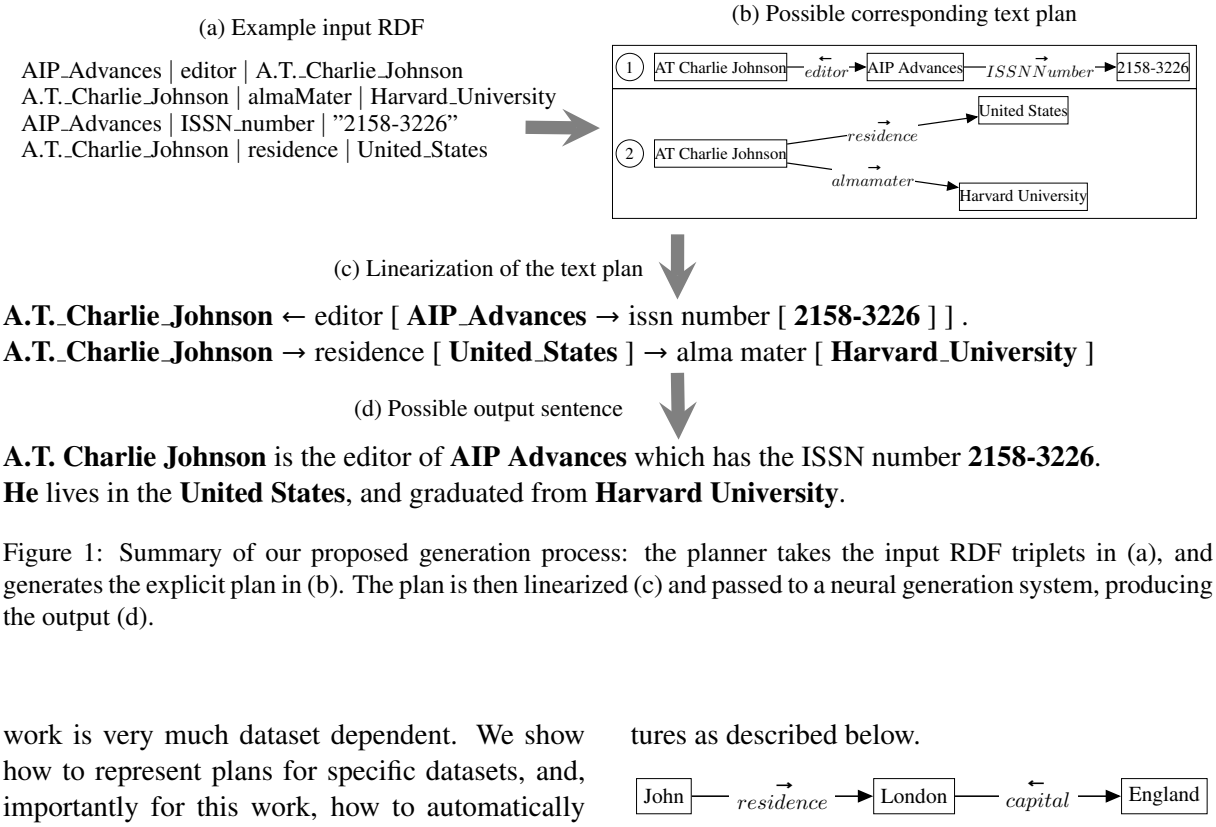

Figure 1: Summary of our proposed generation process: the planner takes the input RDF triplets in (a), and generates the explicit plan in (b). The plan is then linearized (c) and passed to a neural generation system, producing the output (d).

work is very much dataset dependent. We show how to represent plans for specific datasets, and, importantly for this work, how to automatically construct plans for this dataset given inputs and expected natural language outputs. The method of plan construction will likely not generalize "as is" to other datasets, and the plan structure itself may also be found to be lacking for more demanding generation tasks. However, on a higher level, our proposal is very general: intermediary plan structures can be helpful, and one should consider ways of obtaining them, and of using them. In the short term, this will likely take the form of ad-hoc explorations of plan structures for specific tasks, as we do here, to establish their utility. In the longer term, research may evolve to looking into how general-purpose plan are structured. Our main message is that the separation of planning from realization, even in the context of neural generation, is a useful one to be considered.

3 Text Planning

Plan structure Our text plans capture the division of facts to sentences and the ordering of the sentences. Additionally, for each sentence, the plan captures (1) the ordering of facts within the sentence; (2) The ordering of entities within a fact, which we call the *direction* of the relation. For example, the {`A, location, B`} relation can be expressed as either *A is located in B* or *B is the location of A*; (3) the structure between facts that share an entity, namely chains and sibling structures as described below.

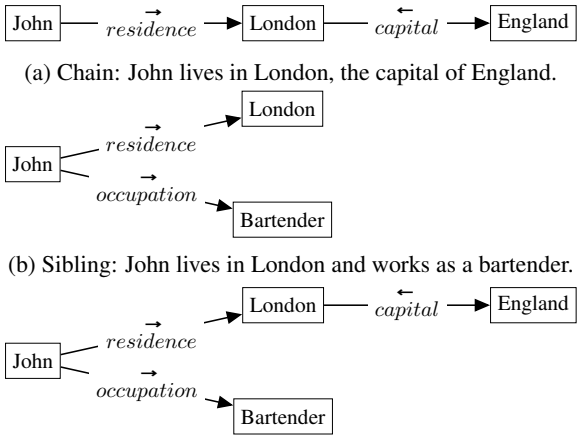

Figure 2: Fact construction structure.

A text plan is modeled as a sequence of sentence plans, to be realized in order. Each sentence plan is modeled as an ordered tree, specifying the structure in which the information should be realized. Structuring each sentence as a tree enables a clear succession between different facts through shared entities. Our text-plan design assumes that each entity is mentioned only once in a sentence, which holds in the WebNLG corpus. The ordering of the entities and relations within a sentence is determined by a pre-order traversal of the tree.

Figure 1b shows an example of a text plan. Formally, given the input G, a text plan T is a sequences of sentence plans $T = s_1, ..., s_{N_T}$. A sen-

tence plan s is a labeled, ordered tree, with arcs of the form (h, ℓ, m), where $h, m \in V$ are head and modifier nodes, each corresponding to an input entity, and $\ell = (r, d)$ is the relation between nodes, where $r \in R$ is the RDF relation, and $d \in \{\rightarrow, \leftarrow\}$ denotes the direction in which the relation is expressed: $d = \rightarrow$ if $(h, r, m) \in G$, and $d = \leftarrow$ if $(m, r, h) \in G$. A text plan T is said to match an input G iff every triplet (s, r, o) in G is expressed in T exactly once, either as an edge $(s, (r, \rightarrow), o)$ or as an edge $(o, (r_i, \leftarrow), s)$.

Chains $(h, \ell_1, m), (m, \ell_2, x)$ represent a succession of facts that share a middle entity (Figure 2a), while siblings — nodes with the same parent — $(h, \ell_1, m_1), (h, \ell_2, m_2)$ represents a succession of facts about the same entity (Figure 2b). Sibling and chain structures can be combined (Figure 2c). An example of an input we addressed in the WebNLG corpus, and matching text plan is given in Figure 1b.

Exhaustive generation For small-ish input graphs G—such as those in the WebNLG task we consider here—it is trivial to generate all possible plans by first considering all the ways of grouping the input into sets, then from each set generating all possible trees by arranging it as an undirected graph and performing several DFS traversals starting from each node, where each DFS traversal follows a different order of children.[3]

3.1 Adding Plans to Training Data

While the input RDFs and references are present in the training dataset, the plans are not. We devise a method to recover the latent plans for most of the input-reference pairs in the training set, constructing a new dataset of (G, ref, T) triplets of inputs, reference texts, and corresponding plans.

We define the reference ref, and the text-plan T to be consistent with each other iff (a) they exhibit the same splitting into sentences—the facts in every sentence in ref are grouped as a sentence plan in T, and (b) for each corresponding sentence and sentence-plan, the order of the entities is identical.

The matching of plans to references is based on the observations that (a) it is relatively easy to identify entities in the reference texts, and a pair of entities in an input is unique to a fact; (b) it is relatively easy to identify sentence splits; (c) a reference text and its matching plan must share the same entities in the same order, and with the same sentence splits.

Sentence split consistency We define a set of triplets to be *potentially consistent* with a sentence iff each triplet contains at least one entity from the sentence (either its subject or object appear in the sentence), and each entity in the sentence is covered by at least one triplet. Given a reference text, we split it into sentences using NLTK (Bird and Loper, 2004), and look for divisions of G into disjoint sets such that each set is consistent with a corresponding sentence. For each such division, we consider the exhaustive set of all induced plans.

Facts order consistency A natural criterion would be to consider a reference sentence and a sentence-plan originating from the corresponding RDF as matching iff the sets of entities in the sentence and the plan are identical, and all entities appear in the same order.[4] Based on this, we could represent each sentence and each plan as a sequence of entities, and verify the sequences match.

However, using this criterion is complicated by the fact that it is not trivial to map between the entities in the plan (that originate from the RDF triplets) and the entities in the text. In particular, due to language variability, the same plan entity may appear in several forms in the textual sentences. Some of these variations (i.e. "A.F.C Fylde" vs. "AFC Fylde") can be recognized heuristically, while others require external knowledge ("UK conservative party" vs. "the Tories"), and some are ambiguous and require full-fledged co-reference resolution ("them", "he", "the former"). Hence, we relax our matching criterion to allow for possible unrecognized entities in the text.

Concretely, we represent each sentence plan as a sequence of its entities $(pe_1, ..., pe_k)$, and each sentence as the sequence of its entities which we managed to recognize and to match with an input entity $(se_1, ..., se_m), m \leq k$.[5]

We then consider a sentence and a sentence-plan to be *consistent* if the following two condi-

[3] If a graph includes a cycle (0.4% of the graphs in the WebNLG corpus contain cycles) we skip it, as it is guaranteed that a different split will result in cycle-free graphs.

[4] An additional constraint is that no two triplets in the RDFs set share the same entities. This is to ensure that if two entities appeared in a structure, only one relation could have been expressed there. This almost always holds in the WebNLG corpus, failing on only 15 out of 6,940 input sets.

[5] We match plan entities to sentence entities using greedy string matching with Levenshtein distance (Levenshtein, 1966) for each token and a manually tuned threshold for a match. While this approach results in occasional false positives, most cases are detected correctly. We match dates by using the chrono-python package that parses dates from natural language texts.

tions hold: (1) The sentence entities $(se_1, ..., se_m)$ are a proper sub-sequence of the plan entities $(pe_1, ..., pe_k)$; and (2) each of the remaining entities in the plan already appeared previously in the plan. The second condition accounts for the fact that most un-identified entities are due to pronouns and similar non-lexicalized referring expressions, and that these only appear after a previous occurrence of the same entity in the text.[6]

3.2 Test-time Plan Selection

To select the plan to be realized, we propose a mechanism for ranking the possible plans. Our plan scoring method is a product-of-experts model, where each expert is a conditional probability estimate for some property of the plan. The conditional probabilities are MLE estimates based on the plans in the training set constructed in section 3.1. Estimates involving relation names are smoothed using Lidstone smoothing to account for unseen relations. We use the following experts:

Relation direction For every relation $r \in R$, we compute its probability to be expressed in the plan in its original order ($d = \rightarrow$) or in the reverse order ($d = \leftarrow$): $p_{dir}(d = \rightarrow \mid R)$. This captures the tendency of certain relations to be realized in the reversed order to how they are defined in the knowledge base. For example, in the WebNLG corpus the relation "manager" is expressed as a variation of "is managed by" instead of one of "is the manager of" in 68% of its occurrences ($p_{dir}(d = \leftarrow \mid \text{manager}) = 0.68$).

Global direction We find that while the probability of each relation to be realized in a reversed order is usually below 0.5, still in most plans of longer texts there are one or two relations that appear in the reversed order. We capture this tendency using an expert that considers the conditional probability $p_{gd}(nr = n \mid |G|)$ of observing n reversed edges in an input with $|G|$ triplets.

Splitting tendencies For each input size, we keep track of the possible ways in which the set of facts can be split to subsets of particular sizes. That is, we keep track of probabilities such as $p_s(s = [3, 2, 2] \mid 7)$ of realizing an input of 7 RDF triplets as three sentences, each realizing the corresponding number of facts.

Relation transitions We consider each sentence plan as a sequence of the relation types expressed in it $r_1, \ldots, r_k$ followed by an EOS symbol, and compute the markov transition probabilities over this sequence: $p_{trans}(r_1, r_2, \ldots, r_k, EOS) = \prod_{i=1,k} p_t(r_{i+1}|r_i)$. The expert is the product of the transition probabilities of the individual sentence plans in the text plan. This captures the tendencies of relations to follow each other and in particular, the tendencies of related relations such as birth-place and birth-date to group, allowing their aggregation in the generated text (*John was born in London on Dec 12th, 1980*).

Each of the possible plans are then scored based on the product of the above quantities.[7]

The scores work well for separating good from lousy text plans, and we observe a threshold above which most generated plans result in adequate texts. We demonstrate in Section 6 that realizing highly-ranked plans manages to obtain good automatic realization scores. We note that the plan in Figure 1b is the one our ranking algorithm ranked first for the input in Figure 1a.

Possible Alternatives In addition to the single plan selection, the explicit planning stage opens up additional possibilities. Instead of choosing and realizing a single plan, we can realize **a diverse set** of high-scoring plans, or realizing a random high-scoring plan, resulting in a diverse and less templatic set of texts across runs. This relies on the combination of two factors: the ability of the scoring component to select plans that correspond to plausible human-authored texts, and the ability of the neural realizer to faithfully realize the plan into fluent text. While it is challenging to directly evaluate the plans adequacy, we later show an evaluation of the plan realization component. Figure 3 shows three random plans for the same graph and their realizations. Further examples of the diversity of generation are given in the appendix.

The explicit and symbolic planning stage also allows for **user control** over the generated text, either by supplying constraints on the possible plans (e.g., number of sentences, entities to focus on, the order of entities/relations, or others) or by supplying complete plans. We leave these options for future work.

[6] A sensible alternative would be to use a coreference resolution system at this stage. In our case it turned out to not help, and even performed somewhat worse.

[7] We note that for an input of n triplets, there are $\mathcal{O}(2^{2n} + n * n!)$ possible plans, making this method prohibitive for even moderately sized input graphs. However, it is sufficient for the WebNLG dataset in which $n \leq 7$. For larger graphs, better plan scoring and more efficient search algorithms should be devised. We leave this for future work.

(a) **Dessert** ← course [**Bionico** → country [**Mexico**] → ingredient [**Granola**] → region [**Jalisco**]]
*The **Dessert Bionico** requires **Granola** as one of its ingredients and originates from the **Jalisco** region of **Mexico** .*

(b) **Bionico** → country [**Mexico**] → region [**Jalisco**] .
Dessert ← course [**Bionico** → ingredient [**Granola**]]
__Bionico__ is a food found in the __Mexico__ region __Jalisco__.
The __Dessert Bionico__ requires __Granola__ as an ingredient.

(c) **Bionico** → ingredient [**Granola**] → course [**Dessert**] .
Bionico → region [**Jalisco**] → country [**Mexico**]
__Bionico__ contains __Granola__ and is served as a __Dessert__.
__Bionico__ is a food found in the region of __Jalisco__, __Mexico__

Figure 3: Three random linearized plans for the same input graph, and their text realizations. All taken from the top 10% scoring plans. (a) structures the output as a single sentence, while (b) and (c) as two sentences. The second sentence in (b) puts emphasis on Bionico being a dessert, while in (c) the emphasis is on the ingredients.

4 Plan Realization

For plan realization, we use an off-the-shelf vanilla neural machine translation (NMT) system to translate plans to texts. The explicit division to sentences in the text plan allows us to realize each sentence plan individually which allows the realizer to follow the plan structure within each (rather short) sentence, reducing the amount of information that the model needs to remember. As a result, we expect a significant reduction in over- and under-generation of facts, which are common when generating longer texts. Currently, this comes at the expense of not modeling discourse structure (i.e., referring expressions). This deficiency may be handled by integrating the discourse into the text plan, or as a post-processing step.[8]. We leave this for future work.

To use text plans as inputs to the NMT, we linearize each sentence plan by performing a preorder traversal of the tree, while indicating the tree structure with brackets (Figure 1c). The directed relations (r, d) are expressed as a sequence of two or more tokens, the first indicating the direction and the rest expressing the relation.[9] Entities that are identified in the reference text are replaced with single, entity-unique tokens. This allows the NMT system to copy such entities from the input rather than generating them. Figure 1d is an example of possible text resulting from such linearization.

Training details We use a standard NMT setup with a copy-attention mechanism (Gulcehre et al., 2016)[10] and the pre-trained GloVe.6B word embeddings[11] (Pennington et al., 2014). The pre-trained embeddings are used to initialize the relation tokens in the plans, as well as the tokens in the reference texts.

Generation details We translate each sentence plan individually. Once the text is generated, we replace the entity tokens with the full entity string as it appears in the input graph, and lexicalize all dates as *Month DAY+ordinal, YEAR* (i.e., *July 4th, 1776*) and for numbers with units (i.e., *"5"(minutes)*) we remove the parenthesis and quotation marks (*5 minutes*).

5 Experimental Setup

The WebNLG challenge (Colin et al., 2016) consists of mapping sets of RDF triplets to text including referring expression generation, aggregation, lexicalization, surface realization, and sentence segmentation. It contains sets with up to 7 triplets each along with one or more reference texts for each set. The test set is split into two parts: *seen*, containing inputs created for entities and relations belonging to DBpedia categories that were seen in the training data, and *unseen*, containing inputs extracted for entities and relations belonging to 5 unseen categories. While the unseen category is conceptually appealing, we view the seen category as the more relevant setup: generating fluent, adequate and diverse text for a mix of known relation types is enough of a challenge also without requiring the system to invent verbalizations for unknown relation types. Any realistic generation system could afford to provide at least a few verbalizations for each relation of interest. We thus focus our attention mostly on the seen case (though our system does also perform well on the unseen case).

Following Section 3.1, we manage to match a

[8] Minimally, each entity occurrence can keep track of the number of times it was already mentioned in the plan. Other alternatives include using a full-fledged referring expression generation system such as NeuralREG (Ferreira et al., 2018)

[9] We map DBPedia relations to sequences of tokens by splitting on underscores and CamelCase.

[10] Concretely, we use the OpenNMT toolkit (Klein et al., 2017) with the `copy_attn` flag. Exact parameter values are detailed in the appendix.

[11] `nlp.stanford.edu/data/glove.6B.zip`

consistent plan for 76% of the reference texts and use these plan-text pairs to train the plan realization NMT component. Overall, the WebNLG training set contains 18,102 RDF-text pairs while our plan-enhanced corpus contains 13,828 plan-text pairs.[12]

Compared Systems We compare to the best submissions in the WebNLG challenge (Gardent et al., 2017): Melbourne, an end-to-end system that scored best on all categories in the automatic evaluation, and UPF-FORGe (Mille et al., 2017), a classic grammar-based NLG system that scored best in the human evaluation.

Additionally, we developed an end-to-end neural baseline which outperforms the WebNLG neural systems. It uses a set encoder, an LSTM (Hochreiter and Schmidhuber, 1997) decoder with attention (Bahdanau et al., 2014), a copy-attention mechanism (Gulcehre et al., 2016) and a neural checklist model (Kiddon et al., 2016), as well as applying entity dropout. The entity-dropout and checklist component are the key differentiators from previous systems. We refer to this system as **StrongNeural**.

6 Experiments and Results

6.1 Automatic Metrics

We begin by comparing our plan-based system (**BestPlan**) to the state-of-the-art using the common automatic metrics: BLEU (Papineni et al., 2002), Meteor (Banerjee and Lavie, 2005), ROUGE$_L$ (Lin, 2004) and CIDEr (Vedantam et al., 2015), using the nlg-eval[13] tool (Sharma et al., 2017) on the entire test set and on each part separately (seen and unseen).

In the original challenge, the best performing system in automatic metric was based on end-to-end NMT (**Melbourne**). Both the **StrongNeural** and **BestPlan** systems outperform all the WebNLG participating systems on all automatic metrics (Table 1). **BestPlan** is competitive with **StrongNeural** in all metrics, with small differences either way per metric.[14]

[12] Note that this only affects the training stage. At test time, we do not require gold plans, and evaluate on all sentences.

[13] https://github.com/Maluuba/nlg-eval

[14] At least part of the stronger results for **StrongNeural** can be attributed to its ability to generate referring expressions, which we currently do not support.

	BLEU	METEOR	ROUGE$_L$	CIDEr
UPF-FORGe ♥	38.5	0.390	60.9	2.500
Melbourne ♦	45.0	0.376	63.5	2.814
RandomPlan-1 ♠	43.3	0.384	57.6	2.342
RandomPlan-2 ♠	43.5	0.384	57.4	2.332
RandomPlan-3 ♠	43.5	0.384	57.4	2.303
StrongNeural ♦	46.5	**0.392**	**65.4**	**2.866**
BestPlan ♠	**47.4**	0.391	63.1	2.692

Table 1: Results for all categories. Team color indicates the type of system used (NMT ♦, Rule-Based ♥, Rule-Based + NMT ♠).

6.2 Manual Evaluation

Next, we turn to manually evaluate our system's performance regarding faithfulness to the input on the one hand and fluency on the other. We describe here the main points of the manual evaluation setup, with finer details in the appendix.

Faithfulness As explained in Section 3, the first benefit we expect of our plan-based architecture is to make the neural systems task simpler, helping it to remain faithful to the semantics expressed in the plan which in turn is guaranteed to be faithful to the original RDF input (by faithfulness, we mean expressing all facts in the graph and only facts from the graph: not dropping, repeating or hallucinating facts). We conduct a manual evaluation over the seen portion of the WebNLG human evaluated test set (139 input sets). We compare **BestPlan** and **StrongNeural**.[15] For each output text, we manually mark which relations are expressed in it, which are omitted, and which relations exist with the wrong lexicalization. We also count the number of relations the system over generated, either repeating facts or inventing new facts.[16]

Table 2 shows the results. **BestPlan** reduces all error types compared to **StrongNeural**, by 85%, 56% and 90% respectively. While on-par regarding automatic metrics, **BestPlan** substantially outperforms the new state-of-the-art end-to-end neural system in semantic faithfulness.

For example, Figure 4 compares the output of

[15] We do not evaluate **UPF-FORGe** as it is a verifiable grammar-based system that is fully faithful by design.

[16] This evaluation was conducted by the first author, on a set of shuffled examples from the **BestPlan** and **StrongNeural** systems, without knowing which outputs belongs to which system. We further note that evaluating for faithfulness requires careful attention to detail (making it less suitable for crowd-workers), but has a precise task definition which does not involve subjective judgment, making it possible to annotate without annotator biases influencing the results. We release our judgments for this stage together with the code.

1. William_Anders | dateOfRetirement | "1969-09-01"
2. William_Anders | was selected by NASA | 1963
3. William_Anders | timeInSpace | "8820.0"(minutes)
4. William_Anders | birthDate | "1933-10-17"
5. William_Anders | occupation | Fighter_pilot
6. William_Anders | birthPlace | British_Hong_Kong
7. William_Anders | was a crew member of | Apollo_8

(a) The last RDF in the seen test-set

William Anders was born on **October 17th, 1933** in **British Hong Kong**.
He was selected by nasa in **1963** and became a crew member on the **Apollo 8** flight mission.
He retired on **September 1st, 1969**.

(b) Output from StrongNeural

William Anders was a **fighter pilot** who joined nasa in **1963** and served as a crew member of **Apollo 8**.
William Anders retired on **September 1st, 1969** and spent **8820.0 minutes** in space.
William Anders was born in **British Hong Kong** on october **October 17th, 1933**.

(c) Output from BestPlan

Figure 4: Comparing end-to-end neural generation with our plan based system.

StrongNeural (4b) and **BestPlan** (4c) on the last input in the seen test set (4b). While both systems chose three sentences split and aggregated details about birth in one sentence and details about the occupation in another, **StrongNeural** also expressed the information in chronological order. However, **StrongNeural** failed to generate facts 3 and 5. **BestPlan** made a lexicalization mistake in the third sentence by expressing "October" before the actual date, which is probably caused by faulty entity matching for one of the references, and (by design) did not generate any referring expression, which we leave for future work.

	BestPlan	**StrongNeural**
Expressed	417	360
Omitted	6	41
Wrong-lexicalization	17	39
Over-generation	3	29

Table 2: Semantic faithfulness of each system regarding 440 RDF triplets from 139 input sets in the seen part of the manually evaluated test set.

Fluency Next, we assess whether our systems succeed at maintaining the high-quality fluency of the neural systems. We perform pairwise evaluation via Amazon Mechanical Turk wherein each task the worker is presented with an RDF set (both in a graph form, and textually), and two texts in random order, one from **BestPlan**, the other from a competing system. We compare **BestPlan** against a strong end-to-end neural system (**StrongNeural**), a grammar-based system which is the state-of-the-art in human evaluation (**UPF-FORGe**), and the human-supplied WebNLG references (**Reference**). The workers were presented with three possible answers: **BestPlan** text is better (scored as 1), the other text is better (scored as -1), and both texts are equally fluent (scored as 0). Table 3 shows the average worker score given to each pair divided by the number of texts compared. **BestPlan** performed on-par with **StrongNeural**, and surpassed the previous state-of-the-art **UPF-FORGe**. It, however, scored worse than the reference texts, which is expected given that it does not produce referring expressions. Our approach manages to keep the same fluency level typical to end-to-end neural systems, thanks to the NMT realization component.

	StrongNeural	**Reference**	**UPF-FORGe**
BestPlan	-0.6%	-5.4%	+5.1%

Table 3: MTurk average worker score for *BestPlan* compared to each system. It is a worse than the reference texts, on-par with the neural end-to-end system, and a better than the previous state-of-the-art.

6.3 Plan Realization Consistency

We test the extent to which the realizer generates texts that are consistent with the plans. For several subsets of ranked plans (best plan, top 1%, and top 10%) for the seen and unseen test sets separately, we realize up to 100 randomly selected text-plans per input. We realize each sentence plan and evaluate using two criteria: (1) Do all entities from the plan appear in the realization; (2) Like the consis-

	Best Plan		Top 1% Plans		Top 10% Plans	
	Entities	**Order**	**Entities**	**Order**	**Entities**	**Order**
Seen	98.9%	100%	95.9%	99.9%	93.6%	100%
Unseen	66.7%	100%	45.3%	100%	41.3%	100%

Table 4: Surface realizer performance. *Entities:* Percent of sentence plans that were realized with all the requested entities. *Order*: of the sentences that were realized with all requested entities, percentage of realizations that followed the requested entity order.

tency we defined above, do all entities appear in the same order in the plan and the realization.

Table 4 indicates that for decreasingly probable plans our realizer does worse in the first criterion. However, for both parts of the test set, if the realizer managed to express all of the entities, it expressed them in the requested order, meaning the outputs are consistent with plans. This opens up a potential for user control and diverse outputs, by choosing different plans for realization.

Finally, we verify that the realization of potentially diverse plans is not only consistent with each given plan but also preserves output quality. For each input, we realize a random plan from the top 10%. We repeat this process three times with different random seeds to generate different outputs, and mark these systems as **RandomPlan-1/2/3**. Table 1 shows that these random plans maintain decent quality on the automatic metrics, with a limited performance drop, and the automatic score is stable across random seeds.[17]

7 Related Work

Text planning is a major component in classic NLG. For example, Stent et al. (2004) shows a method of producing coherent sentence plans by exhaustively generating as many as 20 sentence plan trees for each document plan, manually tagging them, and learning to rank them using the RankBoost algorithm (Schapire, 1999). Our planning approach is similar, but we only have a set of "good" reference plans without internal ranks. While the sentence planning decides on the aggregation, one crucial decision left is sentence order. We currently determine order based on a splitting heuristic which relies on the number of facts in every sentence, not on the content. Lapata (2003) devised a probabilistic model for sentence ordering which correlated well with human ordering. Our plan selection procedure is admittedly simple, and can be improved by integrating insights from previous text planning works (Barzilay and Lapata, 2006; Konstas and Lapata, 2012, 2013).

Many generation systems (Gardent et al., 2017; Dušek et al., 2018) are based on a black-box NMT component, with various pre-processing transformation of the inputs (such as delexicalization) and outputs to aid the generation process.

Generation from structured data often requires referring to a knowledge base (Mei et al., 2015; Kiddon et al., 2016; Wen et al., 2015). This led to input-coverage tracking neural components such as the checklist model (Kiddon et al., 2016) and copy-mechanism (Gulcehre et al., 2016). Such methods are effective for ensuring coverage and reducing the number of over-generated facts and are in some ways orthogonal to our approach. While our explicit planning stage reduces the amount of over-generation, our realizer may be further improved by using a checklist model.

More complex tasks, like RotoWire (Wiseman et al., 2017) require modeling also document-level planning. Puduppully et al. (2018) explored a method to explicitly model document planning using the attention mechanism.

The neural text generation community has also recently been interested in "controllable" text generation (Hu et al., 2017), where various aspects of the text (often sentiment) are manipulated (Ficler and Goldberg, 2017) or transferred (Shen et al., 2017; Zhao et al., 2017; Li et al., 2018). In contrast, like in (Wiseman et al., 2018), here we focused on controlling either the content of a generation or the way it is expressed by manipulating the sentence plan used in realizing the generation.

8 Conclusion

We proposed adding an explicit symbolic planning component to a neural data-to-text NLG system, which eases the burden on the neural component concerning text structuring and fact tracking. Consequently, while the plan-based system performs on par with a strong end-to-end neural system regarding automatic evaluation metrics and human fluency evaluation, it substantially outperforms the end-to-end system regarding faithfulness to the input. Additionally, the planning stage allows explicit user-control and generating diverse sentences, to be pursued in future work.

[17] While the scores for the different sets are very similar, the plans are very different from each other. See for examples the plans in Figure 3.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473.

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.

Regina Barzilay and Mirella Lapata. 2006. Aggregation via set partitioning for natural language generation. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 359–366. Association for Computational Linguistics.

Steven Bird and Edward Loper. 2004. Nltk: the natural language toolkit. In *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*, page 31. Association for Computational Linguistics.

Emilie Colin, Claire Gardent, Yassine Mrabet, Shashi Narayan, and Laura Perez-Beltrachini. 2016. The webnlg challenge: Generating text from dbpedia data. In *Proceedings of the 9th International Natural Language Generation conference*, pages 163–167.

Ondřej Dušek, Jekaterina Novikova, and Verena Rieser. 2018. Findings of the e2e nlg challenge. *arXiv preprint arXiv:1810.01170*.

Thiago Castro Ferreira, Diego Moussallem, Ákos Kádár, Sander Wubben, and Emiel Krahmer. 2018. Neuralreg: An end-to-end approach to referring expression generation. *arXiv preprint arXiv:1805.08093*.

Jessica Ficler and Yoav Goldberg. 2017. Controlling linguistic style aspects in neural language generation. *arXiv preprint arXiv:1707.02633*.

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.

Albert Gatt and Emiel Krahmer. 2017. Survey of the state of the art in natural language generation: Core tasks, applications and evaluation. *CoRR*, abs/1703.09902.

Caglar Gulcehre, Sungjin Ahn, Ramesh Nallapati, Bowen Zhou, and Yoshua Bengio. 2016. Pointing the unknown words. *arXiv preprint arXiv:1603.08148*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Zhiting Hu, Zichao Yang, Xiaodan Liang, Ruslan Salakhutdinov, and Eric P Xing. 2017. Toward controlled generation of text. *arXiv preprint arXiv:1703.00955*.

Chloé Kiddon, Luke Zettlemoyer, and Yejin Choi. 2016. Globally coherent text generation with neural checklist models. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 329–339.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M Rush. 2017. Opennmt: Open-source toolkit for neural machine translation. *arXiv preprint arXiv:1701.02810*.

Ioannis Konstas and Mirella Lapata. 2012. Unsupervised concept-to-text generation with hypergraphs. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 752–761. Association for Computational Linguistics.

Ioannis Konstas and Mirella Lapata. 2013. Inducing document plans for concept-to-text generation. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1503–1514.

Mirella Lapata. 2003. Probabilistic text structuring: Experiments with sentence ordering. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 545–552. Association for Computational Linguistics.

Vladimir I Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710.

Juncen Li, Robin Jia, He He, and Percy Liang. 2018. Delete, retrieve, generate: A simple approach to sentiment and style transfer. *arXiv preprint arXiv:1804.06437*.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. *Text Summarization Branches Out*.

Hongyuan Mei, Mohit Bansal, and Matthew R Walter. 2015. What to talk about and how? selective generation using lstms with coarse-to-fine alignment. *arXiv preprint arXiv:1509.00838*.

Simon Mille, Roberto Carlini, Alicia Burga, and Leo Wanner. 2017. Forge at semeval-2017 task 9: Deep sentence generation based on a sequence of graph transducers. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 920–923.

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 on association for computational linguistics*, pages 311–318. Association for Computational Linguistics.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Ratish Puduppully, Li Dong, and Mirella Lapata. 2018. Data-to-text generation with content selection and planning. *arXiv preprint arXiv:1809.00582*.

Yevgeniy Puzikov and Iryna Gurevych. 2018. E2e nlg challenge: Neural models vs. templates. In *Proceedings of the 11th International Conference on Natural Language Generation*, pages 463–471.

Ehud Reiter and Robert Dale. 2000. *Building natural language generation systems*. Cambridge university press.

Robert E Schapire. 1999. A brief introduction to boosting. In *Ijcai*, volume 99, pages 1401–1406.

Shikhar Sharma, Layla El Asri, Hannes Schulz, and Jeremie Zumer. 2017. Relevance of unsupervised metrics in task-oriented dialogue for evaluating natural language generation. *CoRR*, abs/1706.09799.

Tianxiao Shen, Tao Lei, Regina Barzilay, and Tommi Jaakkola. 2017. Style transfer from non-parallel text by cross-alignment. In *Advances in Neural Information Processing Systems*, pages 6830–6841.

Amanda Stent, Rashmi Prasad, and Marilyn Walker. 2004. Trainable sentence planning for complex information presentation in spoken dialog systems. In *Proceedings of the 42nd annual meeting on association for computational linguistics*, page 79. Association for Computational Linguistics.

Ramakrishna Vedantam, C Lawrence Zitnick, and Devi Parikh. 2015. Cider: Consensus-based image description evaluation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4566–4575.

Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Pei-Hao Su, David Vandyke, and Steve Young. 2015. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. *arXiv preprint arXiv:1508.01745*.

Sam Wiseman, Stuart M Shieber, and Alexander M Rush. 2017. Challenges in data-to-document generation. *arXiv preprint arXiv:1707.08052*.

Sam Wiseman, Stuart M Shieber, and Alexander M Rush. 2018. Learning neural templates for text generation. *arXiv preprint arXiv:1808.10122*.

Junbo Jake Zhao, Yoon Kim, Kelly Zhang, Alexander M. Rush, and Yann LeCun. 2017. Adversarially regularized autoencoders for generating discrete structures. *CoRR*, abs/1706.04223.