Automatic Decomposition of Text Editing Examples into Primitive Edit Operations: Toward Analytic Evaluation of Editing Systems

Daichi Yamaguchi[1], **Rei Miyata**[2],
Atsushi Fujita[3], **Tomoyuki Kajiwara**[4], **Satoshi Sato**[1]
[1]Nagoya University, [2]The University of Tokyo,
[3]National Institute of Information and Communications Technology, [4]Ehime University
[1]Nagoya, Japan, [2]Tokyo, Japan, [3]Kyoto, Japan, [4]Ehime, Japan
yamaguchi.daichi.nlp@gmail.com, miyata@p.u-tokyo.ac.jp,
atsushi.fujita@nict.go.jp, kajiwara@cs.ehime-u.ac.jp, sato.satoshi.g9@f.mail.nagoya-u.ac.jp

Abstract

This paper presents our work on a task of automatic decomposition of text editing examples into primitive edit operations. Toward a detailed analysis of the behavior of text editing systems, identification of fine-grained edit operations performed by the systems is essential. Given a pair of source and edited sentences, the goal of our task is to generate a non-redundant sequence of primitive edit operations, i.e., the semantically minimal edit operations preserving grammaticality, that iteratively converts the source sentence to the edited sentence. First, we formalize this task, explaining its significant features and specifying the constraints that primitive edit operations should satisfy. Then, we propose a method to automate this task, which consists of two steps: generation of an edit operation lattice and selection of an optimal path. To obtain a wide range of edit operation candidates in the first step, we combine a phrase aligner and a large language model. Experimental results show that our method perfectly decomposes 44% and 64% of editing examples in the text simplification and machine translation post-editing datasets, respectively. Detailed analyses also provide insights into the difficulties of this task, suggesting directions for improvement.

Keywords: Text Editing, Analytic Evaluation, Text Generation

1. Introduction

Text editing plays an important role in human writing activities, which widely covers various tasks, such as text simplification, grammatical error correction, post-editing and pre-editing for machine translation, and sentence compression. Many studies have attempted to automate these tasks and made significant progress by using deep learning methods (e.g., Filippova et al., 2015; Nisioi et al., 2017; Junczys-Dowmunt et al., 2018; Correia and Martins, 2019; Feng et al., 2023; Raunak et al., 2023).

One of the challenges in developing and evaluating text editing systems is to concretely understand the behavior of the systems, i.e., how the system rewrites a source text to generate an edited text. Deep learning methods are by nature black-boxed; they are usually trained to predict an edited sentence from a source sentence in an end-to-end fashion. Moreover, evaluation practices in text editing tasks have tended to focus on numerically summative evaluation metrics. Widely-used automatic metrics, such as BLEU (Papineni et al., 2002), ROUGE (Lin, 2004), and BLEURT (Sellam et al., 2020), can gauge the overlap or similarity between system outputs and human references, but they are not designed to capture the detailed edit operations performed by the systems. The problem of low interpretability of such metrics has long been acknowledged (Sai et al., 2022).

To understand and explain the system's behavior in detail, it is essential to develop tools to analytically evaluate system outputs in terms of edit operations that are understandable by humans. Analytic evaluation generally consists of two subtasks: identification of edit operations from a pair of source and edited sentences and classification of each operation. For the former, identification of edit operations can be addressed by decomposing a given pair of source and edited sentences into multiple minimal edit operations (Miyata and Fujita, 2021). However, the procedure to identify the unit of editing has been neither formalized nor automated. For the latter, taxonomies of edit operations have been proposed for various editing tasks (e.g., Vila et al., 2014; Kovatchev et al., 2018; Yamamoto and Yamada, 2022), some of which include an annotation scheme for helping consistent classification (Yamaguchi et al., 2023).

Against this backdrop, we propose to formalize the task of decomposing text editing examples and attempt to automate this process. For instance, a pair of source and edited sentences in Figure 1 can be decomposed into three *primitive edit operations*, i.e., semantically minimal edit operations that preserve grammaticality. Such a decomposition can scaffold fine-grained analyses of edit operations in the next step. We implemented an automatic decomposition method with

LREC-COLING 2024, pages 1899–1914

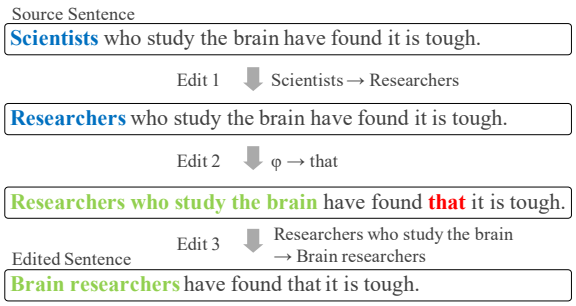

Figure 1: Decomposition of a text editing example into primitive edit operations.

an alignment tool and a pre-trained large language model (LLM). Then, we experimentally verified it using text simplification and machine translation post-editing datasets. The results showed that our proposed method perfectly decomposed 44% and 64% of editing examples in the datasets, respectively. Our in-depth analyses of both erroneous and successful cases revealed the difficult types of problems and suggested future research directions.

2. Related Work

Understanding the system's behavior has been addressed in the field of explainable AI, which aims to help the system users understand how the model makes the prediction (Burkart and Huber, 2021). The way of explanation can be classified into *self-explaining* or *post-hoc* (Danilevsky et al., 2020). The self-explaining methods use information obtained when the model predicts. For instance, editing systems that generate outputs based on an intermediate prediction of edit operation sequence (Stahlberg and Kumar, 2020; Dong et al., 2019) are classified into the former. In contrast, the post-hoc methods use information obtained from an additional process after the prediction has been done. The task tackled in our study, i.e., the decomposition of any given text editing examples, falls into the latter.

To evaluate editing systems, several studies have analyzed the statistics of surface edit operations (e.g., insertion, deletion, substitution, and shift) performed by the systems (Zhang and Lapata, 2017; Sun et al., 2020; Bhattacharyya et al., 2022). The distribution of edit operations is usually obtained by using automatic methods, such as TER (Snover et al., 2006) and SARI (Xu et al., 2016), in a post-hoc manner. Although the general characteristics of adopted edit operations can be observed, the identified textual unit of edit operations does not guarantee grammatical and semantic cohesiveness and thus may not necessarily be understandable for humans.

As more human-interpretable methods, many attempts have been made to develop taxonomies of editing types or strategies for various tasks, including paraphrasing (Bhagat and Hovy, 2013; Vila et al., 2014; Kovatchev et al., 2018), text simplification (Shardlow, 2014; Yamaguchi et al., 2023), grammatical error correction (Bryant et al., 2017), pre-editing (Miyata and Fujita, 2021), post-editing (Tatsumi, 2010), and target text editing in translation (Yamamoto and Yamada, 2022). As for text simplification, Yamaguchi et al. (2023) also provided a decision tree to classify an edit operation based on the simplification strategy taxonomy.

Nevertheless, these past studies possess the following two limitations: (1) procedures to identify the unit of edit operation have yet to be formalized, and (2) analysis of edit operations has been conducted manually. Identification of edit operations in the given pair of source and edited texts chiefly depends on the annotators' expertise, which may hinder the consistent analysis between annotators.[1] In the context of pre-editing, Miyata and Fujita (2021) have defined a primitive edit operation as an edit operation that cannot be decomposed while maintaining its original meaning.[2] However, their definition is not detailed enough for consistently identifying primitive edit operations and may not be applicable to other editing tasks in which original information may change, such as text simplification. Furthermore, among other evaluation methods, analytic evaluation is particularly labor-intensive. To implement such evaluation in the rapid cycle of research, automation of the evaluation process is highly demanded.

Text morphing is a related task, which automatically generates fluent intermediate sentences that smoothly change sentence A into sentence B (Huang et al., 2018). While this task is similar to our task of decomposing text editing examples, the input of text morphing is an arbitrary pair of sentences rather than a pair of source and edited sentences. The constraint on the intermediate sentence is more relaxed than ours, which may permit the generation of deviated intermediate sentences that are unacceptable as an edit operation. In addition, Huang et al. (2018) do not explicitly consider the primitiveness.

[1] For the task of translation error classification, the difficulty in identifying erroneous spans in text has been suggested (Lommel et al., 2014; Fujita et al., 2017).

[2] Miyata and Fujita (2021) use the term "minimum edit" rather than "primitive edit operation."

3. Problem Statement

3.1. Problem to be Solved

Given a pair of source and edited sentences (X_{src}, X_{edt}), we generate a sequence of edit operations that connects the pair of sentences. In general, there exist multiple orders of edit operations. Consider the following pair of sentences.

X_{src}: **Tom** said he loved **The Smiths**.
X_{edt}: **He** said he loved **the band**.

This pair of sentences can be explained in two ways: "Tom" is replaced with "He" before/after "The Smiths" is replaced with "the band." Therefore, the result of this decomposition forms a lattice of edit operations in general. Formally,

$$\mathbb{E} = \{(E_1, ..., E_n) \mid X_{edt} = E_n(...E_1(X_{src}))\}$$

where E_i is an edit operation, a function that edits one part of the input sentence X_{i-1} into another expression and returns X_i $(= E_i(X_{i-1}))$. The index i indicates the order of the application of an edit operation. Given (X_s, X_t), such that $s < t$, we call X_s a source-side sentence and X_t an edited-side sentence. We denote an edit operation E_i as "A $\Rightarrow$ B," which means the edit operation changes A of X_{i-1} into another expression B to form X_i. We refer to A as a source side of an edit operation and B as an edited side of it.

Our goal is to obtain one sequence of edit operations within such a lattice, i.e., $\mathbf{E} \in \mathbb{E}$, due to the following two reasons. First, to understand the system's behavior, the entire lattice is unnecessary. Second, there could be a true order of the edit operations but it cannot be identified from only the given sentence pair.

3.2. Features of the Problem

Dependency of Edit Operations: An edit operation can depend on another edit operation; in other words, there exist strict orders of edit operations. For instance, the following pair of source and edited sentences can be explained by two edit operations: "me $\Rightarrow \phi$ " and "tell $\Rightarrow$ state." The latter can only be applied after the former has been applied.

X_{src}: Please **tell me** the truth.
X_{edt}: Please **state** the truth.

Hidden Edit Operation: An edit operation can be composed of words that appear in neither source nor edited sentences. For instance, the pair

X_{src}: I **start using** that.
X_{edt}: I **do** that.

can be explained by two edit operations in two ways:

1. ("using $\Rightarrow$ **doing**", "start **doing** $\Rightarrow$ do")
2. ("start using $\Rightarrow$ **use**", "**use** $\Rightarrow$ do")

where neither "doing" nor "use" appear in the source and edited sentences.

Uncontinuous Edit Operation: Each side of edit operation can be composed of an uncontinuous span of words. For instance, the following pair of sentences can be explained by an edit operation "like ... very much $\Rightarrow$ love," where "like" in the source side is apart from "very much."

X_{src}: I **like** David Bowie **very much**.
X_{edt}: I **love** David Bowie.

3.3. Constraints

Considering the features of the problem (Section 3.2), we impose the following three constraints on the edit operations to be generated in this task.

Grammaticality: We consider that an edit operation must generate a sentence with the newly rendered part having no grammatical error. The term "edit operation" is used in the context of edit distance, and individual edit operations identified by the existing algorithms for edit distance (e.g., Levenshtein Distance) do not necessarily ensure grammatical correctness of the textual part when individual edit operation is applied. By contrast, in our study, we consider only edit operations that guarantee *Grammaticality* because we focus on the edit operations that are understandable by humans.

Unswervingness: Given a pair of sentences, the edit operation sequence for it must not contain redundant edit operations. We define such a status as *Unswervingness* and refer to an edit operation which satisfies *Unswervingness* as an unswerving edit operation. We regard any sentences generated with unswerving edit operations as intermediate sentences. Consider the following triplet of sentences $(i < j < k)$.

X_i: I **like** David Bowie **very much**.
X_j: I **like Ziggy Stardust very much**.
X_k: I **love** David Bowie.

X_j does not qualify as an intermediate sentence because "David Bowie $\Rightarrow$ Ziggy Stardust" is redundant. *Unswervingness* of an edit operation is evaluated referring to an edited-side sentence.

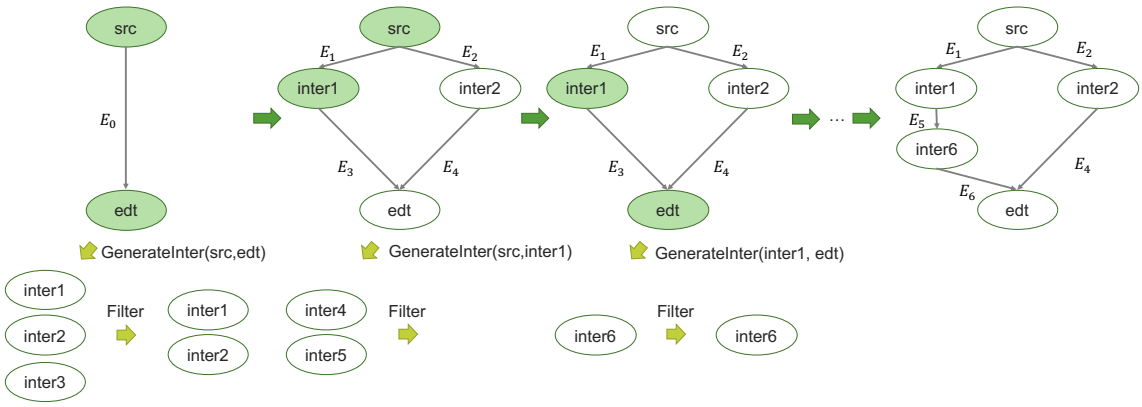

Figure 2: Incremental process for generating a lattice of edit operations.

Primitiveness: We regard an edit operation as primitive iff a source side and an edited side of an edit operation form a group of minimal words related to each other. We impose the constraint of *Primitiveness* on edit operations because this can lead to consistent identification of a fine-grained unit of edit operation. *Primitiveness* of an edit operation depends on the pair of source- and edited-side sentences. Consider the following triplet of sentences ($i < j$).

X_{i-1}: I **like** David Bowie very much, **too**.
X_i : I **love** David Bowie very much, **too**.
X_j : I **love** David Bowie very much.

For this triplet of sentences, "like ⇒ love" is primitive. In contrast, the same edit operation is not primitive for the following triplet of sentences because "like ... very much" and "love" form a group of minimal words related to each other.

X_{i-1}: I **like** David Bowie **very much**, **too**.
X_i : I **love** David Bowie very much, **too**.
X_j : I **love** David Bowie.

Although both examples above consist of the same X_{i-1} and X_i, the judgment of *Primitiveness* differs depending on X_j. Therefore, to evaluate *Primitiveness* of an edit operation that generates an intermediate sentence, an edited-side sentence must be considered.

4. Our Proposed Method

4.1. Overview

To generate a sequence of edit operations for a given pair of source and edited sentences, we first generate a lattice whose nodes are intermediate sentences and edges are edit operations, and then choose one of its paths. We conduct an exhaustive search assuming that the sequence length is limited and the number of paths does not increase exponentially. The lattice makes it possible to identify an optimal path if a proper objective function is available, unlike beam search or greedy search.

4.2. Lattice Generation

We generate a lattice by iteratively generating intermediate sentences of any pair of sentences (X_s, X_t), such that $s < t$, with an Intermediate Sentence Generator and filtering them by judging their *Grammaticality* and *Unswervingness* with an Edit Operation Filter. Figure 2 illustrates this process. We denote an intermediate sentence as X_{inter} ($s < inter < t$).[3]

4.2.1. Intermediate Sentence Generators

As a way of generating intermediate sentences, we examine a phrase aligner and a pre-trained large language model (LLM).

Alignment-based Method: This method considers an alignment as an action of altering aligned words/phrases and thus a candidate of edit operation. To ensure that the edit operations satisfy *Grammaticality*, we employ a phrase aligner instead of a word aligner.[4] Given a pair of sentences, we first identify their phrase alignments. Then, we generate their intermediate sentences, separately applying each alignment as an edit operation to the source-side sentence. This method can generate edit operations composed of continuous words appearing in the given sentences, but not hidden edit operations and uncontinuous edit operations (See Section 3.2).

LLM-based Method: This method generates an intermediate sentence for a given pair of sentences with few-shot prompting to LLM. While it

[3] An algorithm we used is given in Appendix D.

[4] With the term "phrase," we refer to those in the context of constituency parsing.

is difficult to force LLM to generate only unswerving intermediate sentences, unlike the alignment-based method, this may generate any intermediate sentences, including the ones generated with hidden edit operations and uncontinuous edit operations. We collect reliable (X_s, X_{inter}, X_t) triplets and use them for the few-shot prompt, following the framework of Retrieval Augmented Generation (RAG) (Lewis et al., 2021), regarding the given pair of sentences as a query to search for the triplets that are similar to the query. The format of the prompt we use is shown in Section 5.1.

4.2.2. Edit Operation Filter

This discards swerving edit operations by judging their *Grammaticality* and *Unswervingness*, using two components: an Edit Operation Scorer and a text similarity metric. Given a triplet of (X_s, X_{inter}, X_t), the Edit Operation Scorer computes p_{gu}, the likelihood of an edit operation E, such that $X_{inter} = E(X_s)$, being grammatical and unswerving. We retain an edit operation iff X_{inter} generated with it satisfies all of the following conditions.

- p_{gu} is greater than a threshold
- $sim(X_s, X_{inter}) > sim(X_s, X_t)$ and $sim(X_{inter}, X_t) > sim(X_s, X_t)$

where $sim(\cdot, \cdot)$ is a text similarity metric. The former concerns *Grammaticality* and *Unswervingness*, whereas the latter concerns only *Unswervingness*. To compute p_{gu}, we implement a model through finetuning a pre-trained language model as detailed in Section 5.1.2.

4.3. Path Search

Having obtained a lattice of intermediate sentences, we score *Primitiveness* of each edit operation, i.e., edge, in the lattice with a Primitiveness Scorer, and select the most likely path. Then, we extract a sequence of edit operations $(E_1, ..., E_n)$ from a sequence of sentences $(X_{src}, ..., X_{edt})$.

Primitiveness Scorer: Given a triplet of (X_s, X_{inter}, X_t), this scorer computes p_{pri}, the likelihood of edit operation E, such that $X_{inter} = E(X_s)$, being primitive. To compute p_{pri}, we implement a model through finetuning a pre-trained language model as we do for the Edit Operation Scorer. The detailed implementation is shown in Section 5.1.3.

Search of the Path: We choose one lattice path with the minimum negative log-likelihood ($-\sum \log p_{pri}$) by the Viterbi algorithm.

Edit Operation Extractor: The sequence of sentences $(X_{src}, ..., X_{edt})$ in the selected path is finally converted to a sequence of edit operations $(E_1, ..., E_n)$. Assuming that E_i is primitive, we determine the difference between X_{i-1} and X_i ($X_i = E_i(X_{i-1})$) with a word matching method and regard the mismatched pair of spans "A ⇒ B" as an edit operation E_i.

5. Experiment

We evaluated how well our proposed method decomposes pairs of source and edited sentences.

5.1. Implementation Details

5.1.1. Intermediate Sentence Generators

Alignment-based Method: We used the Arase and Tsujii (2020)'s aligner,[5] which outputs a set of alignments based on Enju.[6]

We set the threshold for the phrase aligner to 0.5 and 0.8 and obtained alignments separately. We used the union of the two sets. This enabled us to get alignments at high coverage.

LLM-based Method: We used Llama-2.cpp 70B,[7] which is a four-bit quantized model of Llama 2 70B (Touvron et al., 2023b).[8]

To create a database containing (X_s, X_{inter}, X_t) triplets for the few-shot prompting, we utilized Newsela-Auto (Jiang et al., 2020) created by automatically aligning manually simplified data. In Newsela-Auto, each original document (L0) has four simplified versions (L1–L4). The higher the level is, the simpler the document is. We extracted triplets of adjacent three versions iff all the three were different from each other and each one was composed of a single sentence.[9] We discarded triplets that violated one of the following three criteria considering that such X_{inter} is swerving.

- $sim(X_s, X_{inter}) > sim(X_s, X_t)$
- $TER(X_s, X_{inter}) \leq TER(X_s, X_t) \leq 0.50$[10]

[5] https://github.com/yukiar/phrase_alignment_cted with a pre-trained model available at https://zenodo.org/record/4686663

[6] https://github.com/mynlp/enju

[7] https://huggingface.co/TheBloke/Llama-2-70B-GGUF

[8] We evaluated several models and model sizes, including Llama 1 (Touvron et al., 2023a), regarding the performance of generating intermediate sentences with the validation data. The Llama 2 70B was the best model. Refer to Appendix A.

[9] This is because we targeted the sentence-level edit operation.

[10] https://github.com/mjpost/sacrebleu/blob/master/sacrebleu/metrics/ter.py

SRC: Do you love Large Language Model?
TGT: Do you like LLM?
Rewrite SRC step by step until the rewritten sentence matches TGT.
Do you love Large Language Model?
Do you love LLM?
Do you like LLM?
…
SRC: Long live Large Language Model!
TGT: Hail to LLM
Rewrite SRC step by step until the rewritten sentence matches TGT.
Long live Large Language Model!

Figure 3: Prompt for LLM-based intermediate sentence generator. The first *k* sentence pairs (shown in red) are the examples, and the last one (shown in black) is the sentence pair to be decomposed.

- $\mathit{diff}(X_s, X_{\mathit{inter}}) \subseteq \mathit{diff}(X_s, X_t)$ and $\mathit{diff}(X_t, X_{\mathit{inter}}) \subseteq \mathit{diff}(X_t, X_s)$

where $\mathit{sim}(\cdot,\cdot)$ is BERTScore (Zhang et al., 2020) and $\mathit{diff}(\cdot,\cdot)$ is a function that returns a set of tokens in the first argument sentence differing from those in the second argument sentence. Specifically, the third criterion prunes edit operations like "A ⇒ B" and then "B ⇒ A." After this data selection, we built the database from the remaining 8,978 triplets.

To search for the *k*-nearest neighbors for the *k*-shot prompting, we used FAISS (Johnson et al., 2021),[11] relying on the Euclidean distance. To vectorize each pair of source- and edited-side sentences, we concatenated the two sentences with a separator token </s> and encoded it with Llama-2.cpp 70B. Then, we used the embedding for the last token as the vector for the sentence pair.

Figure 3 shows an example of the prompt. The LLM-based method generated only one intermediate sentence, which was the output from the end of the prompt to the newline code, per input sentence pair. We set *k* to 5 and initialized the Llama model with a seed of 42 every time before inputting a pair of source and edited sentences.[12]

5.1.2. Edit Operation Filter

Text Similarity Metric: We utilized $(1 - \mathit{TER})$.

Edit Operation Scorer: This was implemented through finetuning RoBERTa (Liu et al., 2019)[13] with one additional fully connected layer. Given a triplet of (X_s,X_{inter},X_t), concatenated with a separator token </s>, it quantifies how likely the edit operation E, such that $X_{\mathit{inter}} = E(X_s)$, is grammatical and unswerving. The model was trained on two datasets: an automatically created dataset and a manually created dataset. The former consists of positive examples of the 8,978 triplets used in the LLM-based method, and the same number of negative examples sampled from the pruned triplets. For the latter, we automatically generated intermediate sentences with pairs of sentences in Newsela-Auto with the alignment-based method, and manually checked whether each sentence was grammatical and unswerving. This resulted in 4,460 examples: 1,077 positive and 3,383 negative examples. The model was first trained on the automatically created dataset. Then, we initialized the last fully connected layer and finetuned the model on the manually created data.[14]

We set the threshold for the Edit Operation Scorer to 0.5.

To evaluate the Edit Operation Scorer, we used an evaluation dataset, which was created in the same way as the manually created training dataset. The evaluation dataset consists of 662 examples: 156 positive and 506 negative examples. Table 1 shows the evaluation result, suggesting that the Edit Operation Scorer is adequately capable of judging whether a given edit operation satisfies *Grammaticality* and *Unswervingness*.

	Recall	Precision	F_1
Positive	0.69	0.89	0.78
Negative	0.97	0.91	0.94

Table 1: Performance of the Edit Operation Scorer.

5.1.3. Primitiveness Scorer

We implemented the Primitiveness Scorer by finetuning RoBERTa in the same way as we did for the Edit Operation Scorer. To finetune the pre-trained model, we used another manually created dataset. To create the dataset, we judged *Primitiveness* of each of the positive examples in the manually created dataset for the Edit Operation Scorer and labeled it as positive if it was judged as primitive. This dataset had 1,077 examples: 589 positive and 488 negative examples.

If the output sequence of edit operations contains an edit operation with p_{pri} less than a threshold, we excluded the edit operation from the evaluation. We set the threshold for the Primitiveness Scorer to 0.5.

To evaluate the Primitiveness Scorer, we used an evaluation dataset which was created in the same way as the training dataset. The evaluation

[11] https://github.com/facebookresearch/faiss

[12] We determined the number of shots and the seed through a preliminary study. For the preliminary experiment, refer to Appendix A.

[13] https://huggingface.co/roberta-large

[14] We tested three models: the one trained only on the manually created dataset and the ones trained on the automatically created and manually created datasets with/without initializing the last fully connected layer. Then, we chose the best model.

	Recall	Precision	F_1
Positive	0.89	0.82	0.85
Negative	0.75	0.83	0.79

Table 2: Performance of the Primitiveness Scorer.

dataset consists of 156 examples: 89 positive and 67 negative examples. Table 2 shows the evaluation result. Precision values are above 0.8 for both positive and negative cases. This indicates the high capability of the Primitiveness Scorer to judge whether a given edit operation satisfies *Primitiveness*.

5.1.4. Edit Operation Extractor

We extracted an edit operation from each pair of X_{i-1} and X_i with word matching obtained by TER. We determined the shortest span of X_{i-1} where the application of the edit operation is uniquely determined as the span of the edit operation. Consider the following sentences.

X_{i-1}**:** She said that it was not that good.
X_i **:** She said that it was not good.

In the example, "that good ⇒ good" is considered as the edit operation rather than "that ⇒ ϕ."

5.2. Evaluation Data

We evaluated our system using text simplification (TS) and post-editing (PE) datasets, both of which are English. We used the sentence pairs with TER scores equal to or less than 0.50 since our preliminary analysis revealed that there exist sentence pairs that are difficult to decompose even manually and they tend to have TER scores greater than 0.50.

For the TS dataset, we randomly sampled a total of 50 sentence pairs from the Newsela-Auto corpus test set, ten sentence pairs in each bin with a 0.1 point width of TER. When sampling the sentence pairs, we automatically excluded trivial ones having an edit distance of one. As the gold reference data, we manually created a lattice, containing all possible intermediate sentences, for each pair of source and edited sentences. To this end, we manually extracted the primitive edit operations for each sentence pair.

For the PE dataset, we used machine-translated and then post-edited data from documents of a local government.[15] The original documents had been written in Japanese, and we machine-translated them into English with TexTra.[16] Then, the professional translator, who is a native English speaker, post-edited the machine-translated sentences. We created the PE dataset in much the same way as the TS dataset, except that the two annotators were involved. The in-house worker and the first author manually extracted the primitive edit operations. If there was a disagreement between their results, they resolved it through discussion.

After extracting the primitive edit operations, for both datasets, we automatically generated all permutations of the order of them and corresponding intermediate sentences. Therefore, all intermediate sentence sequences for a pair of source and edited sentences have the same length. Then, to accord with the span of an edit operation, we extracted edit operations from created intermediate sentences using the Edit Operation Extractor. As a result, we obtained the TS and PE datasets containing 146 and 86 primitive edit operation instances, respectively.[17] The TS dataset has about 1.7 times more primitive edit operations. This may lead the TS dataset to be more difficult to decompose. The distributions of edit operation types are shown in Section 5.5.

5.3. Evaluation Method

We evaluated our system in three metrics: exact match rate of the entire sequence of edit operations, and recall and precision of each edit operation. For the exact match rate, we evaluated whether the output sequence of edit operations exactly matches the one in the references. For recall and precision, we evaluated whether each edit operation in the output is included in the references. While multiple references generally exist, we used the one with the largest edit operation overlap.

5.4. Results

We compared three variants of our proposed method: alignment-based method + LLM-based method (A+L), alignment-based method (A), and LLM-based method (L).

Table 3 lists the results of each system for the TS and PE datasets. For both datasets, the A+L system was the best, achieving an exact match rate of 0.44 and 0.64 for the TS and PE datasets, respectively, followed by the A system. The L system was much worse in recall because this method generated one intermediate sentence per input sentence pair. Although we have used the dataset for TS to train the scorers, the result for the PE dataset was better than the one for TS.

[15] https://github.com/tntc-project/MTPEdocs

[16] https://mt-auto-minhon-mlt.ucri.jgn-x.jp/

[17] IDs of pairs of sentences used for the TS dataset are listed in Appendix C. The PE dataset is available via https://github.com/tntc-project/DecomposedMTPE.

System	Exact Match		Recall		Precision	
A+L	**0.44**	(22/50)	**0.61**	(90/146)	**0.66**	(90/137)
A	0.42	(21/50)	0.56	(82/146)	0.65	(82/126)
L	0.28	(14/50)	0.28	(41/146)	0.62	(41/66)

(a) TS

System	Exact Match		Recall		Precision	
A+L	**0.64**	(32/50)	**0.59**	(51/86)	0.71	(51/72)
A	**0.64**	(32/50)	0.57	(49/86)	**0.72**	(49/68)
L	0.48	(24/50)	0.31	(27/86)	0.60	(27/45)

(b) PE

Table 3: Results of each system (A: alignment-based method, L: LLM-based method) for text simplification dataset (TS) and machine translation post-editing dataset (PE).

TER	Exact Match	Recall		Precision	
(0, 0.1]	0.4	0.52	(11/21)	0.73	(11/15)
(0.1, 0.2]	0.8	0.84	(16/19)	0.80	(16/20)
(0.2, 0.3]	0.6	0.70	(19/27)	0.76	(19/25)
(0.3, 0.4]	0.4	0.74	(25/34)	0.64	(25/39)
(0.4, 0.5]	0.0	0.42	(19/45)	0.50	(19/38)

(a) TS

TER	Exact Match	Recall		Precision	
(0, 0.1]	0.6	0.53	(8/15)	0.62	(8/12)
(0.1, 0.2]	0.6	0.71	(12/17)	0.63	(12/19)
(0.2, 0.3]	0.7	0.53	(10/19)	0.83	(10/12)
(0.3, 0.4]	0.6	0.56	(9/16)	0.64	(9/14)
(0.4, 0.5]	0.7	0.63	(12/19)	0.80	(12/15)

(b) PE

Table 4: Breakdown results of alignment- and LLM-based method (A+L).

Table 4 lists the breakdown results of the A+L system for both datasets. For the TS dataset, the higher the TER was, the lower the precision and exact match rate were. In contrast, for the PE dataset, the precision and exact match rate did not decrease along with the increase of TER.

Table 5 shows the relationship between the number of edit operations, edit distance, and exact match rate when using the A+L system. For both datasets, the more the number of primitive edit operations or edit distance was, the lower the exact match rate was. Nearly half of the sentence pairs with three or fewer primitive edit operations were successfully decomposed.

5.5. Types of Identified Edit Operations

To investigate the characteristics of the A+L system, we classified the obtained edit operations. First, as reference data, we classified 146 and 86 edit operations included in the TS and PE datasets into one of the following edit operation types: substitution, insertion, deletion, and shift. Then, we classified the obtained edit operations that were included in the references: 90 and 51 edit operations corresponding to the TS and PE datasets, respectively. Finally, we calculated the recall of the edit operations.[18]

Table 6 shows the result. The distribution of the reference edit operation types for the TS dataset is different from that for the PE dataset. Specifically, in the TS dataset, insertion (25 cases) and deletion (31 cases) account for a larger portion than in the PE dataset. This may reflect the difference between the task in which original information may greatly change, such as text simplification, and the task in which original information does not change much, such as machine translation post-editing.

In the TS dataset, the highest recall was achieved for substitution, followed in order by deletion and insertion. In contrast, in the PE dataset, the highest recall was achieved for insertion, followed in order by deletion and substitution. This inconsistent trend across datasets suggests that the types of edit operations that the A+L system can correctly obtain depend on the editing task.

5.6. Analyses of System's Behavior

First, to understand whether Lattice Generation or Path Search was the cause of the failure of the decomposition, we conducted an error analysis. We classified the outputs of the best system (A+L) that were not exactly matched with the gold reference. We labeled the lattices that included at least one path in the gold references as "Path Search." Otherwise, we labeled the lattices as "Lattice Generation." We further classified the latter into one of the following three error types.

Over Decomposition: All of the paths had more nodes than a reference path

Under Decomposition: All of the paths had less nodes than a reference path

Both: Otherwise

Table 7 shows the result of the error analysis. For both datasets, most errors (23 out of 28 in the TS dataset, and 18 out of 18 in the PE dataset) happened when generating a lattice. These results show that our system did both over- and under-decomposition, implying that only changing the threshold for the Edit Operation Scorer does not help improve decomposition ability.

[18]The obtained edit operations that were not included in the references were all error cases and it is by nature difficult to classify them into edit operations. Therefore, we do not report precision.

# of Edits	Edit Distance 2	3	4	≥5	Total
1	5/9	-	2/2	-	7/11
2	3/4	4/4	-	0/2	7/10
3	1/3	2/4	1/1	1/5	5/13
4	-	-	0/1	2/11	2/12
≥5	-	-	-	1/4	1/4
Total	9/16	6/8	3/4	4/22	22/50

(a) TS

# of Edits	Edit Distance 2	3	4	≥5	Total
1	12/13	5/6	2/3	2/3	21/25
2	6/8	2/6	1/1	0/3	9/18
3	0/1	1/1	-	1/3	2/5
4	-	-	-	0/1	0/1
≥5	-	-	-	0/1	0/1
Total	18/22	8/13	3/4	3/11	32/50

(b) PE

Table 5: Relationship between the number of edit operations, edit distance, and exact match rate when using alignment- and LLM-based method (A+L).

	TS		PE	
Substitution	0.67	(60/90)	0.56	(35/63)
Insertion	0.40	(10/25)	0.78	(7/9)
Deletion	0.65	(20/31)	0.69	(9/13)
Shift	-	(0/0)	0	(0/1)

Table 6: Recall of edit operation.

Process	Error Type	TS	PE
Path Search	Incorrect Search	5	0
Lattice Generation		23	18
	Over Decomposition	(9)	(7)
	Under Decomposition	(10)	(10)
	Both	(4)	(1)

Table 7: Results of the error analysis for alingment- and LLM-based method.

We also analyzed successful cases, 22 and 32 lattices for TS and PE, respectively, to understand how well our proposed method generated lattices. We classified the lattices of the best systems into one of the following three categories.

Exact Match: Identical to the reference lattice
Included: Included in the reference lattice
Partial Match: Otherwise

For the first two categories, even if the system randomly chooses a path, it should always be correct.

Table 8 shows the result. Over half of the lattices had identical sets of paths in both datasets. Among Exact Match, more than half of the lattices had only one primitive edit operation: 7 out of 14 and 21 out of 26 for TS and PE datasets, respectively. Partial Match tended to happen when the number of primitive edit operations was more than two. The TS dataset had eight such pairs, and our system generated only one Exact Match lattice for the pairs, and the rest were Partial Match. Path Search succeeded 58% (7/12) of lattices that need to be searched.[19] This result suggests our system

[19]The number of Partial Match (Table 8) is divided by the sum of the number of Partial Match and Incorrect Search (Table 7).

Category	TS	PE
Exact Match	14	26
Included	1	5
Partial Match	7	1

Table 8: Analysis of correct outputs in terms of overlap between generated and reference lattices.

can correctly select a path to some extent if the lattice includes a correct path. However, there is potential for improvement.

6. Conclusions and Outlook

Aiming at helping analyses of text editing systems, we attempted to automate the decomposition of text editing examples into primitive edit operations. Our system consists of lattice generation with a phrase aligner and LLM, and path search. We tested three variants of our system on text simplification and machine translation post-editing examples. As a result, our best system perfectly decomposed 44% of examples in the text simplification dataset and 64% of examples in the post-editing dataset. Our system decomposed the sentence pairs with a small number of primitive edit operations well. While the performance of our system is not adequate for fully automated use, it can be used for supporting manual analysis.

As for the direction of improvement, our detailed analyses of the results show that the performance of lattice generation should be improved, including the expansion of training data for the Edit Operation Scorer, because most failures were caused by this. Also, improving the Primitiveness Scorer may lead to better results.

7. Limitations

Language Dependency: We have only experimented in English. Although our proposed method described in Section 4 is language-independent, its implementation depends on the phrase aligner for English and the LLM largely trained on English text data. Therefore, the results presented in this paper do not guarantee the accuracy of our method in other languages.

Coverage of Editing Task and Dataset: The coverage of editing tasks and used datasets is also limited. Our experiments have been conducted with datasets of two different tasks: text simplification and machine translation post-editing. The applicability of our methods to other tasks and datasets remains to be examined. It should also be noted that, in our experiment, we have used editing examples created by human writers or translators. Our long-term objective is to evaluate editing examples produced by systems, which often differ from the human-edited ones because they may contain various errors. How well our proposed method can be applied to such machine outputs is still unclear.

Size of Test Dataset: For our evaluation experiment, as described in Section 5.2, we have used a total of 232 primitive edit operations obtained from 100 pairs of source and edited sentences. Although the size of the datasets is not large and further investigation will be needed, it is comparable to other analytic evaluation experiments (Xu et al., 2015; van Miltenburg et al., 2021; Yamaguchi et al., 2023), in which from about 50 to a few hundred cases have been manually analyzed.

Data Availability: While we released the post-editing dataset with manually decomposed gold references, the text simplification dataset created from Newsela-Auto cannot be made publicly available due to copyright restrictions. To help replicate our experiments, we disclosed the list of sentence pair IDs of Newsela-Auto in Appendix C.

8. Acknowledgements

We are grateful to Newsela for sharing the data. This work was supported by JSPS KAKENHI Grant Numbers JP19H05660 and JP23H03689.

9. Bibliographical References

Yuki Arase and Jun'ichi Tsujii. 2020. Compositional phrase alignment and beyond. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1611–1623.

Rahul Bhagat and Eduard Hovy. 2013. Squibs: What is a paraphrase? *Computational Linguistics*, 39(3):463–472.

Pushpak Bhattacharyya, Rajen Chatterjee, Markus Freitag, Diptesh Kanojia, Matteo Negri, and Marco Turchi. 2022. Findings of the WMT 2022 Shared Task on Automatic Post-Editing. In *Proceedings of the 7th Conference on Machine Translation (WMT)*, pages 109–117.

Christopher Bryant, Mariano Felice, and Ted Briscoe. 2017. Automatic annotation and evaluation of error types for grammatical error correction. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 793–805.

Nadia Burkart and Marco F. Huber. 2021. A survey on the explainability of supervised machine learning. *Journal of Artificial Intelligence Research (JAIR)*, 70:245–317.

Gonçalo M. Correia and André F. T. Martins. 2019. A simple and effective approach to automatic post-editing with transfer learning. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 3050–3056.

Marina Danilevsky, Kun Qian, Ranit Aharonov, Yannis Katsis, Ban Kawas, and Prithviraj Sen. 2020. A survey of the state of explainable AI for natural language processing. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing (AACL-IJCNLP)*, pages 447–459.

Yue Dong, Zichao Li, Mehdi Rezagholizadeh, and Jackie Chi Kit Cheung. 2019. EditNTS: An neural programmer-interpreter model for sentence simplification through explicit editing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 3393–3402.

Yutao Feng, Jipeng Qiang, Yun Li, Yunhao Yuan, and Yi Zhu. 2023. Sentence simplification via large language models. *CoRR*, abs/2302.11957.

Katja Filippova, Enrique Alfonseca, Carlos A. Colmenares, Lukasz Kaiser, and Oriol Vinyals. 2015. Sentence compression by deletion with LSTMs. In *Proceedings of the 2015 Conference*

on Empirical Methods in Natural Language Processing (EMNLP), pages 360–368.

Atsushi Fujita, Kikuko Tanabe, Chiho Toyoshima, Mayuka Yamamoto, Kyo Kageura, and Anthony Hartley. 2017. Consistent classification of translation revisions: A case study of English-Japanese student translations. In *Proceedings of the 11th Linguistic Annotation Workshop (LAW)*, pages 57–66.

Shaohan Huang, Yu Wu, Furu Wei, and Ming Zhou. 2018. Text morphing. *CoRR*, abs/1810.00341.

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

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2021. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547.

Marcin Junczys-Dowmunt, Roman Grundkiewicz, Shubha Guha, and Kenneth Heafield. 2018. Approaching neural grammatical error correction as a low-resource machine translation task. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 595–606.

Venelin Kovatchev, M. Antònia Martí, and Maria Salamó. 2018. ETPC - A paraphrase identification corpus annotated with extended paraphrase typology and negation. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC)*, pages 1384–1392.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2021. Retrieval-augmented generation for knowledge-intensive nlp tasks. *CoRR*, abs/2005.11401.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Arle Lommel, Maja Popović, and Aljoscha Burchardt. 2014. Assessing inter-annotator agreement for translation error annotation. In *Proceedings of the LREC MTE Workshop on Automatic and Manual Metrics for Operational Translation Evaluation*, pages 31–37.

Rei Miyata and Atsushi Fujita. 2021. Understanding pre-editing for black-box neural machine translation. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 1539–1550.

Sergiu Nisioi, Sanja Štajner, Simone Paolo Ponzetto, and Liviu P. Dinu. 2017. Exploring neural text simplification models. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL) Short Papers*, pages 85–91.

OpenAI. 2023. GPT-4 technical report. *CoRR*, abs/2303.08774.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 311–318.

Vikas Raunak, Amr Sharaf, Yiren Wang, Hany Awadalla, and Arul Menezes. 2023. Leveraging GPT-4 for automatic translation post-editing. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 12009–12024.

Ananya B. Sai, Akash Kumar Mohankumar, and Mitesh M. Khapra. 2022. A survey of evaluation metrics used for NLG systems. *ACM Computing Surveys*, 55(2).

Thibault Sellam, Dipanjan Das, and Ankur Parikh. 2020. BLEURT: Learning robust metrics for text generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 7881–7892.

Matthew Shardlow. 2014. A survey of automated text simplification. *International Journal of Advanced Computer Science and Applications (IJACSA), Special Issue on Natural Language Processing 2014*, 4(1):58–70.

Matthew Snover, Bonnie Dorr, Rich Schwartz, Linnea Micciulla, and John Makhoul. 2006. A study of translation edit rate with targeted human annotation. In *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas: Technical Papers (AMTA)*, pages 223–231.

Felix Stahlberg and Shankar Kumar. 2020. Seq2Edits: Sequence transduction using span-level edit operations. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5147–5159.

Renliang Sun, Zhe Lin, and Xiaojun Wan. 2020. On the helpfulness of document context to sentence simplification. In *Proceedings of the 28th International Conference on Computational Linguistics (COLING)*, pages 1411–1423.

Midori Tatsumi. 2010. *Post-Editing Machine Translated Text in A Commercial Setting: Observation and Statistical Analysis*. Ph.D. thesis, Dublin City University.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023a. LLaMA: Open and efficient foundation language models. *CoRR*, abs/2302.13971.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023b. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288.

Emiel van Miltenburg, Miruna Clinciu, Ondřej Dušek, Dimitra Gkatzia, Stephanie Inglis, Leo Leppänen, Saad Mahamood, Emma Manning, Stephanie Schoch, Craig Thomson, and Luou Wen. 2021. Underreporting of errors in NLG output, and what to do about it. In *Proceedings of the 14th International Conference on Natural Language Generation (INLG)*, pages 140–153.

Marta Vila, Maria Antònia Martí, and Horacio Rodríguez. 2014. Is this a paraphrase? What kind? Paraphrase boundaries and typology. *Open Journal of Modern Linguistics (OJML)*, 4:205–218.

Wei Xu, Chris Callison-Burch, and Courtney Napoles. 2015. Problems in current text simplification research: New data can help. *Transactions of the Association for Computational Linguistics (TACL)*, 3:283–297.

Wei Xu, Courtney Napoles, Ellie Pavlick, Quanze Chen, and Chris Callison-Burch. 2016. Optimizing statistical machine translation for text simplification. *Transactions of the Association for Computational Linguistics (TACL)*, 4:401–415.

Daichi Yamaguchi, Rei Miyata, Sayuka Shimada, and Satoshi Sato. 2023. Gauging the gap between human and machine text simplification through analytical evaluation of simplification strategies and errors. In *Findings of the Association for Computational Linguistics: EACL 2023*, pages 359–375.

Mayuka Yamamoto and Masaru Yamada. 2022. Translation strategies for English-to-Japanese translation. In Rei Miyata, Masaru Yamada, and Kyo Kageura, editors, *Metalanguages for Dissecting Translation Processes: Theoretical Development and Practical Applications*, pages 80–91. Routledge, London.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. 2020. BERTScore: Evaluating text generation with BERT. In *International Conference on Learning Representations (ICLR)*.

Xingxing Zhang and Mirella Lapata. 2017. Sentence simplification with deep reinforcement learning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 584–594.

A. Experiment to Determine LLM Method

To determine the LLM method used in the experiment in Section 5, we conducted a preliminary experiment. The following items were determined.

1. LLM model
2. Number of few-shot examples
3. Seed for LLM

As to 1, we decided to choose models that ensure the reproducibility of outputs and excluded private models such as GPT-4 (OpenAI, 2023). To accelerate execution time, a lightweight model is desirable if the capability to generate an intermediate sentence is comparable to larger models. As to 2, in terms of execution time, the smaller number is desirable. As to 3, it is well known that the output of LLM can change drastically depending on the seed. Therefore, we investigated a suitable seed to generate an intermediate sentence.

A.1. Implementation Details

For reproducibility, we selected Llama.cpp and Llama-2.cpp, which are four-bit quantized models of Llama 1 and Llama 2, respectively. Llama 2 was trained with more data than Llama 1 and is generally better than Llama 1 in NLP tasks (Touvron et al., 2023b). For Llama 1, we used 7B,[20] 13B,[21] 30B,[22] and 65B[23] models. For Llama 2, we used 7B,[24] 13B,[25] and 70B[26] models. For each model, we evaluated it with the number of few-shot examples *k* from zero to five and a seed from 42 to 46. The other settings were the same as explained in Section 5.1.

A.2. Evaluation Data and Method

With the Newsela-Auto corpus test set, we created another dataset in the same way as explained in Section 5.2. This dataset contains all intermediate sentences for a pair of source and edited sentences. Although this dataset consists of 50 pairs of sentences, we used 35 pairs with more than one edit operation.[27]

Given a pair of source and edited sentences, LLM generated one sentence. Then, we automatically judged whether the output is correct, i.e., included in the references.

[20] https://huggingface.co/TheBloke/LLaMA-7b-GGUF

[21] https://huggingface.co/TheBloke/LLaMA-13b-GGUF

[22] https://huggingface.co/TheBloke/LLaMA-30b-GGUF

[23] https://huggingface.co/TheBloke/LLaMA-65b-GGUF

[24] https://huggingface.co/TheBloke/Llama-2-7B-GGUF

[25] https://huggingface.co/TheBloke/Llama-2-13B-GGUF

[26] https://huggingface.co/TheBloke/Llama-2-70B-GGUF

[27] This is because an intermediate sentence does not exist for the pair with only one edit operation.

A.3. Results

Table 9 presents the average number of correct intermediate sentences that each model generated over five seeds. The setting in which the average was highest was when using Llama-2 70B given five few-shot examples. Under this setting, LLM generated at the maximum 19 correct intermediate sentences, where the seed was 42. Therefore, we adopted the following settings.

1. LLM model: Llama-2.cpp 70B
2. Number of few-shot examples: 5
3. Seed for LLM: 42

The capability of Llama.cpp and Llama-2.cpp to generate an intermediate sentence increased along with the model size. In some settings, giving more few-shot examples did not contribute to increasing the capability of intermediate sentence generation. In the zero-shot setting, the average numbers of correct intermediate sentences did not vary among model sizes and were all below three. In the one-shot setting, however, the larger the model size was, the more improvement in the generation capability was observed. Compared to the same size of Llama.cpp and Llama-2.cpp in the best setting, Llama.cpp was better than Llama-2.cpp for 7B, while Llama.cpp was worse than Llama-2.cpp for 13B. Thus, although the best performance was achieved by using Llama 2, all model sizes of Llama 2 are not always better than Llama 1 for different settings.

The best seed to generate an intermediate sentence was inconsistent across the number of few-shot examples and LLM models. This result suggests that there is no generally good seed to generate an intermediate sentence with Llama.cpp or Llama-2.cpp.

B. Ablation Study

To verify the effectiveness of the data selection method to create the database used in the LLM-based method in Section 5.1, we conducted an ablation study.

B.1. Settings

The following criteria were used to select triplets of (X_s, X_{inter}, X_t).

1. $sim(X_s, X_{inter}) > sim(X_s, X_t)$
2. $TER(X_s, X_{inter}) \leq TER(X_s, X_t) \leq 0.50$
3. $diff(X_s, X_{inter}) \subseteq diff(X_s, X_t)$ and $diff(X_t, X_{inter}) \subseteq diff(X_t, X_s)$

LLM	# of Parameters	# of Few-shot Examples					
		0	1	2	3	4	5
Llama 1 (Llama.cpp)	7B	1.0	**5.0**	2.6	3.0	3.6	3.6
	13B	2.0	4.8	4.2	4.4	3.6	**5.4**
	30B	3.0	5.8	7.4	**10.2**	9.8	9.0
	65B	2.8	9.8	10.6	**11.8**	10.8	11.0
Llama 2 (Llama-2.cpp)	7B	1.2	**3.4**	2.4	1.4	2.8	2.4
	13B	2.0	4.4	5.8	**6.6**	6.4	**6.6**
	70B	2.6	9.8	12.0	11.8	12.0	**13.0**

Table 9: The average number of correct intermediate sentences that each model generated over five seeds.

Data Selection Criteria	# of Entries	Seed					Avg.
		42	43	44	45	46	
1+2+3	8,978	**19**	13	15	10	**8**	**13.0**
1+2	10,304	18	13	14	7	6	11.6
1+3	52,948	15	14	**18**	9	**8**	12.8
2+3	9,323	16	**15**	14	**11**	**8**	12.8
1	70,745	13	11	13	7	7	10.2
2	10,876	15	12	12	8	**8**	11.0
3	56,894	12	13	15	5	7	10.4
No data selection	84,784	9	10	8	4	7	7.6

Table 10: The number of correct intermediate sentences for each condition and seed.

where *sim*$(\cdot, \cdot)$ is BERTScore and *diff*$(\cdot, \cdot)$ is a function that returns a set of tokens in the first argument sentence differing from those in the second argument sentence. Specifically, the third criterion prunes redundant edit operations like "A $\Rightarrow$ B" and then "B $\Rightarrow$ A."

We verified the effectiveness of the way to create the database by excluding one or more criteria and generating an intermediate sentence using it. For the LLM model, we used Llama-2.cpp 70B with the seed of 42 to 46 and gave five few-shot examples. The evaluation data and method were the same as explained in Section A.2.

B.2. Results

Table 10 presents the number of correct intermediate sentences for each condition, i.e., combination of data selection criteria, and seed. When excluding one of the three criteria, the average numbers of correct intermediate sentences decreased in all conditions. In particular, excluding criterion 3 consistently degraded the performance across different seeds.

When using only one of the three criteria, criterion 2 achieved the best result, and the difference between the conditions using criteria 1 and 3 was small. The number of entries in the database using only criterion 2 (10,876 entries) is almost the same as that using criteria 1 and 2 (10,304 entries). This means that most (X_s, X_{inter}, X_t) triplets following criterion 2 also followed criterion 1.

The average number of correct cases for the condition without all criteria was much less than that with all criteria, suggesting our data selection method to create the database was effective.

C. IDs of Newsela-Auto Used for TS Dataset

Table 11 lists sentence IDs of Newsela-Auto comprising the TS dataset used in the experiment explained in Section 5 and the number of edit operations needed to convert a source sentence into an edited sentence.

D. Algorithm to Generate a Lattice

Algorithm 1 shows how to generate a lattice.

No.	Source Sentence ID	Edited Sentence ID	# of Edits
1	lead-violence.en-2-24-1	lead-violence.en-3-24-1	2
2	fourthgrade-sports.en-1-27-0	fourthgrade-sports.en-3-20-0	3
3	malala-shooters.en-1-6-0	malala-shooters.en-2-6-0	1
4	beef-grassfed.en-0-22-0	beef-grassfed.en-1-28-0	3
5	samesexmarriage-alabama.en-0-11-1	samesexmarriage-alabama.en-1-16-1	2
6	liberia-ebolafree.en-1-16-1	liberia-ebolafree.en-2-16-1	1
7	climate-surfing.en-1-19-0	climate-surfing.en-3-18-0	1
8	naacp-ethnicity.en-2-15-0	naacp-ethnicity.en-3-16-0	3
9	oregon-plague.en-0-9-2	oregon-plague.en-1-13-2	2
10	syrian-swimmers.en-1-17-0	syrian-swimmers.en-3-17-0	3
11	kraft-yellowdye.en-1-8-2	kraft-yellowdye.en-3-8-2	2
12	gilgamesh-lines.en-3-7-0	gilgamesh-lines.en-4-4-0	1
13	johnnash-obit.en-0-19-0	johnnash-obit.en-2-16-0	3
14	samesexmarriage-alabama.en-3-13-1	samesexmarriage-alabama.en-4-15-1	1
15	fake-leonardo.en-1-4-2	fake-leonardo.en-2-4-3	1
16	challenger-anniversary.en-2-18-1	challenger-anniversary.en-3-20-1	3
17	jamestown-discovery.en-1-4-0	jamestown-discovery.en-2-4-0	1
18	iraq-books.en-1-25-0	iraq-books.en-2-21-0	4
19	projectsyndicate-labor.en-3-8-3	projectsyndicate-labor.en-3-8-3 projectsyndicate-labor.en-4-8-2	1
20	isisattack-procon.en-2-40-1	isisattack-procon.en-4-40-2	2
21	samesexmarriage-alabama.en-1-3-0	samesexmarriage-alabama.en-3-4-0	3
22	mars-liquidwater.en-0-5-1	mars-liquidwater.en-2-4-3	4
23	fisherman-found.en-0-12-0	fisherman-found.en-4-12-0	4
24	scienceart-institute.en-1-25-0	scienceart-institute.en-4-23-0	3
25	poverty-braindevelopment.en-3-8-2	poverty-braindevelopment.en-4-8-2	3
26	femaleengineer-cars.en-3-3-0	femaleengineer-cars.en-4-3-0	2
27	karate-80yo.en-0-15-2	karate-80yo.en-4-10-1	1
28	rejectrefugees-procon.en-2-30-3	rejectrefugees-procon.en-3-34-2	1
29	slaveship-artifacts.en-1-8-0	slaveship-artifacts.en-3-7-0	4
30	museums-selfies.en-3-4-0	museums-selfies.en-4-2-0	2
31	nobel-chemistry.en-0-15-1	nobel-chemistry.en-2-18-1	4
32	obama-policetaskforce.en-3-20-0	obama-policetaskforce.en-4-20-0	2
33	china-kitemakers.en-0-21-0	china-kitemakers.en-3-20-0	6
34	climate-surfing.en-0-4-1	climate-surfing.en-2-5-2	4
35	arthropod-fossil.en-0-18-0	arthropod-fossil.en-2-22-0	4
36	student-testing.en-0-6-0	student-testing.en-1-5-0	4
37	fourthgrade-sports.en-3-19-1	fourthgrade-sports.en-4-18-1	3
38	hispanics-education.en-3-2-2	hispanics-education.en-4-2-3	2
39	quake-doomsday.en-0-3-0	quake-doomsday.en-2-3-0	4
40	sledding-ban.en-1-1-1	sledding-ban.en-2-1-1	1
41	clinton-presannounce.en-2-17-0	clinton-presannounce.en-3-14-0	4
42	bat-survival.en-3-2-2	bat-survival.en-4-6-1	3
43	millennials-justicesurvey.en-1-14-0	millennials-justicesurvey.en-4-18-0	3
44	sugar-label.en-0-1-0	sugar-label.en-2-1-0	4
45	kraft-yellowdye.en-1-12-1	kraft-yellowdye.en-3-10-1	6
46	lead-violence.en-0-14-1	lead-violence.en-2-18-1	8
47	shakespeare-analysis.en-2-9-5	shakespeare-analysis.en-4-8-5	3
48	bat-survival.en-0-12-0	bat-survival.en-2-13-0	8
49	zika-virus.en-3-5-0	zika-virus.en-4-3-0	2
50	gmo-salmon.en-0-15-0	gmo-salmon.en-3-16-0	4

Table 11: Sentence IDs of Newsela-Auto comprising the TS dataset and the number of edit operations needed to convert a source sentence into an edited sentence.

Algorithm 1 Generate Lattice

Require: **input:** SOURCE SENTENCE *src*
Require: **input:** EDITED SENTENCE *edt*
Require: **input:** THRESHOLD OF GRAMMATICALITY AND UNSWERVINGNESS $threshold_{gu}$
openList $\leftarrow$ [(*src*, *edt*)] // initialize a openList
closedList $\leftarrow$ [] // initialize a closedList
while *openList.length* > 0 **do**
  $(s, t) \leftarrow$ *openList.pop*()
  closedList.push((s, t))
  editNum $\leftarrow$ TranslationEditNum(s,t) // calculate the number of edits by TER
  if *editNum* $== 1$ **then**
    CONTINUE // cannot make any intermediate sentences
  end if
  inters $\leftarrow$ GenerateInter(s,t) // generate candidates of intermediate sentences
  $sim_{s,t} \leftarrow$ CalSim(s,t) // calculate similarity by arbitrary text similarity metrics
  for *inter* $\in$ *inters* **do**
    sntsPriorS $\leftarrow$ GetPrecedingSnts(s,*openList*,*closedList*) // get all sentences preceding to s
    sntsAfterT $\leftarrow$ GetSucceedingSnts(t,*openList*,*closedList*) // get all sentences succeeding to t
    if *inter* $\in$ *sntsPriorS* or *inter* $\in$ *sntsAfterT* **then**
      CONTINUE // do not generate a cycle
    end if
    $sim_{s,inter} \leftarrow$ CalSim(s,*inter*)
    $sim_{inter,t} \leftarrow$ CalSim(*inter*,t)
    $prob_{gu} \leftarrow$ CalProb(s,*inter*,t) // calculate grammaticality and unswervingness of $(s,$ *inter*$)$
    if $sim_{s,t} < sim_{s,inter}$ and $sim_{s,t} < sim_{inter,t}$ and $prob_{gu} > threshold_{gu}$ **then**
      if (s,*inter*) $\notin$ *openList* and (s,*inter*) $\notin$ *closedList* **then**
        openList.push($(s,$ *inter*$)$)
      end if
      if (*inter*,t) $\notin$ *openList* and (*inter*,t) $\notin$ *closedList* **then**
        openList.push($($*inter*$, t)$)
      end if
    end if
  end for
end while
lattice $\leftarrow$ []
for $(s, t) \in$ *closedList* **do**
  $A \leftarrow \{a | (s, a) \in$ *closedList*$\}$
  if $\forall a \in A; (a, t) \notin$ *closedList* **then**
    lattice.push((s, t))
  end if
end for
return *lattice*