Abstract

Most existing sequence generation models produce outputs in one pass, usually left-to-right. However, this is in contrast with a more natural approach that humans use in generating content; iterative refinement and editing. Recent work has introduced edit-based models for various tasks (such as neural machine translation and text style transfer), but these generally model a single edit step. In this work, we propose modeling editing processes, modeling the whole process of iteratively generating sequences. We form a conceptual framework to describe the likelihood of multi-step edits, and describe neural models that can learn a generative model of sequences based on these multi-step edits. We introduce baseline results and metrics on this task, finding that modeling editing processes improves performance on a variety of axes on both our proposed task and related downstream tasks compared to previous single-step models of edits.

1 Introduction

Revising and editing are a central part of the human creative workflow, with most original content (e.g. art, books, articles, source code) being developed not in a single iteration, but in many iterations with each more refined than the last. How can we model these editing processes from inception to completion? In this paper, we attempt to provide a first answer to this question, specifically focusing on generation of sequential data such as natural language documents or source code.

Most current work on language generation tasks such as machine translation (Vaswani et al., 2017), language modeling (Baevski and Auli, 2018), or summarization (See et al., 2017) generates the target sentence or document in a single pass (usually from left to right). There has been a reasonable amount of work that can generate edits to existing sequences for the purposes of post-editing, grammatical error correction (Omelianchuk et al., 2020), text style transfer (Mallinson et al., 2020; Malmi et al., 2020; Reid and Zhong, 2021), sentence fusion (Malmi et al., 2019), or machine translation (Gu et al., 2019). However, these works all 1) model only a single editing step and 2) do not fully define a model of incrementally editing a document from a blank slate to the final text, and thus do not stand in for the one-pass generative models of sequences described above.

In this context, we propose the task of modeling editing processes, in which we look to explicitly model the likelihood of the entire process of revising a document to a polished form. In particular, and in contrast to previous works on modeling edits, we hypothesize that in order to edit more accurately, instead of simply learning to predict the next revision given the current revision, we should have context of multiple previous revisions when deciding when and how to edit the document next. Given the novelty of framing generation problems in this way, this paper simultaneously 1) proposes both baseline and novel models for the task, 2) creates evaluation datasets that can be used to compare models, and 3) discusses intrinsic and extrinsic evaluation methodology.

The proposed multi-step editing model predicts
discrete edit operations (Levenshtein, 1966) to enable progressive refinement as shown in Figure 1, rather than framing sequence editing as a sequence to sequence task (Reid and Zhong, 2021; Fallings et al., 2021). In the figure, for each step of the editing process discrete operations (insert, replace, delete, keep) are predicted and then actions (such as generating a replaced span) are performed based on this. This has two benefits: 1) it allows the model to scale well with respect to input sequence length, and 2) allows us to make substantial changes with fewer actions (Grangier and Auli, 2018). We use these edit operations to condition a semi-autoregressive model that is able to insert and replace multiple spans at once. Combined with an encoder that is able to quickly specify which spans of text need to be changed and how, this allows for considerable changes to be made to the text (including insertion, deletion, re-ordering, and replacement) on a given span, and how likely the model thinks the generated text for a given span is. As we are modeling editing processes, and hypothesize that context from edits applied to the sequence are helpful, we propose a method for edit-aware sequence compression which can compress sequences into their edit operations and use relative edit positional embeddings to specify the position of edits relative to each other.

Given that the task of modeling natural editing processes in itself is novel, we collect new datasets to study this behavior: WIKIREVISIONS and CODEREVISIONS. These datasets, in the code and natural language domains respectively, cover over 2.5M and 2.3M natural sequential revisions. We also discuss evaluation methodology, describing a metric of edit perplexity (ePPL), the perplexity of generating an edit given the current state of a document, as well as applications to downstream tasks.

We train and evaluate our proposed models on these datasets and find that the proposed methodology of modeling the entire editing process, referencing previous edits while generating the next one, significantly improves both intrinsic and extrinsic performance baselines that model edits in isolation. In particular, our method reduces perplexity by up to 22.9% relative over a state-of-the-art editing baseline, and 11.3% relative over a version of our model that does not consider editing history. We also demonstrate the ability of the model to generate qualitatively natural edit sequences, and the utility of the learned representations on downstream tasks of commit message generation (Loyola et al., 2017) and edit intention classification (Yang et al., 2017a).

2 Problem Definition
Let $X = \{x_0, x_1, \ldots, x_N\}$ be a series of $N$ versions of a document, where the $i$th revised document is denoted by $x_i$. $x_0$ represents an initial state (generally the null string), and $x_N$ represents the current state of the edited document. The probability of this series of document versions occurring can be decomposed as

$$p(X) = \prod_{i=1}^{N} p(x_i|x_{i-1}),$$

where $x_{i-1} := x_0, \ldots, x_{i-1}$ (similarly below). The right hand side is the likelihood of the transformation of the previous document version $x_{i-1}$ to the current document version $x_i$ given the previous revision history $x_{<i}$. We refer to the likelihood of the whole revision process as the edit likelihood, and judge learned models based on their ability to achieve high edit likelihood on held-out data.

Note that standard generative models (specifically language models; LMs) calculate the probability of only the final version $p(x_N)$, whereas the proposed formulation calculates the probability of the entire sequence of document edits. It nonetheless could theoretically be used to calculate the final version’s likelihood by treating the editing process as latent and marginalizing over it

$$p(x_N) = \sum_{\tilde{X} \in \{x_N\}} p(\tilde{X}).$$

Thus, our formulation, in contrast to previous single-step models of edits (Yin et al., 2019a; Malmi et al., 2019; Reid and Zhong, 2021), can also be used to define a generative model over single documents. It is also worth noting that the final document likelihood is lower-bounded by the edit likelihood; i.e. $p(x_N) \geq p(X)$.

3 Modeling Editing Processes
In this section, we now describe our approach to actually modeling these sequences of edits through
(1) a decomposition of the modeling process into a sequential process of modeling edit operations then actual edits, and (2) neural model of modeling these operations and edits.

### 3.1 Modeling Operations and Operation-conditioned Edits

While the probability $p(x_i|x_{0:i-1})$ of the next document given all previous document versions could theoretically be modeled with a single neural sequence model, this is infeasible computationally (and likely infeasible from learning perspective as well). To simplify this problem, we employ the $n$-th order Markov assumption, assuming that the probability of the next document is conditioned only on the previous $n$ documents $p(x_i|x_{i-n:n})$. This probability could be modeled directly, and in fact in the case of $n = 1$ this becomes analogous to the single-step editing problem tackled by previous work (Yin et al., 2019a; Malmi et al., 2019; Reid and Zhong, 2021; Faltings et al., 2021). To our knowledge, no previous work has modeled natural editing processes with $n > 1$.

However, in the interest of both efficiency and efficacy, we take an alternative approach where we first predict a set of edit operations $e_i$, and then predict the next document version based on the previous documents and these edit operations:

$$p(x_i|x_{i-1:n}) = p(x_i, e_i|x_{i-1:n})$$

$$= p(x_i|e_i, x_{i-n:n})p(e_i|x_{i-n:n}).$$

The first approximation becomes an equality when the edit operations can be deterministically derived from $x_i$ and $x_{i-1}$, i.e. $p(e_i|x_i, x_{i-1}) = 1$, as is the case described below.

**Edit Operations.** We base the edit operations in $e$ on those calculated by the Levenshtein algorithm (Levenshtein, 1966), including token-level insertions, deletions, and substitutions. These are expressed as four operations insert, delete, keep, and replace denoted by \{INSERT, DELETE, KEEP, REPLACE\}. For multi-word insertions and replacements, e.g. a replacement of a contiguous span of words, we apply the the same REPLACE label to all tokens in this span. An example of each operation is shown in Figure 1.

**Decomposed Edit Likelihood.** We can then redefine our previous formulation of edit likelihood:

$$P(x_i^N) = \prod_{i=1}^N p(x_i|e_i, x_{i-n-1}^i)p(e_i|x_{i-n-1}^i).$$

and analogously define edit log-likelihood

$$\mathcal{L}_{xe} := \log P(x_i^N)$$

$$= \sum_{i=1}^N \log p(x_i|e_i, x_{i-n-1}^i) + \log p(e_i|x_{i-n-1}^i).$$

We can further decompose this into only the components corresponding to the edit operations $\mathcal{L}_{e} := \sum_{i=1}^N \log p(e_i|x_{i-1:n})$, or the operation-conditioned edits $\mathcal{L}_{xe} := \sum_{i=1}^N \log p(x_i|e_i, x_{i-n-1}^i)$, both of which we will utilize for devising evaluation metrics in Section 5.2 below.

### 3.2 EditPRO

In this section, we propose a model of multi-step editing processes, EditPRO, which is based on a semi-autoregressive edit-conditioned encoder-decoder model with a Transformer (Vaswani et al., 2017). The model (depicted in Figure 2) contains three main components: (1) an edit encoder, (2) an operation classifier and (3) an insertion-replacement decoder.

**Edit Encoder.** The encoder $f_{enc}$ takes in a document version $x_{i-1}$ and feeds it through multiple self-attention and feedforward layers (Vaswani et al., 2017) to produce contextual representations for each token. In the case that we perform variable-order edit modeling, we use cross-attention to feed in representations of previous edit steps. For models where $n > 1$, we feed in $n − 1$ additional edit sequences – we describe this process after describing our methods for edit sequence prediction.

**Edit Operation Prediction.** We use an autoregressive tagger, using a single Transformer layer with a causal attention mask, that models the probability of each edit in edit operation sequence $e = e_1^M$ from left to right, $p(e_j|e_{j-1}^{j-1})$. Notably, we also performed preliminary experiments with a tagger that predicts operations independently, but found it was heavily biased towards the KEEP operation as most words are kept in any single document revision, and thus did not produce coherent multi-word edit sequences when sampling sequences of edits.

**Generating Replacements and Insertions.** When editing, given our four Levenshtein operations (INSERT, REPLACE, KEEP, DELETE), two of them — INSERT and REPLACE — entail generation of new content conditioned on the current revision of the document. Given our predicted edit operations $e$, we propose a semi-autoregressive
model with a causal Transformer decoder that can decode multiple spans in parallel for efficiency purposes. Each edit span contains the following properties: it has a start index (denoted by \( s_{\text{start}} \)), end index (denoted by \( s_{\text{end}} \)), and an operation type (denoted by \( s_{\text{type}} \)). Note that these can be simply be extracted by looking at contiguous spans of a certain type in an edit (e.g. REPLACE for \textit{descended from} \( \rightarrow \) \textit{domesticated descendant of} in Figure 1).

We use a mean pooling operation to aggregate the contextual vectors produced by \( f_{\text{enc}}(x) \) into span representation \( \hat{x}_s \):

\[
\hat{x}_s = \frac{1}{s_{\text{end}} - s_{\text{start}}} \sum_{t=s_{\text{start}}}^{s_{\text{end}}} f_{\text{enc}}(x)_t \quad (7)
\]

We then update the span representation \( \hat{x}_s \) by taking the sum of the appropriate operation embedding for the span type and the current span representation and feed it to a multi-layer perceptron with an intermediate non-linearity: \( \hat{x}_s \leftarrow \text{MLP}(W_{\text{op}}(e)_s + \hat{x}_s) \), where \( W_{\text{op}} \) denotes an embedding matrix for each operation. \( \hat{x}_s \) is then used to initialize the \(<s>\) token for the decoder span to further condition the generative process.

**Encoding Edit History.** As we look to investigate variable order edit modeling over long sequences of text, we need a way to be able to represent edits in a way useful for predicting the next editing steps. Previous work (Yin et al., 2019b; Marrese-Taylor et al., 2021; Yao et al., 2021) has focused largely on learning a single-vector representation for edits which is compressed but limited in expressiveness. One the other hand, a perhaps more intuitive way taken from common Transformer-based (Vaswani et al., 2017) models would be to use cross-attention between all \( n \) previous documents, which is more expressive but prohibitively expensive when \( n \) is scaled upwards.

Instead, we make a compromise between the above approaches, leveraging predicted edits \( e_{i-1}^{t-1} \) to compress the sequence and their derived spans (as discussed above). Given each of these spans, we compute the edit-compressed sequence, composed of a sequence of vector representations with each vector representing a different span. For each span in each of the previous revisions in \( x_{i-1}^{t-1} \), we mean pool the encoder (pre-edit) and the decoder (post-edit) representations for that span. We then sum this representation with the operation representing its edit operation and feed it into an MLP. Once we have done this for each span, we sum a learned \textit{relative edit positional embedding}, where we learn an embedding matrix where each index in the matrix represents positions \( i - 1 \) to \( i - n \). We do this to specify the order of the previous edits. Finally, we compose these into a sequence and treat that as the “edit-compressed” sequence representation for that edit.

**Turning Pre-trained Encoder-Decoder Models into Editors.** Despite the fact that our model introduces both an edit prediction and a semi-autoregressive component, it is easy to finetune a pre-trained language model into an editor with our method as it uses vanilla Transformer layers as a backbone. We perform this by batching various spans and their conditioning variables together and training the model to adapt to decode these in parallel.

4 Data

While some datasets of edits exist (Faruqui et al., 2018; Marrese-Taylor et al., 2021), to our knowledge they only consider a single editing step, i.e. dealing with a document \( X = \{x_0, x_1\}, N = 1 \). As we propose learning to model multi-step
edits, we develop new datasets in both the code and natural language domains. In addition, previous datasets have only concerned themselves with atomic edits (Faruqui et al., 2018) which only occur at a small scale (usually sentence-level), and we instead look to model larger-scale edits as document level changes, which are more representative of the natural editing process.

4.1 WikiRevisions

In order to model the creative process for natural language text, we gather data from Wikipedia, which has extensive logs of the editing process that gave rise to Wikipedia articles, which have been used in a variety of previous works on single-step editing (Marrese-Taylor et al., 2019, 2021; Yang et al., 2017b; Faruqui et al., 2018). We collect data for each revision using dumps from English Wikipedia. Given that the dumps are provided in the XML format, we extract the text with beautifulsoup and remove wikitext (custom Wikipedia markup) with wikiextractor. With this sanitized data, we gather revision of each document in chronological order removing any metadata-based edits which were stripped as a result of the sanitization process. Now, with our sets of revisions we tokenize all text with the sentence-piece model used by Radford et al. (2018); Liu et al. (2019) for congruence with pre-trained models (see Section 3.2). We pre-compute Levenshtein operations using python-Levenshtein for use during training. In the case that an article exceeds 2000 tokens, we split the articles into its subsections and treat each subsection as an article (for the purpose of modeling editing processes). Dataset statistics are shown in Table 1. We note that there is a significant imbalance for the INSERT operation, this is because we define insertions to be applied to the token preceding the insertion (as shown in Figure 1), rather than applied to an entire span (as we do for the deletion, replacement, and keep operations).

Edit Summaries. When extracting each edit we keep the edit summary (akin to a commit message) supplied by the editor at time of editing. We then curate these comments and develop a dataset for usage on downstream tasks—for both edit summary generation (Loyola et al., 2017) and edit-summary-conditioned text editing (Faltings et al., 2021).

4.2 CodeRevisions

Another place where the incremental creative process is on display is in the creation of program source code. When building CodeRevisions, we scrape a total of 700 Python GitHub repositories using the MIT License with at least 1000 commits and 500 stars. We extract line-level patches from each repository’s commit history when forming our code-based corpus and progressively apply each patch and compute the token-level Levenshtein operations between each revision. Note that we also keep commit messages for each commit. For this dataset we operate on the file level. For each series of revisions, we precompute Levenshtein operations based on tokens derived from a sentencepiece (Kudo and Richardson, 2018) model with a 10k vocabulary. We also curate a dataset of revisions with commit messages as described in the previous subsection.

5 Experimental Setup

5.1 Baselines

We use the following baselines for our edit modeling task: (1) Seq2Seq, a standard sequence to sequence model trained to map $x_i \rightarrow x_{i+1}$, (2) LEWIS (Reid and Zhong, 2021), a state-of-the-art single-step editing model, which uses a separate encoder-only tagger and sequence-to-sequence generator setup during training, and (3) LaserTagger (Malmi et al., 2019), a simple editing model which learns how to apply a restricted set of edits.

5.2 Metrics

Many previous works on editing have focused on conditional language modeling tasks such as machine translation or editing source code based on commit messages (Malmi et al., 2019; Gu et al., 2019; Reid and Zhong, 2021), and thus have used non-likelihood based metrics such as BLEU or F1 score. However as we look to model the standard unconditional LM objective as shown in Equation 2, we instead adopt a small twist on standard perplexity-based metrics from language modeling as our main intrinsic metrics. Note that $|x|$ refers to the token count for the newly generated/inserted...
Dataset | Num. Edits | Avg. Len (Max/Min) | % Keep | % Insert | % Replace | % Delete  
--- | --- | --- | --- | --- | --- | ---  
WIKIREVISIONS | 2.5M | 333 (9992/1) | 82.4% | 0.1% | 8.7% | 8.8%  
CODEREVISIONS | 2.3M | 774 (9725/1) | 76.9% | 0.1% | 11.3% | 11.7%  

Table 1: Dataset statistics on CODEREVISIONS and WIKIREVISIONS, average length is measured by whitespace tokenization.

spans, and |e| refers to the number of edit operations:

**Edit Perplexity (ePPL)** is the exponent of the edit likelihood, divided by the length of both sequences, exp(-\frac{\text{loss}}{|x|+|e|}).

**Generation Perplexity (gPPL)** measures the likelihood of generating replaced or inserted spans when compared with the ground truth edit sequence as follows \(\text{exp}\left(-\frac{\text{loss}}{|e|}\right)\).

**Operation Perplexity (oPPL)** \(\text{exp}\left(-\frac{\text{loss}}{|e|}\right)\) is the likelihood of predicting a set of edit operations.

### 5.3 Training Setup

We train our models using the Transformer implementation using HuggingFace (Wolf et al., 2020). We tokenize data using SentencePiece (Kudo and Richardson, 2018), using the same vocabulary used in Lewis et al. (2019) for natural language, and a custom 10k vocabulary for code. We use the Transformer architecture with a hidden dimension of 768, feed-forward size of 3072, and 6 layers for both the encoder and decoder. We initialize all natural language models with BART (Lewis et al., 2019), and code models randomly. We set the maximum sequence length = 2048, using a batch size of 65K tokens distributed over 8 A100 GPUs.

### 5.4 Downstream tasks

In addition to assessing our proposed model’s generative capacity, we assess the quality of the learned representations on downstream tasks:

**Conditional Editing** We also continue training using the commit messages gathered during the cleaning process as a conditioning variable, essentially reformulating our \(p(x_i|x_{i-1},c)\) to \(p(x_i|x_{i-1},e,c)\) to add the additional conditional variable \(c\), which we set to be the edit summary or commit message in this setting. With our model, we append the comment to each document, delimiting with a separator token </s> as follows: DOCUMENT </s>

### 6 Results

#### 6.1 Edit Modeling

Results on edit modeling for both CODEREVISIONS and WIKIREVISIONS can be seen in Table 2, where we measure edit perplexity, operation perplexity, and generative perplexity. We first note that our model significantly outperforms LEWIS (Reid and Zhong, 2021) on WIKIREVISIONS, by 8.6 ePPL, suggesting that our model formulation is superior at this task. We believe that this stems from the fact that our model is trained to explicitly generate the newly added spans, and because it was used.

[1] We contacted the authors, but they were no longer able to release the folds they used in their K-fold cross-validation setup, hence our creation of splits.
Table 2: Results on Edit Modeling

<table>
<thead>
<tr>
<th>DATASET</th>
<th>Model</th>
<th>ePPL</th>
<th>gPPL</th>
<th>oPPL</th>
<th>DEL</th>
<th>KEEP</th>
<th>REPL</th>
<th>INS</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIKIREVISIONS</td>
<td>LEWIS</td>
<td>65.94</td>
<td>48.85</td>
<td>24.29</td>
<td>1.09</td>
<td>19.49</td>
<td>507.76</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Seq2Seq</td>
<td>—</td>
<td>41.81</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>LaserTagger</td>
<td>57.32</td>
<td>42.43</td>
<td>25.53</td>
<td>1.09</td>
<td>18.45</td>
<td>889.24</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EditPro (1-order)</td>
<td>53.91</td>
<td>39.87</td>
<td>20.70</td>
<td>1.13</td>
<td>15.49</td>
<td>376.15</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EditPro (2-order)</td>
<td>50.84</td>
<td>37.66</td>
<td>19.30</td>
<td>1.13</td>
<td>14.88</td>
<td>252.14</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EditPro (3-order)</td>
<td>34.22</td>
<td>28.02</td>
<td>125.21</td>
<td>1.05</td>
<td>10.38</td>
<td>544.57</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EditPro (1-order)</td>
<td>30.85</td>
<td>26.26</td>
<td>84.77</td>
<td>1.05</td>
<td>9.30</td>
<td>304.90</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EditPro (2-order)</td>
<td>29.47</td>
<td>25.37</td>
<td>75.19</td>
<td>1.06</td>
<td>8.16</td>
<td>441.42</td>
<td></td>
</tr>
<tr>
<td>CODEREVISIONS</td>
<td>EditPro (1-order)</td>
<td>34.22</td>
<td>28.02</td>
<td>125.21</td>
<td>1.05</td>
<td>10.38</td>
<td>544.57</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EditPro (2-order)</td>
<td>30.85</td>
<td>26.26</td>
<td>84.77</td>
<td>1.05</td>
<td>9.30</td>
<td>304.90</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EditPro (3-order)</td>
<td>29.47</td>
<td>25.37</td>
<td>75.19</td>
<td>1.06</td>
<td>8.16</td>
<td>441.42</td>
<td></td>
</tr>
</tbody>
</table>

directly connects the operation prediction and generation processes. We also remark that although Seq2Seq gPPL is slightly lower than our baseline, it tends to learn copy given the large portion of small edits and the lack of fine-grained control over edits enabled by edit operations. LaserTagger has the opposite issue: given that they select a set of “most common” phrases as the model was initially proposed for sentence fusion, despite the fine-grained control provided by edit operations, generative capability is limited. For EditPro we also take note that ePPL decreases when the order of context increases. In particular, we take note of the significant gain when introducing the notion of editing processes to the model in our 2-order setting (in contrast to a single edit step), with a reduction of 3.4 ePPL on natural language and 4.4 ePPL on source code.

We also note that while the gPPL consistently decreases as the number of orders increases, oPPL does not perform as consistently. Interestingly, we find that single-order models tend to be more confident with respect to keeping tokens (the overwhelmingly dominant edit operation), while other operations (deletions, replacements and insertions) are not predicted as well. In contrast, learning higher-order editing processes almost universally decreases the oPPL for non-KEEP operations, indicating the necessity of longer context to capture these rarer edit operations.

**Likely and Unlikely Edits.** We perform a qualitative analysis on a subsample 4,000 natural language edits, examining which edits are judged to be likely (or unlikely) and with respect to which metrics. We do this by identifying outlier values for each metric (significantly above or below the average) and further analysing these for unique properties.

As a result, we found that many of the edits with higher oPPL were spam and vandalism-type edits, as many of the edit operations have more of a random nature. However we notice that generative perplexity was much lower as these edits tend to be repetitive in nature with the same ngrams often being repeated for long spans. However, we notice that, irrespective of the number of orders, when editing reverted spam-like content, the oPPL for the REPLACE and DELETE operations are extremely low (on average 1.07 and 4.4 respectively). The importance of variable-order modeling was particularly evident these revisions where the gPPL in the single-order setting averages at 123.90 gPPL, however when using 2-orders we are able to attain 67.83 gPPL indicating that the edit-compressed sequences provide useful context about the previous revisions. We also notice that models are able to predict insertions (2.25 INSERT oPPL) significantly better when they come after the end of a sentence, representative of many insertions in Wikipedia. We also notice that outside of the above settings, models with extra context generally predict more likely edits supporting the notion of modeling edit processes compared to modeling changes individually.

6.2 Downstream Performance

Results on conditional edit generation, edit classification and edit-conditioned generation can be seen in Table 3. The findings generally follow the edit modeling results, with additional context improving performance further giving supporting evidence to modeling editing processes. Specifically, increasing context from single-order to 3-order improves commit message generation performance by 1.9 and 0.7 BLEU for both natural language and source code respectively. We also note that ePPL decreases similarly when we add natural language
Table 3: Results on Edit Generation (BLEU), Edit Classification (measured with micro-F1), and Conditional Edit Generation (measured Edit Perplexity = ePPL). The Δ symbol refers to the change between the model’s non-message conditioned version in Table 2.

6.3 Human Evaluation

We additionally perform a human evaluation using 3 Amazon Mechanical Turk crowdworkers to annotate 100 samples from our edit models at inference time. We compare our LEWIS, Seq2Seq, EditPRO (1-order) and EditPRO (2-order) models. Annotators are initially given 100 gold examples from the training set to in order to prime them on the form of natural edits. Annotators are then given samples to annotate on a scale of 1 to 5, where 1 corresponds to no edit and 5 corresponds to natural edit, where 2,3,4 represent somewhat natural, moderately natural, almost natural, respectively. We take the average of the three annotators’ scores to provide the following results: Seq2Seq (1.7), LEWIS (2.6), EditPRO(1-order; 2.5), EditPRO (2-order; 3.2). These results support our findings in Tables 2 and 3, demonstrating the perceptible impact of increased previous context when edit modeling (from the 2-order model), and the tendency to copy of Seq2Seq models faced with fine-grained editing.

7 Related Work

Learning Edit Representations. Previous work on learning properties inherent to editing has largely focused on learning distributed representations of edits. Yin et al. (2019b) proposed this task, using an attentional sequence-to-sequence model to learn representations. Marrese-Taylor et al. (2021) expands upon this approach, introducing an auxiliary loss and a variational setup. More recently, Yao et al. (2021) worked on using this approach for editing tree-structured data. However, in contrast with this work, these approaches only consider modeling single editing steps instead of the more general multi-step setting tackled here.

Semi-Autoregressive Sequence Generation. Work in machine translation has explored non-autoregressive methods that use an iterative generation process. This was first proposed by Lee et al. (2018) using latent variables and denoising autoencoding. Iterative refinement was then explored with conditional masked language models (CMLM; Ghazvininejad et al., 2019), simplifying previously proposed methods, by iteratively replacing predicted tokens with low confidence. Gu et al. (2019), introduced the Levenshtein Transformer, making this approach more flexible by introducing insertion and deletion operations. However, these methods have not yet considered
Europe is a continent located entirely in the Northern Hemisphere and mostly in the Eastern Hemisphere.

Table 4: Example generation when sampling with an edit model. We notice that the 2nd order model is able to perform text revision with a more holistic goal in mind. The Western South Eastman Islands are also located in Europe.

Other Editing-based Work. Other work on editing has included editing for sentence fusion (Malmi et al., 2019), in which one can perform minimal operations to join two sentences together grammatically. Furthermore, with text style transfer in which the difference between sentences in different styles (e.g. positive vs negative) can be relatively minimal (Reid and Zhong, 2021; Mallinson et al., 2020). Furthermore, Faltings et al. (2021) explored natural language conditioned editing as a means for controllable text generation using a T5-based (Raffel et al., 2020) sequence to sequence model. Also related to our work is text morphing (Huang et al., 2018), in which they look at an edit-based model to interoperate between two sentences. We also note that diffusion models (Sohl-Dickstein et al., 2015; Ho et al., 2020) can be formulated as a flavor of editing models, where the model learns to iteratively edit some representation of information in order to construct a final version of said representation. Editing-based lexically-constrained generation has been explored by (Miao et al., 2019) and Li et al. (2020) propose a search-based method for improving conditional text generation by way of heuristics — an approach similar in spirit to editing, however often directed towards a specific task (which benefits from specific constraints), rather than inherently modeling sequential editing processes. We also note work (Du et al., 2022a,b) looking at modeling edit intentions and modeling text revision with a more holistic goal in mind.

8 Conclusions

In this work, we proposed the novel task of modeling editing processes, in which we model the likelihood of documents by way of their natural editing processes. We develop new datasets and curate existing datasets for downstream tasks. We find that modeling editing processes is beneficial to this end, in contrast to modeling single-order edits, as has been done in much of previous literature. More broadly, we believe that tackling iterative refinement tasks in broader NLP may be beneficial given its resemblance to the natural generative and creative process.

In future work, we look to investigate methods for transferring pre-trained edit models to a wider range of NLP tasks. For instance, there are other widely acknowledged editing tasks in NLP such as grammatical error correction (Dahlmeier and Ng, 2012) or style transfer (Shen et al., 2017). The technical challenge in applying the models presented here to these tasks lies in the lack of datasets annotated with multi-step edits, and potential methods to overcome this challenge would be creative use of existing datasets (e.g. identifying relevant edits in WIKIREVISIONS), or latent variable learning methods to approximate the marginal in Equation 2 such as variational auto-encoders (Kingma and Welling, 2013).

Acknowledgements

We thank Vincent Hellendoorn for help when formulating the idea and providing scripts to develop CodeRevisions.
References


Wanyu Du, Vipul Raheja, Dhruv Kumar, Zae Myung Kim, Melissa Lopez, and Dongyeop Kang. 2022b. Understanding iterative revision from human-written text. ACL.


Appetite

A.1 Further training details

When training, we employ data sharding to enable cheaper, on the fly data processing. We shard each documents’ into 10 shards and form splits based on these shards. Our train-valid-test splits are split 90%, 5%, 5% for commit message generation, commit-conditioned edit modeling, and edit modeling. We use a dropout rate of 0.3 and use the GELU activation for all MLPs. We use a learning rate of 1e-4 warmed up for 1000 iterations. We also note that we extended the positional embedding matrix for BART to handle longer sequences.