Document Level Hierarchical Transformer

Najam Zaidi
Monash University
syed.zaidi1@monash.edu

Trevor Cohn
University of Melbourne
trevor.cohn@unimelb.edu.au

Gholamreza Haffari
Monash University
gholamreza.haffari@monash.edu

Abstract

Generating long and coherent text is an important and challenging task encompassing many application areas such as summarization, document level machine translation and story generation. Despite the success in modeling intra-sentence coherence, existing long text generation models (e.g., BART and GPT-3) still struggle to maintain a coherent event sequence throughout the generated text. We conjecture that this is because of the difficulty for the model to revise, replace or revoke any part that has been generated by the model.

In this chapter, we present a novel semi-autoregressive document generation model capable of revising and editing the generated text. Building on recent models by (Gu et al., 2019; Xu and Carpuat, 2020), we propose document generation as a hierarchical Markov decision process with a two level hierarchy, where the high and low level editing programs generate and refine the document. We train our model using imitation learning and introduce roll-in policy such that each policy learns on the output of applying the previous action. Experiments applying the proposed approach convey various insights on the problems of long text generation using our model. We suggest various remedies such as using distilled dataset, designing better attention mechanisms and using autoregressive models as a low level program.

1 Introduction

Generating long and coherent text encompass various tasks such as summarization, story generation, document level machine translation and document level post editing. Each task is characterised by modelling long range dependencies to make the document coherent as well as modelling a high level plot to make the document thematically consistent (Fan et al., 2018). This is challenging as the models need to plan content, while producing local words consistent with the global context in a timely manner.

Recent work on autoregressive generation models, such as GPT-3 and BART (Lewis et al., 2019; Brown et al., 2020), have shown impressive performance in generating short fluent text with a maximum length ranging from 150 to 350 tokens (Bosselut et al., 2018; Shen et al., 2019; Zhao et al., 2020b). But applying the same model to generate longer passages of text (e.g., 1000 tokens) has resulted in syntactic and semantic errors throughout the document requiring extensive human curations (Tan et al., 2020). These massive language models are usually pre-trained using large corpora of generic text, and then fine-tuned with small domain-specific data. Most of the time, the models are not publicly available to adapt to arbitrary desired domains.

On the other hand, recent non-autoregressive approaches allow generation to be done within a much smaller number of decoding iterations (Gu et al., 2017; Wang et al., 2019; Kasai et al., 2020). But due to its problems with modelling dependencies among the tokens, the approach still lags behind its autoregressive counterparts and has not yet been applied to long text generation (Zhou et al., 2019; Gu and Kong, 2020). In both of these model families, the length of generated sequences is either fixed or monotonically increased as the decoding proceeds. This makes them incompatible with human-level intelligence where humans can revise and edit any part of their generated text.

In this paper, we present a novel semi-autoregressive document generation model capable of revising and editing the generated text. We build on recent models by (Gu et al., 2019; Xu and Carpuat, 2020), who framed generation as a Markov decision process (Garcia and Rachelson, 2013) and showed that iteratively refining output sequences via insertions and repositions yields a fast and flexible generation process for machine translation and automatic post editing task. We extend their model by proposing document generation as a hierarchical Markov decision (Liu et al., 2018) process with a two level hierarchy. The high level program produce actions $a_H \in$ *{reposition, insert, update}* which tries to capture global context and plan content while the low level program produce actions $a_L \in$ *{reposition, insert}* to generate local words in a consistent and timely manner. Due to unavailability of large-scale data to train our model, we propose a noising process to simulate the error patterns observed in document level tasks such as redundancy of words, key information omission and disordered sentences. The noising process can be reversed by applying a set of high and low level actions to get back the original document. This serve as an efficient oracle to train our model using imitation learning (Hussein et al., 2017). The roll-in policy is defined such that each policy learns on the output of applying the previous action.

2 Problem formulation

2.1 Hierarchical Markov decision process

We cast document generation and refinement as a hierarchical Markov decision process (HMDP) with a two level hierarchy. The high level program is defined by the tuple $(\mathscr{D}, \mathscr{A_H}, \mathscr{E}, \mathscr{R}, \mathbf{d_0})$ where a state $\mathbf{d} \in \mathscr{D}$ corresponds to a set of sequences $\mathbf{d} = (\mathbf{s_1}, \mathbf{s_2}, ..., \mathbf{s_L})$ up to length L, and $\mathbf{d_0} \in \mathscr{D}$ is the initial document. The low level program corresponds to the tuple $(\mathscr{S}, \mathscr{A_L}, \mathscr{E}, \mathscr{R}, \mathbf{s_0})$ where a state $\mathbf{s} \in \mathscr{S}$ corresponds to a sequence of tokens $\mathbf{s} = (w_1, w_2, ..., w_n)$ from the vocabulary V up to length n, and $\mathbf{s_0} \in \mathscr{S}$ is the initial sequence.

At any time step t, the model takes as input $\mathbf{d_{t-1}}$, the output from the previous iteration, chooses an action $a_H \in \mathscr{A_H}$ to refine the sequence into $\mathbf{d_t} = \mathscr{E}(\mathbf{d_{t-1}}, a_H)$, and receives a reward $r_t = \mathscr{R}(\mathbf{d_t})$. The policy π_H maps the input sequence $\mathbf{d_{t-1}}$ to a probability distribution $P(A_H)$ over the action space $\mathscr{A_H}$. A high level program may call a low level program with the initial input $\mathbf{s_0}$. It is similar to high level program with its set of actions $a_L \in \mathscr{A_L}$, reward function $r_t = \mathscr{R}(\mathbf{s_t})$ and the policy π_L. Instead of sequences, the low level actions are applied to individual tokens. This results in a trajectory $\sigma := \{\mathbf{d_1}, a_H^1, \tau_1, r_1, \mathbf{d_2}, ..., \mathbf{d_N}, a_H^N, \tau_N, r_N, \mathbf{d_{N+1}}\}$ which is the concatenation of high-level trajectory $\tau_H := (\mathbf{d}_1, a_H^1, r_1, \mathbf{d}_2, a_H^2, r_2, ..., \mathbf{d}_{H+1})$ and the low level trajectory $\tau_L := (\mathbf{s}_1, a_L^1, \mathbf{s}_2, a_L^2, ..., \mathbf{s}_{T+1})$. We define a reward function $R = dist(\mathbf{D}, \mathbf{D}^*)$ which measures the distance between the generation and the ground-truth sequence. We use Levenstein distance (**?**) as our distance metric.

2.2 HMDP policies:

Following the formulation of HDMP, we define a high level policy $\pi_H : \mathbf{d} \rightarrow A_H$, as well as the low level policy $\pi_L : \mathbf{s} \rightarrow A_L$ as a mapping from state to actions. The high level actions consist of $a_H \in \{reposition, insert, update\}$ and the low level actions consist of $a_L \in \{reposition, insert\}$.

INSERT$_\mathbf{H}$: The insertion policy reads the input document $\mathbf{d}$ consisting of set of sequences $\{\mathbf{s_1}, \mathbf{s_2}, ...\mathbf{s_i}, \mathbf{s_{i+1}}, ...\mathbf{s_L}\}$, and for every possible slot $i, i+1$, the insertion policy $\pi_H^{ins}(x|i, \mathbf{d})$ makes a binary decision which is 1 (insert here) or 0 (do not insert). For each insertion position, low level MDP is called to generate the new sequence from scratch. This allows the model to generate a sentence conditioned on the surrounding context resulting in outputs that are consistent with the theme and plot of the document.

UPDATE$_\mathbf{H}$: The update policy reads the input document $\mathbf{d}$, consisting of set of sequences $\{\mathbf{s_1}, \mathbf{s_2}, ...\mathbf{s_i}, \mathbf{s_{i+1}}, ...\mathbf{s_L}\}$, and for every sequence position i, the update policy $\pi_H^{upd}(x|i, \mathbf{d})$ makes a binary decision which is 1 (update this sentence) or 0 (do not update). In order to make the update, the low level MDP is called to refine the given sequence. This allows the model to correct mistakes and improve the sentences generated by the insert policy.

REPOSITION$_\mathbf{H}$: The reposition policy reads in the document $\mathbf{d}$ consisting of set of sequences $\{\mathbf{s_1}, \mathbf{s_2}, ...\mathbf{s_i}, \mathbf{s_{i+1}}, ...\mathbf{s_L}\}$. For every sentence position i, the reposition policy $\pi_H^{rep}(x|i, \mathbf{d})$ makes a categorical decision between 0 and $L+1$ where L is the number of sequences in the document. The given sequence is repositioned to the output value. If x is 0 then the sequence is deleted. This policy allows the model to observe the complete document and make it more coherent by repositioning and deleteing sentences.

INSERT$_\mathbf{L}$, REPOSITION$_\mathbf{L}$: The Low level MDP is made up of actions reposition and insert. They work in a similar manner as defined in the paper (Gu et al., 2019; Xu and Carpuat, 2020) with the difference that the conditioning context contains document d along with the sentence s. Therefore the reposition policy at the word level is defined by $\pi_L^{rep}(x|i,\mathbf{y},\mathbf{d})$. The insertion policy is made up of a placeholder and token prediction policy as defined by $\pi_L^{plh}(x|i,\mathbf{y},\mathbf{d})$ and $\pi_L^{tok}(x|i,\mathbf{y},\mathbf{d})$ respectively. The placeholder policy first determines the number of words that need to be inserted at a given position. Special <*mask*> tokens are then inserted. These <*mask*> tokens are filled by the token prediction policy.

2.3 Generative process:

The generative process is outlined in algorithm 1. The combination of high and low level policies can either generate a document from scratch or edit a given initial document. The insertion and update policy calls the low level program in Lines 6 and 11. Line 2 in algorithm 2 builds the initial scaffolding which is later used by the algorithm for its set of actions. If the low level program is called by the high level update action the initial scaffolding is created by concatenating the sentences identified by the high level update policy. Otherwise in case of high level insert action, it is the concatenation of empty sentences. Although one iteration is made up of multiple stages, within each stage an action is performed in parallel.

3 Hierarchical Transformer

3.1 Architectures

Our model is based on the Transformer encoder-decoder architecture (Vaswani et al., 2017). We extract the hidden representations $(\mathbf{h_1}, ..., \mathbf{h_n})$ to make the policy predictions. We extract sentence representations by concatenating all sentences with a special <*sep*> token. The hidden states corresponding to these special tokens are then used as sentence representation by the policies. Along with position embeddings for individual tokens, we also introduce segment embeddings for sentences, which identify the position of a sentence in a document. We show the illustration of the proposed model in Figure 1.

3.2 Policy classifiers

We implement policies as classifiers whose prediction depends upon the hidden state representations generated by the transformer layers.

Reposition classifier: The reposition classifier gives a categorical distribution over the index of the input, where the input can be the representation of a sentence or a word. The input sequence is then repositioned accordingly. Along with reordering, this classifier can also perform deletion by predicting special delete token. This classifier is implemented as:

$$\pi_\theta^{rep}(r|\mathbf{s}_i,\mathbf{d}) = \text{softmax}(\mathbf{h}_i \cdot [\mathbf{b}, \mathbf{e_1}, ..., \mathbf{e_n}])$$

for $i \in \{1..n\}$ where $\mathbf{e}$ can be the embedding of a sentence or token and $\mathbf{b} \in \mathbb{R}^{\mathbf{d_{model}}}$ is a special token to predict deletion. Note that in case of low level program, we also condition on the complete document. This is done by having cross-attention on the hidden representation of the sentences.

Insertion classifier: The high level insert classifer scans over the consecutive sentences and make a binary decision to insert or not.

$$\pi_\theta^{ins}(p|\mathbf{s}_i,\mathbf{d}) = \text{softmax}([\mathbf{h}_i;\mathbf{h}_{i+1}] \cdot \mathbf{A})$$

for $i \in \{1..n\}$ and $\mathbf{A} \in \mathbb{R}^{2\times\mathbf{d_{model}}}$ is a parameter to be learned. The low level insert classifier is made up of placeholder insertion followed by token insertion. The placeholder classifier predicts the number of tokens to be inserted at every consecutive position pairs, by casting the representation to a categorical distribution

$$\pi_\theta^{ins}(p|w_i,\mathbf{s},\mathbf{d}) = \text{softmax}([\mathbf{h}_i,\mathbf{h}_{i+1}] \cdot \mathbf{B})$$

for $i \in \{1..n\}$ and $\mathbf{B} \in \mathbb{R}^{(k_{max}+1)\times(2\mathbf{d_{model}})}$ is a parameter to be learned. Following (Gu et al., 2019), k_{max} is 255. Token classifier then fill the placeholders

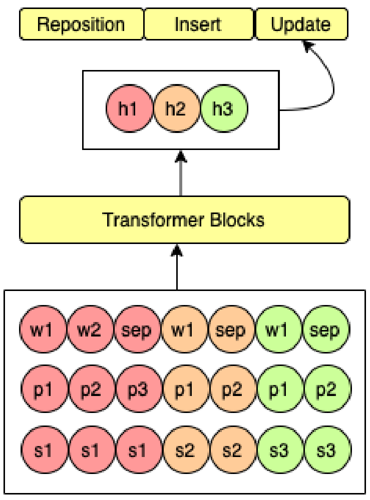
$$\pi_\theta^{tok}(t|w_i,\mathbf{s},\mathbf{d}) = \text{softmax}(\mathbf{h_i} \cdot \mathbf{C})$$

for $i \in \{1..n\}$ where w_i is a placeholder and $\mathbf{C} \in \mathbb{R}^{|\mathcal{V}|\times\mathbf{d_{model}}}$ is a parameter to be learned.

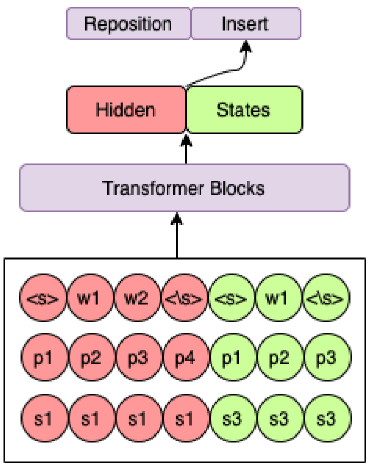
Update classifier: The update classifier is only present in the high level program. It scans over the sentences and make a binary decision to update a given sentence

$$\pi_\theta^{upd}(u|\mathbf{s}_i,\mathbf{d}) = \text{softmax}(\mathbf{h}_i \cdot \mathbf{D})$$

for $i \in \{1..n\}$ and $\mathbf{D} \in \mathbb{R}^{2\times\mathbf{d_{model}}}$ is a parameter to be learned.



(a) Transformer blocks extract the sentence representations which are used by high level policy classifiers. Suppose that the update policy predicts to refine sentence 1 and 3



(b) The input to the low level transformer is the concatenated sentences identified by the high level update policy.

Figure 1: The illustration of the proposed model for the update iteration. The same architecture can be applied for different tasks with specific classifiers. We have omitted attention from transformer blocks for simplicity. p stands for position embedding wheras s is for segment embedding

3.3 Noise

There is no large-scale labeled training dataset for document-level rewriting. Accordingly we train on synthetic dataset. To generate artificial broken text, we apply transformation techniques both at the sentence and word level and then learn to reverse the transformation to recover the original document. The techniques we use at the sentence level include: i) *sentences reordering* where sentences are randomly shuffled and/or deleted; ii) *sentence insertion* that a totally independent sentence is inserted into the source. iii) *sentence update* the sentence is slightly modified. For the lower-level transformation, we apply: i) *word insertion* that we insert a random word from another pre-defined vocabulary into the source. ii) *shuffle and delete* that we shuffle and delete some words. Each transformation is applied with a uniform probability between 0 and 1 leads to different trajectories of noise.

3.4 Oracle

Expert policy actions $\mathbf{a}^*$ are created by reversing the noise in the data. This is done by keeping track of the noise actions that have been used to create a corrupted output. In order to get alignment among sentences, we create a bipartite graph where the nodes are the sentences and the edge weight is the Levenstein distance between those sentences. We use max-flow min-cut algorithm to get the alignment (Dantzig and Fulkerson, 2003).

3.5 Training

Training is done by imitating the expert policy. We design roll-in policy such that each classifier is trained on the output of the other classifier. This reduces exposure bias as the model is trained on conditions it will encounter at decoding. The algorithm for training is shown in algorithm 3. The objective function is the product of decisions made during the generation process. It is the loses incurred by both the high level and low level program and is shown on line 14.

4 Experiments and Analysis

4.1 Experimental Setup

Data sets. We conduct experiments on synthetically generated dataset consisting of sorted and unsorted sequence pairs. Each sequence contains 5 - 10 and each line has between 20 to 100 tokens. The document is sorted in numerical order with tens coming before hundreds. The numbers lie between 1 and 1000. We generated 300K such pairs for training consisting of unsorted sequence as input and sorted sequence as output.

We further use real world datasets including ROC stories (Mostafazadeh et al., 2016), consisting of multiple 5 lines stories to check the capabilities of our model. We also conducted experiemnts on Multi-news and DUC-2004 for multi-document summarization (MDS), which is a sub-task of summarization tasks. Multi-news (Lebanoff et al., 2018) is a large-scale dataset for MDS and DUC-2004 (Over and Yen, 2004) is a benchmark dataset in MDS and its source documents are truncated to 1,500 tokens. To generate our input and output pairs, we inserted noise in the output sequences as outlined in section 3.3.

Evaluation Metrics. Rouge (Hovy et al., 2006), an automatic evaluation metric, is commonly used in Summarization to evaluate the quality of summaries. We use Rouge-l, Rouge-2 and Rouge-L to measure unigram-overlap, bigram-overlap, and the longtest common sequence between system and actual summaries. Synthetic and ROC stories are evaluated with BLEU score (Papineni et al., 2002).

Baselines. We compare three models: i) *Copy*: the original text is copied without any change, which establishes the lower bound for the task. ii) *Transformer*: a vanilla Transformer (Vaswani et al., 2017) is used to generate a sequence of text by reconstructing the source text. Without explicit editing guidance, we have little control over its generation process. iii) *Levenshtein Transformer (LevT)*: LevT is a semi autoregressive model for parallel sentence-level sequence generation (Gu et al., 2019). It refines a given sequence in an iterative manner with three operations, including *deletion*, *placeholder prediction* and *token prediction*. The iteration terminates when a certain stopping criterion is met. iv) *Editor transformer*: It is similar to the LevT, with the exception that it introduce a reposition operator instead of the deletion operator (Xu and Carpuat, 2020).

Implementation Details. To train the our models, we follow most of the hyper-parameter settings in (Gu et al., 2019). The only differences are that we use 3 Nvidia V100 GPUs and adopt fastbpe (**?**).

4.2 Results

The main results for summarization are shown in table 1. The best result is obtained by copy across both dataset indicating that post editing of long sequences may hurt its quality. Copy consist of output from SummPip system (Zhao et al., 2020a). SummPip uses graph clustering to find relevant sentences which are then used to generate the summary. Among other models, the Vanilla transformer performed better showing a strong bias present in the languages for autoregressive monotone generation. Levenshtein and the Editor transformer performed comparably whereas as our model showed no improvement over the baselines. We see similar performance in Synthetic and ROC-stories dataset in table 2 with Vanilla transformer performing better then the other models.

	Multi-News			DUC-2004		
	R-1	R-2	R-L	R-1	R-2	R-L
Copy	42.32	13.28	37.86	36.30	8.47	32.52
Transformer	40.62	12.42	36.37	35.4	7.78	31.71
LevT	25.93	8.59	28.95	23.45	4.89	25.12
Editor	25.56	8.13	28.33	23.17	4.21	25.01
Ours	21.67	5.89	24.03	18.22	2.17	20.87

Table 1: Experiment Results on Multi-News and DUC2004 dataset

	Synthetic	ROC-Stories
Copy	23.59	28.82
Transformer	30.17	35.72
LevT	22.42	25.29
Editor	22.78	25.89
Ours	20.63	23.10

Table 2: Experiment Results on Synthetic and ROC-stories dataset. We report the BLEU score in the table.

4.3 Analysis

We outlines various ways to improve the results of our model:

Evaluation metrics sensitivity towards document level ordering: We measure the sensitivity of our evaluation metrics towards capturing sentence reordering. We permuted sentences in a document and measure the metric's mean and standard deviation. The results in table 2 shows the inadequacy of using these metrics(BLEU, ROGUE) towards document level phenomenons. This suggest a training approach where a low level program is initially trained separately and then kept frozen while the high level program is trained.

	Mean	Standard Deviation
Synthetic	97.84	±0.05
ROC stories	98.94	±0.03
Multi-News	97.95	±0.05
DUC-2004	97.73	±0.05

Table 3: Sensitivity of metrics towards capturing sentence reordering. We synthetic and ROC stories we report the BLEU score. For Multi-news and DOC-2004 we report the R1 score. Mean and standard deviation is measured over 10 runs.

Distilled Dataset: Semi/non-autoregressive models struggle to achieve quality similar to autoregressive models. As the dependencies are broken, it become difficult for the model to generalise across multimodal dataset. The situation is further aggravated when the sequences are long. Distilled dataset has been found useful in dealing with multomodality problem in non-autoregressive modals (Zhou et al., 2019). Instead of using the actual output, the outputs generated from an autoregressive teacher modal are used with the input sequence. It is not directly clear as to how we can use distilled data in our model. One way is to insert the noise in distilled dataset to get input sequences. Another way is to use curriculum learning (Bengio et al., 2009), starting with distilled dataset and then moving to harder actual examples.

Better Training: Pre-training and fine-tuning approach has been found useful in various tasks. Our model consist of various components including classifiers at two levels. These classifiers can be individually pre-trained. Once the pre-training step is done, the whole model can be fine tuned for better model generalisation.

Use of Autoregressive model: The low level program is responsible for word generation. Due to the inherent left to right generation bias, autoregressive models have shown better results in our experiments. We can take advantage of this bias by using autoregressive model as a low level program but this can lead to longer decoding times.

Attention Mechanism: Wider context have been shown to improve results for various document level task (Kim et al., 2019). Designing an attention mechanism such that more attention is given to the sentences around the given sentence than those far away in the document can improve results. This can be done by having more attention heads for the near context then the far away context.

5 Related Work

Previous work on long text generation has mostly focused on generating tokens up to three hundred words. These method usually employ the idea of planning a document before generating it (Shen et al., 2019; Zhao et al., 2020b; Rashkin et al., 2020). Another line of work, focus on extending transformer architecture to model long sequences (Wang et al., 2020; Choromanski et al., 2020). Recent work by (Tan et al., 2020) used pre-train language models to progressively generate longer text greater than 300 tokens. Our work differs from previous approaches as it allows editing the generated text while it is being written. Previous work on non-monotonic generation and refinement (Welleck et al., 2019; Stern et al., 2019; Lee et al., 2018) has mostly focused on generating shorter text. Our proposed approach, differs from prior works by extending non-monotonic generation towards longer texts.

6 Conclusion

We present a hierarchical document generation model, that is capable of revising and editing its generated text thus bringing it closer to human-level intelligence. Although results showed that our approach lags behind the baselines, it did shed light into various problems present in semi-autoregressive models and long document generation. In the future, we will be incorporating these insights into our model to make it more robust.

Acknowledgments

References

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48.

Antoine Bosselut, Asli Celikyilmaz, Xiaodong He, Jianfeng Gao, Po-Sen Huang, and Yejin Choi. 2018. Discourse-aware neural rewards for coherent text generation. *arXiv preprint arXiv:1805.03766*.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. 2020. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*.

G Dantzig and Delbert Ray Fulkerson. 2003. On the max flow min cut theorem of networks. *Linear inequalities and related systems*, 38:225–231.

Angela Fan, Mike Lewis, and Yann Dauphin. 2018. Hierarchical neural story generation. *arXiv preprint arXiv:1805.04833*.

Frédérick Garcia and Emmanuel Rachelson. 2013. Markov decision processes. *Markov Decision Processes in Artificial Intelligence*, pages 1–38.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor OK Li, and Richard Socher. 2017. Non-autoregressive neural machine translation. *arXiv preprint arXiv:1711.02281*.

Jiatao Gu and Xiang Kong. 2020. Fully non-autoregressive neural machine translation: Tricks of the trade. *arXiv preprint arXiv:2012.15833*.

Jiatao Gu, Changhan Wang, and Jake Zhao. 2019. Levenshtein transformer. *arXiv preprint arXiv:1905.11006*.

Eduard H Hovy, Chin-Yew Lin, Liang Zhou, and Junichi Fukumoto. 2006. Automated summarization evaluation with basic elements. In *LREC*, volume 6, pages 604–611. Citeseer.

Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. 2017. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):1–35.

Jungo Kasai, James Cross, Marjan Ghazvininejad, and Jiatao Gu. 2020. Non-autoregressive machine translation with disentangled context transformer. In *International Conference on Machine Learning*, pages 5144–5155. PMLR.

Yunsu Kim, Duc Thanh Tran, and Hermann Ney. 2019. When and why is document-level context useful in neural machine translation? *arXiv preprint arXiv:1910.00294*.

Logan Lebanoff, Kaiqiang Song, and Fei Liu. 2018. Adapting the neural encoder-decoder framework from single to multi-document summarization. *arXiv preprint arXiv:1808.06218*.

Jason Lee, Elman Mansimov, and Kyunghyun Cho. 2018. Deterministic non-autoregressive neural sequence modeling by iterative refinement. *arXiv preprint arXiv:1802.06901*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*.

Ming Liu, Wray Buntine, and Gholamreza Haffari. 2018. Learning to actively learn neural machine translation. In *Proceedings of the 22nd Conference on Computational Natural Language Learning*, pages 334–344.

Nasrin Mostafazadeh, Nathanael Chambers, Xiaodong He, Devi Parikh, Dhruv Batra, Lucy Vanderwende, Pushmeet Kohli, and James Allen. 2016. A corpus and evaluation framework for deeper understanding of commonsense stories. *arXiv preprint arXiv:1604.01696*.

Paul Over and James Yen. 2004. An introduction to duc-2004. *National Institute of Standards and Technology*.

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

Hannah Rashkin, Asli Celikyilmaz, Yejin Choi, and Jianfeng Gao. 2020. Plotmachines: Outline-conditioned generation with dynamic plot state tracking. *arXiv preprint arXiv:2004.14967*.

Dinghan Shen, Asli Celikyilmaz, Yizhe Zhang, Liqun Chen, Xin Wang, Jianfeng Gao, and Lawrence Carin. 2019. Towards generating long and coherent text with multi-level latent variable models. *arXiv preprint arXiv:1902.00154*.

Mitchell Stern, William Chan, Jamie Kiros, and Jakob Uszkoreit. 2019. Insertion transformer: Flexible sequence generation via insertion operations. In *International Conference on Machine Learning*, pages 5976–5985. PMLR.

Bowen Tan, Zichao Yang, Maruan AI-Shedivat, Eric P Xing, and Zhiting Hu. 2020. Progressive generation of long text with pretrained language models. *arXiv preprint arXiv:2006.15720*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. 2020. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*.

Yiren Wang, Fei Tian, Di He, Tao Qin, ChengXiang Zhai, and Tie-Yan Liu. 2019. Non-autoregressive machine translation with auxiliary regularization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 5377–5384.

Sean Welleck, Kianté Brantley, Hal Daumé III, and Kyunghyun Cho. 2019. Non-monotonic sequential text generation. *arXiv preprint arXiv:1902.02192*.

Weijia Xu and Marine Carpuat. 2020. Editor: an edit-based transformer with repositioning for neural machine translation with soft lexical constraints. *arXiv preprint arXiv:2011.06868*.

Jinming Zhao, Ming Liu, Longxiang Gao, Yuan Jin, Lan Du, He Zhao, He Zhang, and Gholamreza Haffari. 2020a. Summpip: Unsupervised multi-document summarization with sentence graph compression. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1949–1952.

Liang Zhao, Jingjing Xu, Junyang Lin, Yichang Zhang, Hongxia Yang, and Xu Sun. 2020b. Graph-based multi-hop reasoning for long text generation. *arXiv preprint arXiv:2009.13282*.

Chunting Zhou, Graham Neubig, and Jiatao Gu. 2019. Understanding knowledge distillation in non-autoregressive machine translation. *arXiv preprint arXiv:1911.02727*.

A Appendices

A.1 Generation Algorithm

Algorithm 1 Generation in HMDP

```
Require: Initial document $\mathbf{d}_0$, policy: $\pi_{\theta_H}$
 1: $\mathbf{d} \leftarrow \mathbf{d}_0$
 2: while Termination condition is not met do
 3:     $\mathbf{rep\_index} \leftarrow \arg\max_{\mathbf{r}} \sum_{\mathbf{s}_i \in \mathbf{d}} \log \pi^{rep}_{\theta_H}(r_i|\mathbf{s}_i, \mathbf{d})$          ▷ Do reposition
 4:     $\mathbf{d} \leftarrow \mathscr{E}(\mathbf{d}, \mathbf{rep\_index})$
 5:     $\mathbf{ins\_index} \leftarrow \arg\max_{\mathbf{p}} \sum_{\mathbf{s}_i, \mathbf{s}_{i+1} \in \mathbf{d}} \log \pi^{ins}_{\theta_H}(p_i|\mathbf{s}_i, \mathbf{s}_{i+1}, \mathbf{d})$          ▷ Do insertion
 6:     $\mathbf{d} \leftarrow \mathscr{E}(\mathbf{d}, \mathbf{ins\_index})$          ▷ Call to Low level MDP
 7:     $\mathbf{upd\_index} \leftarrow \arg\max_{\mathbf{u}} \sum_{\mathbf{s}_i \in \mathbf{d}} \log \pi^{upd}_{\theta_H}(u_i|\mathbf{s}_i, \mathbf{d})$          ▷ Do update
 8:     $\mathbf{d} \leftarrow \mathscr{E}(\mathbf{d}, \mathbf{upd\_index})$          ▷ Call to Low level MDP
 9: end while
```

Algorithm 2 Low Level MDP

```
Require: Document $\mathbf{d}$, policy: $\pi_{\theta_L}$, Hi Level MDP action: $\mathbf{H}$
 1: while Termination condition is not met do
 2:     $\mathbf{s_0} \leftarrow \text{buildFrame}(\mathbf{d}, \mathbf{H})$
 3:     if $\mathbf{s_0}$ is empty then
 4:         $\mathbf{s} \leftarrow \mathbf{s_0}$          ▷ Skip reposition
 5:     else
 6:         $\mathbf{rep\_index} \leftarrow \arg\max_{\mathbf{r}} \sum_{w_i \in \mathbf{s}} \log \pi^{rep}_{\theta_L}(r_i|w_i, \mathbf{s}, \mathbf{d})$          ▷ Do reposition
 7:         $\mathbf{d} \leftarrow \mathscr{E}(\mathbf{s}, \mathbf{rep\_index})$
 8:     end if
 9:     $\mathbf{plh\_index} \leftarrow \arg\max_{\mathbf{p}} \sum_{w_i, w_{i+1} \in \mathbf{s}} \log \pi^{ins}_{\theta_L}(p_i|w_i, w_{i+1}, \mathbf{s}, \mathbf{d})$          ▷ Insert placeholders
10:     $\mathbf{s} \leftarrow \mathscr{E}(\mathbf{s}, \mathbf{plh\_index})$
11:     $\mathbf{tok\_index} \leftarrow \arg\max_{\mathbf{t}} \sum_{w_i \in \mathbf{s}, w_i == <mask>} \log \pi^{tok}_{\theta_L}(t_i|w_i, \mathbf{s}, \mathbf{d})$          ▷ Fill placeholders
12:     $\mathbf{s} \leftarrow \mathscr{E}(\mathbf{s}, \mathbf{tok\_index})$
13: end while
14: $\mathbf{d} \leftarrow \text{documentUpdate}(\mathbf{d}, \mathbf{s})$
```

A.2 Training Algorithm

Algorithm 3 Training for Hierarchical Levenshtein Transformer

```
Require: Training data $\mathscr{T}$, Model policy: $\pi_\theta$, Expert policy: $\pi_*$
 1: while Maximum training steps reached do
 2:     $(\mathbf{d}, \mathbf{d}_*) \sim \mathscr{T}$                                        ▷ Sample a training pair

 3:     $\mathbf{repH}^*, \mathbf{insH}^*, \mathbf{updH}^* \leftarrow \pi_*^H(\mathbf{d}, \mathbf{d}_*)$                        ▷ Get oracle actions
 4:     $\mathbf{repL1}^*, \mathbf{insL1}^*, \mathbf{tokL1}^*, \mathbf{repL2}^*, \mathbf{insL2}^*, \mathbf{tokL2}^* \leftarrow \pi_*^L(\mathbf{d}, \mathbf{d}_*)$

 5:     $\mathscr{L}_{\theta_H}^{rep} \leftarrow -\sum_{\mathbf{s}_i \in \mathbf{d}} \log \pi_{\theta_H}^{rep}(repH_i^* | \mathbf{s}_i, \mathbf{d})$
 6:     $\mathbf{d} \leftarrow \text{applyAction}(\mathbf{d}, \mathbf{repH}^*)$

 7:     $\mathscr{L}_{\theta_H}^{ins} \leftarrow -\sum_{\mathbf{s}_i, \mathbf{s}_{i+1} \in \mathbf{d}} \log \pi_{\theta_H}^{ins}(insH_i^* | \mathbf{s}_i, \mathbf{s}_{i+1}, \mathbf{d})$
 8:     $\mathbf{s} \leftarrow \text{buildFrame}(\mathbf{insH}^*, \mathbf{d})$

 9:     $\mathscr{L}_{\theta_L}^{rep1} \leftarrow -\sum_{w_i \in \mathbf{s}} \log \pi_{\theta_L}^{rep}(repL1_i^* | w_i, \mathbf{s}, \mathbf{d})$                        ▷ Low Level
10:     $\mathbf{s} \leftarrow \text{applyAction}(\mathbf{s}, \mathbf{repL1}^*)$
11:     $\mathscr{L}_{\theta_L}^{ins1} \leftarrow -\sum_{w_i, w_{i+1} \in \mathbf{s}} \log \pi_{\theta_L}^{ins}(insL1_i^* | w_i, w_{i+1}, \mathbf{s}, \mathbf{d})$
12:     $\mathbf{s} \leftarrow \text{applyAction}(\mathbf{s}, \mathbf{insL1}^*)$
13:     $\mathscr{L}_{\theta_L}^{tok1} \leftarrow -\sum_{w_i \in \mathbf{s}, w_i = <mask>} \log \pi_{\theta_L}^{tok}(tokL1_i^* | w_i, \mathbf{s}, \mathbf{d})$
14:     $\mathbf{d} \leftarrow \text{applyAction}(\mathbf{d}, \mathbf{insH}^*)$

15:     $\mathscr{L}_{\theta_H}^{upd} \leftarrow -\sum_{\mathbf{s}_i \in \mathbf{d}} \log \pi_{\theta_H}^{upd}(updH_i^* | \mathbf{s}_i, \mathbf{d})$
16:     $\mathbf{s} \leftarrow \text{buildFrame}(\mathbf{updH}^*, \mathbf{d})$

17:     $\mathscr{L}_{\theta_L}^{rep2} \leftarrow -\sum_{w_i \in \mathbf{s}} \log \pi_{\theta_L}^{rep}(repL2_i^* | w_i, \mathbf{s}, \mathbf{d})$                        ▷ Low Level
18:     $\mathbf{s} \leftarrow \text{applyAction}(\mathbf{s}, \mathbf{repL2}^*)$
19:     $\mathscr{L}_{\theta_L}^{ins2} \leftarrow -\sum_{w_i, w_{i+1} \in \mathbf{s}} \log \pi_{\theta_L}^{ins}(insL2_i^* | w_i, w_{i+1}, \mathbf{s}, \mathbf{d})$
20:     $\mathbf{s} \leftarrow \text{applyAction}(\mathbf{s}, \mathbf{insL2}^*)$
21:     $\mathscr{L}_{\theta_L}^{tok2} \leftarrow -\sum_{w_i \in \mathbf{s}, w_i = <mask>} \log \pi_{\theta_L}^{tok}(tokL2_i^* | w_i, \mathbf{s}, \mathbf{d})$

22:     $\theta \leftarrow \theta - \lambda \nabla [\mathscr{L}_{\theta_H}^{rep} + \mathscr{L}_{\theta_H}^{ins} + \mathscr{L}_{\theta_H}^{upd} + \mathscr{L}_{\theta_L}^{rep1} + \mathscr{L}_{\theta_L}^{ins1} + \mathscr{L}_{\theta_L}^{tok1} + \mathscr{L}_{\theta_L}^{rep2} + \mathscr{L}_{\theta_L}^{ins2} + \mathscr{L}_{\theta_L}^{tok2}]$
23: end while
```