Dual-Stage Multi-Task Syntax-Oriented Pre-Training for Syntactically Controlled Paraphrase Generation

Hongxu Liu[1], **Xiaojie Wang**[1], **Jiashen Sun**[2], **Ke Zeng**[2], **Guanglu Wan**[2]

[1]School of Artificial Intelligence, Beijing University of Posts and Telecommunications
[2]Meituan
{hxliu,xjwang}@bupt.edu.cn
{sunjiashen,zengke02,wanguanglu}@meituan.com

Abstract

Syntactically **C**ontrolled **P**araphrase **G**eneration (**SCPG**), which aims at generating sentences having syntactic structures resembling given exemplars, is attracting more research efforts in recent years. We took an empirical survey on previous SCPG datasets and methods and found three tacitly approved while seldom mentioned intrinsic shortcomings/trade-offs in terms of data obtaining, task formulation, and pre-training strategies. As a mitigation to these shortcomings, we proposed a novel **D**ual-**S**tage **M**ulti-**T**ask (DSMT) pre-training scheme, involving a series of structure-oriented and syntax-oriented tasks, which, in our opinion, gives sequential text models the ability of comprehending intrinsically non-sequential structures like **L**inearized **C**onstituency **T**rees (LCTs), understanding the underlying syntactics, and even generating them by parsing sentences. We performed further pre-training of the popular T5 model on these novel tasks and fine-tuned the trained model on every possible variant of SCPG task in literature, finding that our models significantly outperformed (up to 10+ BLEU-4) previous state-of-the-art methods. Finally, we carried out ablation studies which demonstrated the effectiveness of our DSMT methods and emphasized on the SCPG performance gains compared to vanilla T5 models, especially on hard samples or under few-shot settings.

1 Introduction

Given a source sentence X, (e.g., '*How exciting that is.*'), and a template sentence Y (e.g., '*That's exactly right*'), the goal of Syntactically Controlled Paraphrase Generation (SCPG) is to produce a paraphrase sentence Z whose syntactic structure stays as close to the Y as possible, while retaining X's original semantics (e.g., '*That's really exciting.*').

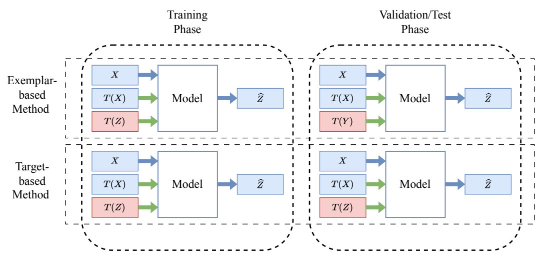


Figure 1: Differences between exemplar-based and target-based SCPG. Blue bars and red bars stand for model inputs/outputs while red bars stand for the part with differences (syntactical guidance inputs). $T(\cdot)$ indicates the constituency tree of a sentence, while $\hat{Z}$ stands for model's predictions for target sentence Z.

It is a simple yet crucial task, beneficial to a great many NLP tasks, such as neural machine translation (Yang et al.. 2019), abstractive text summarization (Cao et al., 2018), dialogue generation (Gao et al., 2020), as well as data augmentation (Sun et al., 2021) and improving model robustness (Iyyer et al., 2018; Huang et al., 2021).

In principle, supervised learning for SCPG task requires a set of (X, Y, Z) triplets. Practically, it's possible to obtain large-scale (X, Z) pairwise datasets by means of neural machine translation inference (Wieting et al., 2018), or various data-mining & web-crawling techniques (Iyer et al., 2017; Dolan et al., 2004; Potthast et al., 2010). However, obtaining (X, Y, Z) triplets constitutes a more delicate situation, usually involving heavy human workload or intolerable computational costs (Chen et al., 2019; Kumar et al., 2020). As a compromise to this problem, preliminary SCPG datasets (Chen et al., 2019; Kumar et al., 2020) use a great quantity of easy-to-gain (X, Z) pairs as training sets, together with a small portion of them delicately annotated and edited to form (X, Y, Z) triplets, constituting the validation/test set.

Findings of the Association for Computational Linguistics: ACL 2024, pages 14215–14231
August 11-16, 2024 ©2024 Association for Computational Linguistics

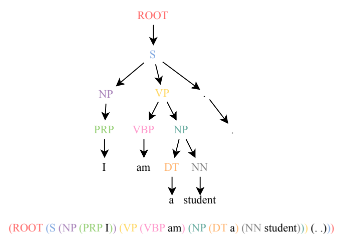

ROOT
S
NP
VP
.
PRP
VBP
NP
.
I
am
DT
NN
a
student
(ROOT (S (NP (PRP I)) (VP (VBP am) (NP (DT a) (NN student))) (. .)))

Figure 2: The Linearized Constituency Tree (LCT) example for sentence *I am a student.*, while a sequence wrapped in a pair of brackets stands for a subtree. Nodes and their corresponding parts in LCT are marked in the same color. For example, *(DT a)* stands for the subtree with *DT* as root node and *a* as child node.

Due to the absence of template sentence (Y) in the training set, as shown in Figure 1, preliminary methods are manifestly divided into two genres: *exemplar-based* and *target-based* methods. The former ones (Chen et al., 2019; Kumar et al., 2020; Sun et al,. 2021) train models to predict Z given Z's constituency tree ($T(Z)$) as syntactical inputs together with sentence X as sentential inputs on the (X, Y) pairwise training set. During inference, they replace $T(Z)$ with Y's constituency tree $T(Y)$ to incorporate Y into model's learned predicting procedure. The latter ones (Li et al., 2020; Yang et al., 2022a) adopt the same training criteria as the former ones, while keep using $T(Z)$ as syntactical inputs during evaluation. It's obvious that there is a trade-off between *bringing a train-validation inconsistency* and *deviating from the original SCPG task formulation* which requires Y, if Y is not present during training. What's more, they both introduce a reduction bias for a constituency tree (but not sentential) template as syntactical guidance, since Z itself cannot be provided to models during training.

Since the aforementioned SCPG methods require constituency trees which are difficult to feed to seq2seq models as syntactical inputs, another trade-off, *task* vs. *model structure trade-off* occurs. Some works (Kumar et al., 2020; Yang et al., 2022a) design dedicated tree-encoding model structures to assure fine-grained encoding of tree structures and precise syntactical control, while others (Iyyer et al., 2018; Sun et al., 2021; Yang et al., 2022b) convert trees to linearized constituency trees (LCTs) in a *root-first-bracket-separated* manner as shown in Figure 2 and treat them as sequential inputs like natural. The former ones add to model complexity while the latter ones inevitably introduce losses of topological information during linearizing.

Apart from supervised methods, there is also a growing trend of unsupervised pre-training on a large set of single sentences. Previous unsupervised works (Huang et al., 2021; Huang et al., 2022) mainly focus on training model to reconstruct sentences from disentangled syntactic and semantic embeddings learned by separate encoders. In spite of their effectiveness, their models still lack syntactical understanding abilities and can only adapt to a small range of downstream tasks due to their dedicated model structures. Thus, we firmly believe that the extent and power of syntactical pre-training is still underestimated, which is the third issue of previous methods.

Drawing from the three issues of previous works, we proposed a novel **D**ual-**S**tage **M**ulti-**T**ask (DSMT) syntax-oriented pre-training scheme based on a series of tasks which correspond to subtasks of understanding, manipulating, comprehending and generating LCT sequences. Upon pretraining, through analyzing the performance of our models on these tasks, we found that our models gain sufficient knowledge about the structure and intrinsic syntactics of LCT sequences. Afterwards, we fine-tuned our models on target-based and exemplar-based SCPG tasks adding no external tree-encoding structures or specifically designed input/output schemes and found that **1.** our models significantly outperform previous SOTA methods (up to 10+ BLEU-4) on target-based SCPG, **2.** training our models even on (X, Y, Z) triplet dataset may diminish the *train-validation inconsistency* for exemplar-based SCPG.

In our opinions, our contributions are as follows:

- We proposed a novel dual-stage multi-task pre-training scheme, which is the **first** that emphasize on understanding linearized tree sequences which were previously widely used but seldom considered. This also well remits the *third* issue of previous SCPG methods.

- We achieved SOTA performance without introducing dedicated structures by fine-tuning DSMT pre-trained models on all SCPG tasks, which resolves the second tradeoff. We built triplet training data and achieved SOTA performance on exemplar-based SCPG by using exemplar sentences as syntactical guidance inputs, which resolves the first issue.

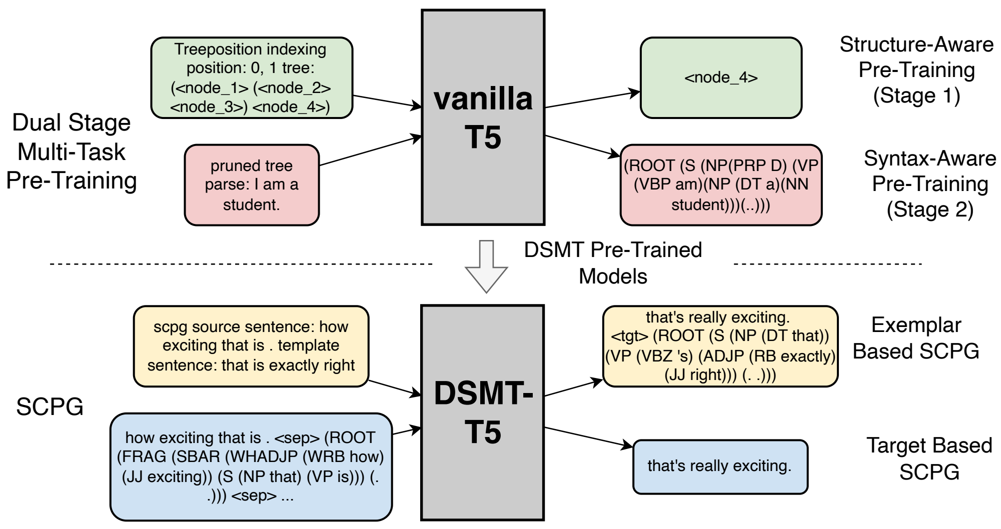

Dual Stage Multi-Task Pre-Training
Treeposition indexing position: 0, 1 tree: (<node_1> (<node_2> <node_3>) <node_4>)
pruned tree parse: I am a student.
vanilla T5
<node_4>
Structure-Aware Pre-Training (Stage 1)
(ROOT (S (NP(PRP D) (VP (VBP am)(NP (DT a)(NN student)))(..)))
Syntax-Aware Pre-Training (Stage 2)
DSMT Pre-Trained Models
SCPG
scpg source sentence: how exciting that is . template sentence: that is exactly right
how exciting that is . <sep> (ROOT (FRAG (SBAR (WHADJP (WRB how) (JJ exciting)) (S (NP that) (VP is))) (. .))) <sep> ...
DSMT-T5
that's really exciting. <tgt> (ROOT (S (NP (DT that)) (VP (VBZ 's) (ADJP (RB exactly) (JJ right))) (. .)))
Exemplar Based SCPG
that's really exciting.
Target Based SCPG

Figure 3: Architecture of our **D**ual-**S**tage **M**ulti-**T**ask pre-training and fine-tuning framework. Our models are trained on a series of tasks which are split into two stages, focusing on learning to comprehend LCTs structurally and semantically. After DSMT pre-training, our model is fine-tuned on both exemplar based and target based SCPG still under the text2text setting without introducing any external structures.

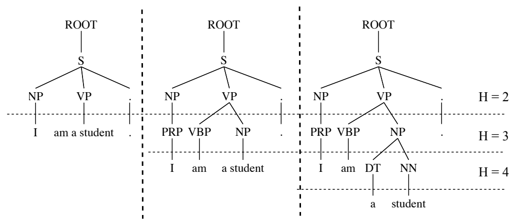
We released our code and data on https://github.com/ChristLBUPT/DSMT-T5, we'll actively maintain the repo in the following years.

2 Related Work

Prior works about SCPG can be categorized according to different perspectives. In terms of **model structures**, Early practices of SCPG such as (Iyyer et al., 2018; Kumar et al., 2020) utilize RNN (LSTM, GRU) or RNN-based VAE models (Chen et al., 2019; Bao et al., 2019; Zhang et al., 2019), while recent works mainly focus on randomly-initialized or pre-trained encoder-decoder Transformers (Li et al., 2020; Sun et al., 2021; Huang et al., 2021; Yang et al., 2022a; Huang et al., 2022) with reasonable parameter magnitudes. In terms of **syntactic control**, some of these VAE approaches (Chen et al., 2019; Bao et al., 2019) exploit syntactical latent variables while others use explicit constituency tree inputs. Among those who feed models explicit constituency trees, the way that they input the trees also differs, with some of them linearize (Iyyer et al., 2018; Sun et al., 2021) the constituency trees and directly feed them into seq2seq models just like sentences, while others design dedicated model structures such as Tree-LSTM (Kumar et al., 2020) or Tree-Transformer (Yang et al., 2022a) for intensive tree modeling. This essentially constitutes *the second issue* as mentioned in Section 1, in our opinion. Moreover, apart from supervised SCPG methods trained on parallel corpus, **unsupervised** methods on standalone sentences are also becoming a growing strand, which mainly achieve unsupervised training by separately encoding semantics and syntactics utilizing corresponding encoders and learn to reconstruct the sentences (Huang et al., 2021; Huang et al., 2022). These methods can adapt to SCPG tasks directly by mixing source sentence's semantics and template sentence's syntactics. These methods are spiritly similar to early VAE practices like Chen et al., 2019 and Bao et al., 2019 which, in our opinion, lacks variety in terms of tasks and model structures.

3 Dual-Stage Multi-Task Syntax-Oriented Pre-Training: Overview

The framework of our methods is shown in Figure 3. We base our work on T5 model (Raffel et al., 2020) and ParaNMT-50m (Wieting et al., 2018) dataset. Our DSMT pre-training involves *structure-aware pre-training* and *syntax-aware pre-training*. The former one focus on the structural aspects of understanding LCTs, like understanding the topologies which bracketing sequences (as in Figure 2) stand for, or inferring useful information such as heights or parent/sibling relationships from these sequences. The latter one focus on learning the meanings of different syntax tree constituents and understanding the co-relationship between constituency trees and sentences. Like T5 pre-training, we also cast all tasks to text2text format by introducing task-specific prefixes. Upon pre-training, we fine-



Corresponding Linearized Sequences	H = 2: (ROOT (S (NP I) (VP am a student) (. .)))
	H = 3: (ROOT (S (NP (PRP I)) (VP (VBP am) (NP a student)) (. .)))
	H = 4: (ROOT (S (NP (PRP I)) (VP (VBP am) (NP (DT a) (NN student))) (. .)))

Figure 4: Examples of pruning a constituency tree to different heights.

tune our pre-trained models on exemplar-based and target-based SCPG tasks, still in a text2text manner.

We'll demonstrate the two stages of our pre-training and the process of fine-tuning in the following sections.

4 Stage 1: Structure-Aware Pre-Training

In the following section, we'll be discussing the first stage which makes our model understand LCT sequences topologically, namely the *structure-aware pre-training* stage.

4.1 Datasets

Since most of the preliminary works including these unsupervised works use ParaNMT-50m (Wieting et al., 2018) or ParaNMT-small (Chen et al., 2019) as training set, in consideration of proving our method's intrinsic advantage rather than showcasing a technique of data engineering, we also use ParaNMT-50m as our pre-training dataset. Since it's a pairwise paraphrastic dataset derived from neural machine translation results, to utilize more various data, we only use the *machine translation* part of each translation pair. After a series of heuristics of filtering (details are discussed in Appendix B), around 22-million machine translation sentences are used for training. We split the sentences into samples for each task, and divide out 5k examples for each task as validation set. Finally, we use Stanford CoreNLP parser (Manning et al., 2014) to parse all sentences into constituency trees. We linearize the trees to gain LCTs. This processed ParaNMT-50m dataset will be used in both stages of our pre-training.

4.2 Data Preprocessing

Prior than training, we apply a series of critical preprocessing methods for our model and data. Remind that we use Stanford CoreNLP (Manning et al., 2014) parsers to parse the sentences, it's ok to directly use these LCT as inputs to these structure-aware tasks. However, since constituency trees are derived from a series of probabilistic context free grammar (PCFG) rules, they exhibit strong and easy-to-find regularities (*e.g.*, *S* is always at the beginning). These regularities give our model more possibility to remember LCT by rote and might thus hinder our models from focusing on learning topologies. Therefore, we add special tokens (`<node_1>`, `<node_2>`, … `<node_99>`) to our model's vocabulary and replace all constituency nodes randomly to these special nodes. We also all a `<sep>` token to our model's vocabulary for tasks with multiple input/output constituents and add tokens representing *node heights* (`<H_1>`, `<H_2>`, …) and *ranks among siblings* (`<S_1>`, `<S_2>`, …) for tree interpreting task.

4.3 Tasks

The key issue to structure-aware pre-training lies on tasks. We propose a series of structure-related tasks. We'll introduce these tasks in detail in the following paragraphs.

Treeposition Indexing. *Treeposition* is a sequence which can be used to locate a node in a tree. Assume the treeposition sequence is $\{t_1, t_2, \ldots t_m\}$, then t_i indicates that it is the t_i-th child of the $(i-1)$-th node. The treeposition of the root node will be an empty sequence. We train our model to locate a node by giving a treeposition sequence and a linearized tree sequence. A typical example is shown as follows:

```
  Inputs: treeposition indexing posi-
tion: 2 1: tree: (<node_1> <node_11>
(<node_12> <node_21>) <node_13>)
  Outputs: <node_21>
```

We can see that samples are prepended with task prefixes, with trees linearized. Other details, such as **sampling strategies** (probabilities of a node to be chosen, etc.), and **examples of all tasks** from each stage can be found in Appendix A.

Tree Forming: A tree can be constructed according to a series of context free grammar (CFG) productions. The left part of a production corresponds to the root of a subtree while the right part stands for its children. We train our model to predict the linearized sequence of the tree constructed by a set of productions, with the left part of the first production as *the root of the entire tree*.

Task (Metric)	Mean	Std
Treeposition Indexing (Acc)	96.20	1.59
Node Deleting (F1)	99.87	0.05
Tree Forming (F1)	99.79	0.08
Height Selection (F1)	97.80	0.91
Tree Interpreting (Bleu)	95.71	0.03
Stage 1 Average	97.88	0.28

Table 1: Model's performance on stage-1 structural aware pre-training tasks.

Node Deletion: Node deleting is a common tree operation, especially during construction a balanced tree. For a subtree `(A (B C D))`, we lift up `C` and `D` to make them as `A`'s new children upon deleting `B`. We train our model to predict the linearized sequence of the new tree after deleting, giving the linearized sequence of a tree and the treeposition of the node to be deleted.

Height selection: In this task, we train our model to select all nodes left-to-right at a specified height, given a tree's linearized sequence and a specified height.

Tree Interpreting: We selected several important topology-related attributes of tree nodes including height, parent node and rank among siblings, and train our models to predict them for each node given a tree's linearized sequence. These attributes include a node's parent, height, and rank among its siblings.

We can see that our tasks cover the adding, deleting, manipulating and indexing of linearized trees. We also incorporate parent-descendant and sibling relationships. These tasks are challenging and representative and will monitor our models' understanding of linearized tree sequences. Readers might refer to Appendix A for more details.

4.4 Experiment Settings and Results

We convert the processed ParaNMT-50m dataset into samples of different tasks with specific proportions, and train our model on a small subset of it. 5k samples are split out for each task as validation set. Compared to T5's multi-task settings (Raffel et al., 2020), we employ a stricter multitask setting under which we do not fine-tune our models on each task individually, but train a unified model. Details about training like task sample proportions, training and evaluation strategies, and hyperparameter settings are listed in Appendix B and Appendix C. We run experiments on 5 different seeds, with data reshuffled, and calculate the mean and standard deviation of each run.

Results of validation are shown in Table 1. We can see that our model gets considerable scores on those tasks, especially on those tree-generation tasks with trees as outputs such as Tree Forming. Considering these tasks' difficulty and representativeness, we suppose that our model gains yet sufficient knowledge of the topology of linearized tree sequences, and we'll explore them in experiments in Section 7.

5 Stage 2: Syntax-Aware Pre-Training

This section is about the second stage where our models learn the about LCT themselves (like what syntax nodes like *NP/VP* usually stand for), and the relationships between LCTs and sentences (like a sentence's corresponding LCT). We name it the **syntax-aware pre-training stage**.

5.1 Data Preprocessing

Since we use random node tokens in stage-1 while this stage involves real **syntax nodes** like *NP/VP*, we add these nodes to our models' vocabulary and embeddings. To better transfer knowledge learned from stage 1, we calculate the element-wise means and variances of the embeddings of the random nodes (added in stage 1) and initialize syntax nodes' embeddings by sampling from a high dimensional gaussian distribution featured by those means and variances, with covariances as zeros (*I.I.D.* for each dimension).

5.2 Tasks

We conduct the following tasks for constituency semantics learning:

Unsupervised Mask Filling: This is also the basic unsupervised task of T5's pre-training. We mask continuous spans from training LCT sequences and train our models to predict them. Since masked spans might correspond to constituency nodes, leaf nodes (words), brackets, and any arbitrary composition of all them above, this is a crucial way for our models to learn generalizable basic knowledge about the meaning of syntax nodes together with the structure of LCTs.

Apart from the *span-level* unsupervised mask filling task which is less intensive, we also introduce *sequence-level* tasks which focus on certain aspects of LCT comprehending. Some of them focus more on structural understanding, and are designed in order to prevent catastrophic forgetting of

Task (Metric)	Mean	Std
POS Tagging (Bleu)	95.74	0.40
Tree Pruning (F1)	96.80	0.26
Production Detection (F1)	98.80	0.35
Constituency Discrimination (Acc)	92.85	0.76
Constituency Searching (F1)	78.84	0.72
Prune Tree Parse (F1)	80.99	0.27
Stage 2 Average	91.19	0.21

Table 2: Model's performance on stage 2syntax-aware pre-training tasks.

structural knowledge learned from stage 1, we call them *auxiliary tasks*. These tasks include:

Tree Pruning: Given a tree's LCT sequence and a specified height h, we train our model to predict the LCT sequence of that tree pruned (as shown in Figure 4) at height h.

Production Detection: Given a LCT sequence, we train our model to list all CFG productions constituting the tree. Productions are listed in root-first traversal order.

Pruned Tree Completion: Given the linearized sequence of a sentence's pruned constituency tree, we train our model to predict the linearized sequence of the full-fledge constituency tree corresponding to the same sentence. This task involves parsing of find-grained syntactical structures.

Besides, we also introduce tasks which are more comprehensive, requiring deeper understanding of the syntactics of sentences and involving more parsing abilities. These tasks are called comprehensive task and are shown as follows:

POS Tagging: This is the standard part-of-speech (POS) tagging task in a text2text manner without any external sequence labeling structures. Models are trained to predict each token's POS tag sequentially given a sentence as inputs.

Constituency Searching: This task only accepts sentences as inputs. Given a sentence and a specified syntax node, our models are trained to predict all spans corresponding to that node.

Constituency Discrimination: Binary form of the previous task. Given a sentence, a span from that sentence and a constituency node, models are trained to deciding whether the span constitutes the constituency node. We add this task as an easy-to-learn alternative of the previous task.

Pruned Tree Parse: Given a sentence, our models are trained to parse the corresponding constituency tree pruned at a specified height.

In conclusion, like the tasks on which T5 was trained, these tasks cover almost all granularities of constituency syntactics, ranging from low level POS tags to high level pruned trees. They are also progressive, with some of them acting as the basics of others. Again, readers might refer to Appendix A for more details of these tasks.

5.3 Experiment Settings and Results

Similar to the settings of stage 1, we also convert the dataset to samples of each task with specific proportions and train models on 5 runs of different randomize settings. More details like sample proportions, data usage and training settings are introduced in Appendix B and Appendix C. Results are shown in Table 2. These results indicate similar conclusions as in stage 1, that our model gains considerable and sufficient performance on syntax-oriented tasks. We'll further compare with baselines and interpret the efficiency of our DSMT training in Section 7.

6 Downstream Task: Constituency-Based and Sentence-Based SCPG

A further pre-trained model is gained after dual-stage multi-task pre-training. We call the model **D**ual-**S**tage **M**ulti-**T**ask pre-trained **T5** (**DSMT-T5**). With respect to the two genres of previous supervised SCPG methods mentioned in section 1, it's time to answer the following two questions:

- **Q1**: Is DSMT-T5 well capable of *target-based* SCPG, when constituency trees are given as syntactical constraints?
- **Q2**: Now that *exemplar-based* SCPG has an obvious *train-validation gap* problem, how can we diminish that gap and will DSMT-T5 fit in well in that solution?

6.1 Target-Based SCPG

To answer **Q1**, we fine-tune and evaluate DSMT-T5 on ParaNMT-small (Chen et al., 2019), using <*source, paraphrase* > (*X, Z*) pairs in both training and validation/test set, using the LCTs of *paraphrase sentences* ($T(Z)$) as syntactic constraints.

Methods. Like AESOP (Sun et al., 2021) which makes no structural engineering to models, we employ a barely seq2seq input/output scheme. Our model's input includes source sentence, source sentence's pruned LCT and paraphrase sentence's pruned LCT without leaf nodes (words), with all of

Model	B-4	R-1/R-2/R-L	MTR	TED↓
SCPN	21.2	55.1 / 31.3 / 57.4	33.0	6.3
GuiG	26.3	60.7 / 37.1 / 62.5	38.0	6.4
SI-SCP	27.8	62.8 / 39.5 / 64.4	39.9	5.7
DSMT	**42.3**	**72.7/53.8/74.7**	**49.9**	**3.9**

Table 3: Results of target-based SCPG. Metrics stand for BLEU-4, Rouge-1/2/L, Meteor and Tree-Edit Distance between model outputs and target sentences. ↓ means smaller is better.

them concatenated by a special `<sep>` token. Our model's output only includes predicted paraphrase sentences.

Baselines. Under target-based setting, we compare our methods with preliminary methods also focusing on target-based SCPG together with their baselines: SCPN (Iyyer et al., 2018), GuiG (Li et al., 2020) and SI-SCP (Yang et al., 2022a).

Experiment Setup. We use the constituency trees provided alongside the public available ParaNMT-small dataset (Kumar et al., 2020). We prune the constituency trees to a height of 5 (including root) for fair comparison, since most of our baselines are prone to do so and using more find-grained constituency tree will significantly close up predicted sentences with ground truth under target-based setting. Details of training hyperparameters are discussed in Appendix D. In order to alleviate the effects of randomness, we run our experiments on *5 set of seeds* and report the averaged metrics.

Evaluation Metrics. Following previous works, we employ two sets of metrics, namely semantics preserving metrics and syntactics conformation metrics. The former includes BLEU (Papineni et al., 2002), ROUGE (Lin, 2004) and METEOR (Iyer et al., 2016). The latter includes Tree-Edit-Distance (Zhang and Shasha, 1989) between constituency trees of generated sentences target sentences.

Experiment Results. Experiment results are shown in Table 3. Through comparing semantically sand syntactically between different baselines and our models, we can see that our model achieves a significantly huger leap (more than 10 BLEU-4 and 1.8 TED-R) and is far better than any of these baselines. This is caused not only by the powerfulness of T5 but also by our DSMT pre-training. We'll discuss it in detail in section 7.

6.2 Exemplar-Based SCPG

To answer **Q2**, we first employ a series of heuristic filtering and calculating rules to add exemplar sentences (Y) to each (X, Z) pair of ParaNMT-small's

Model	B-4	R-1/R-2/R-L	MTR	TED↓
CGEN	13.6	44.8/21.0/48.3	24.8	6.7/6.0
SCGP	16.4	49.4/22.9/50.3	28.8	8.7/7.0
AESOP	22.9	54.5/29.8/56.4	32.7	6.9/5.7
Parafra-GPT	14.5	49.7/22.4/51.3	27.8	8.2/-
GCPG	26.2	63.6/40.8/ 65.0	39.8	8.3/-
Target-Based	**26.8**	**58.0/34.1/60.0**	**35.2**	**6.4/5.5**
DSMT-T5/C	**27.8**	**59.2/35.6/61.1**	**36.5**	**6.5/6.1**
DSMT-T5/S	**30.3**	**60.9/37.7/62.4**	**38.2**	**6.1/5.9**

Table 4: Results of exemplar-based SCPG. TED is calculated between predicted paraphrases and *gold paraphrases/exemplars*. *DSMT-T5/C(S)* refers to models trained on SCPG/C(S), while *Target-Based* stands for the inference result of pairwise trained model (as in 6.1) on SCPG/C.

training set, and then train and evaluate DSMT-T5 under two settings, one taking exemplar *constituency trees* as syntactical inputs while the other directly taking exemplar *sentence* as syntactical inputs. We call them **SCPG** with **C**onstituency Trees (**SCPG/C**) and **SCPG** with Exemplar **S**entences (**SCPG/S**)

Data Building. In order to get Ys, we use the Sequence Edit Distance (Levenshtein distance) of LCTs as an approximated measure of syntactical similarity. We find a syntactically closest yet lexically (bag-of-words) varied sentence for each target sentence from (X, Z) pairs of ParaNMT-small training set and obtained a training set consisting of around 250k (X, Y, Z) triplets. More details about triplet data building are described in Appendix D.

Methods. For experiments of **SCPG/C**, we adopt almost the same settings with target-based SCPG as illustrated in Section 6.1, apart from replacing *target* LCTs with *exemplar* LCTs. For **SCPG/S**, we still adopt a text-to-text input/output scheme, with source sentences and exemplar sentences as inputs and target sentences together with exemplar LCT as outputs. We add exemplar LCTs to increase the interpretability of SCPG/S since it doesn't involve constituency structures in inputs. Exemplar LCTs and prepended before paraphrase sentence, splitting with a `<tgt>` token. Details about sample formats are discussed in Appendix D.

Baselines. We compare our methods with previous exemplar-based SCPG advances, CGEN (Chen et al., 2019), SGCP (Kumar et al., 2020), AESOP (Sun et al., 2021), ParafraGPT (Bui et al., 2021) and GCPG (Yang et al., 2022b).

Model	Avg. Metric
DSMT Stage-1	97.88 ± 0.28
Ind. Stage-1	94.85
DSMT Stage-2	91.19 ± 0.21
Ind. Stage-2	85.14
w/o/stage-1	90.7

Table 5: Results of DSMT ablation studies (± means the standard deviation of a metric). We only report average metrics of each task due to space constraints. Ind. Stage-1/2 means the average metric of individually fine-tuning vanilla T5/stage-1-trained-T5 on each task in Stage-1/2 (for **Q2**) and w/o/stage-1 means directly training vanilla T5 model on stage-2 tasks (for **Q1**).

Experiment Setup and Evaluation Metrics. We employ similar metrics and other setups as our target-based experiments. Specially, we also compare the results (on exemplar based SCPG) of models trained on triplet datasets and pairwise datasets, by applying models in Section 6.1, which are trained on (X, Y) pairs, on SCPG/C validation set consisting of (X, Y, Z). This is the common practice of previous exemplar-based methods, yielding the train/validation gap as discussed in 1.

Experiment results. Results are shown in Table 4. In general, exemplar-based methods exhibit lower semantical and syntactical scores compared to target-based ones since exemplar sentences inevitably introduce syntactical noises, while we can still see a huge leap of performance of our methods compared with state-of-the-are baselines, except for GCPG (Yang et al., 2020b), which utilizes far more constraints other than syntactics, and performs undoubtedly better than SCPG methods only involving syntactical constraints. Specially, there are also two noticeable phenomena: **1.** Models trained on triplet datasets have better performance than those trained on pairwise dataset, even when trained on less data (our triplet dataset has ~250k triplets while the original training set has ~490k pairs). This is the well alleviation of the *train-validation inconstancy* discussed in Section 1. **2.** Models have better performance when accepting sentential syntactical inputs, meaning that models are understanding the underlying syntactics. There is no surprise since the stage-2 of DSMT consists of tasks incorporating LCTs with natural sentences.

7 Ablation Studies

In this section, we're dedicated to answer the following questions:

Model	B-4	R-1/R-2/R-L	MTR	TED↓
Target-Based				
DSMT	42.3	72.7/53.8/74.7	49.9	3.9
T5	41.7	72.1/52.7/74.1	49.1	4.1
SCPG/S				
DSMT	30.3	60.9/37.7/62.4	38.2	6.1/5.9
T5	30.0	60.6/37.0/62.1	38.1	6.1/6.0

Table 6: Results of SCPG ablation studies. DSMT means our pre-trained model's performance (same as that shown in section 6) while T5 means directly fine-tuning vanilla T5 on SCPG tasks.

For **DSMT pre-training**:

- **Q1**: Is structure-aware training stage (stage 1) beneficial to task performances of syntax-aware training stage (stage 2)?
- **Q2**: Are tasks corresponding to each stage exhibit mutual benefits, i.e., does multi-task setting provide performance gains compared with single-task setting?

For **exemplar and target based SCPG**:

- **Q3**: Since T5 itself is a powerful model, is it good enough to directly fine-tune T5 on SCPG task, without DSMT pre-training?
- **Q4**: If DSMT do gives a performance gain, in which situation will DSMT be more necessary?

Experiment Settings. To answer **Q1**, we directly train T5 model on stage 2 syntax-aware tasks using the same multitask training settings as what is described in Section 5.

To answer **Q2**, we fine-tune vanilla T5 model on stage 1 tasks individually, and fine-tune T5 model, which is already trained on stage 1, on stage 2 tasks individually. We train using the same quantities of data, steps of optimization and hyperparameters as DSMT training for each task for fair comparison.

To answer **Q3**, we directly fine-tune T5 model under target-based and SCPG/S (exemplar-sentence-based) SCPG settings as shown in Section 6. We call it **vanilla T5**. We compare SCPG performance of vanilla T5 and our DSMT-T5.

Results and Analysis. Results for DSMT pre-training ablation studies are shown in Table 5. We can see that models' performances degrade when individually trained on each task of each stage, or stripping out stage 1 pre-training, proving the effectiveness of DSMT pre-training.

Results for SCPG ablation studies are shown in Table 6. Readers may notice that DSMT do provides a performance gain compared with vanilla T5, while seemingly marginal. Even though results are averaged over 5 runs, which alleviates the effect of randomness to a great extent, to make our work more rigorous, we yielded the following questions:

- **Q1**: Are the *seemingly marginal* results statistically marginal, or explicit?
- **Q2**: If not, in which case will DSMT offer an explicit, never to be doubted, performance gain compared to vanilla T5?

To answer **Q1**, the obvious solution is performing statistical tests. Since we have metrics on 5 runs on different seeds, which constitutes a small sample count, we perform **Student t-test**, which well suits this case, on the metrics of target and exemplar based SCPG. To answer **Q2**, we make two assumptions, that is, DSMT will be more necessary when **1.** model cannot learn structures about LCT directly from SCPG training data (since target-based SCPG data itself involves LCTs), or when **2.** samples require understanding complex LCT structures. The corresponding cases are **few-shot SCPG** in which models are trained only on a proportion of SCPG training data, and **hard-sample SCPG** in which we evaluate the performance on the long samples of SCPG, which are commonly considered more syntactically complex.

In practice, we train our models on 20%, 10% and 5% of the training data of target-based SCPG to justify **assumption 1**. We compare the performances on top-400 (avg. 12.4 words), top-200 (avg. 14.4 words) and top-100 (avg. 16.2 words) long samples of the test set of ParaNMT-small, which has average word count of 9.6 words, to justify **assumption 2**.

Besides, to explore the mutual effects between **Q1** and **Q2**, we also combined the above solutions and perform t-tests on top-100 long samples.

Results and Analysis. Results of t-tests of all samples and top-100 long samples are shown in Table 7 of Appendix E. We can see that all p-values of target-based methods and most p-values of exemplar-based methods indicate that DSMT provides statistically significant performance gains, since they are below 0.05, even with some of them rounded to 0.000. We can also see that the p-values of long samples are smaller, simultaneously answering **Q1** and **Q2**. Note that since we only have 5 samples, p-values might still suffer from the small sample problem and some of them appear large, but still in a reasonable magnitude.

Results of few-shot SCPG and hard-sample SCPG are shown in Figure 5 and Figure 6 of Appendix E. We can see that as training data reduces, vanilla T5's performance degrades quickly, and the improvements become significantly larger. This indicates that DSMT gives our models general-purpose knowledge about syntactic structures, and gives them few-shot abilities, which vanilla T5 lacks, to adapt to syntax-intensive tasks rapidly. We can also see a growing trend for improvements when data becomes longer and longer. Since ParaNMT-small is far shorter than syntax-intensive datasets like PTB or SST, we believe the improvements will become more significant if tested on longer datasets. These two cases well demonstrate DSMT's effectiveness and necessity, drawing more research effort onto investigating the mutual relationship between our nova, DSMT pre-training, and various syntax-intensive tasks.

8 Conclusion and Future Work

We found three issues of previous SCPG works. We proposed a novel dual-stage multi-task pre-training framework which offers a possibility to apply pre-trained models on SCPG tasks with no external structures or specifically-designed input/output schemes. Upon resolving the three issues and proving the effectiveness of our pre-trained models on SCPG tasks, we have two critical findings. **First**, we found that exemplar-based SCPG exhibits better performance when trained on triplet dataset, or directly using sentential exemplars. This paved the way for subsequent exemplar-based SCPG researchers and suggests them to pay more attention on how to better utilize natural language based syntactical constraints and dive deeper on model's understanding of syntactics of sentences. **Second**, we found that transformer based seq2seq models may also have strong performance on tree comprehending tasks even with constituency trees fully linearized. This, to our best knowledge, is the first solid step on exploring the extent for seq2seq models to understand and generate LCTs, appealing for various syntax-intensive downstream applications, and also appealing further constituency parsing works to focus not only on working under CKY framework and taking neural models as *a neural alternative of PCFG rules*, but also focus on the networks themselves and explore various seq2seq solutions.

9 Limitations & Ethical Considerations

We further pre-trained T5 model on ParaNMT-50m dataset using various tasks. Generally, pre-training involves training on a large-scale dataset thus randomness such as model initialization or data reshuffling might have strong effects. Theoretically, it's encouraged that we run every experiment on different seeds and calculate mean accordingly. However, it's much too computational expensive for us so we only run the main DSMT pre-training experiments several times and record their means and variances. This is a similar practice as T5 (Raffel et al., 2020) pre-training but might still introduce the side-effects caused by randomness. What's more, due to the computational resource limits, we only trained our model on a subset of ParaNMT-50m for experiments on different seeds and ablation studies. This might not fully showcase the powerfulness of DSMT pre-training and might increase the effect of randomness. Moreover, since DSMT pre-training, as well as obtaining exemplar sentences (Ys) requires constituency trees of sentences, we used the trees produced by Stanford CoreNLP (Manning et al., 2014). Since it's an automated toolkit, wrong constituency trees are inevitable and noise might thus be added to our training data. Finally, we've only trained our model on ParaNMT-50m/ParaNMT-small dataset. This single-dataset setting might also introduce biases to our results and might hinder the demonstration our training framework's generalization, and might also raise a *data leakage* issue, since ParaNMT-small itself is a subset of ParaNMT-50m. However, since DSMT pre-training and SCPG exhibit quite difference formulations, we argue that the effect of data leakage is limited.

Moreover, we're using T5 model and ParaNMT-50m dataset as model and dataset. The former one is unsupervised trained on a large unlabeled corpus C4 (Raffel et al., 2020) while the latter one is derived from machine translation of a large corpus mainly consisting of movie subtitles. Partly due to the colloquial and dramatic nature of movie lines, through manually inspecting some of the samples from our training data, we found biased opinions or impolite words from those training samples. This means that we can't deny the possibility that our models generate toxic or harmful contents. We strongly suggest following researchers do safety-check prior than deploying DSMT-T5 on production environments.

Last but not least, as illustrated in Section 8, applications of DSMT pre-trained models on various downstream tasks (and various models, especially for LLMs which are ubiquitous nowadays) is desirable, but not explored by ourselves, due to the time and computational resource limit. Again, we would like to appeal for further researchers to explore more on this newly founded syntax-intensive pre-training scheme and reveal the immeasurable possibilities beneath it.

Acknowledgments

We would like to thank Meituan Inc. for offering computational resources, which is essential to our work. We'd also like to thank the anonymous reviewers who paid efforts to offer precious suggestions and made our paper more rigorous and self-contained.

References

Alfred. V. Aho and Jeffrey D. Ullman. 1972. The Theory of Parsing, Translation and Compiling, volume 1. Prentice-Hall, Englewood Cliffs, NJ.

Yu Bao, Hao Zhou, Shujian Huang, Lei Li, Lili Mou, Olga Vechtomova, Xin-yu Dai, and Jiajun Chen. 2019. Generating Sentences from Disentangled Syntactic and Semantic Spaces. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6008–6019, Florence, Italy. Association for Computational Linguistics.

Tien-Cuong Bui, Van-Duc Le, Hai-Thien To, and SangKyun Cha. 2021. Generative pre-training for paraphrase generation by representing and predicting spans in exemplars. In *IEEE BigComp,* pages 8390. IEEE.

Ziqiang Cao, Wenjie Li, Sujian Li, and Furu Wei. 2018. Retrieve, Rerank and Rewrite: Soft Template Based Neural Summarization. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 152–161, Melbourne, Australia. Association for Computational Linguistics.

Mingda Chen, Qingming Tang, Sam Wiseman, and Kevin Gimpel. 2019. Controllable Paraphrase Generation with a Syntactic Exemplar. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5972–5984, Florence, Italy. Association for Computational Linguistics.

Raffel Colin, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. 2020. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, *21*(1), 5485-5551.

Bill Dolan, Chris Quirk, and Chris Brockett. 2004. Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources. In *20th International Conference on Computational Linguistics (COLING)*.

Silin Gao, Yichi Zhang, Zhijian Ou, and Zhou Yu. 2020. Paraphrase Augmented Task-Oriented Dialog Generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 639–649, Online. Association for Computational Linguistics.

Kuanhao Huang and Kaiwei Chang. 2021. Generating Syntactically Controlled Paraphrases without Using Annotated Parallel Pairs. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pp. 1022-1033. 2021.

Kuanhao Huang, Varun Iyer, Anoop Kumar, Sriram Venkatapathy, Kai-Wei Chang, and Aram Galstyan. 2022. Unsupervised Syntactically Controlled Paraphrase Generation with Abstract Meaning Representations. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pp. 1547-1554. 2022.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2016. Summarizing source code using a neural attention model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics.

Shankar Iyer, Nikhil Dandekar, and Korn el Csernai. 2017. First quora dataset release: Question pairs. *data.quora.com*.

Mohit Iyyer, John Wieting, Kevin Gimpel, and Luke Zettlemoyer. 2018. Adversarial Example Generation with Syntactically Controlled Paraphrase Networks. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1875–1885, New Orleans, Louisiana. Association for Computational Linguistics.

Ashutosh Kumar, Kabir Ahuja, Raghuram Vadapalli, and Partha Talukdar. 2020. Syntax-Guided Controlled Generation of Paraphrases. Transactions of the Association for Computational Linguistics, 8:329–345.

Yinghao Li, Rui Feng, Isaac Rehg, and Chao Zhang. 2020. Transformer-based neural text generation with syntactic guidance. *arXiv preprint arXiv:2010.01737*.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*. Association for Computational Linguistics.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60.

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*. Association for Computational Linguistics

Martin Potthast, Benno Stein, Alberto Barrón-Cedeño, and Paolo Rosso. 2010. An Evaluation Framework for Plagiarism Detection. In *Coling 2010: Posters,* pages 997–1005, Beijing, China. Coling 2010 Organizing Committee.

Jiao Sun, Xuezhe Ma, and Nanyun Peng. 2021. AESOP: Paraphrase Generation with Adaptive Syntactic Control. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5176–5189, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

John Wieting and Kevin Gimpel. 2018. Paranmt-50m: Pushing the limits of paraphrastic sentence embeddings with millions of machine translations. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers),* pages 451–462.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics.

Xuewen Yang, Yingru Liu, Dongliang Xie, Xin Wang, and Niranjan Balasubramanian. 2019. Latent partofspeech sequences for neural machine translation. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics.

Erguang Yang, Chenglin Bai, Deyi Xiong, Yujie Zhang, Yao Meng, Jinan Xu, and Yufeng Chen. 2022a. Learning Structural Information for Syntax-Controlled Paraphrase Generation. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 2079–2090, Seattle, United States. Association for Computational Linguistics.

Kexin Yang, Dayiheng Liu, Wenqiang Lei, Baosong Yang, Haibo Zhang, Xue Zhao, Wenqing Yao and Boxing Chen. 2022b. GCPG: A General Framework for Controllable Paraphrase Generation. In *Findings of the Association for Computational Linguistics: ACL 2022*, 4035–47. Dublin, Ireland: Association for Computational Linguistics, 2022.

Kaizhong Zhang and Dennis Shasha. 1989. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, 18:1245–1262.

Xinyuan Zhang, Yi Yang, Siyang Yuan, Dinghan Shen, and Lawrence Carin. 2019. Syntax-Infused Variational Autoencoder for Text Generation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2069–2078, Florence, Italy. Association for Computational Linguistics.

Appendix A. DSMT Task Details

Below are examples of each tasks' training data and evaluation methods. If a involves sampling like choosing a span or a specified tree node, we'll also introduce the introduction of sampling strategies.

A.1 Treeeposition Indexing (Stage 1).

Example:

```
Inputs: treeposition indexing position: 1 2 1: tree: (<node_1> (<node_3> <node_9> (<node_11> <node_13> <node_33>)) (<node_4> <node_27))
Outputs: <node_33>
```

Sampling Strategy: Longer treepositions (treepositions of nodes close to leaf) are sampled more frequently. We sample from each treeposition of a tree, with the length of each treeposition as sampling weights.

Evaluation Metrics: We calculate the accuracy (proportion of properly predicted nodes).

A.2 Tree Forming (Stage 1):

Example:

```
Inputs: tree forming: <node_1> -> <node_2> <node_7> <sep> <node_2> -> <node_8> -> <node_11> <sep> <node_7> -> <node_25>
Outputs: (<node_1 (<node_2> <node_8> <node_11>) (<node_7> <node_25>))
```

Evaluation Metrics: We calculate the bracket F1 score widely used by constituency works. Bracket F1 score is based on bracket precision and recall. They are defined as follows:

$$Br_{\text{prec}} = \frac{\#(\text{correct brackets})}{\#(\text{brackets in predicted LCT})}$$

$$Br_{\text{recall}} = \frac{\#(\text{correct brackets})}{\#(\text{brackets in ground-truth LCT})}$$

$$Br_{\text{F1}} = \frac{2 \cdot Br_{\text{prec}} \cdot Br_{\text{recall}}}{Br_{\text{prec}} + Br_{\text{recall}}}$$

Where $\#(\cdot)$ means *the number of.*

A.3 Node Deleting (Stage 1):

Example:

```
Inputs: node deleting position: 1 2 tree: (<node_1> (<node_3> <node_9> <node_13> <node_33>) (<node_4> <node_27))
Outputs: (<node_1> (<node_3> <node_9> (<node_11> <node_13> <node_33>)) (<node_4> <node_27))
```

Sampling Strategy: Nodes are sampled with even possibilities.

Evaluation Metrics: We use bracket F1 described in A.3 as evaluation metrics.

A.4 Height Selection (Stage 1):

Example:

```
Inputs: height selection height: 3 tree: (<node_1> (<node_3> <node_5> <node_11>) (<node_2> <node_4>)
Outputs: <node_5> <node_11> <node_4>
```

Sampling Strategy: We sample heights with number of nodes of each height as weights.

Evaluation Metrics: We take output and label sentences as bag-of-words and calculate F1.

A.5 Tree Interpreting (Stage 1):

```
Inputs: tree interpreting: (<node_1> (<node_11> <node_21>) <node_12>)
Outputs: (<node_1> <H_1> <S_1> <none> (<node_11> <H_2> <S_1> <node_1> <node_21> <H_3> <S_1> <node_11> <node_12> <H_2> <S_2> <node_1>)
```

Evaluation Metrics: We calculate Bleu-4 (Papineni et al., 2002) between generated tree interpreting sequences and ground truth sequences.

A.6 Unsupervised Mask-Filling.

Example:

```
Inputs: low difficulty inter span: (ROOT (S (NP (<extra_id_0>)) <extra_id_1>am) (NP DT a) (NN student<extra_id_2> (. .)))
Outputs: <extra_id_0>PRP I<extra_id_1>(VP VBP<extra_id_2>)))<extra_id_3>
```

We have 4 sets of unsupervised mask-filling task settings: low difficulty intra span, low difficulty inter span, high difficulty intra span, high difficulty inter span. Intra span and inter span means whether masked spans should be within brackets or across brackets. Low difficulty and high difficulty mean the different proportions of length of masked spans. Low means masking around 15% of the sequence while High means masking around 30% of the sequence.

We do not evaluate mask-filling during evaluation phases.

A.7 Tree Pruning.

Example:

```
Inputs: tree pruning: (ROOT (S (NP (DT these) (NNS people)) (VP (VBP have) (VP (VBN been) (NP (NP (DT an) (JJ integral) (NN part)) (PP (IN of) (NP (DT this) (NN program)))))) (..)))
Outputs: (ROOT (S (NP (DT these) (NNS people)) (VP (VBP have) (VP (VBN been) (NP an integral part of this program))) (..)))
```

Evaluation Metrics: We use bracket F1 described in A.3 as evaluation metrics.

A.8 Pruned Tree Completion.

Example:

```
Inputs: pruned tree completion: (ROOT (S (NP (DT these) (NNS people)) (VP (VBP have) (VP (VBN been) (NP an integral part of this program))) (..)))
Outputs: (ROOT (S (NP (DT these) (NNS people)) (VP (VBP have) (VP (VBN been) (NP (NP (DT an) (JJ integral) (NN part)) (PP (IN of) (NP (DT this) (NN program)))))) (..)))
```

Evaluation Metrics: We also use bracket F1 described in A.3 as evaluation metrics.

A.9 Production Detection.

Example:

```
Inputs: production detection: (ROOT (S (NP (PRP I)) (VP (VBP am) (NP (DT a) (NN student))) (. .)))
Outputs: ROOT -> S <sep> S -> NP VP . <sep> NP -> PRP <sep> PRP -> I <sep> VP -> VBP NP <sep> VBP -> am <sep> NP -> DT NN <sep> DT -> a <sep> NN -> student <sep> . -> .
```

Evaluation Metrics: We calculate the F1 score of productions. Recall and Precision is calculated by #(exactly matched productions) divided by #(ground truth productions) and #(predicted productions), where #(·) means *the number of.*

A.10 POS Tagging.

Example:

```
Inputs: pos tagging: I am a student.
Outputs: PRP I VBP am DT a NN student . .
```

Evaluation Metrics: We calculate Bleu-4 (Papineni et al., 2002) between generated sentence-POS-tag sequences and ground truth sequences.

A.11 Constituency Searching.

Example:

```
Inputs: constituency searching node: NP sentence: I am a student.
Outputs: I <sep> a student
```

Sampling Strategy: We sample from each constituency node with equal possibilities.

Evaluation Metrics: Like production detection, we calculate the F1 score of constituency spans.

A.12 Constituency Discrimination.

Example:

```
Inputs: constituency discrimination node: NP span: I am sentence: I am a student.
Outputs: False
```

Sampling strategy: We sample the node using the same strategy as constituency searching. We sample 50% positive samples and 50% negative samples. For negative samples, we sample two offsets from {-3, -2, -1, 0, 1, 2, 3} and move the left bound & right bound according to offsets and clamp them within [0, length of sentence). This sampling strategy will produce harder samples than random sampling.

Evaluation Metrics: We calculate the accuracy, that is, proportion of correct predictions as metrics.

A.13 Pruned Tree Parse.

Example:

```
Inputs: pruned tree parse: I am a student.
Outputs: (ROOT (S (NP (PRP I)) (VP (VBP am) (NP (DT a) (NN student))) (. .)))
```

Note that we use a fixed prune height of 5, which is the optimal or sub-optimal prune height of previous SCPG tasks (Kumar et al., 2020; Sun et al., 2021).

Evaluation metrics: We use bracket F1 described in A.3 as evaluation metrics.

Appendix B. DSMT Training Strategies

This section of appendix will be discussing the training strategies, like proportions of each task in dataset, the overall proportion of dataset together with training and evaluation strategies, of DSMT training.

B.1 Data Preprocessing & Filtering

The original ParaNMT-50m dataset is a noisy dataset consisting of 50-million <reference-sentence,

translate-sentence> pairs. We adopt the following filtering strategies:

- We filter out sentences shorter than 5 tokens and longer than 25 tokens.
- Some sentences contain repeatedly generated tokens due NMT model's limitations. We filter out sentences consisting of 5 or more repeat tokens.
- Some sentences are barely upper-case. We lowercase these sentences.
- We filter out sentences having 40% or the characters as digits, since these sentences are numerical sentences which might correspond to serial numbers or bookpages.
- We clean the punctuations to make quotes or double tildes match.

After filtering and processing, a dataset consisting of around 22-million sentences is obtained.

B.2 Data Partitioning & Training Strategies

Structure Aware Training Stage. We perform training on a portion of the processed (like shown in section 4.2) ParaNMT-50m dataset, with around 250k linearized tree sequences. We divide out 25k LCT sequences from that dataset as validation samples (5k for each task) and do evaluation every 8192 steps. In terms of the proportions of each task, we first split samples equally, observe validation performance alongside training, and adjust data splitting strategy accordingly in order to make model's performance harden on each task. As a trade-off between sufficient learning and preventing overfitting, we choose to train on around 15% data of our processed ParaNMT-50m dataset (around 3.2M samples), after which our model's validation performance harden. What's more, Following T5's multi-task training practice (Raffel et al., 2020), we separately fine-tune our model on each task and after multi-task training regard model's corresponding performance as final performance.

Syntax-Aware Pre-Training Stage. For models used as seed repetition experiments and ablation studies, we train them on around 20% of our dataset (around 4.3M samples). For models used for SCPG fine-tuning, we train them on 80% of our dataset (around 17M samples). 60% of the sentences are randomly selected and masked as unsupervised samples while the rest of them are equally partitioned for each auxiliary and comprehensive task. These samples are pre-processed and converted from constituency annotated sentences to samples corresponding to each auxiliary/comprehensive tasks. Like what has been done in stage 1 pre-training, we also picked out 35000 sentences of our training set as the validation set, which are also equally partitioned for each auxiliary/comprehensive tasks (total 7 tasks).

Appendix C. DSMT Implementation Details

C.1 Training Hyperparameters.

We use PyTorch T5 model implemented by huggingface transformers (Wolf et al., 2020) library. Our transformers library version is 4.27.1 and our PyTorch version is 1.13.1. We expand the tokenizer vocabulary and model embedding layers prior than each stage's training as described in Section 4 and Section 5. We use Adam optimizer with $\beta_1 = 0.9$ and β_2 =0.99 and use a learning rate of $5e-5$ in constant schedule with 1000-step warmups. We use a per-device batch size of 16 and set gradient accumulation steps to 16. During the evaluation phase, we use greedy decoding with `generation_kwargs` as default settings.

C.2 Computational Costs.

Our models are trained on machines with 2 ×32G Nvidia V100 GPUs with Distributed Data Parallel implemented by huggingface accelerate library. Training elapses around 10-15 hours for stage 1 and stage 2 pre-training on a 15%/20% subsets and around 2 days on 80% of our dataset. This is roughly 4 × computational costs of target-based SCPG, which, in our opinion, is acceptable considering its gains.

Appendix D. SCPG Implementing Details.

D.1 Building of Triplet ParaNMT Dataset.

The procedure of building a training set consisting of (X, Y, Z) triplets is that for each (X, Z) pair, find a Y which has the least syntactical variance compared to Z according to a specific syntactical measurement $D_S(Y, Z)$, while having enough lexical variance according to lexical measurement $D_L(Y, X; Z)$, in order to effectively prevent template sentence word copying problem (Chen et al., 2019; Yang et al., 2022b).

Now that deciding Y involves $\Theta(n^2)$ outer loop among the training set, as a balance between syntactical representation and computational efficiency, we regard the edit distance of the LCT sequences of Y, Z LevEdit(LCT_Y, LCT_Z) as $D_S(Y, Z)$, and regard bag-of-word F1 as $D_L(Y, X)$. For each (X, Z) pair, we apply the aforementioned heuristics on $\{Y | Y \in \{X\} \cup \{Z\}, \big||Y| - |Z|\big| < 2, Y \notin \{X, Z\}\}$, that is, sentences in training set within 2-token length difference and other than (X, Y), to find the appropriate Y. Pairs which have little syntactical variance are filtered out by calculating Tree-Edit Distances between (X, Z)s and filtering out the low-distance ones. Eventually, around 250k (X, Y, Z) triplets are obtained.

D.2 Model Inputs & Outputs.

For **target-based SCPG**, we use the special token `<sep>`, as used in DSMT pre-training, to split source sentences and LCTs. Inputs & outputs are shown as follows:

```
  Inputs: [source sentence] <sep>
[source LCT] <sep> [paraphrase LCT].
  Outputs: [parphrase sentence].
```

Note that compared with *source LCT*, *paraphrase LCT* has stripped leaf nodes (words) out.

For **SCPG/C**, we use similar input/output schemes as target based SCPG. Inputs & outputs are shown as follows:

```
  Inputs: [source sentence] <sep>
[source LCT] <sep> [exemplar LCT].
  Outputs: [parphrase sentence].
```

For **SCPG/S**, we use different input/output schemes, which is shown as follows:

```
  Inputs: no-tree-scpg source sen-
tence: [source sentence] template
sentence: [exemplar sentence].
  Outputs: [parphrase sentence] <tgt>
[exemplar LCT].
```

Note that this text2text format is similar to those of DSMT pre-training tasks but different from target-based SCPG. Actually, during experiments, we tried a great many formats and found that sample formats has quite limited effect on task performance. Therefore, for target-based SCPG and SCPG/C, we choose a similar scheme as AESOP (Sun et al., 2021), which is the previous text2text SOTA (corresponding to methods *adapting to model structures* mentioned in Section 1), for better comparison.

D.3 Training Details.

We use the same hyperparameter settings in all sets of experiments. We train our model 8 epochs with a batch size of 16 and gradient accumulation steps of 8. We use Adam Optimizer with ($\beta_1 = 0.9$, $\beta_2 = 0.99$) using cosine schedule. We also use Nvidia V100-32G for training, which elapses around 24 hours for target based SCPG on original ParaNMT-small dataset and 9 hours for exemplar sentence based SCPG, 12 hours for exemplar tree based SCPG on our processed and filtered triplet ParaNMT-small dataset.

Appendix E. SCPG Ablation Results.

E.1 Student t-Test Results.

Below are results of Student t-Test of whether or not model results of DSMT-T5 are *significantly higher* than those of vanilla T5, under target-based and sentential exemplar based (SCPG/S) settings:

Model	B-4	R-1/R-2/R-L	MTR	TED↓
		Target-Based		
DSMT	42.3	72.7/53.8/74.7	49.9	3.9
T5	41.7	72.1/52.7/74.1	49.1	4.1
		SCPG/S		
DSMT	30.3	60.9/37.7/62.4	38.2	6.1/5.9
T5	30.0	60.6/37.0/62.1	38.1	6.1/6.0

Table 7: p-values of ablation study results of target-based SCPG and SCPG/S.

E.2 Few-Shot / Hard-Sample SCPG Results.

Below are results of vanilla T5 together with DSMT-T5 under few-shot and hard-sample SCPG settings:

Few-shot SCPG:

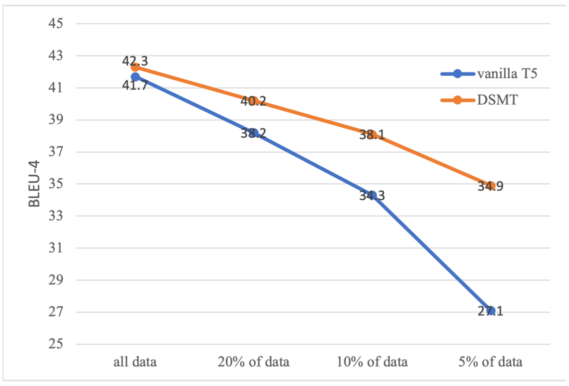


Figure 5: Results (BLEU-4) of vanilla/DSTM-T5 on target-based SCPG, when trained on all, 20%, 10% and 5% of ParaNMT-small training data.

Hard-Sample SCPG:

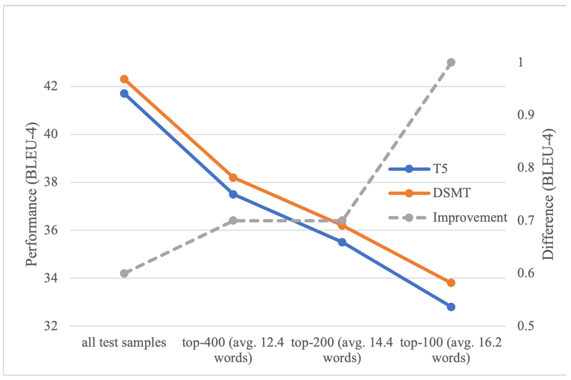


Figure 6: Results of vanilla/DSMT-T5 (in blue and orange solid lines) and their differences (in grey dashed lines) when evaluated on all, top-400, top-200 and top-100 long samples.