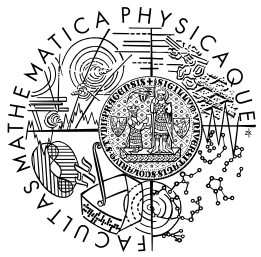

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Bc. Peter Grajcar

Data-to-Text Generation With Text-Editing Models

Institute of Formal and Applied Linguistics

Supervisor of the master thesis:	Mgr. et Mgr. Ondřej Dušek, PhD.
Study programme:	Computer Science
Study branch:	Language Technologies and Computational Linguistics

Prague 2023

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

Acknowledgements

Computational resources were provided by the e-INFRA CZ project (ID:90140), supported by the Ministry of Education, Youth and Sports of the Czech Republic.

Title: Data-to-Text Generation With Text-Editing Models

Author: Bc. Peter Grajcar

Institute: Institute of Formal and Applied Linguistics

Supervisor: Mgr. et Mgr. Ondřej Dušek, PhD., Institute of Formal and Applied Linguistics

Abstract: We explore the use of different model extensions of the FELIX neural transformer-based text-editing model for data-to-text generation. Our approach is based on iterative text-editing – transforming the individual items of the input data into short sentences using trivial templates and then iteratively improving the text by fusing the sentences using a text-editing model. Our extensions include replacing the FELIX's non-autoregressive decoder with an autoregressive transformer decoder, extending the decoding so that it can preserve the input data in the output text, and adding a pointer network-based clause-level reordering mechanism. Furthermore, we propose our own new dataset versions of the WebNLG and DiscoFuse datasets for training the text-editing models. We evaluate our models on the WebNLG dataset with automatic metrics and manually analyse the outputs of selected models.

Keywords: natural language generation | data-to-text generation | text-editing models | natural language processing

Contents

Chapter 1

Introduction

Autoregressive language models employing word-by-word generation have been state-of-the-art in data-to-text generation in recent years. While they are able to produce very fluent outputs, they suffer from a lack of control over the output and are susceptible to hallucinations, i.e. generating unintended text (Ji et al., 2022).

Text-editing models, such as the recent FELIX (Mallinson et al., 2020), may offer a potential alternative (Malmi et al., 2022) by editing an initial text instead of generating a new one. This initial text can be generated with a simple rule-based system. Editing the initial text in place using delete, insert, replace, and reorder operations on tokens could provide more control over the output and interpretability of the model decisions and possibly lower the number of hallucinations.

In this thesis, we explore the use of different extended variants of the text-editing model FELIX (Mallinson et al., 2020) for data-to-text generation. Data-to-text generation is the task of generating an understandable natural language text from structured data (Gatt and Krahmer, 2018). The structured data are represented in various forms, such as semantic triples, tables, or graphs. Our experiments are based on the WebNLG dataset that provides data structured into semantic triples along with reference sentences. The dataset contains examples from a range of various domains. The examples contain facts about sports teams, artists, buildings, and more (Gardent et al., 2017). The structured data are first transformed into a basic preliminary textual form using simple templates (Kale and Rastogi, 2020; Kasner and Dušek, 2020). Then we apply our text-editing model to improve the text and increase its fluency.

The goal of this thesis is to explore possible extensions to the basic text-editing setup that could lead to better results in terms of fluency and faithfulness. The extensions include general-domain pre-training (Kasner and Dusek, 2022) on the DiscoFuse dataset (Geva et al., 2019), higher-level reordering (Calizzano,

Ostendorff, and Rehm, 2021), autoregressive decoding, and post-processing of the predicted editing operations. We compare the text-editing setup against standard autoregressive generation with the BART language model (Lewis et al., 2020) using both automatic metrics and small-scale manual evaluation.

The thesis has the following structure. In Chapter 2, we provide a general theoretical background. In Chapter 3 we discuss previous works relating to the datasets, models, and algorithms we base our thesis on. Chapter 4 describes the data processing and experimental setup, including the extensions to the model. In Chapter 5, we present and discuss the results of the experiments. Chapter 6 sums up our results and proposes possible directions of future research. Appendix A contains information about the contents of the attachments.

Chapter 2

Theoretical Background

This chapter provides the necessary background for our task and models. First, in Section 2.1, we give an introduction to the natural language generation task (NLG) and data-to-text generation in particular. Next, we describe the transformer architecture in Section 2.2, give an overview of transformer-based pre-trained language models Section 2.3, and discuss metrics commonly used in NLG in Section 2.4. We assume that the reader has some basic prior knowledge of neural networks, as we do not cover the basics here. Otherwise, we refer the reader to a comprehensive introductory book on the topic of neural networks by Goodfellow, Bengio, and Courville, 2016.

2.1 Natural Language Generation

The Natural Language Generation (NLG) task is the process of generating natural language text from a given input, such as a set of facts, structured data, or instructions (Gatt and Krahmer, 2018). Data-to-text is a task of generating text conditioned on structured data such as tables, databases, or knowledge graphs. In Table 2.1, we show an example of input data and the corresponding reference text from the WebNLG dataset (Gardent et al., 2017) for the data-to-text generation task. While the natural language generators can be as simple as template-based systems, the majority of the state-of-the-art systems are based on transformer-based models (see Section 2.2) and pre-trained language models (see Section 2.3).

One of the major challenges in deep learning-based NLG is generating unintended texts. Generated content that is nonsensical or unfaithful to the provided source content is referred to as hallucination. There are two main types of hallucinations – intrinsic and extrinsic. Ji et al., 2022 give the following hallucination definition specific to the task of data-to-text generation:

Input Data	Reference Text
(Aarhus Airport, city, Aarhus)	Aarhus airport serves the city of Aarhus.
(1. FC Köln, season, 2014) (1. FC Köln, capacity, 50000)	1. FC Köln has 50000 members and played in the 2014 season.
(Alan Bean, birth date, 1932-03-15) (Alan Bean, was selected by NASA, 1963) (Alan Bean, status, Retired)	Alan Bean (born on 1932-03-15) was selected by NASA in 1963 and now is retired.

Table 2.1 Examples of data-to-text input data and reference texts taken from the WebNLG dataset (Gardent et al., 2017). The input data are triples of the form (*subject*, *predicate*, *object*), also referred to as the Resource Description Framework (RDF) triples.

1. Intrinsic hallucination: the generated text contains information that is contradicted by the input data.
2. Extrinsic hallucinations: the generated text contains extra information irrelevant to the input.

In addition to hallucinations, in the data-to-text generation task, the generated text should contain all the information from the input data without any omissions.

2.2 Transformer

The transformer is a neural network architecture initially designed for sequence-to-sequence modelling (Vaswani et al., 2017), machine translation in particular. However, the use of transformers is not limited to sequence-to-sequence modelling but can be used for encoding-only and decoding-only tasks as well. These transformer-based models achieve state-of-the-art results on many natural language processing tasks, including NLG (Devlin et al., 2019; Lewis et al., 2020; Radford et al., 2019a; Radford et al., 2019b; Brown et al., 2020). In this section, we introduce the sequence-to-sequence models in general and give an overview of the building blocks of the transformer architecture.

2.2.1 Sequence-to-Sequence Models

Sequence-to-sequence (Seq2Seq) models are a class of neural network models that are capable of processing sequences of tokens in one domain and generating sequences of tokens in another domain. These models have been widely used in natural language processing tasks, including NLG.

2.2.2 Encoder-Decoder Architecture

The Encoder-Decoder architecture is a general framework for Seq2Seq models. In the Encoder-Decoder architecture, the encoder processes the input sequence and produces a fixed-length vector representation of the input. The decoder then generates the output sequence based on the encoder's representation and the previous output tokens (Cho et al., 2014). Transformers are not always used in the encoder-decoder setting. Often, models use only the decoder or the encoder part of the transformer. Several transformer-based models like BERT (Devlin et al., 2019) (described in further detail in Section 2.3) or others use an architecture that only uses the encoder part of the transformer.

2.2.3 Attention

Attention is a mechanism used in Seq2Seq models to improve the quality of generated output by allowing the decoder to focus on different parts of the input sequence when generating each output token. In recurrent neural networks, the attention is computed from a weighted sum of the encoder's hidden states, where the weights are learned based on the decoder's state. The attention mechanism allows the model to selectively attend to the most relevant parts of the input sequence, which is especially useful when generating long and complex output sequences (Bahdanau, Cho, and Bengio, 2015). The transformer architecture is based on the attention mechanism without recurrence or convolutions. Transformers are composed of multiple layers of multi-head -attention and feed-forward neural networks. The attention layers allow the model to attend to different parts of the input sequence, while the feed-forward layers allow the model to perform non-linear transformations on the input sequence representations. The single head of the attention layer projects the input sequence X into three spaces – query Q, key K, and value V. The output is then computed as

$$Y = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

where d_k is the key dimension. The multi-head attention computes the above multiple times with a different set of weights for query, key, and value projections. Then the outputs of each head are concatenated into a single output. (Vaswani et al., 2017).

2.2.4 Architecture

The entire original transformer architecture by Vaswani et al., 2017 is illustrated in Figure 2.1. The encoder and decoder contains N blocks, each consisting of a

multi-head attention layer and a feed-forward network with two linear layers and ReLU activation between them. The multi-head attention described in the previous section is used differently for the encoder, decoder, and between the encoder and decoder. The attention is used in the following ways:

1. In the encoder, keys, queries, and values are the output of the previous encoder layer.

2. In the decoder, keys, queries, and values are the output of the previous decoder layer. However, the decoder can at every position attend to all the previous positions, including the current. The other positions are masked out.

3. The attention between the encoder and decoder uses the previous decoder layer's output as the query and the encoder's output as the key and value.

The inputs are embedded into a dimension d_{model}. The model uses positional embeddings to give the model information about the absolute and relative positions of tokens in the sentence. The positional embeddings are summed with the input token embeddings. The output of the last decoder layer is fed into a linear layer, and then softmax is applied to the projected outputs.

2.2.5 Decoding

The decoding process is autoregressive, meaning that the decoder generates the output sequence one token at a time, taking into account the previously generated tokens at each step. The encoder-only architectures allow the model to generate the entire output sequence simultaneously, not taking into account the output positions. Such decoding is also known as non-autoregressive decoding. The autoregressive models may employ different decoding strategies for generating the output sequence from the probability distribution over the vocabulary at each decoding step (Zarrieß, Voigt, and Schüz, 2021). In the following list, we describe the most common decoding strategies:

- **Greedy decoding** is the simplest decoding strategy. Greedy decoding selects the token with the highest probability from the probability distribution at each decoding step. This approach may lead to low-quality, outputs as it does not consider alternative options.

- **Top-k decoding** selects the top-k most probable tokens from the probability distribution at each decoding step. Then, the final sequence needs to be sampled from the k candidates.

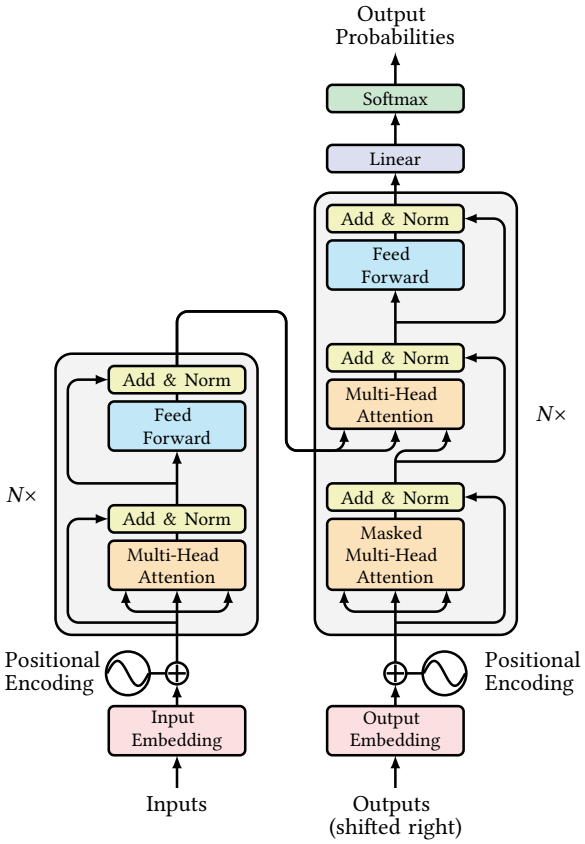

Figure 2.1 The original transformer architecture by Vaswani et al. The gray boxes represent the encoder and decoder blocks.

- **Beam search** is a heuristic search algorithm that keeps track of the k most likely generated sequences at each time step, expanding each sequence to k next candidates (i.e. k^2 in total), then retaining only k best for the next step. The algorithm selects the most likely sequence from the k candidates at the end of the generation process.

2.3 Pre-trained Language Models

Pre-trained language models are a class of deep learning models that are trained on large amounts of text data in an unsupervised manner. They can be used as a starting point for fine-tuning on specific tasks – taking a pre-trained language model and adapting it to a specific task or domain by training it on a smaller labelled dataset. This is made possible by the transfer learning paradigm, where the knowledge gained from one task can be used to improve performance on

another related task. The pre-training models differ in their architecture and the pre-training tasks used. Some models use the full encoder-decoder transformer architecture (Lewis et al., 2020), while others use only the encoder part (Devlin et al., 2019) or only the decoder part (Radford et al., 2019a). In Sections 2.3.1 to 2.3.3, we describe three of these pre-trained language models that are relevant to our work in more detail.

2.3.1 BERT

BERT (Devlin et al., 2019) is a transformer-based model trained using two unsupervised tasks: masked language modelling and next-sentence prediction. The masked language modelling task involves masking out 15% of the input tokens from a sentence and predicting the masked words based on the context. The masked-out token is either replaced by a special `[MASK]` token, replaced by a random token or kept unchanged. The next sentence prediction task involves predicting whether two sentences are consecutive in a text corpus or not. English Wikipedia and BookCorpus (Zhu et al., 2015) were used as the pre-training corpora with 3,300M words altogether. The model achieved state-of-the-art results on multiple classification tasks with a single fine-tuned layer on top of the pre-trained model.

2.3.2 BART

BART (Lewis et al., 2020) is trained using a combination of denoising autoencoder and sequence-to-sequence pre-training objectives. The denoising autoencoder objective involves corrupting a sequence of text and then reconstructing the original sequence from the corrupted version. The training involves multiple strategies for sentence corruption. These are token masking, text infilling and sentence permutation. The sequence-to-sequence objective is to reconstruct the original sequence from the corrupted one. As the training data, a corpus with 160GB of news, books, stories, and web text was used. The model has been shown to be effective when fine-tuned on text-generation tasks.

2.3.3 GPT

GPT (Radford et al., 2019a) is trained using an unsupervised language modelling objective, where the model is trained to predict the next word in a sequence of text based on the previous words. Unlike BERT, GPT is a left-to-right language model that generates text in a sequential manner. The first GPT model was pre-trained on the BookCorpus dataset (Zhu et al., 2015). The following models, GPT-2 (Radford et al., 2019b) and GPT3 (Brown et al., 2020), mainly differ in the

number of trainable parameters. GPT-2 was trained on a custom WebText dataset created from millions of web pages containing 40GB of text. GPT-3 was trained on a combination of multiple datasets, including data from CommonCrawl, an expanded version of WebText, and Wikipedia. Although these large language models have been shown to be effective in various tasks, their large size makes them impossible to use with regular hardware.

2.4 Metrics

Besides human evaluation, which is both slow and costly, NLG evaluation heavily relies on automatic metrics. The most commonly used metrics are heuristic, word-overlap-based approaches such as BLEU, METEOR, and ROUGE. However, these metrics rarely correlate well with human judgements and are inadequate for various NLG tasks (Novikova et al., 2017). In the context of data-to-text generation, we need to consider criteria like fluency, faithfulness (i.e. preserving facts from the input), and coverage (i.e. expressing all the information in the input). Fluency is often evaluated using perplexity with respect to a language model such as GPT-2 (Sai, Mohankumar, and Khapra, 2022). In recent years, several automatic task-specific metrics trained on human judgements have been proposed that can evaluate the mentioned criteria. In the rest of this section, we describe two such metrics used in this thesis, NLGI (Dušek and Kasner, 2020) and BLEURT (Sellam, Das, and Parikh, 2020).

2.4.1 NLGI

NLGI (Dušek and Kasner, 2020) is a metric for the semantic evaluation of data-to-text generation tasks. It uses a natural language inference (NLI) model to check textual entailment between input and output texts, which reveals omissions and hallucinations, thus measuring the coverage and faithfulness of the outputs. The metric is based on RoBERTa (Zhuang et al., 2021), an encoder pre-trained model similar to BERT (Section 2.3.1), fine-tuned on the MultiNLI dataset (Williams, Nangia, and Bowman, 2018). For a given premise and hypothesis text, the model predicts distribution over three results: *entailment*, *contradiction*, and *neutral*. The input to the metric is an RDF triple of facts and an output from the data-to-text generation model. The input triples are transformed into natural language using simple templates that are then used to construct the hypotheses. Each of the hypotheses is evaluated using the NLI model with the data-to-text output as the premise. The hypothesis passes the NLI check if the *entailment* has the highest probability. A failed check is considered an omission. Hallucinations are detected by concatenating the hypotheses into one and checking it against the premise

the other way around – premise and hypothesis are swapped.

2.4.2 BLEURT

BLEURT (Sellam, Das, and Parikh, 2020) is a learned automatic metric based on BERT (Devlin et al., 2019, see Section 2.3.1) fine-tuned for quality evaluation assessing both fluency and semantic quality. The metric correlates with the human evaluation better than heuristic word-overlap-based approaches. BLEURT was trained on a set of various pre-training signals, i.a. BLEU, ROUGE, and another automatic metric BERTscore (Zhang et al., 2020b), a metric based on computing the similarity between pre-trained BART contextual embeddings. The training data are synthetic. The dataset was created by perturbing segments from Wikipedia using three types of perturbations: mask-filling, back-translation, and randomly dropping words. The scores range roughly between 0 and 1, where 0 indicates random output, and 1 indicates a perfect match with the reference.

Chapter 3

Foundations of Our Approach

In this chapter, we give an overview of related research and previous work on which we base our approach. In Section 3.1, we describe the iterative text-editing approach proposed by Kasner and Dušek, 2020, which serves as a starting point for our experiments. In Section 3.2, we introduce the text-editing task and give an overview of the existing text-editing models. In Section 3.3, describe the FELIX text-editing model (Mallinson et al., 2020) in more detail. Next, in Section 3.4, we describe pointer networks for sequence ordering, which are a part of the FELIX model and are also used in our experiments for clause reordering. In Section 3.5, we present the datasets used in our experiments. In Section 3.6, we describe the clause extraction algorithm by Zhang et al., 2020a that we use to prepare a new dataset for our experiments.

3.1 Iterative Text-Editing

Kasner and Dušek, 2020 proposed a novel text-editing approach to data-to-text generation. Their approach is based on transforming the input data using simple templates, which are then iteratively improved by a text-editing model. Kasner and Dušek automatically extracted or handcrafted sets of templates for single RDF triple predicates. Then they train the text-editing model on pairs of examples that consist of $(n, n+1)$ triples and have n common triples. The model is trained on the task of transforming input with n triples to a sentence with $n+1$ triples. The process is illustrated in Table 3.1.

3.2 Text-Editing Models

Text-editing models are a family of models that modify a source sequence by applying a set of edit operations, unlike the traditional Seq2Seq models, which generate

Triples	(Aarhus Airport, location, Tirstrup), (Tirstrup, country, Denmark)
1 Triple Example	Aarhus Airport is located in Tirstrup.
2 Triples Example	Aarhus Airport is located in Tirstrup, Denmark.
Template	<subject> can be found in the country of <object>.
Input	Aarhus Airport is located in Tirstrup. Tirstrup can be found in the country of Denmark.
Reference	Aarhus Airport is located in Tirstrup, Denmark.

Table 3.1 The process of generating training examples for iterative text-editing. We start from two examples that have n common triples (n being 1 in the table), and one of the examples has one more triple. We then find a template for the predicate of the one triple and use it to concatenate it with the example that has n triples. The resulting example is then used as input. The reference is the example with $n + 1$ triples.

the output from scratch. The output sequence is transformed by adding, replacing, or deleting tokens in the input sequence. Text-editing models have become an alternative to Seq2Seq models in tasks in which there is a large overlap between the source and target sequence, such as grammatical error correction, simplification, and style transfer. In contrast to the Seq2Seq models, the text-editing models achieve faster inference speeds and better control and interpretability of the results (Malmi et al., 2022).

The text-editing models differ in the type of operations they use, architecture, and token reordering capability. While some models use autoregressive decoding (Stahlberg and Kumar, 2020; Malmi et al., 2019), others are non-autoregressive (Gu, Wang, and Zhao, 2019; Mallinson et al., 2020). LaserTagger (Malmi et al., 2019) and FELIX (Mallinson et al., 2020) are built on top of the pre-trained BERT model (Devlin et al., 2019). In addition, FELIX is capable of reordering tokens without deleting the tokens using a pointer network proposed by Vinyals, Fortunato, and Jaitly. This way, FELIX addresses LaserTagger's inability to arbitrarily reorder the tokens, which may have led to a limited expressiveness of the model, as stated by Kasner and Dušek. Further details of the FELIX model are described in the following section (Kasner and Dušek, 2020).

3.3 FELIX

FELIX (Mallinson et al., 2020) is a text-editing model that follows up on the LaserTagger (Malmi et al., 2019) model. Both models decompose the text-editing task into two steps – tagging and insertion. The two steps are carried out by two independent models, both based on BERT and trained separately. The key difference between the FELIX and LaserTagger is that FELIX has an additional pointer

network that allows it to reorder tokens without deleting and reinserting them (see the next Section 3.4 for a more detailed description of the pointer network). This mechanism results in a reduction in the number of insert operations. Another difference is that FELIX uses a non-autoregressive decoder for the tagging step, unlike LaserTagger, which uses an autoregressive transformer decoder. FELIX outperforms LaserTagger on several tasks, including sentence fusion. Figure 3.1 provides an overview of the FELIX architecture and its operation on an example input sentence. Felix's tagging and insertion models are described in further detail in Sections 3.3.1 and 3.3.2.

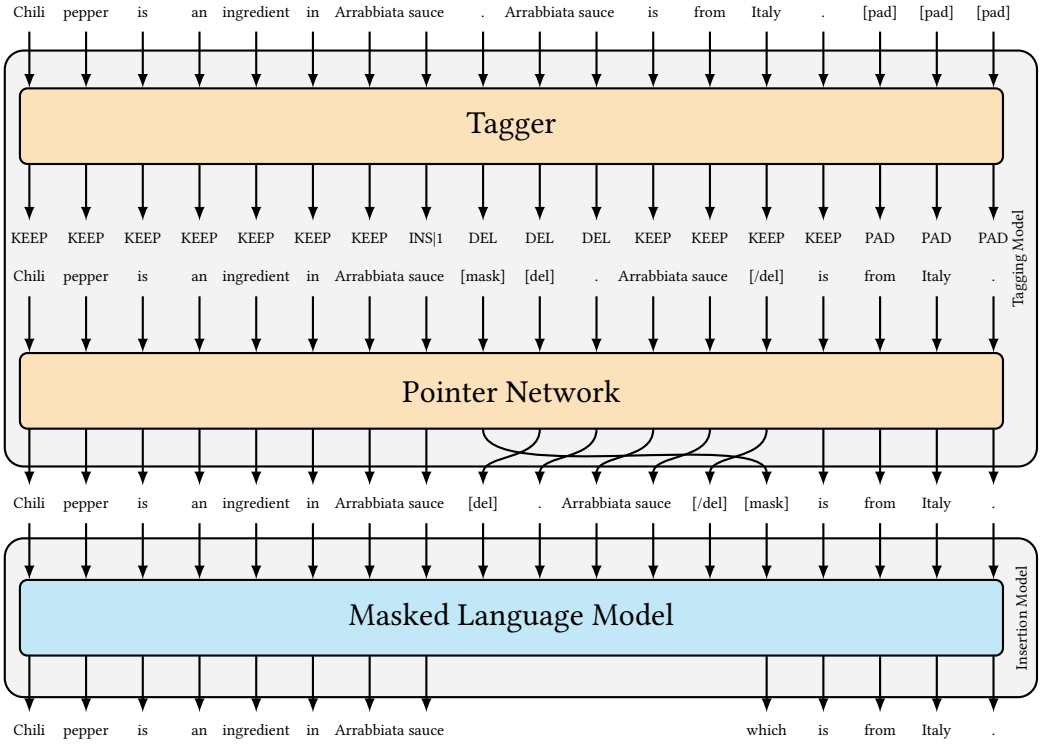

Figure 3.1 FELIX transforming the input sentence *"Chili pepper is an ingredient in Arrabbiata sauce. Arrabbiata sauce is from Italy."* into *"Chili pepper is an ingredient in Arrabbiata sauce which is from Italy."*

3.3.1 Tagging Model

The tagging model performs three subtasks – **encoding**, **tagging**, and **pointing**.

- **Encoding** is done by the pre-trained BERT model.

- **Tagging** is a multi-class classification using a single feed-forward layer on top of the BERT encoder. For each input token, the tagger predicts one of the following classes: `KEEP`, `DELETE`, `INSERT_N` (where `N` is an integer signifying how many tokens are to be added).
- **Pointing** via the pointer network uses an attention mechanism applied to the encoder's hidden states to point to the next token (see Section 3.4). This mechanism allows the model to move tokens around without deleting and reinserting them, reducing the total number of edit operations.

3.3.2 Insertion Model

After the input sequence is tagged, a new sequence is constructed by surrounding the deleted tokens with special `[DEL]` and `[/DEL]` tokens and inserting `[MASK]` tokens after the tokens that were tagged with the `INSERT_N` tag. This new sequence is then fed to the insertion model, which is a fine-tuned masked language model. The insertion model predicts the tokens that are to be inserted in place of the `[MASK]` tokens. As the deleted tokens are kept surrounded by the `[DEL]` and `[/DEL]` tokens, the insertion model can also attend to them. The deleted tokens are removed from the final output sequence.

3.4 Pointer Networks

The pointer network (Vinyals, Fortunato, and Jaitly, 2015) is a neural network architecture which, for a given input sequence, outputs a sequence of pointers to the elements from the input sequence. These pointers designate a new order of the sequence elements. The network is a recurrent sequence-to-sequence model with attention. Since the model works with variable-length sequences, the model takes into account only the non-padding elements of the input sequence by modelling the output by applying softmax to an attention mask over the input, which is computed as follows:

$$u_{ij} = v^T \tanh(W_1 e_i + W_2 d_j)$$

where v, W_1, and W_2 are weights of the model, e^i is the i-th hidden state of the encoder, d_j is the j-th hidden state of the decoder. Then the i-th element of the output sequence is modelled with the following distribution:

$$o_i = \text{softmax}(u_i)$$

Originally the network was designed for solving geometric problems, such as the travelling salesman problem, finding a planar convex hull, or Delaunay

triangulation. However, the network has also been applied to other tasks. An extended version of the network (Wang and Wan, 2019), which uses a hierarchical attention mechanism, has been used for sentence and paragraph ordering (Calizzano, Ostendorff, and Rehm, 2021). FELIX's tagging model, described in Section 3.3.1, uses a pointer network based on the original network by Vinyals, Fortunato, and Jaitly, 2015 to reorder the tokens from the input sequence.

3.5 Datasets

This section is dedicated to two datasets used throughout our experiments for the training and evaluation of our models. The first one is the WebNLG dataset, an important data-to-text generation benchmark. The second one, DiscoFuse is a dataset suitable for training text-editing models for sentence fusion task. In this section, we provide an overview of the datasets and their contents.

3.5.1 WebNLG

The WebNLG dataset is a crowd-sourced dataset designed for the training and evaluation of data-to-text generation systems. The dataset consists of RDF triples extracted from DBPedia[1] and corresponding human-written texts that verbalise the triples. The data are extracted from 15 DBPedia categories: *Astronaut*, *University*, *Monument*, *Building*, *Comics Character*, *Food*, *Airport*, *Sports Team*, *Written Work*, *Athlete*, *Artist*, City, Mean of Transportation, Celestial Body, Politician. To give a few example of the human-written texts:

- Aaron Boogaard was born in Saskatchewan and his club is Wichita Thunder.
- Anderson is part of Adams Township, Madison County, Indiana located in the U.S.

The dataset serves as a common benchmark for comparing data-to-text generation systems (Gardent et al., 2017). In this thesis, we work with an enriched version of the dataset that contains additional information, including delexicalised texts (Castro Ferreira et al., 2018).

3.5.2 DiscoFuse

DiscoFuse (Geva et al., 2019) is an automatically generated large-scale dataset for sentence fusion task, i.e. joining multiple sentences into a more coherent text. The dataset was constructed using a rule-based approach for decomposing sentences

[1] `https://www.dbpedia.org/`

into two independent sentences. The dataset consists of source sentence pairs and fused sentence or sentences, which are illustrated in Table 3.2. The dataset also includes a discourse type, which indicates the structure of the sentences. This approach was applied to articles from Wikipedia and sport report articles. In this thesis, we use only the Wikipedia dataset, which consists of more than 16 million examples. The large scale of the dataset makes it well-suited for training neural models.

Discourse Type	`SINGLE_VP_COORD`
Incoherent First Sentence	This line survived a merger into the Burlington Northern.
Incoherent Second Sentence	This line was abandoned in the 1980s.
Coherent First Sentence	This line survived a merger into the Burlington Northern but was abandoned in the 1980s.
Coherent Second Sentence	—

Table 3.2 Examples taken from the DiscoFuse dataset. The datasets contains pair of incoherent and two coherent sentences created by fusing the two incoherent sentences. In this example, the second coherent sentence is empty in this example as the coherent text consists of one sentence. The discourse type `SINGLE_VP_COORD` signifies how the example was generated – from a single sentence with verb-phrase coordination.

3.6 Clause Extraction

In this thesis, we employ a higher-order operation of clause reordering in our text-editing pipeline. The clause reordering is based on an approach for sentence ordering used in the neural pipeline for data-to-text generation by Kasner and Dusek (2022).

We use an algorithm proposed by Zhang et al., 2020a. The algorithm uses dependency parsing and semantic role labelling to identify clauses in complex sentences. The complex sentences are then split into individual clauses. Semantic role labelling is a shallow semantic parsing technique capturing the relationships between predicates and their associated arguments. The semantic role labelling used by the algorithm is based on the PropBank annotation scheme (Palmer, Kingsbury, and Gildea, 2005). The clause-splitting algorithm works in three steps: **Wh handling**, **conjunction handling**, and **insertion handling**. In Figure 3.2, we illustrate the conjunction handling step on an annotated example.

- **Wh handling** looks for relational arguments (`R-ARG`) and subject argument (`ARG`) preceding a relational predicate. The split is done by replacing the relational argument with the subject argument. For instance, in the sentence *"The instruments that Alison O'Donnell plays are bodhrán, percussion, autoharp plus she also sings"*, the word *that* is a relational argument of verb *plays* and *The instruments* is a subject argument of the same verb. By replacing the relational argument with the subject argument, we obtain clause: *"The instruments are bodhrán, percussion, autoharp"*.

- **Conjunction handling** looks for the conjunction *and*. If the word *and* is followed by an argument (`ARG`), or if the word *and* is followed by a verb (`V`), this is identified as a clause boundary, and the sentence is split. In the latter case, the argument preceding the verb is considered to be a subject argument. The split is done by replacing the word *and* by the subject argument. Conjunction handling is illustrated in Figure 3.2.

- **Insertion handling** looks for nodes in the dependency tree of one of the following types: *participle modifier*, *relative clause modifier*, *prepositional modifier*, *adjective modifier*, or *appositional modifier*. The sentence is split by extracting a clause with the node as the root and prepending the extracted clause with the subject. The rest of the original sentence is passed on for further processing. For example, the sentence *"The first club Adam Maher played for was Netherlands national under-17 football team."* is split by extracting node *played*, being a relative clause modifier. The extracted clause is *"The first club Adam Maher played for"*, and the rest of the sentence with the prepended subject is *"The first club was Netherlands national under-17 football team."*.

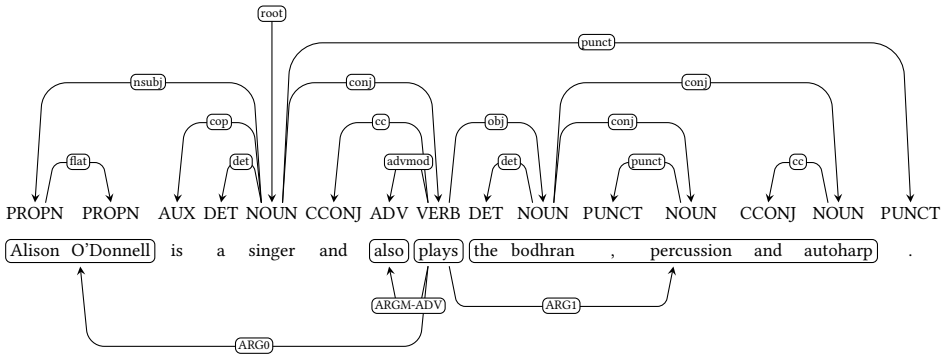

Figure 3.2 Example of the conjunction split. The conjunction *and* is followed by an argument (`ARGM-ADV`); hence the split is made. The word *and* is replaced by the subject argument (`ARG0`), resulting in the following two sentences: *"Alison O'Donnell is a singer"* and *"Alison O'Donnell also plays the bodhran, percussion and autoharp."*

Chapter 4

Experiments

In the following chapter, we describe the details of our own experiments. We start with the description of our data-to-text generation pipeline with the FELIX model in Section 4.1. In the rest of the chapter, individual parts of the pipeline are newly introduced in our system. In Section 4.2, we cover the data preparation and preprocessing, including the implementation of the clause extraction algorithm in Section 3.6 and how we use it in the text-editing pipeline. Next, in Section 4.3, we describe our extensions to the FELIX model. Finally, in Section 4.4, we detail all the variants of our pipeline, which will be later experimentally evaluated in Chapter 5.

4.1 Pipeline

Our data-to-text generation pipeline is based on the iterative text editing approach of Kasner and Dušek, 2020, discussed in Section 3.1. The pipeline consists of two main steps – template infilling and iterative text-editing, described in Section 3.1. In the template selection step, we select appropriate templates for the input triples. The selected templates are filled with the input data. In the iterative text-editing step, we iteratively merge the templated sentences into one sentence using the FELIX text-editing model. Our experiments concern three areas of the pipeline – FELIX training data preparation, FELIX model extensions, and template generation. Our approach differs from the one by Kasner and Dušek, 2020 in the used text-editing model, data preprocessing, and training data. The pipeline is illustrated in Figure 4.1.

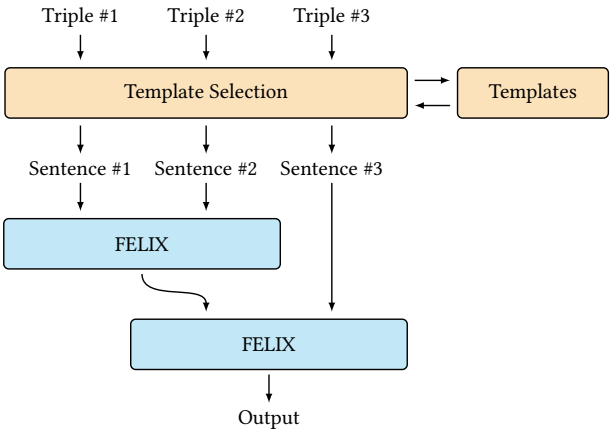

Figure 4.1 Iterative text-editing pipeline diagram.

4.2 Data Processing

This section describes the details of our data preparation and preprocessing for the training of the text-editing model (Sections 4.2.1 and 4.2.2). In Section 4.2.4, we add the implementation details of the clause extraction, previously described in Section 3.6, for the preparation of new training data. Next, in Section 4.2.3, we explain the preparation of new training datasets obtained by filtering and reordering the DiscoFuse. In Section 4.2.5, we introduce clause-level ordering into our pipeline. Both DiscoFuse and WebNLG were introduced in Section 3.5.

4.2.1 Obtaining Single-Triple Templates

As mentioned in Section 3.1, the initial conversion from data to text is handled by simple single-triple templates in Kasner and Dušek's system. We use their approach, including their provided templates, by default. In addition to the provided templates, we use an alternative approach for obtaining the templates by fine-tuning the BART-base model obtained from the HuggingFace Transformers repository[1] (Lewis et al., 2020; Section 2.3.2). We fine-tune the model on the WebNLG dataset to generate reference sentences from the input triples. Then we use this fine-tuned model to construct the inputs for training and inference of the FELIX model. These templates might make the input sequence more similar to the references and thus further reduce the number of required edits.

[1] `https://huggingface.co/facebook/bart-base`

4.2.2 Combining DiscoFuse and WebNLG data

The task of the text-editing model in the iterative text-editing approach described earlier is to join two input sentences into one more fluent sentence, in other words, sentence fusion. Kasner and Dušek use the sentence fusion dataset DiscoFuse for zero-shot domain adaptation (Kasner and Dušek, 2020). We combine the WebNLG dataset with the DiscoFuse dataset using two approaches.

1. We first train the model on the DiscoFuse dataset and then fine-tune it on the WebNLG dataset.

2. We oversample the WebNLG dataset to match the size of the DiscoFuse dataset and then train the model on the combined dataset.

4.2.3 DiscoFuse Filtering and Reordering

DiscoFuse and WebNLG datasets differ in their complexity. Figure 4.2 shows the difference in the number of edits necessary to transform the input text to the output text in both datasets. It is clear that the DiscoFuse dataset is much simpler than the WebNLG dataset, requiring much fewer edits on average. In addition, unlike the WebNLG dataset, the DiscoFuse dataset contains many examples, requiring much fewer edits on average, such as replacing a proper noun with a pronoun. To compensate for this difference and better prepare our models for working on the WebNLG set, we create a new version of the dataset by filtering out the "easiest" examples in terms of the number of edits. By comparing the vocabulary used in the two incoherent and the two coherent sentences, we also reorder the DiscoFuse examples that have reversed order in the original dataset. The distribution of the number of edits in the filtered dataset is plotted in Figure 4.3.

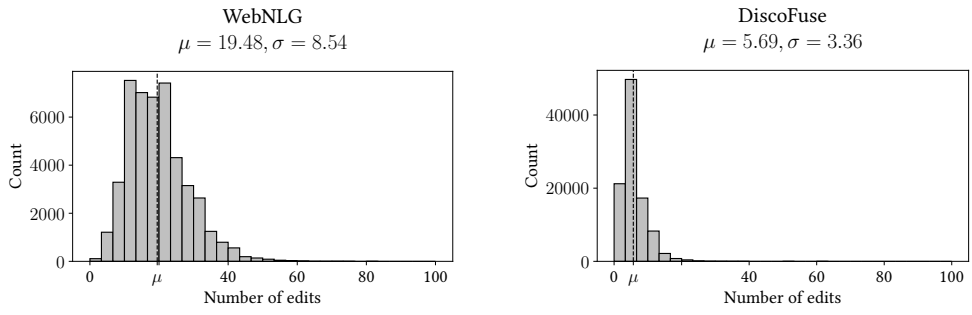

Figure 4.2 Distribution of the number of edits necessary to transform the input text to the output text in the WebNLG and DiscoFuse datasets. The distributions illustrate the difference in complexity and variety of the two datasets.

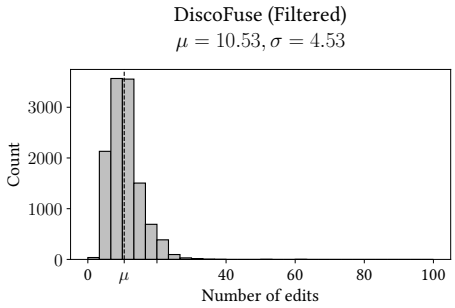

Figure 4.3 Distribution of the number of edits necessary to transform the input text to the output text in the Filtered DiscoFuse datasets.

4.2.4 Clause Extraction

Since the WebNLG and DiscoFuse datasets differ in their complexity (see Figure 4.2) in terms of the number of edit operations necessary to transform the input text to the output text, we simplify the WebNLG dataset by extracting clauses from the original sentences.

We implement a rule-based algorithm for sentence splitting proposed by Zhang et al., 2020a described in Section 3.6, to simplify the WebNLG dataset. We use this algorithm to build a new dataset for sentence fusion on the clause-level. We use this dataset to train the FELIX model. Our implementation uses the AllenNLP semantic role labeller (Shi and Lin, 2019) and the UDPipe toolkit for dependency parsing (Straka, 2018).

Figures 4.2 and 4.4 show the distribution of the number of edits in the original WebNLG dataset, the DiscoFuse dataset and the WebNLG dataset after clause extraction. The clause extraction did reduce the mean number of edits necessary to transform the input text to the output text (close to the mean number of edits in the filtered DiscoFuse dataset); however, the variance has increased.

4.2.5 Clause Reordering

Although FELIX is capable of reordering tokens without deleting them and inserting them again, reordering the input clauses beforehand may reduce the necessary number of edits and thus reduce the load on the model.

We reuse the ordering module used in the neural NLG pipeline[2] by Kasner and Dusek, 2022. The module is a pointer network model based on an architecture by Calizzano, Ostendorff, and Rehm that works on a similar principle as the pointer

[2] `https://github.com/kasnerz/zeroshot-d2t-pipeline`

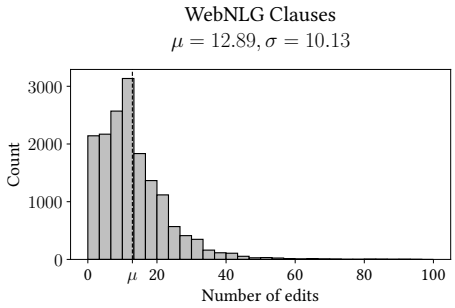

Figure 4.4 Distribution of the number of edits necessary to transform the input text to the output text in the WebNLG dataset after clause extraction. The mean number of edits is reduced compared to the original WebNLG dataset (compare Figure 4.3

network model used in the FELIX model described in Section 3.3. For a more detailed description of the pointer networks, refer to Section 3.4. The model is trained on the WebNLG dataset. Instead of predicting the next token, the model predicts the following clause (Kasner and Dusek, 2022; Calizzano, Ostendorff, and Rehm, 2021).

The training data for the reordering model are constructed by reordering the input clauses by comparing the words used in each input clause to words used in the output clauses – We map each input clause to an output clause based on the cosine similarity of bag-of-words vectors used in the clauses. Then we reorder the input clauses based on the order of the output clauses they are mapped to. On this data, we fine-tune the ordering model. The mapping is illustrated in Figure 4.5.

We modify the inference process such that in each iteration step, we split the current input text into clauses using the clause extraction algorithm described in Section 4.2.4. Then we construct a new text by reordering the clauses using the ordering model. Finally, we feed the reordered text to the FELIX model.

4.3 FELIX Model Extensions

We extend the basic FELIX model described in Section 3.3 by replacing the tagger's feed-forward layer on top of the BERT encoder, i.e. non-autoregressive decoder, with a transformer decoder, i.e. an autoregressive generator. Such extension has been proven to improve the results of the tagger in the original LaserTagger paper (Malmi et al., 2019).

In addition, we modify the decoding mechanism of the transformer decoder such that we preserve the input triples in the output text. We do so by modifying

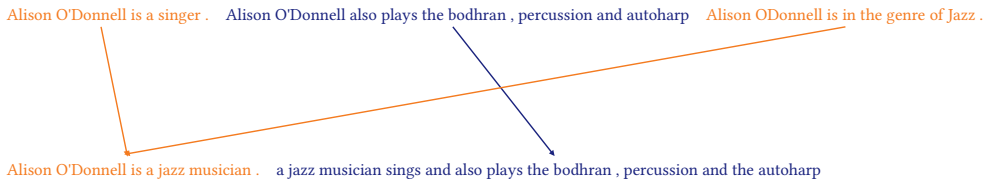

Figure 4.5 Mapping of input clauses to output clauses extracted using the algorithm from Section 4.2.4. Original input: *"Alison O'Donnell is a singer and also plays the bodhran, percussion and autoharp. Alison ODonnell is in the genre of Jazz."* Original reference: *"Alison O'Donnell is a jazz musician who sings and also plays the bodhran, percussion and the autoharp."*

the edit tag probabilities of the tokens that are part of the input triples. We set the probability of the KEEP tag to 1 and other probabilities to 0. This forces the decoder to keep the facts from the input triples in the output sequence. With this modification, the model needs another input besides the input sequence (i.e. template-based natural language sentence, see Section 4.2.1) – a set of the original RDF triples. This approach is an alternative to the approach of Kasner and Dušek, 2020, in which the outputs that did not contain the input triples were discarded and replaced with the original input fallback.

4.4 Setup

In this section, we summarise the experimental settings based on previously described data and model extensions, which are later referred to in the evaluation in Chapter 5. We start by describing the baselines we compare our models with (Section 4.4.1). Then list the training data variants (Section 4.4.2) and the model settings (Section 4.4.3) used in the experiments. Finally, in Section 4.4.4, we describe the metrics used for the evaluation of our models.

4.4.1 Baselines

We compare our models to baseline results obtained by copying the input verbatim to the input (copy baseline) and BART fine-tuned for direct data-to-text generation. We refer to this baseline as a **Copy Baseline**. The BART model was fine-tuned on the WebNLG dataset using a trivial template-based system, with one triple per sentence. The model was trained for 10 epochs with a batch size of 8, the AdamW optimiser (Loshchilov and Hutter, 2019) with a learning rate of 1×10^{-4} and a weight decay set to 1×10^{-2}, and linear schedule with 200 warm-up steps. We refer to this baseline as a **BART End-to-End Baseline**.

4.4.2 Training Data

In the following list, we summarise the training data variants used in the experiments. We will refer to these variants in Chapter 5.

- **WebNLG** – the original iterative-text-editing training data based on the enriched version of WebNLG (Castro Ferreira et al., 2018; see Sections 3.1 and 3.5.1) dataset prepared using the data processing pipeline[3].
- **DiscoFuse** – the original DiscoFuse dataset (Geva et al., 2019; see Section 3.5.2).
- **DiscoFuse** oversampled with WebNLG – the DiscoFuse dataset oversampled with the WebNLG dataset such that it makes up 50% of the training data.
- **Filtered DiscoFuse** – The filtered version of the DiscoFuse dataset (see Section 4.2.3).
- **Filtered DiscoFuse oversampled with WebNLG** – The filtered version of the DiscoFuse dataset oversampled with the WebNLG dataset such that it makes up 50% of the training data.
- **WebNLG clauses** – our dataset constructed from clauses extracted from the WebNLG dataset (see Section 4.2.4).
- **Filtered DiscoFuse oversampled with WebNLG clauses** – The filtered version of the DiscoFuse dataset oversampled with the dataset constructed from clauses extracted from the WebNLG dataset such that it makes up 50% of the training data.
- **Filtered DiscoFuse oversampled with WebNLG clauses and BART templates** – The filtered DiscoFuse dataset oversampled with the dataset constructed from clauses extracted from the WebNLG dataset with the use of templated prepared with BART (see Section 4.2.1) such that it makes up 50% of the training data.

4.4.3 Model Settings

The following list is a brief overview of the experimental settings that utilise the model extensions and data-processing methods described in this chapter. We will refer to these settings in Chapter 5.

[3] `https://github.com/kasnerz/d2t_iterative_editing`

- **Basic Setup** – In the basic setup we employ the pipeline proposed by Kasner and Dušek adapted for the FELIX model without any modifications. Throughout the experiments, we use the model hyperparameters provided by the authors of the model used for the sentence fusion task. Due to the hardware limitations, we use a lower batch size of 4 and 4×10^6 training steps.
- **Transformer Decoder** – In this experimental setup, we use the transformer decoder extension described in Section 4.3. We use the hyperparameters from the LaserTagger paper (Malmi et al., 2019) – 1 layer, 4 heads, hidden size of 768, and feed-forward size of 3072, all with randomly initialised weights.
- **Force Keep Triples** – This setup uses modified decoding, which forces the tagger to keep the facts from the input triples described in Section 4.3.
- **Clause Extraction** – In Section 4.2, we described our simplified WebNLG dataset created by extracting clauses from the original dataset. We use this newly created dataset to train a new FELIX model but also use the extracted clauses for inference.
- **BART templates** – This setup uses the alternative templates generated by BART discussed in Section 4.2.1. The templates are used for inference only but also for training a new model.

4.4.4 Evaluation

All the experimental models are evaluated on the iterative-text-editing test set based on the WebNLG dataset using the following automatic metrics: NLGI,[4] BLEURT,[5] GPT-2 (version with 124M parameters) perplexity from the Hugging Face repository,[6] and average output length. For more information about the metrics and underlying models refer to Sections 2.3.3, 2.4.1 and 2.4.2. Details of the metrics are described in Section 2.4. We also evaluate selected models manually. The manual evaluation is focused on the quality of the generated text in terms of fluency and faithfulness.

[4] `https://github.com/ufal/nlgi_eval`
[5] `https://github.com/google-research/bleurt`
[6] `https://huggingface.co/spaces/evaluate-metric/perplexity`

Chapter 5

Results

In this chapter, we present and discuss the results of the experiments described in Chapter 4. Sections 5.1 to 5.4 discuss the results obtained in individual experiments with model settings described in Section 4.4.3. These sections are focused on the results of automatic evaluation using the metrics described in Section 4.4.4. Section 5.5 presents the results of the manual evaluation of selected models and compares them with the results of the automatic evaluation, adding specific insights regarding individual model behaviour.

5.1 Initial Experiments With the Basic Setup

Our initial experiments compared the performance of the model with the basic setup (i.e. FELIX, see Section 3.3) trained on the WebNLG dataset, on the DiscoFuse dataset (with both base and filtered variant) and on a combination of both datasets – pre-training on the base DiscoFuse and fine-tuning on the WebNLG dataset. The results of the experiments are shown in Table 5.1, comparing to both the copy and BART end-to-end baselines, as described in Section 4.4.1.

We found that training the model solely on the WebNLG dataset is prone to overfitting. Both models trained or fine-tuned on WebNLG perform poorly compared to models trained on the DiscoFuse dataset in both NLGI and BLEURT scores.

The predictions of our models tend to be longer than the human references, suggesting a lower number of edits (coming from the copy baseline with one sentence per triple, i.e. the information density still does not match the references). For comparison, the average length of the reference sentences in the test set is 25.72 words. We have analysed the edit tags predicted by FELIX's tagger model and compared them with the gold tags required to transform the input text into the reference text. Figure 5.1 shows the distribution of the labels in the dataset, and

Setting	OK	O	H	O+H	BLEURT	Avg. Length
Basic / WebNLG	159	47	49	28	0.561	30.98
Basic / DiscoFuse, WebNLG fine-tuning	136	84	21	42	0.271	25.59
Basic / DiscoFuse	189	37	32	25	0.597	31.24
Basic / Filtered DiscoFuse	189	42	26	26	0.605	31.23
Copy Baseline	221	40	11	11	0.152	30.31
BART End-to-End Baseline	205	37	14	27	0.638	30.36

Table 5.1 Results of initial experiments. The second to sixth column contains the results of the NLGI metric as counts out of the 283 examples, which were correct (OK), contained omission (O), hallucination (H), or both (O+H). The next column contains the BLEURT scores. The last column contains the average length of the output sentences in terms of words.

Figure 5.2 contains a confusion matrix of predictions by the model trained on the WebNLG dataset. These figures support the idea that the model is more inclined towards predicting fewer edit operations than the number of edits required to transform the input triple into the reference sentence.

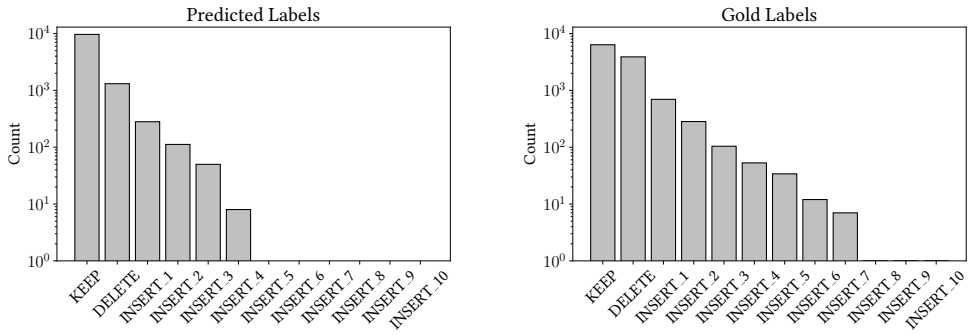

Figure 5.1 Distribution of predicted (left) and gold (right) edit tags in the WebNLG test set. Note that the y-axis has a logarithmic scale.

Filtering and reordering the DiscoFuse dataset does not bring any improvement in the performance of the model. The results of the model trained on the filtered and reordered dataset are nearly identical to the results of the model trained on the original dataset.

Note that even the copy baseline does not achieve a perfect score in the NLGI metric, even though the copy baseline, by definition, should always meet the conditions to pass the NLI check (see Section 2.4.1). These results indicate the limitations of this metric and the difficulty of automatic semantic evaluation.

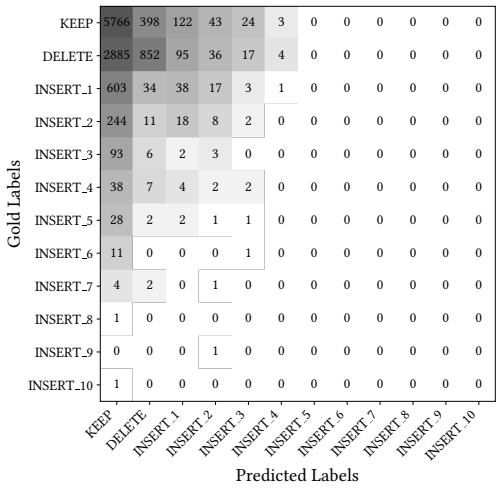

Figure 5.2 Edit tag confusion matrix of the model trained on the WebNLG dataset.

5.2 Transformer Decoder

In the following experiments, we use the modified model with the autoregressive transformer decoder, as described in Section 4.3. As a comparison of results in Table 5.2 with the basic setup in Table 5.1 shows, models trained with the transformer decoder achieve better results in both NLGI and BLEURT scores compared to the models trained on the same data without the transformer decoder.

Setting	OK	O	H	O+H	BLEURT	Avg. Length
Transformer Decoder / WebNLG	159	71	20	33	0.561	26.93
Transformer Decoder / DiscoFuse	195	46	27	15	0.618	30.92
Transformer Decoder / Filtered DiscoFuse oversampled with WebNLG	159	76	17	31	0.564	26.23
Trainsformer Decoder / Filtered DiscoFuse, force keep triples	208	45	19	11	0.613	30.44
Copy baseline	221	40	11	11	0.152	30.31
BART End-to-End baseline Baseline	205	37	14	27	0.638	30.36

Table 5.2 Results of experiments with the transformer decoder.

The lower performance when training directly on the WebNLG dataset led us to oversampling the WebNLG dataset and combining it with the DiscoFuse dataset. However, this approach leads to similar results as the ones achieved by the model trained solely on the WebNLG dataset.

Further improvements were achieved by forcing the model to keep the input triples in the output text (see Section 4.3). Hence, the transformer decoder and the mechanism for keeping the facts from the input triples are used in all the following experiments.

5.3 Clause Extraction and Reordering

Experiments in this section explore the effect of clause reordering (as described in Section 4.2.5) on the performance of the model (using FELIX with the autoregressive decoder and force keep triples setting, as described in Section 5.2) trained using our own new dataset created from the extracted templates (see Section 4.2.4). In some of the experiments, we use the model trained on the filtered version of the DiscoFuse dataset. In these experiments, we compare the effects of clause reordering (described in Section 4.2.5) during inference. Other experiments also use the new clause-based dataset for training. The results of the experiments are shown in Table 5.3.

Setting	OK	O	H	O+H	BLEURT	Avg. Length
WebNLG clauses, no ordering	189	62	9	23	0.562	31.55
Filtered DiscoFuse, inference with clauses, no ordering	215	46	12	10	0.586	35.23
Filtered DiscoFuse, inference with clauses, gold ordering	217	43	12	11	0.589	35.19
Filtered DiscoFuse, inference with clauses, predicted ordering	215	45	11	12	0.582	35.16
Filtered DiscoFuse oversampled with WebNLG clauses	211	52	9	11	0.583	34.98
Copy Baseline	221	40	11	11	0.152	30.31
BART End-to-End baseline Baseline	205	37	14	27	0.638	30.36

Table 5.3 Results of experiments with clause extraction.

With clause extraction, we were able to achieve results comparable to the previous ones by only training on WebNLG without using the DiscoFuse dataset. However, better results were achieved using the DiscoFuse dataset for training than the inference-only clause reordering. Table 5.3 contains results of a model trained on the filtered DiscoFuse dataset from Section 5.2 with clause reordering during inference, i.e. splitting the first (possibly complex) sentence of the input into clauses and reordering as described in Section 4.2.5. We compare three variants: without any ordering, ordering predicted by the model, and gold ordering estimated using the method described in Section 4.2.5. The differences between

the three variants were minimal. As the clause reordering has little effect, we do not use it in the subsequent models. These results suggest that the token-level pointing mechanism of the FELIX model might be sufficient for reordering.

5.4 BART Templates

Experiments listed in Table 5.4 employ the BART during inference and during training instead of the base handcrafted templates (see Section 4.2.1). This approach brings a further improvement in both NLGI and BLEURT metrics, compared to previous results in Table Table 5.3. The best results were achieved when the BART templates were used for both training and inference. These results are already comparable to the copy baseline in terms of NLGI and to the BART end-to-end baseline in terms of BLEURT. In addition, in terms of NLGI, our model performs better than the BART end-to-end baseline.

Setting	OK	O	H	O+H	BLEURT	Avg. Length
Filtered DiscoFuse with WebNLG clauses oversampling, inference with BART templates	218	53	4	8	0.619	32.13
Filtered DiscoFuse with WebNLG clauses and BART templates, inference with BART templates	223	50	1	9	0.619	31.90
Copy Baseline	221	40	11	11	0.152	30.31
BART End-to-End baseline Baseline	205	37	14	27	0.638	30.36

Table 5.4 Results of experiments with BART templates.

5.5 Manual Evaluation

The manual evaluation was performed on 100 sentences sampled from the outputs on the test set for selected models. All the selected models were trained on the combination of the DiscoFuse and WebNLG-based datasets, so the models were trained on data of comparable size. Each of the sampled sentences was evaluated for hallucinations and disfluencies. By disfluency, we mean a sentence that is not syntactically or grammatically correct. For hallucination, we use the definition from Section 2.1. The results are shown in Table 5.5.

The manual evaluation shows significant improvement in fluency with the use of the transformer decoder. However, fluency still poses a problem. We identified several types of disfluencies that occur in the outputs. The identified classes of disfluencies are:

Setting	Disfluencies	Hallucinations
Basic Setup	43	15
Transformer Decoder	25	31
Clause Extraction	21	28
BART Templates	22	26
BART End-to-End Baseline	0	28

Table 5.5 Results of the manual evaluation. The results are given as the absolute counts of sentences containing at least one disfluency or at least one hallucination..

1. Connecting without a connective – locally fluent, e.g. *"300 North La salle is located in Chicago 300 North LaSalle has a floor count of 60."*
2. Missing words and incomplete sentences; e.g. *"Alan Frew is a performer of rock music, the fusion genre of Bhangra, is."*
3. Trailing words and punctuation; e.g. *"Alex Tyus plays in the Turkish Basketball Super League and Maccabi Tel Aviv B.C. is"*, *"Adam Maher is the PSV Eindhoven played for the Netherlands national under – 17 football team,."*

While there is an improvement in fluency between the basic setup and the experiments using the transformer decoder, the number of hallucinations doubled. The number of hallucinations is comparable to the BART baseline, but the nature of hallucinations is different. Text-edited output exhibit hallucinations that result from misuse of connectives and subjunctives, e.g. *"The musical genre of Alex Day is Synthpop, but the stylistic origin of Synthpop is New wave music."* In all three settings that use the transformer decoder, a third of the hallucinations are caused by the misuse of connectives.

We have identified that the misuse of connectives causes a large number of hallucinations. The reason for this behaviour is most likely the fact that some of the connectives are over-represented in the DiscoFuse dataset compared to the WebNLG dataset. The distributions of connectives in both datasets are listed in Table 5.6. In addition, the WebNLG dataset contains a large number of relative clauses. Our analysis of the training set of the WebNLG dataset shows that 52.9% of the sentences in the WebNLG examples contain relative clauses, while only 6.5% of the DiscoFuse dataset contains relative clauses. Geva et al., 2019 also found that connectives like *"however"*, *"although"*, and *"for example"* were harder to learn for their model – the error rate of sentences with these connectives was higher than the error rate of the other connectives. The higher error rate is likely

caused by the fact that the use of these connectives is more complex, as they require a broader context to be used correctly.

Connective	DiscoFuse	WebNLG
and	12.5%	52.06%
but	10.7%	0.73%
although	8.4%	0.03%
however	8.2%	0.15%
because	7.7%	0%
so that	2.1%	0%
while	2.0%	0.11%
or	1.8%	0.50%
so	1.2%	0.06%
for example	1.0%	0%

Table 5.6 Distribution of connectives in the DiscoFuse dataset and WebNLG dataset. Percentages signify the proportion of sentences containing a given connective. DiscoFuse statistics are taken from Geva et al., 2019. Numbers for WebNLG are based on our own analysis of the training set. In the case of connectives *"and"* and *"or"*, we include only connectives that connect clauses or sentences based on a dependency parse by UDPipe.

Setting	Perplexity
Basic Setup	51.8
Transformer Decoder	40.0
Clause Extraction	38.3
BART Templates	34.8
BART End-to-End Baseline	31.6
Copy Baseline	41.6

Table 5.7 Mean perplexity of GPT-2 over the selected model outputs.

The differences between the transformer decoder, clause extraction, and BART template models are subtle in terms of both fluency and hallucinations. This result does not match the NLGI metric, which showed improvements in successive experiments. One of the reasons might be that the NLGI metric does not take into account the fluency of the output. By inspection of the NLGI outputs on our manually evaluated sample, we found that sentences connected without a connective or with a wrong connective are considered by the metric as OK.

Hence the locally fluent outputs may be classified as OK by the NLGI metric. In Table 5.7, we list the perplexity of GPT-2 on the outputs of our selected models and baselines. The perplexity measure shows larger differences between the models.

Besides, the manual evaluation revealed that a number of input sequences are left intact by the models. In Table 5.8, we show the percentage of outputs with no edits. Kasner and Dušek, 2020 did not implement any decoding-time mechanism that would guarantee retaining of the input facts. In their approach, outputs are checked after decoding, and those that do not contain the facts are replaced by a fallback, which copies the input. They reported that 28% of the steps were fallbacks, i.e. no edits were performed. In our approach, we ensure that the facts are kept in the output by modifying the edit tag probabilities during decoding. In spite of the fact that our numbers are lower than 28%, outputs without edits still represent a significant percentage of the outputs.

Setting	**Zero Edits**
Basic Setup	5.30%
Transformer Decoder	10.25%
Clause Extraction	7.07%
BART Templates	9.19%

Table 5.8 Percentage of the selected model outputs without edits.

Chapter 6

Conclusion

In this thesis, we created a data-to-text generation system based on the text-editing model FELIX (Mallinson et al., 2020) and the iterative text-editing approach (Kasner and Dušek, 2020). We explored several extensions to the original FELIX model and template preparation pipeline. The source code of our implementation is attached to this thesis (see Appendix A. We trained the model on different datasets, including our own new versions of the WebNLG and DiscoFuse datasets.

Both automatic and manual evaluation has proven our transformer decoder extension of the FELIX's tagging model to be effective in reducing some artefacts of the non-autoregressive feed-forward model, such as trailing punctuation. Even with this extension, the outputs of the model still contain some disfluencies. Although the FELIX model is able to arbitrarily reorder the input tokens without deleting and re-inserting them, the model still cannot consistently generate fluent outputs. Malmi et al., 2022 recommend studying the effects of scaling up the text editing models as a potential future direction of research. Scaling up proves to be effective in many other transformer-based text-generation tasks and may also increase the fluency and overall quality of the text-editing models' outputs (Brown et al., 2020; Chowdhery et al., 2022; Touvron et al., 2023).

Using the clause extraction, we were able to use the WebNLG data together with the DiscoFuse dataset to achieve better results than using the WebNLG dataset or the DiscoFuse dataset alone. Extracting clauses and filtering the DiscoFuse dataset has brought the datasets closer to each other in terms of the number of edits required. However, the zero-edit outputs (i.e. inputs that are left intact by the model, discussed in Section 5.5) represent a significant fraction of the outputs being higher in models trained on datasets with fewer edits required on average. Our mechanism for keeping the facts from the input RDF triples, which differs from the approach used in the original iterative text-editing approach by Kasner and Dušek, 2020, reduced the number of zero-edit outputs, compared to

the original approach, but the number of such outputs is still significant. We also found that additional clause reordering has little effect on the results, suggesting that FELIX's pointer network is sufficient for the task.

Despite the ability to preserve all the facts from the input using the modified decoding, our manual evaluation revealed that the number of hallucinations in the outputs is comparable to our end-to-end BART baseline results. However, a third of hallucinations of our models amount simply to incorrect use of certain connectives, whilst the output contains all the correct facts. By analysing the training data, we found a large discrepancy between the distributions of connectives in the DiscoFuse and WebNLG datasets. Therefore, the pre-training on the DiscoFuse dataset leads to the overuse of some connectives that shift the meanings in an undesirable way. For future works, it might be beneficial to sample the DiscoFuse dataset in a way that would better match the distribution of connectives in the WebNLG dataset. Automatically generated datasets, such as DiscoFuse and our own clause-based WebNLG dataset, have proven to be effective for training text-editing models. Creating new, possibly in-domain, synthetic datasets could be a promising direction for future research.

Bibliography

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2015). "Neural Machine Translation by Jointly Learning to Align and Translate". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. URL: http://arxiv.org/abs/1409.0473.

Brown, Tom et al. (2020). "Language Models are Few-Shot Learners". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., pp. 1877–1901. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

Calizzano, Rémi, Malte Ostendorff, and Georg Rehm (2021). "Ordering sentences and paragraphs with pre-trained encoder-decoder transformers and pointer ensembles". In: *DocEng '21: ACM Symposium on Document Engineering 2021, Limerick, Ireland, August 24-27, 2021*. Ed. by Patrick Healy, Mihai Bilauca, and Alexandra Bonnici. ACM, 10:1–10:9. DOI: 10.1145/3469096.3469874. URL: https://doi.org/10.1145/3469096.3469874.

Castro Ferreira, Thiago et al. (Nov. 2018). "Enriching the WebNLG corpus". In: *Proceedings of the 11th International Conference on Natural Language Generation*. Tilburg University, The Netherlands: Association for Computational Linguistics, pp. 171–176. DOI: 10.18653/v1/W18-6521. URL: https://aclanthology.org/W18-6521.

Cho, Kyunghyun et al. (2014). "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*. Ed. by Alessandro Moschitti, Bo Pang, and Walter Daelemans. ACL, pp. 1724–1734. DOI: 10.3115/v1/d14-1179. URL: https://doi.org/10.3115/v1/d14-1179.

Chowdhery, Aakanksha et al. (2022). "PaLM: Scaling Language Modeling with Pathways". In: *CoRR* abs/2204.02311. DOI: 10.48550/arXiv.2204.02311.

arXiv: 2204.02311. URL: https://doi.org/10.48550/arXiv.2204.02311.

Devlin, Jacob et al. (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, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*. Ed. by Jill Burstein, Christy Doran, and Thamar Solorio. Association for Computational Linguistics, pp. 4171–4186. DOI: 10.18653/v1/n19-1423. URL: https://doi.org/10.18653/v1/n19-1423.

Dušek, Ondřej and Zdeněk Kasner (Dec. 2020). "Evaluating Semantic Accuracy of Data-to-Text Generation with Natural Language Inference". In: *Proceedings of the 13th International Conference on Natural Language Generation*. Dublin, Ireland: Association for Computational Linguistics, pp. 131–137. URL: https://aclanthology.org/2020.inlg-1.19.

Gardent, Claire et al. (Sept. 2017). "The WebNLG Challenge: Generating Text from RDF Data". In: *Proceedings of the 10th International Conference on Natural Language Generation*. Santiago de Compostela, Spain: Association for Computational Linguistics, pp. 124–133. DOI: 10.18653/v1/W17-3518. URL: https://aclanthology.org/W17-3518.

Gatt, Albert and Emiel Krahmer (2018). "Survey of the State of the Art in Natural Language Generation: Core tasks, applications and evaluation". In: *J. Artif. Intell. Res.* 61, pp. 65–170. DOI: 10.1613/jair.5477. URL: https://doi.org/10.1613/jair.5477.

Geva, Mor et al. (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, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*. Ed. by Jill Burstein, Christy Doran, and Thamar Solorio. Association for Computational Linguistics, pp. 3443–3455. DOI: 10.18653/v1/n19-1348. URL: https://doi.org/10.18653/v1/n19-1348.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). "Deep Learning". In: http://www.deeplearningbook.org.

Gu, Jiatao, Changhan Wang, and Junbo Zhao (2019). "Levenshtein Transformer". In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. Ed. by Hanna M. Wallach et al., pp. 11179–11189. URL: https://proceedings.neurips.cc/paper/2019/hash/675f9820626f5bc0afb47b57890b466e-Abstract.html.

Ji, Ziwei et al. (2022). “Survey of Hallucination in Natural Language Generation”. In: *CoRR* abs/2202.03629. arXiv: 2202.03629. URL: https://arxiv.org/abs/2202.03629.

Kale, Mihir and Abhinav Rastogi (2020). “Template Guided Text Generation for Task-Oriented Dialogue”. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*. Ed. by Bonnie Webber et al. Association for Computational Linguistics, pp. 6505–6520. DOI: 10.18653/v1/2020.emnlp-main.527. URL: https://doi.org/10.18653/v1/2020.emnlp-main.527.

Kasner, Zdeněk and Ondřej Dušek (Dec. 2020). “Data-to-Text Generation with Iterative Text Editing”. In: *Proceedings of the 13th International Conference on Natural Language Generation*. Dublin, Ireland: Association for Computational Linguistics, pp. 60–67. URL: https://aclanthology.org/2020.inlg-1.9.

Kasner, Zdeněk and Ondrej Dusek (May 2022). “Neural Pipeline for Zero-Shot Data-to-Text Generation”. In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Dublin, Ireland: Association for Computational Linguistics, pp. 3914–3932. DOI: 10.18653/v1/2022.acl-long.271. URL: https://aclanthology.org/2022.acl-long.271.

Lewis, Mike et al. (July 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*. Online: Association for Computational Linguistics, pp. 7871–7880. DOI: 10.18653/v1/2020.acl-main.703. URL: https://aclanthology.org/2020.acl-main.703.

Loshchilov, Ilya and Frank Hutter (2019). “Decoupled Weight Decay Regularization”. In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. URL: https://openreview.net/forum?id=Bkg6RiCqY7.

Mallinson, Jonathan et al. (Nov. 2020). “FELIX: Flexible Text Editing Through Tagging and Insertion”. In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online: Association for Computational Linguistics, pp. 1244–1255. DOI: 10.18653/v1/2020.findings-emnlp.111. URL: https://aclanthology.org/2020.findings-emnlp.111.

Malmi, Eric et al. (Nov. 2019). “Encode, Tag, Realize: High-Precision Text Editing”. 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)*. Hong Kong, China: Association for Computational Linguistics, pp. 5054–5065. DOI: 10.18653/v1/D19-1510. URL: https://aclanthology.org/D19-1510.

Malmi, Eric et al. (July 2022). "Text Generation with Text-Editing Models". In: *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Tutorial Abstracts*. Seattle, United States: Association for Computational Linguistics, pp. 1–7. DOI: 10.18653/v1/2022.naacl-tutorials.1. URL: https://aclanthology.org/2022.naacl-tutorials.1.

Novikova, Jekaterina et al. (2017). "Why We Need New Evaluation Metrics for NLG". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*. Ed. by Martha Palmer, Rebecca Hwa, and Sebastian Riedel. Association for Computational Linguistics, pp. 2241–2252. DOI: 10.18653/v1/d17-1238. URL: https://doi.org/10.18653/v1/d17-1238.

Palmer, Martha, Paul R. Kingsbury, and Daniel Gildea (2005). "The Proposition Bank: An Annotated Corpus of Semantic Roles". In: *Comput. Linguistics* 31.1, pp. 71–106. DOI: 10.1162/0891201053630264. URL: https://doi.org/10.1162/0891201053630264.

Radford, Alec et al. (2019a). "Language models are unsupervised multitask learners". In: *OpenAI blog* 1.8, p. 9.

— (2019b). "Language models are unsupervised multitask learners". In: *OpenAI blog* 1.8, p. 9.

Sai, Ananya B., Akash Kumar Mohankumar, and Mitesh M. Khapra (2022). "A Survey of Evaluation Metrics Used for NLG Systems". In: *ACM Comput. Surv.* 55.2. ISSN: 0360-0300. DOI: 10.1145/3485766. URL: https://doi.org/10.1145/3485766.

Sellam, Thibault, Dipanjan Das, and Ankur Parikh (July 2020). "BLEURT: Learning Robust Metrics for Text Generation". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, pp. 7881–7892. DOI: 10.18653/v1/2020.acl-main.704. URL: https://aclanthology.org/2020.acl-main.704.

Shi, Peng and Jimmy Lin (2019). "Simple BERT Models for Relation Extraction and Semantic Role Labeling". In: *CoRR* abs/1904.05255. arXiv: 1904.05255. URL: http://arxiv.org/abs/1904.05255.

Stahlberg, Felix and Shankar Kumar (Nov. 2020). "Seq2Edits: Sequence Transduction Using Span-level Edit Operations". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Online: Association for Computational Linguistics, pp. 5147–5159. DOI: 10.18653/v1/2020.emnlp-main.418. URL: https://aclanthology.org/2020.emnlp-main.418.

Straka, Milan (2018). "UDPipe 2.0 Prototype at CoNLL 2018 UD Shared Task". In: *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies, Brussels, Belgium, October 31 - November 1, 2018*.

Ed. by Daniel Zeman and Jan Hajic. Association for Computational Linguistics, pp. 197–207. DOI: 10.18653/v1/k18-2020. URL: https://doi.org/10.18653/v1/k18-2020.

Touvron, Hugo et al. (2023). "LLaMA: Open and Efficient Foundation Language Models". In: *CoRR* abs/2302.13971. DOI: 10.48550/arXiv.2302.13971. arXiv: 2302.13971. URL: https://doi.org/10.48550/arXiv.2302.13971.

Vaswani, Ashish et al. (2017). "Attention is All you Need". In: ed. by Isabelle Guyon et al., pp. 5998–6008. URL: https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Vinyals, Oriol, Meire Fortunato, and Navdeep Jaitly (2015). "Pointer Networks". In: *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*. Ed. by Corinna Cortes et al., pp. 2692–2700. URL: https://proceedings.neurips.cc/paper/2015/hash/29921001f2f04bd3baee84a12e98098f-Abstract.html.

Wang, Tianming and Xiaojun Wan (2019). "Hierarchical Attention Networks for Sentence Ordering". In: *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, pp. 7184–7191. DOI: 10.1609/aaai.v33i01.33017184. URL: https://doi.org/10.1609/aaai.v33i01.33017184.

Williams, Adina, 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)*. New Orleans, Louisiana: Association for Computational Linguistics, pp. 1112–1122. URL: http://aclweb.org/anthology/N18-1101.

Zarrieß, Sina, Henrik Voigt, and Simeon Schüz (2021). "Decoding Methods in Neural Language Generation: A Survey". In: *Inf.* 12.9, p. 355. DOI: 10.3390/info12090355. URL: https://doi.org/10.3390/info12090355.

Zhang, Li et al. (Nov. 2020a). "Small but Mighty: New Benchmarks for Split and Rephrase". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Online: Association for Computational Linguistics, pp. 1198–1205. DOI: 10.18653/v1/2020.emnlp-main.91. URL: https://aclanthology.org/2020.emnlp-main.91.

Zhang, Tianyi et al. (2020b). "BERTScore: Evaluating Text Generation with BERT". In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. URL: https://openreview.net/forum?id=SkeHuCVFDr.

Zhu, Yukun et al. (2015). “Aligning Books and Movies: Towards Story-Like Visual Explanations by Watching Movies and Reading Books”. In: *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*. IEEE Computer Society, pp. 19–27. DOI: 10.1109/ICCV.2015.11. URL: https://doi.org/10.1109/ICCV.2015.11.

Zhuang, Liu et al. (Aug. 2021). “A Robustly Optimized BERT Pre-training Approach with Post-training”. English. In: *Proceedings of the 20th Chinese National Conference on Computational Linguistics*. Huhhot, China: Chinese Information Processing Society of China, pp. 1218–1227. URL: https://aclanthology.org/2021.ccl-1.108.

Appendix A

Attachments

The attached archive contains a modified version of the original FELIX model source code.[1] The modified code implements the extensions described in Section 4.3 and inference with clause reordering described in Section 4.2.5. The archive also contains the implementation of the clause extraction algorithm described in Section 3.6. For further details, see the `README.md` files in the archive.

[1] `https://felixmodel.page.link/code`