Syntactically Supervised Transformers for Faster Neural Machine Translation

Nader Akoury, Kalpesh Krishna, Mohit Iyyer
College of Information and Computer Sciences
University of Massachusetts Amherst
{nsa,kalpesh,miyyer}@cs.umass.edu

Abstract

Standard decoders for neural machine translation *autoregressively* generate a single target token per time step, which slows inference especially for long outputs. While architectural advances such as the Transformer fully parallelize the decoder computations at training time, inference still proceeds sequentially. Recent developments in *non-* and *semi-autoregressive* decoding produce multiple tokens per time step independently of the others, which improves inference speed but deteriorates translation quality. In this work, we propose the syntactically supervised Transformer (SynST), which first autoregressively predicts a chunked parse tree before generating all of the target tokens in one shot conditioned on the predicted parse. A series of controlled experiments demonstrates that SynST decodes sentences $\sim 5\times$ faster than the baseline autoregressive Transformer while achieving higher BLEU scores than most competing methods on En-De and En-Fr datasets.

1 Introduction

Most models for neural machine translation (NMT) rely on *autoregressive* decoders, which predict each token t_i in the target language one by one conditioned on all previously-generated target tokens $t_{1\cdots i-1}$ and the source sentence s. For downstream applications of NMT that prioritize low latency (e.g., real-time translation), autoregressive decoding proves expensive, as decoding time in state-of-the-art attentional models such as the Transformer (Vaswani et al., 2017) scales quadratically with the number of target tokens.

In order to speed up test-time translation, *non-autoregressive* decoding methods produce all target tokens at once independently of each other (Gu et al., 2018; Lee et al., 2018), while *semi-autoregressive* decoding (Wang et al., 2018; Stern et al., 2018) trade off speed for quality by reducing (but not completely eliminating) the number of sequential computations in the decoder (Figure 1). We choose the latent Transformer (LT) of Kaiser et al. (2018) as a starting point, which merges both of these approaches by autoregressively generating a short sequence of discrete latent variables before non-autoregressively producing all target tokens conditioned on the generated latent sequence.

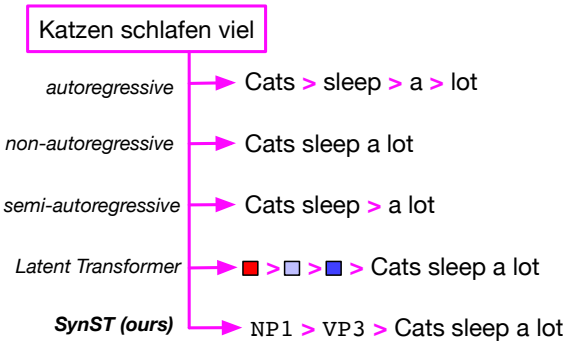

Figure 1: Comparison of different methods designed to increase decoding speed. The arrow > indicates the beginning of a new decode step conditioned on everything that came previously. The latent Transformer produces a sequence of discrete latent variables, whereas SynST produces a sequence of syntactic constituent identifiers.

Kaiser et al. (2018) experiment with increasingly complex ways of learning their discrete latent space, some of which obtain small BLEU improvements over a purely non-autoregressive baseline with similar decoding speedups. In this work, we propose to syntactically supervise the latent space, which results in a simpler model that produces better and faster translations.[1] Our model, the syntactically supervised Transformer (SynST, Section 3), first autoregressively predicts a sequence of target syntactic chunks, and then non-autoregressively

[1] Source code to reproduce our results is available at https://github.com/dojoteef/synst

Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pages 1269–1281
Florence, Italy, July 28 - August 2, 2019.

generates all of the target tokens conditioned on the predicted chunk sequence. During training, the chunks are derived from the output of an external constituency parser. We propose a simple algorithm on top of these parses that allows us to control the average chunk size, which in turn limits the number of autoregressive decoding steps we have to perform.

SynST improves on the published LT results for WMT 2014 En→De in terms of both BLEU (20.7 vs. 19.8) and decoding speed (4.9× speedup vs. 3.9×). While we replicate the setup of Kaiser et al. (2018) to the best of our ability, other work in this area does not adhere to the same set of datasets, base models, or "training tricks", so a legitimate comparison with published results is difficult. For a more rigorous comparison, we re-implement another related model within our framework, the semi-autoregressive transformer (SAT) of Wang et al. (2018), and observe improvements in BLEU and decoding speed on both En↔ De and En→ Fr language pairs (Section 4).

While we build on a rich line of work that integrates syntax into both NMT (Aharoni and Goldberg, 2017; Eriguchi et al., 2017) and other language processing tasks (Strubell et al., 2018; Swayamdipta et al., 2018), we aim to use syntax to speed up decoding, not improve downstream performance (i.e., translation quality). An in-depth analysis (Section 5) reveals that syntax is a powerful abstraction for non-autoregressive translation: for example, removing information about the constituent type of each chunk results in a drop of 15 BLEU on IWSLT En→De.

2 Decoding in Transformers

Our work extends the Transformer architecture (Vaswani et al., 2017), which is an instance of the encoder-decoder framework for language generation that uses stacked layers of self-attention to both encode a source sentence and decode the corresponding target sequence. In this section, we briefly review[2] the essential components of the Transformer architecture before stepping through the decoding process in both the vanilla autoregressive Transformer and non- and semi-autoregressive extensions of the model.

[2]We omit several architectural details in our overview, which can be found in full in Vaswani et al. (2017).

2.1 Transformers for NMT

The Transformer encoder takes a sequence of source word embeddings $\boldsymbol{s}_{1,\dots,}\boldsymbol{s}_n$ as input and passes it through multiple blocks of self-attention and feed-forward layers to finally produce contextualized token representations $\boldsymbol{e}_{1,\dots,}\boldsymbol{e}_n$. Unlike recurrent architectures (Hochreiter and Schmidhuber, 1997; Bahdanau et al., 2014), the computation of $\boldsymbol{e}_n$ does not depend on $\boldsymbol{e}_{n-1}$, which enables full parallelization of the encoder's computations at both training and inference. To retain information about the order of the input words, the Transformer also includes positional encodings, which are added to the source word embeddings.

The decoder of the Transformer operates very similarly to the encoder during training: it takes a *shifted* sequence of target word embeddings $\boldsymbol{t}_{1,\dots,}\boldsymbol{t}_m$ as input and produces contextualized token representations $\boldsymbol{d}_{1,\dots,}\boldsymbol{d}_m$, from which the target tokens are predicted by a softmax layer. Unlike the encoder, each block of the decoder also performs *source attention* over the representations $\boldsymbol{e}_{1\dots n}$ produced by the encoder. Another difference during training time is *target-side masking*: at position i, the decoder's self attention should not be able to look at the representations of later positions $i+1,\dots,i+m$, as otherwise predicting the next token becomes trivial. To impose this constraint, the self-attention can be masked by using a lower triangular matrix with ones below and along the diagonal.

2.2 Autoregressive decoding

While at training time, the decoder's computations can be parallelized using masked self-attention on the ground-truth target word embeddings, inference still proceeds token-by-token. Formally, the vanilla Transformer factorizes the probability of target tokens $t_{1,\dots,}t_m$ conditioned on the source sentence s into a product of token-level conditional probabilities using the chain rule,

$$p(t_{1,\dots,}t_m|s) = \prod_{i=1}^{m} p(t_i|t_{1,\dots,}t_{i-1}, s).$$

During inference, computing $\arg\max_t p(t|s)$ is intractable, which necessitates the use of approximate algorithms such as beam search. Decoding requires a separate decode step to generate each target token t_i; as each decode step involves a full pass through every block of the decoder, autoregressive decoding becomes time-consuming especially

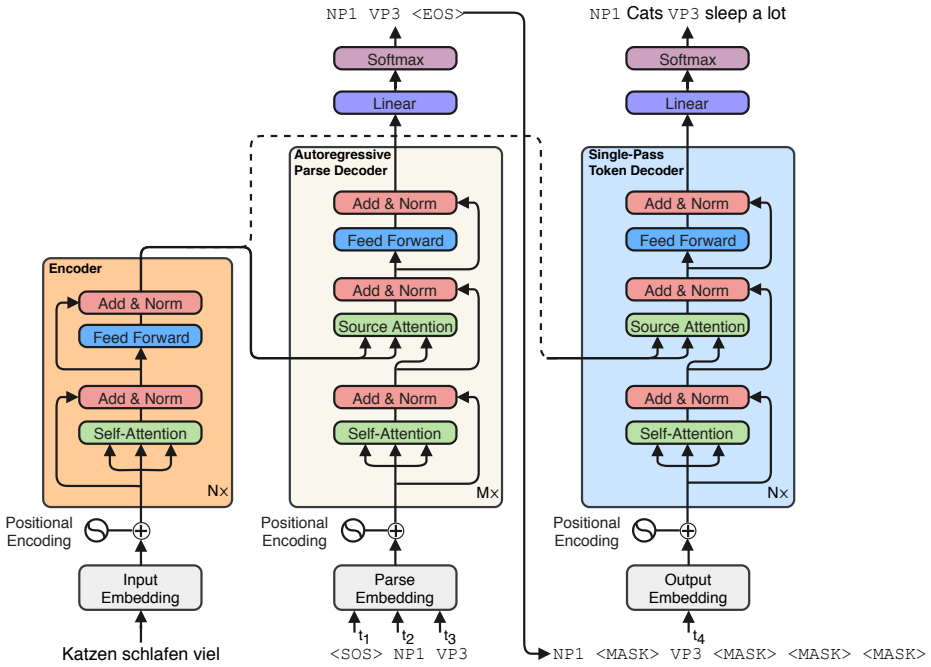

Figure 2: A high-level overview of the SynST architecture. During training, the parse decoder learns to autoregressively predict all chunk identifiers in parallel (time steps $t_{1,2,3}$), while the token decoder conditions on the "ground truth" chunk identifiers to predict the target tokens in one shot (time step t_4). During inference (shown here), the token decoder conditions on the autoregressively predicted chunk identifiers. The encoder and token decoder contain $N \geq 1$ layers, while the parse decoder only requires $M = 1$ layers (see Table 4).

for longer target sequences in which there are more tokens to attend to at every block.

2.3 Generating multiple tokens per time step

As decoding time is a function of the number of decoding time steps (and consequently the number of passes through the decoder), faster inference can be obtained using methods that reduce the number of time steps. In autoregressive decoding, the number of time steps is equal to the target sentence length m; the most extreme alternative is (naturally) *non-autoregressive* decoding, which requires just a single time step by factorizing the target sequence probability as

$$p(t_{1,\cdots},t_m|s) = \prod_{i=1}^{m} p(t_i|s).$$

Here, all target tokens are produced *independently* of each other. While this formulation does indeed provide significant decoding speedups, translation quality suffers after dropping the dependencies between target tokens without additional expensive reranking steps (Gu et al., 2018, NAT) or iterative refinement with multiple decoders (Lee et al., 2018).

As fully non-autoregressive decoding results in poor translation quality, another class of methods produce k tokens at a single time step where $1 < k < m$. The *semi-autoregressive Transformer* (SAT) of Wang et al. (2018) produces a fixed k tokens per time step, thus modifying the target sequence probability to:

$$p(t_{1,\cdots},t_m|s) = \prod_{i=1}^{|G|} p(G_t|G_{1,\cdots},G_{t-1},x),$$

where each of $G_{1,\cdots},G_{\lfloor\frac{m-1}{k}\rfloor+1}$ is a group of contiguous non-overlapping target tokens of the form $t_{i,\cdots},t_{i+k}$. In conjunction with training techniques like knowledge distillation (Kim and Rush, 2016) and initialization with an autoregressive model, SATs maintain better translation quality than non-autoregressive approaches with competitive speedups. Stern et al. (2018) follow a similar approach but dynamically select a different k at each step, which results in further quality improvements with a corresponding decrease in speed.

2.4 Latent Transformer

While current semi-autoregressive methods achieve both better quality and faster speedups than their non-autoregressive counterparts, largely due to the number of tricks required to train the latter, the

theoretical speedup for non-autoregressive models is of course larger. The latent Transformer (Kaiser et al., 2018, LT) is similar to both of these lines of work: its decoder first autoregressively generates a sequence of discrete latent variables $l_1,\ldots,l_j$ and then non-autoregressively produces the entire target sentence $t_i,\ldots,t_m$ conditioned on the latent sequence. Two parameters control the magnitude of the speedup in this framework: the length of the latent sequence (j), and the size of the discrete latent space (K).

The LT is significantly more difficult to train than any of the previously-discussed models, as it requires passing the target sequence through what Kaiser et al. (2018) term a *discretization bottleneck* that must also maintain differentiability through the decoder. While LT outperforms the NAT variant of non-autoregressive decoding in terms of BLEU, it takes longer to decode. In the next section, we describe how we use syntax to address the following three weaknesses of LT:

1. generating the same number of latent variables j regardless of the length of the source sentence, which hampers output quality
2. relying on a large value of K (the authors report that in the base configuration as few as $\sim$3000 latents are used out of 2^{16} available), which hurts translation speed
3. the complexity of implementation and optimization of the discretization bottleneck, which negatively impacts both quality and speed.

3 Syntactically Supervised Transformers

Our key insight is that we can use syntactic information as a prox the learned discrete latent space of the LT. Specifically, instead of producing a sequence of latent discrete variables, our model produces a sequence of phrasal chunks derived from a constituency parser. During training, the chunk sequence prediction task is supervised, which removes the need for a complicated discretization bottleneck and a fixed sequence length j. Additionally, our chunk vocabulary is much smaller than that of the LT, which improves decoding speed.

Our model, the syntactically supervised Transformer (SynST), follows the two-stage decoding setup of the LT. First, an autoregressive decoder generates the phrasal chunk sequence, and then all of the target tokens are generated at once, conditioned on the chunks (Figure 2). The rest of this section fully specifies each of these two stages.

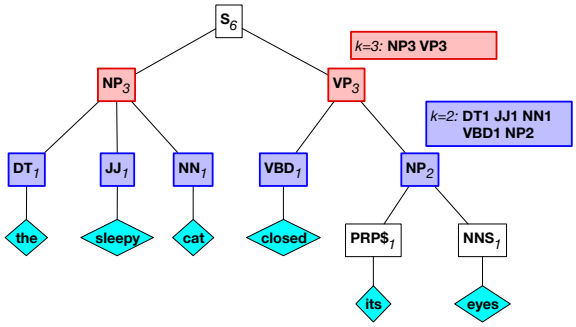

Figure 3: Example of our parse chunk algorithm with max span sizes $k = 2, 3$. At each visited node during an in-order traversal of the parse, if the subtree size is less than or equal to k, we append a corresponding chunk identifier to our sequence.

3.1 Autoregressive chunk decoding

Intuitively, our model uses syntax as a scaffold for the generated target sentence. During training, we acquire supervision for the syntactic prediction task through an external parser in the target language. While we could simply force the model to predict the entire linearized parse minus the terminals,[3] this approach would dramatically increase the number of autoregressive steps, which we want to keep at a minimum to prioritize speed. To balance syntactic expressivity with the number of decoding time steps, we apply a simple chunking algorithm to the constituency parse.

Extracting chunk sequences: Similar to the SAT method, we first choose a maximum chunk size k. Then, for every target sentence in the training data, we perform an in-orde versal of its constituency p tree. At each visited node, if the number of leaves spanned by that node is less than or equal to k, we append a descriptive *chunk identifier* to the parse sequence before moving onto its sibling; otherwise, we proceed to the left child and try again. This process is shown for two different values of k on the same sentence in Figure 3. Each unique chunk identifier, which is formed by the concatenation of the constituent type and subtree size (e.g., `NP3`), is considered as an element of our first decoder's vocabulary; thus, the maximum size of this vocabulary is $|P| \times k$ where P is the

[3] This approach is used for paraphrase generation by Iyyer et al. (2018), who were not focused on decoding speed.

set of all unique constituent types.[4] Both parts of the chunk identifier (the constituent type and its size) are crucial to the performance of SynST, as demonstrated by the ablations in Section 5.

Predicting chunk sequences: Because we are fully supervising the chunk sequence prediction, both the encoder and parse decoder are architecturally identical to the encoder and decoder of the vanilla Transformer, respectively. The parse decoder differs in its target vocabulary, which is made up of chunk identifiers instead of word types, and in the number of layers (we use 1 layer instead of 6, as we observe diminishing returns from bigger parse decoders as shown in Section 5). Formally, the parse decoder autoregressively predicts a sequence of chunk identifiers $c_{1,\cdots,}c_p$ conditioned on the source sentence s[5] by modeling

$$p(c_{1,\cdots,}c_p|s) = \prod_{i=1}^{p} p(c_i|c_{1,\cdots,}c_{i-1}, s).$$

Unlike LT, the length p of the chunk sequence changes dynamically based on the length of the target sentence, which is reminiscent of the token decoding process in the SAT.

3.2 Non-autoregressive token decoding

In the second phase of decoding, we apply a single non-autoregressive step to produce the tokens of the target sentence by factorizing the target sequence probability as

$$p(t_{1,\cdots,}t_m|s) = \prod_{i=1}^{m} p(t_i|c_{1,\cdots,}c_p, s).$$

Here, all target tokens are produced independently of each other, but in contrast to the previously-described non-autoregressive models, we additionally condition each prediction on the entire chunk sequence. To implement this decoding step, we feed a chunk sequence as input to a second Transformer decoder, whose parameters are separate from those of the parse decoder. During training, we use the ground-truth chunk sequence as input, while at inference we use the predicted chunks.

Implementation details: To ensure that the number of input and output tokens in the second decoder are equal, which is a requirement of the Transformer decoder, we add placeholder `<MASK>` tokens to the chunk sequence, using the size component of each chunk identifier to determine where to place these tokens. For example, if the first decoder produces the chunk sequence `NP2 PP3`, our second decoder's input becomes `NP2 <MASK> <MASK> PP3 <MASK> <MASK> <MASK>`; this formulation also allows us to better leverage the Transformer's positional encodings. Then, we apply unmasked self-attention over this input sequence and predict target language tokens at each position associated with a `<MASK>` token.

4 Experiments

We evaluate the translation quality (in terms of BLEU) and the decoding speedup (average time to decode a sentence) of SynST compared to competing approaches. In a controlled series of experiments on four different datasets (En↔ De and En→ Fr language pairs),[6] we find that SynST achieves a strong balance between quality and speed, consistently outperforming the semi-autoregressive SAT on all datasets and the similar LT on the only translation dataset for which Kaiser et al. (2018) report results. In this section, we first describe our experimental setup and its differences to those of previous work before providing a summary of the key results.

4.1 Controlled experiments

Existing papers in non- and semi-autoregressive approaches do not adhere to a standard set of datasets, base model architectures, training tricks, or even evaluation scripts. This unfortunate disparity in evaluation setups means that numbers between different papers are uncomparable, making it difficult for practitioners to decide which method to choose. In an effort to offer a more meaningful comparison, we strive to keep our experimental conditions as close to those of Kaiser et al. (2018) as possible, as the LT is the most similar existing model to ours. In doing so, we made the following decisions:

- Our base model is the base vanilla Transformer (Vaswani et al., 2017) without any ar-

[4] In practice, this vocabulary is significantly smaller than the discrete latent space of the LT for reasonable values of k.

[5] In preliminary experiments, we also tried conditioning this decoder on the source parse, but we did not notice significant differences in translation quality.

[6] We explored translating to other languages previously evaluated in the non- and semi-autoregressive decoding literature, but could not find publicly-available, reliable constituency parsers for them.

Model	WMT En-De		WMT De-En		IWSLT En-De		WMT En-Fr	
	BLEU	Speedup	BLEU	Speedup	BLEU	Speedup	BLEU	Speedup
Baseline ($b = 1$)	25.82	1.15×	29.83	1.14×	28.66	1.16×	39.41	1.18×
Baseline ($b = 4$)	26.87	1.00×	30.73	1.00×	30.00	1.00×	40.22	1.00×
SAT ($k = 2$)	22.81	2.05×	26.78	2.04×	25.48	2.03×	36.62	2.14×
SAT ($k = 4$)	16.44	3.61×	21.27	3.58×	20.25	3.45×	28.07	3.34×
SAT ($k = 6$)	12.55	4.86×	15.23	4.27×	14.02	4.39×	24.63	4.77×
LT*	19.8	3.89×	-	-	-	-	-	-
SynST($k = 6$)	20.74	4.86×	25.50	5.06×	23.82	3.78×	33.47	5.32×

Table 1: Controlled experiments comparing SynST to a baseline Transformer, SAT, and LT on four different datasets (two language pairs) demonstrate speed and BLEU improvements. Wall-clock speedup is measured on a single Nvidia TitanX Pascal by computing the average time taken to decode a single sentence in the dev/test set, averaged over five runs. When beam width b is not specified, we perform greedy decoding (i.e., $b = 1$). Note that the LT results are reported by Kaiser et al. (2018) and not from our own implementation;[9] as such, they are not directly comparable to the other results.

chitectural upgrades.[7]

- We use all of the hyperparameter values from the original Transformer paper and do not attempt to tune them further, except for: (1) the number of layers in the parse decoder, (2) the decoders do not use label smoothing.
- We do not use sequence-level knowledge distillation, which augments the training data with translations produced by an external autoregressive model. The choice of model used for distillation plays a part in the final BLEU score, so we remove this variable.
- We report all our BLEU numbers using sacreBLEU (Post, 2018) to ensure comparability with future work.[8]
- We report wall-clock speedups by measuring the average time to decode one sentence (batch size of one) in the dev/test set.

As the code for LT is not readily available[9], we also reimplement the SAT model using our setup, as it is the most similar model outside of LT to our own.[10] For SynST, we set the maximum chunk size $k = 6$ and compare this model to the SAT trained with $k = 2, 4, 6$.

4.2 Datasets

We experiment with English-German and English-French datasets, relying on constituency parsers in all three languages. We use the Stanford CoreNLP (Manning et al., 2014) shift-reduce parsers for English, German, and French. For English-German, we evaluate on WMT 2014 En↔De as well as IWSLT 2016 En→De, while for English-French we train on the Europarl / Common Crawl subset of the full WMT 2014 En→Fr data and evaluate over the full dev/test sets. WMT 2014 En↔De consists of around 4.5 million sentence pairs encoded using byte pair encoding (Sennrich et al., 2016) with a shared source-target vocabulary of roughly 37000 tokens. We use the same preprocessed dataset used in the original Transformer paper and also by many subsequent papers that have investigated improving decoding speed, evaluating on the `newstest2013` dataset for validation and the `newstest2014` dataset for testing. For the IWSLT dataset we use `tst2013` for validation and utilize the same hyperparameters as Lee et al. (2018).

4.3 Results

Table 1 contains the results on all four datasets. SynST achieves speedups of $\sim 4 - 5\times$ that of the vanilla Transformer, which is larger than nearly all

[7] As the popular Tensor2Tensor implementation is constantly being tweaked, we instead re-implement the Transformer as originally published and verify that its results closely match the published ones. Our implementation achieves a BLEU of 27.69 on WMT'14 En-De, when using `multi-bleu.perl` from Moses SMT.

[8] SacreBLEU signature: BLEU+case.mixed+lang.LANG +numrefs.1+smooth.exp+test.TEST+tok.intl+version.1.2.11, with LANG ∈ {en-de, de-en, en-fr} and TEST ∈ {wmt14/full, iwslt2017/tst2013}

[9] We attempted to use the publicly available code in Tensor2Tensor, but were unable to successfully train a model.

[10] The published SAT results use knowledge distillation and different hyperparameters than the vanilla Transformer, most notably a tenfold decrease in training steps due to initializing from a pre-trained Transformer.

	Chunk types	SynST predictions with | separating syntax chunks
Words repeated in two separate syntax chunks (**blue**, **red**)	NP1, NP3	But | it | is | **enthusiasm** | in | **a great enthusiasm**
	NP3, PP4	... Enrique | Pena | Nieto | is | facing | **a difficult start** | **on a difficult start**
	NP2, PP3	Do | you | not | turn | **your voters** | **on your voters**
	Output type	**Output for a single example**
SynST reorders **syntax chunks**, which is fixed with gold parses (GP) as input	ground truth	Canada | was | the first country | to | make | photograph warnings | **mandatory in 2001**
	SynST	Canada | was | the first country | **in 2001** | to | propose | photographic warnings
	predicted parse	NP1 VBD1 NP3 **PP2** TO1 VB1 NP4
	SynST + GP	Canada | was | the first country | to | make | photographic warnings | **available in 2001**
	True chunk	**SynST predictions with @@ as subword divisions**
Wrong subword completion within a **syntax chunk**	ignores them	I | simply | **ign@@ it them**
	beforehand	Most ST@@ I | can | be | cur@@ ed | **be@@ foreh@@ ly**
	examines	Beg@@ inning | of | the course | which | **exam@@ ates** | the ...

Table 2: Common error made by SynST due to its syntactically informed semi-autoregressive decoding. Different syntax chunks have been separated by | symbols in all the decoded outputs.

of the SAT configurations. Quality-wise, SynST again significantly outperforms the SAT configurations at comparable speedups on all datasets. On WMT En-De, SynST improves by 1 BLEU over LT (20.74 vs LT's 19.8 without reranking).

Comparisons to other published work: As mentioned earlier, we adopt a very strict set of experimental conditions to evaluate our work against LT and SAT. For completeness, we also offer an unscientific comparison to other numbers in Table A1.

5 Analysis

In this section, we perform several analysis and ablation experiments on the IWSLT En-De dev set to shed more light on how SynST works. Specifically, we explore common classes of translation errors, important factors behind SynST's speedup, and the performance of SynST's parse decoder.

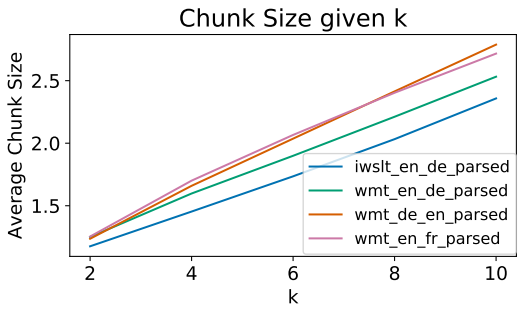

Figure 4: The average size of a chunk given a particular value of the max chunk size k.

5.1 Analyzing SynST's translation quality

What types of translation errors does SynST make? Through a qualitative inspection of SynST's output translations, we identify three types of errors that SynST makes more frequently than the vanilla Transformer: subword repetition, phrasal reordering, and inaccurate subword completions. Table 2 contains examples of each error type.

Do we need to include the constituent type in the chunk identifier? SynST's chunk identifiers contain both the constituent type as well as chunk size. Is the syntactic information actually useful during decoding, or is most of the benefit from the chunk size? To answer this question, we train a variant of SynST without the constituent identifiers, so instead of predicting NP3 VP2 PP4, for example, the parse decoder would predict 3 2 4. This model substantially underperforms, achieving a BLEU of 8.19 compared to 23.82 for SynST, which indicates that the syntactic information is of considerable value.

How much does BLEU improve when we provide the ground-truth chunk sequence? To get an upper bound on how much we can gain by improving SynST's parse decoder, we replace the input to the second decoder with the ground-truth chunk sequence instead of the one generated by the parse decoder. The BLEU increases from 23.8 to 41.5 with this single change, indicating that future work on SynST's parse decoder could prove very fruitful.

	Predicted parse vs. Gold parse (separate)	Predicted parse vs. Gold parse (joint)	Parsed prediction vs. Gold parse	Parsed prediction vs. Predicted parse
F1	65.48	69.64	79.16	89.90
Exact match	4.23%	5.24%	5.94%	43.10%

Table 3: F1 and exact match comparisons of predicted chunk sequences (from the parse decoder), ground-truth chunk sequences (from an external parser in the target language), and chunk sequences obtained after parsing the translation produced by the token decoder. First two columns show the improvement obtained by jointly training the two decoders. The third column shows that when the token decoder deviates from the predicted chunk sequence, it usually results in a translation that is closer to the ground-truth target syntax, while the fourth column shows that the token decoder closely follows the predicted chunk sequence.

5.2 Analyzing SynST's speedup

What is the impact of average chunk size on our measured speedup? Figure 4 shows that the IWSLT dataset, for which we report the lowest SynST speedup, has a significantly lower average chunk size than that of the other datasets at many different values of k.[11] We observe that our empirical speedup directly correlates with the average chunk size: ranking the datasets by empirical speedups in Table 1 results in the same ordering as Figure 4's ranking by average chunk size.

How does the number of layers in SynST's parse decoder affect the BLEU/speedup tradeoff? All SynST experiments in Table 1 use a single layer for the parse decoder. Table 4 shows that increasing the number of layers from 1 to 5 results in a BLEU increase of only 0.5, while the speedup drops from $3.8\times$ to $1.4\times$. Our experiments indicate that (1) a single layer parse decoder is reasonably sufficient to model the chunked sequence and (2) despite its small output vocabulary, the parse decoder is the bottleneck of SynST in terms of decoding speed.

# Layers	Max Chunk Size	Speedup	BLEU
1	$k = 6$	$3.8\times$	23.82
2	$k = 6$	$2.8\times$	23.98
3	$k = 6$	$2.2\times$	24.54
4	$k = 6$	$1.8\times$	24.04
5	$k = 6$	$1.4\times$	24.34
1	$k \in \{1 \dots 6\}$	$3.1\times$	25.31

Table 4: Increasing the number of layers in SynST's parse decoder significantly lowers the speedup while marginally impacting BLEU. Randomly sampling k from $\{1 \dots 6\}$ during training boosts BLEU significantly with minimal impact on speedup.

5.3 Analyzing SynST's parse decoder

How well does the predicted chunk sequence match the ground truth? We evaluate the generated chunk sequences by the parse decoder to explore how well it can recover the ground-truth chunk sequence (where the "ground truth" is provided by the external parser). Concretely, we compute the chunk-level F1 between the predicted chunk sequence and the ground-truth. We evaluate two configurations of the parse decoder, one in which it is trained separately from the token decoder (first column of Table 3), and the other where both decoders are trained jointly (second column of Table 3). We observe that joint training boosts the chunk F1 from 65.4 to 69.6, although, in both cases the F1 scores are relatively low, which matches our intuition as most source sentences can be translated into multiple target syntactic forms.

How much does the token decoder rely on the predicted chunk sequence? If SynST's token decoder produces the translation "the man went to the store" from the parse decoder's prediction of `PP3 NP3`, it has clearly ignored the predicted chunk sequence. To measure how often the token decoder follows the predicted chunk sequence, we parse the generated translation and compute the F1 between the resulting chunk sequence and the parse decoder's prediction (fourth column of Table 3). Strong results of 89.9 F1 and 43.1% exact match indicate that the token decoder is heavily reliant on the generated chunk sequences.

When the token decoder deviates from the predicted chunk sequence, does it do a better job matching the ground-truth target syntax? Our next experiment investigates why the token decoder sometimes ignores the predicted chunk sequence. One

[11] IWSLT is composed of TED talk subtitles. A small average chunk size is likely due to including many short utterances.

hypothesis is that it does so to correct mistakes made by the parse decoder. To evaluate this hypothesis, we parse the predicted translation (as we did in the previous experiment) and then compute the chunk-level F1 between the resulting chunk sequence and the *ground-truth* chunk sequence. The resulting F1 is indeed almost 10 points higher (third column of Table 3), indicating that the token decoder does have the ability to correct mistakes.

What if we vary the max chunk size k during training? Given a fixed k, our chunking algorithm (see Figure 3) produces a deterministic chunking, allowing better control of SynST's speedup, even if that sequence may not be optimal for the token decoder. During training we investigate using $k' = \min(k, \sqrt{T})$, where T is the target sentence length (to ensure short inputs do not collapse into a single chunk) and randomly sampling $k \in \{1 \dots 6\}$. The final row of Table 4 shows that exposing the parse decoder to multiple possible chunkings of the same sentence during training allows it to choose a sequence of chunks that has a higher likelihood at test time, improving BLEU by 1.5 while decreasing the speedup from $3.8\times$ to $3.1\times$; this is an exciting result for future work (see Table A3 for additional analysis).

6 Related Work

Our work builds on the existing body of literature in both fast decoding methods for neural generation models as well as syntax-based MT; we review each area below.

6.1 Fast neural decoding

While all of the prior work described in Section 2 is relatively recent, non-autoregressive methods for decoding in NMT have been around for longer, although none relies on syntax like SynST. Schwenk (2012) translate short phrases non-autoregressively, while Kaiser and Bengio (2016) implement a non-autoregressive neural GPU architecture and Libovick and Helcl (2018) explore a CTC approach. Guo et al. (2019) use phrase tables and word-level adversarial methods to improve upon the NAT model of Gu et al. (2018), while Wang et al. (2019) regularize NAT by introducing similarity and back-translation terms to the training objective.

6.2 Syntax-based translation

There is a rich history of integrating syntax into machine translation systems. Wu (1997) pioneered this direction by proposing an inverse transduction grammar for building word aligners. Yamada and Knight (2001) convert an externally-derived source parse tree to a target sentence, the reverse of what we do with SynST's parse decoder; later, other variations such as string-to-tree and tree-to-tree translation models followed (Galley et al., 2006; Cowan et al., 2006). The Hiero system of Chiang (2005) employs a learned synchronous context free grammar within phrase-based translation, which follow-up work augmented with syntactic supervision (Zollmann and Venugopal, 2006; Marton and Resnik, 2008; Chiang et al., 2008).

Syntax took a back seat with the advent of neural MT, as early sequence to sequence models (Sutskever et al., 2014; Luong et al., 2015) focused on architectures and optimization. Sennrich and Haddow (2016) demonstrate that augmenting word embeddings with dependency relations helps NMT, while Shi et al. (2016) show that NMT systems do not automatically learn subtle syntactic properties. Stahlberg et al. (2016) incorporate Hiero's translation grammar into NMT systems with improvements; similar follow-up results (Aharoni and Goldberg, 2017; Eriguchi et al., 2017) directly motivated this work.

7 Conclusions & Future Work

We propose SynST, a variant of the Transformer architecture that achieves decoding speedups by autoregressively generating a constituency chunk sequence before non-autoregressively producing all tokens in the target sentence. Controlled experiments show that SynST outperforms competing non- and semi-autoregressive approaches in terms of both BLEU and wall-clock speedup on En-De and En-Fr language pairs. While our method is currently restricted to languages that have reliable constituency parsers, an exciting future direction is to explore unsupervised tree induction methods for low-resource target languages (Drozdov et al., 2019). Finally, we hope that future work in this area will follow our lead in using carefully-controlled experiments to enable meaningful comparisons.

Acknowledgements

We thank the anonymous reviewers for their insightful comments. We also thank Justin Payan and the rest of the UMass NLP group for helpful comments on earlier drafts. Finally, we thank Weiqiu You for additional experimentation efforts.

References

Roee Aharoni and Yoav Goldberg. 2017. Towards string-to-tree neural machine translation. In *Proceedings of the Association for Computational Linguistics*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. In *Proceedings of the International Conference on Learning Representations*.

David Chiang. 2005. A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the Association for Computational Linguistics*, pages 263–270. Association for Computational Linguistics.

David Chiang, Yuval Marton, and Philip Resnik. 2008. Online large-margin training of syntactic and structural translation features. In *Proceedings of Empirical Methods in Natural Language Processing*, pages 224–233. Association for Computational Linguistics.

Brooke Cowan, Ivona Kučerová, and Michael Collins. 2006. A discriminative model for tree-to-tree translation. In *Proceedings of Empirical Methods in Natural Language Processing*, pages 232–241. Association for Computational Linguistics.

Andrew Drozdov, Patrick Verga, Mohit Yadav, Mohit Iyyer, and Andrew McCallum. 2019. Unsupervised latent tree induction with deep inside-outside recursive autoencoders. In *Conference of the North American Chapter of the Association for Computational Linguistics*.

Akiko Eriguchi, Yoshimasa Tsuruoka, and Kyunghyun Cho. 2017. Learning to parse and translate improves neural machine translation. In *Proceedings of the Association for Computational Linguistics*. Association for Computational Linguistics (ACL).

Michel Galley, Jonathan Graehl, Kevin Knight, Daniel Marcu, Steve DeNeefe, Wei Wang, and Ignacio Thayer. 2006. Scalable inference and training of context-rich syntactic translation models. In *Proceedings of the Association for Computational Linguistics*, pages 961–968. Association for Computational Linguistics.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor OK Li, and Richard Socher. 2018. Non-autoregressive neural machine translation. In *Proceedings of International Conference on Learning Representations*.

Junliang Guo, Xu Tan, Di He, Tao Qin, Linli Xu, and Tie-Yan Liu. 2019. Non-Autoregressive Neural Machine Translation with Enhanced Decoder Input. In *Association for the Advancement of Artificial Intelligence*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*.

Mohit Iyyer, John Wieting, Kevin Gimpel, and Luke Zettlemoyer. 2018. Adversarial example generation with syntactically controlled paraphrase networks. In *Conference of the North American Chapter of the Association for Computational Linguistics*.

Łukasz Kaiser and Samy Bengio. 2016. Can active memory replace attention? In *Proceedings of Advances in Neural Information Processing Systems*, pages 3781–3789.

Lukasz Kaiser, Samy Bengio, Aurko Roy, Ashish Vaswani, Niki Parmar, Jakob Uszkoreit, and Noam Shazeer. 2018. Fast decoding in sequence models using discrete latent variables. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2390–2399, Stockholmsmässan, Stockholm Sweden. PMLR.

Yoon Kim and Alexander M Rush. 2016. Sequence-level knowledge distillation. In *Proceedings of Empirical Methods in Natural Language Processing*, pages 1317–1327.

Jason Lee, Elman Mansimov, and Kyunghyun Cho. 2018. Deterministic non-autoregressive neural sequence modeling by iterative refinement. In *Proceedings of Empirical Methods in Natural Language Processing*, pages 1173–1182.

Jindich Libovick and Jindich Helcl. 2018. End-to-End Non-Autoregressive Neural Machine Translation with Connectionist Temporal Classification. In *Proceedings of Empirical Methods in Natural Language Processing*.

Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of Empirical Methods in Natural Language Processing*, pages 1412–1421.

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.

Yuval Marton and Philip Resnik. 2008. Soft syntactic constraints for hierarchical phrased-based translation. *Proceedings of the Association for Computational Linguistics*, pages 1003–1011.

Matt Post. 2018. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191. Association for Computational Linguistics.

Holger Schwenk. 2012. Continuous space translation models for phrase-based statistical machine translation. *Proceedings of International Conference on Computational Linguistics*, pages 1071–1080.

Rico Sennrich and Barry Haddow. 2016. Linguistic input features improve neural machine translation. In *Proceedings of the First Conference on Machine Translation: Volume 1, Research Papers*, volume 1, pages 83–91.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1715–1725.

Xing Shi, Inkit Padhi, and Kevin Knight. 2016. Does string-based neural mt learn source syntax? In *Proceedings of Empirical Methods in Natural Language Processing*, pages 1526–1534.

F Stahlberg, E Hasler, A Waite, and B Byrne. 2016. Syntactically guided neural machine translation. In *Proceedings of the Association for Computational Linguistics*, volume 2, pages 299–305. Association for Computational Linguistics.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. 2018. Blockwise parallel decoding for deep autoregressive models. In *Advances in Neural Information Processing Systems*, pages 10106–10115.

Emma Strubell, Patrick Verga, Daniel Andor, David Weiss, and Andrew McCallum. 2018. Linguistically-Informed Self-Attention for Semantic Role Labeling. In *Proceedings of Empirical Methods in Natural Language Processing*, Brussels, Belgium.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Proceedings of Advances in Neural Information Processing Systems*, pages 3104–3112.

Swabha Swayamdipta, Sam Thomson, Kenton Lee, Luke Zettlemoyer, Chris Dyer, and Noah A Smith. 2018. Syntactic scaffolds for semantic structures. In *Proceedings of Empirical Methods in Natural Language Processing*, pages 3772–3782.

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.

Chunqi Wang, Ji Zhang, and Haiqing Chen. 2018. Semi-autoregressive neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 479–488.

Yiren Wang, Fei Tian, Di He, Tao Qin, ChengXiang Zhai, and Tie-Yan Liu. 2019. Non-Autoregressive Machine Translation with Auxiliary Regularization. In *Association for the Advancement of Artificial Intelligence*.

Dekai Wu. 1997. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational linguistics*, 23(3):377–403.

Kenji Yamada and Kevin Knight. 2001. A syntax-based statistical translation model. In *Proceedings of the Association for Computational Linguistics*.

Andreas Zollmann and Ashish Venugopal. 2006. Syntax augmented machine translation via chart parsing. In *Proceedings of the Workshop on Statistical Machine Translation*, pages 138–141. Association for Computational Linguistics.

Appendix

A Unscientific Comparison

We include a reference to previously published work in comparison to our approach. Note, that many of these papers have multiple confounding factors that make direct comparison between approaches very difficult.

Model	WMT En-De	
	BLEU	Speedup
LT rescoring top-100	22.5	-
NAT rescoring top-100	21.54	-
BPT ($k = 6$)	28.11	3.10×
IRT (adaptive)	21.54	2.39×
SAT ($k = 6$)	23.93	5.58×
SynST($k = 6$)	20.74	4.86×

Table A1: Unscientific comparison against previously published works. The numbers of each model are taken from their respective papers. These previous results often have uncomparable hyperparameters, compute their BLEU with `multi-bleu.perl`, and/or require additional steps such as knowledge distillation and re-ranking to achieve their reported numbers. Latent Transformer (LT) (Kaiser et al., 2018), Non-autoregressive Transformer (NAT) (Gu et al., 2018), Blockwise parallel Transformer (BPT) (Stern et al., 2018), Iterative refinement Transformer (IRT) (Lee et al., 2018), Semi-autoregressive Transformer (SAT) (Wang et al., 2018).

B The impact of beam search

In order to more fully understand the interplay of the representations output from the autoregressive parse decoder on the BLEU/speedup tradeoff we examine the impact of beam search for the parse decoder. From Table A2 we see that beam search does not consitently improve the final translation quality in terms of BLEU (it manages to decrease BLEU on IWSLT), while providing a small reduction in overall speedup for SynST.

C SAT replication results

As part of our work, we additionally replicated the results of (Wang et al., 2018). We do so without any of the additional training stabilization techniques they use, such as knowledge distillation or initializing from a pre-trained Transformer. Without the use of these techniques, we notice that the approach sometimes catastrophically fails to converge to a meaningful representation, leading to sub-optimal translation performance, despite achieving adequate perplexity. In order to report accurate translation performance for SAT, we needed to re-train the model for $k = 4$ when it produced BLEU scores in the single digits.

D Parse performance when varying max chunk size k

In Section 5.3 (see the final row of Table 3) we consider the effect of randomly sampling the max chunk size k during training. This provides a considerable boost to BLEU with a minimal impact to speedup. In Table A3 we highlight the impact to the parse decoder's ability to predict the ground-truth chunk sequences and how faithfully it follows the predicted sequence.

Model	Beam Width	WMT En-De		WMT De-En		IWSLT En-De		WMT En-Fr	
		BLEU	Speedup	BLEU	Speedup	BLEU	Speedup	BLEU	Speedup
Transformer	1	25.82	1.15×	29.83	1.14×	28.66	1.16×	39.41	1.18×
Transformer	4	26.87	1.00×	30.73	1.00×	30.00	1.00×	40.22	1.00×
SAT ($k = 2$)	1	22.81	2.05×	26.78	2.04×	25.48	2.03×	36.62	2.14×
SAT ($k = 2$)	4	23.86	1.80×	27.27	1.82×	26.25	1.82×	37.07	1.89×
SAT ($k = 4$)	1	16.44	3.61×	21.27	3.58×	20.25	3.45×	28.07	3.34×
SAT ($k = 4$)	4	18.95	3.25×	23.20	3.19×	20.75	2.97×	32.62	3.08×
SAT ($k = 6$)	1	12.55	4.86×	15.23	4.27×	14.02	4.39×	24.63	4.77×
SAT ($k = 6$)	4	14.99	4.15×	19.51	3.89×	15.51	3.78×	28.16	4.19×
LT*	-	19.8	3.89×	-	-	-	-	-	-
SynST($k = 6$)	1	20.74	4.86×	25.50	5.06×	23.82	3.78×	33.47	5.32×
SynST($k = 6$)	4	21.61	3.89×	25.77	4.07×	23.31	3.11×	34.10	4.47×

Table A2: Controlled experiments comparing SynST to LT and SAT on four different datasets (two language pairs) demonstrate speed and BLEU improvements while varying beam size. Wall-clock speedup is measured on a single Nvidia TitanX Pascal by computing the average time taken to decode a single sentence in the dev/test set, averaged over five runs. Note that the LT results are reported by Kaiser et al. (2018) and not from our own implementation; as such, they are not directly comparable to the other results.

	Max Chunk Size	Predicted parse vs. Gold parse	Parsed prediction vs. Gold parse	Parsed prediction vs. Predicted parse
F1	$k = 6$	69.64	79.16	89.90
Exact match		5.24%	5.94%	43.10%
F1	$k \in \{1 \dots 6\}$	75.35	79.78	95.28
Exact match		4.83%	7.55%	50.15%

Table A3: F1 and exact match comparisons of predicted chunk sequences (from the parse decoder), ground-truth chunk sequences (from an external parser in the target language), and chunk sequences obtained after parsing the translation produced by the token decoder. The first column shows how well the parse decoder is able to predict the ground-truth chunk sequence when trained jointly with the token decoder. The second column shows that when the token decoder deviates from the predicted chunk sequence, it usually results in a translation that is closer to the ground-truth target syntax, while the third column shows that the token decoder closely follows the predicted chunk sequence. Randomly sampling k from $\{1 \dots 6\}$ during training significantly boosts the parse decoder's ability to recover the ground-truth chunk sequence compared to using a fixed $k = 6$. Subsequently the token decoder follows the chunk sequence more faithfully.