GiLT: Augmenting Transformer Language Models with Dependency Graphs

Anonymous ACL submission

Abstract

Augmenting Transformers with linguistic structures effectively enhances the syntactic generalization performance of language models. Previous work in this direction focuses on syntactic tree structures of languages, in particular constituency tree structures. We propose *Graph-Infused Layers Transformer Language Model* (GiLT) which leverages dependency graphs for augmenting Transformers language model. Unlike most previous work, GiLT dose not insert extra structural tokens in language modeling; instead, it injects structural information into language modeling by modulating attention weights in the Transformer with features extracted from the dependency graph that is incrementally constructed along with token prediction. In our experiments, GiLT with semantic dependency graphs achieves better syntactic generalization while maintaining competitive perplexity in comparison with Transformer language model baselines. In addition, GiLT can be finetuned from a pretrained language model to achieve improved downstream task performance. Our code is available for release.

1 Introduction

Transformer-based language models (LMs) have shown excellent performance in language modeling and downstream tasks (Vaswani et al., 2017). Notably, linguistic structures such as syntactic and semantic parses that were deemed essential in traditional natural language processing are absent from the model design and training process of Transformer-based LMs.

Over the past decade, a number of researches have been trying to integrate linguistic structures into neural language models. Among them are syntactic LMs which jointly model syntactic structures and surface words (Choe and Charniak, 2016). These include earlier work such as RNNG, which combines constituency parsing with recurrent neural networks (Dyer et al., 2016; Kim et al., 2019; Noji and Oseki, 2021), and recent studies that incorporate constituency and dependency syntax into Transformers (Yoshida and Oseki, 2022; Qian et al., 2021; Sartran et al., 2022; Murty et al., 2023; Zhao et al., 2024). Experiments have shown their competitive perplexity in language modeling and improved syntactic generalization compared with standard Transformer LMs. However, existing researches in this direction have two major limitations. First, most of them are based on constituency syntactic tree structures. Dependency tree structures, another important form of syntax, receive much less attention (Zhao et al., 2024). In addition, little work has been done to jointly model linguistic structures other than syntactic trees in LMs. Second, most of the existing methods require additional tree-building operations inserted into the input and output sequence, leading to longer sequence lengths and higher computation cost and also making it harder to finetune a pretrained LM into a syntactic LM. The only exception is Pushdown Layers (Murty et al., 2023), which uses syntactic trees to influence attention computation and does not change the input and output space of the LM.

In this paper, we propose Graph-Infused Layer Transformer LM (GiLT) that tackles the above-mentioned limitations in integrating linguistic structures into Transformer-based LMs. GiLT is based on *dependency graphs* that subsume both syntactic dependency trees and *semantic* dependency graphs, thus extending the line of research on syntactic LMs beyond syntax. Inspired by Pushdown Layers (Murty et al., 2023), GiLT incrementally constructs dependency graphs without changing the input and output space of the underlying LM, and moderates attention scores with features extracted from the constructed dependency graphs.

Experimental results show that GiLT achieves competitive perplexity performance in language modeling and improved syntactic generalization

over baselines. Furthermore, finetuning pretrained language models by moderating attention scores with our proposed Graph-Infused layers can effectively improve the performance on downstream tasks, which suggests that the Graph-Infused layer is a competitive replacement for standard self-attention.

In summary, our contributions are as follows:

- We propose GiLT, leveraging dependency graphs to enhance LM without modifying the input or output space.
- We introduce feature tape extracted explicitly from partially built dependency graphs to moderate attention scores.
- Experimental results on language modeling and syntactic generalization validate the effectiveness of Graph-Infused layers.

2 Background

2.1 Pushdown Layers

Transformer LM with Pushdown Layers (Pushdown-LM) (Murty et al., 2023) is a type of syntactic LMs that incrementally build constituency syntactic trees and moderate attention scores based on the constituency trees. Unlike other syntactic LMs, it does not change the input and output space of the underlying LM.

At each decoding step i, Pushdown-LM predicts shift/reduce operations to simulate the status of a pushdown automaton that corresponds to the partially built constituency tree, and records on a *stack tape* W_i the depth of all the tokens that are already generated in the partially built constituency tree.

Pushdown-LM then augments self-attention with stack tape W_i:

$$\tilde{\alpha}^l_{ij} = [h^l_j + d^l_{ij}]^\top \mathbf{W}^\top_{\text{key}} \mathbf{W}_{\text{query}} h^l_i \qquad (1)$$

where $\tilde{\alpha}^l_{ij}$ is the attention score before softmax assigned to the jth token from the ith token at layer l, h^l_j is the hidden states of jth token at the lth attention block, d^l_{ij} is an embedding of the depth of the jth token recorded in W_i, and W_{key} and W_{query} are learnable parameters in self-attention. In this way, structural information from the constituency tree is implicitly introduced into self-attention computation and thus influence the decoding of LM.

2.2 Semantic Dependency Graphs

A Semantic Dependency Graph forms a directed acyclic graph instead of a tree. The dependency arcs in the graph, where nodes correspond to words, illustrate semantic relations (*e.g.*, agent and patient (Palmer et al., 2005)). The graph often includes a virtual root node.

In this paper, we consider three types of semantic dependency graphs from Oepen et al. (2015) as discussed below. DELPH-IN MRS-Derived Bi-Lexical Dependencies (Flickinger et al., 2012) (DM) are derived from Deep Bank (Flickinger, 2000), in which roots designate the highest-scoping predicate in the graph. Enju Predicate–Argument Structures (PAS) originate from Enju Treebank (Miyao, 2006), which is obtained by automatically annotating the PTB. The root of PAS denotes the semantic head in the sentence. Prague Semantic Dependencies (PSD) is based on Prague Czech-English Dependency Treebank (Hajic et al., 2012), where the roots mostly correspond to main verbs.

3 Graph-Infused Layers

We introduce a dependency-graph-based language model, *Graph-Infused Layers Transformer LM* (GiLT), which simultaneously generates tokens that form sentences and dependency arcs that incrementally form dependency graphs over the sentences. We also develop the *graph-based feature tape* to characterize generated tokens in the graph, and utilize it to influence attention computation.

3.1 Dependency Scoring

Whenever a word w_i is generated by the Transformer LM, we score all possible dependencies connected from and to w_i.

Since a word may correspond to multiple tokens, we first define word representation. Suppose word w_i is tokenized into m tokens whose input embeddings are $x_k, ..., x_{k+m-1}$ and hidden states are $h_k, ..., h_{k+m-1}$. We define h_k by concatenating the hidden states from the penultimate layer $L-1$ and middle layer $L/2$ to capture both semantic and syntactic information. We do not use the hidden state of the last layer to leave it only focused on next token prediction. We consider three methods for representing w_i: (i) complete token representation (CTR): the mean of $h_k, ..., h_{k+m-1}$; (ii) first token representation (FTR): the hidden state of the first token, h_k; (iii) representation ahead of time (RA): the concatenation of the hidden state of the

last token of the previous word w_{i-1} and the input embedding of the first token of word w_i, i.e., $[h_{k-1}; x_k]$. We choose RA by default for word representation in this work, because with RA we can predict all the dependencies of w_i before feeding the tokens of w_i into the Transformer, thus being able to infuse structural information from the dependency graph to the hidden states of the tokens in w_i and hence influence next token prediction of the Transformer LM. Empirically, we find that RA outperforms the other two methods in improving language modeling of GiLT. The disadvantage of RA is that its word representation clearly contains less information than the other two methods, resulting in worse dependency prediction accuracy.

We follow the biaffine parsing approach (Dozat and Manning, 2018) to compute the score p_{ij} of the dependency from the word w_i to w_j. Note that for the root node, we use a learnable vector as its word representation.

$$\begin{aligned} \tilde{\mathbf{o}}_i^p &= \text{MLP}_p^2(\text{MLP}_p^1(\mathbf{o}_i) + \mathbf{r}_{ii}) \\ \tilde{\mathbf{o}}_j^c &= \text{MLP}_c^2(\text{MLP}_c^1(\mathbf{o}_j) + \mathbf{r}_{ij}) \\ p_{ij} &= \sigma([\tilde{\mathbf{o}}_i^p; 1]^\top \mathbf{W_p} [\tilde{\mathbf{o}}_j^\mathbf{c}; 1]) \end{aligned} \tag{2}$$

where $\mathbf{o}_i$ is the word representation of w_i as discussed earlier, $\tilde{\mathbf{o}}_i \in \mathbb{R}^n$ is a vector with the same dimension as the hidden states of Transformer, $\mathbf{W_p} \in \mathbb{R}^{(n+1)\times(n+1)}$ is a learnable matrix, $\text{MLP}_{p/c}^{1/2}$ denotes the first/second MLP for computing parent/child representations, σ denotes the Sigmoid function, and $\mathbf{r}_{ij}$ represents positional encodings involving relative position $|i-j|$ and graph-based feature tape G_i (see Section 3.3).

3.2 Graph Update

Given dependency scores $\{p_{ij}, p_{ji}, p_{ii}\}$ $(j = 0, \cdots, i-1)$ for all the dependencies involving word w_i, a naive method to update the dependency graph is to predict the existence of each dependency independently. However, this becomes computationally infeasible when we employ beam search decoding (Section 3.5) because of the exponentially large search space. To address this issue, we follow a two-step method. First, we predict $a_i \in \{0, 1, \cdots, c\}$, the total number of dependencies of word w_i, where c is a constant upper bound.

$$\begin{aligned} \mathbf{s} &= \sum_{j=0}^{i} \tilde{\mathbf{o}}_i^p \odot \mathbf{W}_s \tilde{\mathbf{o}}_j^c + \sum_{j=0}^{i-1} \tilde{\mathbf{o}}_j^p \odot \mathbf{W}_s \tilde{\mathbf{o}}_i^c \\ \boldsymbol{\pi}_i &= \text{softmax}\left(\mathbf{W}_a^\top (\frac{\mathbf{s}}{\sqrt{2i-1}}) + b_a\right) \\ p(a_i) &= \max \pi_i \\ a_i &= \arg\max(\boldsymbol{\pi}_i) \end{aligned} \tag{3}$$

where $\odot$ denotes the hadamard product operation, $\mathbf{W}_a \in \mathbb{R}^{(c+1)\times n}$, $\mathbf{W}_s \in \mathbb{R}^{n\times n}$ and $\mathbf{b}_a \in \mathbb{R}^{c+1}$ are learnable parameters, $\boldsymbol{\pi}_i \in \mathbb{R}^{c+1}$ is the probability distribution with $c+1$ dimensions and a_i is the action we pick when using greedy decoding. Also, we scale $\mathbf{s}$ by $\sqrt{2i-1}$ to normalize the variance.

With the dependency number a_i picked from the distribution $\boldsymbol{\pi}_i$, we then simply rank all the dependencies based on their scores and add the top-a_i dependencies to the dependency graph. This two-step method reduces the search space from exponential to linear for each step.

3.3 Feature Extraction

Given input $x_{<k}$, we extract features from the partially constructed dependency graph and form a graph-based feature tape $G_k \in \mathbb{N}^{4\times k}$ for the token x_k. Note that the graph is word-level, but G_k corresponds to a token. Therefore, the values of $G_k, G_{k+1}, ...G_{k+m-1}$ are always the same for each word w_i consisting of m tokens $x_k, ..., x_{k+m-1}$. The feature tape involves four graph-based features ranging from fine-grained properties of words to coarse-grained properties of subgraphs: (i) degree, an attribute for each word; (ii) distance, measuring local topology between nodes; (iii) depth, representing the global structure of subgraphs; (iv) predicate depth, indicating relations between subgraphs.

Degree. The degree of a word is defined as the number of its incoming and outgoing dependencies c_{in} and c_{out} respectively. The value of out-degree plus in-degree is assigned as the degree feature for each word. Empirically, we discover that the weighted summation achieve better performance, so we assign weight m_{in} to in-degree and m_{out} to out-degree, where $m_{in} < m_{out}$ and update degree as $m_{out}c_{out} + m_{in}c_{in}$.

Distance. The distance between words w_i and w_j is defined to be the minimal sum of the weights

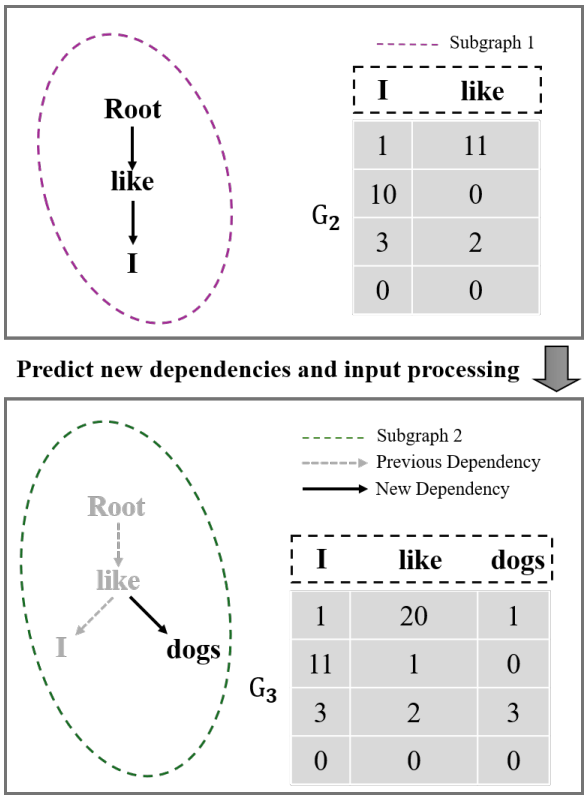

Figure 1: Illustration of how feature tape is updated when constructing a parse graph. Each row in G_2 and G_3 form top to bottom respectively are Degree values, Distance values, Depth values and Subgraph Distance values.$m_{in}$is 1 and m_{out} is 10 for this picture. As *dogs* is predicted, one dependency is added to the PSD graph.

along the dependencies connecting them. We allow both parent-to-child and child-to-parent passes over dependencies when traversing. Intuitively, the distance measures the relevance of the two nodes. In order to incorporate some dependency direction information into the distance measure, we assign weights m_{out} and m_{in} to parent-to-child and child-to-parent passes similar to the setting of Degree.

Depth. For each connected component in the partial dependency graph, we define its root as the first word from left to right. For each other node in the subgraph, we apply the breadth-first search algorithm on the undirected version of the subgraph and assign the depth (from the root) as the total (minimum) number of edges of the path from the root to this node. As an example, the root of one subgraph has a depth 0, and 1-hop nodes from the root defined on the undirected version of the subgraph have 1 as their depth values.

Subgraph Distance. Since we build the dependency graph incrementally, there may exist a sequence of multiple disconnected subgraphs when generating the k-th token. We define the words without incoming dependencies as the roots of each subgraph g, and define the Subgraph Distance of these root words as the number of subgraphs to the right of g (including g itself). We define the Subgraph Distance of non-root words to be 0. Subgraph Distance reflects relative positions between subgraphs.

3.4 Computing Attention Scores

We incorporate information contained in the graph-based feature tape G_k into the Transformer LM by modifying the self-attention module. Specifically, we first map contents of $G_k \in \mathbb{N}^{4\times k}$ onto a global embedding $\tilde{e}_k^G \in \mathbb{R}^{4\times k\times \tilde{d}}$. For each Transformer layer l, we use a linear function f_l for feature fusion, that is, $e_k^{Gl} = f_l(e_k^G) \in \mathbb{R}^{k\times d}$. For index of token $j \in \{0, 1, \cdots, k\}$, the corresponding fused graph feature $e_{kj}^{Gl} \in \mathbb{R}^d$ is directly added to its key in attention computation,

$$\tilde{\alpha}_{kj}^l = [h_j^l + e_{kj}^{Gl}]^\top \mathbf{W}_{\text{key}}^\top \mathbf{W}_{\text{query}} h_k^l \quad (4)$$

In this work, we follow TXL (Dai et al., 2019) for attention computation, so we additionally transform Equation 4 as follows.

$$\begin{aligned}\tilde{\alpha}_{kj}^l = {h_j^l}^\top \mathbf{W}_{\text{key, c}}^\top \mathbf{W}_{\text{query}} h_k^l \\ + (R_{kj} + e_{kj}^{Gl})^\top \mathbf{W}_{\text{key, r}}^\top \mathbf{W}_{\text{query}} h_j^l \\ + u^\top \mathbf{W}_{\text{key, c}} h_k^l + v^\top \mathbf{W}_{\text{key, r}} (R_{kj} + e_{kj}^{Gl}),\end{aligned} \quad (5)$$

where R_{kj} is a vector with sinusoid encoding of $|k-j|$, $W_{\text{key, c}}$ and $W_{\text{key, r}}$ are key matrix for respectively content and relative representation. Content and relative attention scores are divided into two parts. Since $|k-j| \in [0, k]$, TXL precomputes relative scores for k values instead of $k \times k$ matrix.

3.5 Training and Inference

Training. Given a corpus of strings annotated with parse graphs, we precompute graph-based feature tape G_k for each prefix $x_{\leq k}$ baesd on the ground-truth dependencies over $x_{\leq k}$. Afterwards, GiLT can be trained in parallel like a standard Transformer. During training, teacher forcing is applied to both token prediction and dependency prediction. Given a string x with N tokens, ground-truth action sequence $\{a_1, a_2, \cdots, a_M\}$ of length M based on the annotated graphs and matrix $\hat{p}$ where $\hat{p}_{ij} = 1$, if there is a dependency from i to j.

The training loss function is defined as follow:

$$\mathcal{L} = \beta \frac{1}{N} \sum_{i=1}^{N} \text{CE}(x_i, \hat{x_i}) + (1-\beta)(\frac{1}{M^2} \sum_{j=1}^{M} \sum_{k=1}^{M} \text{BCE}(p_{jk}, \hat{p_{jk}}) + \frac{1}{M} \sum_{k=1}^{M} \text{CE}(\pi_k, \hat{\pi_k})) \quad (6)$$

where β is a constant coefficient, N denotes the total token number, M denotes the total word number, x and $\hat{x}$ are the output distribution in each step and ground-truth one-hot vector, and $\hat{\pi_k}$ is based on the action sequence $\{a_1, a_2, \cdots, a_M\}$.

Inference. GiLT jointly models a string and its parse graph. For any string x with length N and a parse graph g with totally M actions $\{a_1, a_2, \cdots, a_M\}$, their joint probability $p(x, g)$ is computed as the product of token and action probabilities:

$$p(x, g) = \prod_{k=1}^{N} p(x_k | x_{<k}; G_{k-1}) \times \prod_{k=1}^{M} p(a_k | x_{<k}; G_{k-1}) \quad (7)$$

Computing the marginal $p(x) = \sum_g p(x, g)$ is computationally intractable due to the huge space of all possible graphs. Following Murty et al. (2023), we approximate it by marginalizing over a relatively small set of parse graphs produced via beam search. The probability $p(x)$ computed in this way is an exact lower bound of its true value. Note that, since GiLT does not generate extra tokens representing parsing actions in the output sequence, we do not need to use complicated word-synchronous beam search decoding (Stern et al., 2017), which has been widely used in previous syntactic LMs.

4 Experiments

4.1 Sentence-Level Language Modeling

Dataset and prepocessing. We use the BLLIP-LG dataset of Charniak et al. (2000), with training splits from Hu et al. (2020). We obtain annotated PSD, PAS and DM parse graphs with unlabelled dependency edges by parsing the dataset with the ACE parser (Wang et al., 2021). Since dependency trees can be seen as special cases of dependency graphs, we also obtain unlabeled projective dependency trees with the Biaffine-roberta parser (Dozat and Manning, 2017) following Zhao et al. (2024). Tokenization is performed with the same scheme as in Sartran et al. (2022) with Sentence Piece (Kudo and Richardson, 2018). We follow Murty et al. (2023) and model each sentence independently.

Model		PPL ↓	10%BLiMP ↑	SG ↑
TXL (baseline)		14.8	71.2	76.1
Pushdown-LM		**14.6**	70.1	78.2
Ours	GiLT-DP	15.6	70.1	75.5
	GiLT-DM	14.9	71.8	76.8
	GiLT-PAS	15.1	71.4	77.7
	GiLT-PSD	14.9	**72.5**	**80.2**

Table 1: Result of our models and baselines. The results of Pushdown-LM are reproduced using a TXL base model. All PPL results except for that of TXL are upper bounds.

Setup. We evaluate the perplexity of the models on the BLLIP-LG dataset. We train a 16-layer Transformer-XL (TXL) (Dai et al., 2019) language model with 252M parameters as the Baseline. To fairly compare with Pushdown-LM (Murty et al., 2023), we reimplement Pushdown-LM based on TXL instead of vanilla Transformer. We use PSD, DM and PAS parse graphs to train our GiLT respectively, resulting in three models: GiLT-PSD, GiLT-DM and GiLT-PAS. We also train GiLT on dependency parse trees, resulting in the GiLT-DP model. We use the same hyperparameters (model size, dropout, learning rate schedulers) as in Zhao et al. (2024) for TXL. We set the hyperparameters of GiLT with $w_{in} = 1$, $w_{out} = 10$, $\beta = \frac{5}{6}$, $\tilde{d} = 256$ (dimension of $\tilde{e}^G_{kj}$), and $d = 1024$ (dimension of e^G_{kj}). This results in 268M parameters for Transformer and 54M parameters for the modules described in Section 3.1- 3.4 in GiLT. For perplexity computation, we follow Pushdown-LM (Murty et al., 2023) and use beam search with a beam size of 300 to estimate the perplexity upper bound of GiLT.

Result. We report the perplexity (PPL) of all models in Table 1. Pushdown-LM achieves the best PPL and even outperforms the baseline, confirming previous observation that Pushdown-LM excels in language modeling among syntactic LMs (Murty et al., 2023). Our GiLT models based on

parse graphs maintain PPL levels that are comparable to both Pushdown-LM and the baseline. Notably, GiLT-DM and GiLT-PSD only underperform the baseline by 0.1. On the other hand, GiLT-DP achieves the worst PPL, implying the limitation of tree structures in comparison with graph structures.

4.2 Syntactic Generalization

We evaluate syntactic generalization on BLIMP (Warstadt et al., 2020) and the SG test suites (Hu et al., 2020).

Setup. We use the same models as in Section 4.1. For BLiMP, models are provided with a pair of sentences of which one is grammatical and the other is ungrammatical, and are tasked with assigning a higher probability to the grammatical sentence. Due to limited computational resources, we experiment with a subset of the BLiMP dataset by selecting every tenth example (i.e., the 1st, 11th, 21st, etc.). The SG test suites consist of six fine-grained syntactic phenomena. For each suite, there exists an inequality about the probability of generating a target span that the models should satisfy. For 10%BLiMP and SG, we use beam search to both compute marginal probability $p(x)$ and conditional probability $-\log p(x_t|x_{<t})$. Note that conditional probability is marginalized based on the beam state at time step t.

Result. The results are presented in Table 1. GiLT-PSD achieves the best performance in both tests, surpassing the baseline by 1.3 points in 10%BLiMP and 4.1 points in SG. GiLT-DM and GiLT-PAS also outperform the baseline in both tests. In contrast, Pushdown-LM exhibits better SG performance but worse 10%BLiMP performance than the baseline. GiLT-DP has the worst performance, underperforming the baseline in both tests, the reason of which is discussed in Appendix B. Figure 2 illustrates the scores of each suite in SG. Our best-performing GiLT-PSD outperforms the baseline in 5 out of 6 suites, while showing a slight disadvantage in *Gross Syntactic State*. Pushdown-LM performs well in most scenarios except licensing, which significantly degrades its overall SG score.

4.3 Finetuning on Pretrained LM

Since GiLT does not change the input and output space of the Transformer LM, it can be finetuned from any pretrained language model on datasets annotated with parse graphs to introduce syntactic inductive bias. We evaluate GiLT trained in this way on downstream tasks.

Setup. We use the pretrained TXL on BLLIP-LG in Section 4.1 to act as the base model. We replace the last *four* layers of pretrained TXL with Graph-Infused Layers, whose parameters involving the embedding for the graph-based feature tape and modules for scoring and constructing graphs are randomly initialized, and then finetune the model with silver parse graphs obtained by running our GiLT-PSD with beam search. We name the resulting model **GiLT-TXL**. We evaluate GiLT-TXL on four downstream text classification tasks from GLUE (Wang et al., 2018): RTE, SST2, MRPC and STS-B. We reduce each task into language modeling via prompting (details are provided in Appendix A).

Result. In Table 2, we report F1 score for SST2 and RTE, accuracy/F1 for MRPC, and Pearson/Spearman correlation for STS-B. We find that our GiLT-TXL outperforms the baseline on all the tasks. However, the F1 scores of RTE are both close to 0.5, indicating that both models struggle with this task, probably because our base model is pretrained on a rather small dataset and have limited ability in complicated tasks.

Model	RTE ↑	SST2 ↑	MRPC ↑	STS-B ↑
TXL	50.1	83.0	67.4/79.5	36.3/33.2
GiLT-TXL	**50.2**	**84.2**	**68.9/80.6**	**38.9/36.8**

Table 2: Results of finetuned models on downstream tasks.

4.4 Ablation for Representing Strategy

We compare three representing strategies mentioned in Section 3.1 for GiLT: (i) representation ahead of time (RA); (ii) first token representation (FTR); (iii) complete token representation (CTR). We train our models on $\frac{1}{4}$ of the BLLIP-LG dataset with PSD dependency graphs and compare the F1 score of dependency prediction and average log joint probability $\frac{1}{N}\log p(x, g)$ on the test set with the best model evaluating on validation set during training in Table 3, where N is the total number of tokens in the test set. Unsurprisingly, RA achieves the lowest F1 score because it receives least informative word representation compared with FTR and CTR. However, RA update the graph earliest and is able to utilize the inductive bias brought by

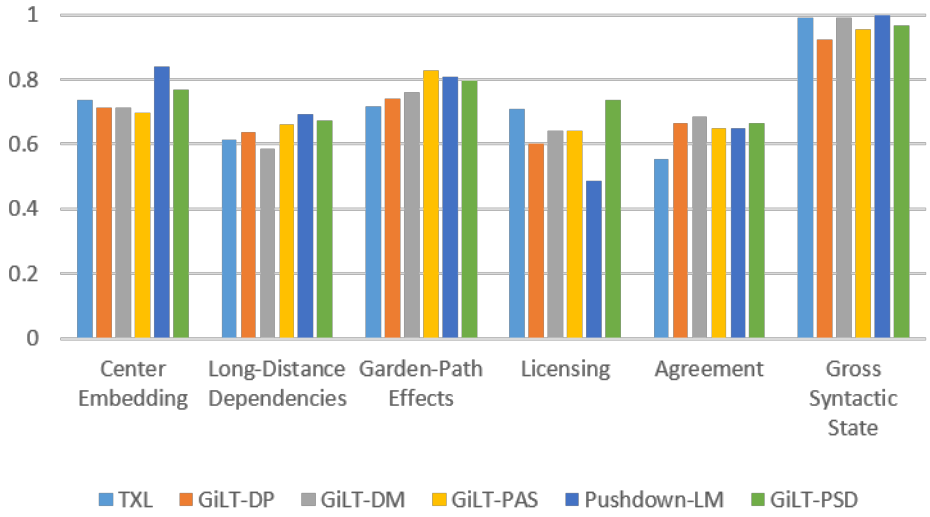

Figure 2: Scores on the six circuits of the SG test suites. The order from left to right in the table corresponds to the order of scores of SG from low to high.

Method	F1 ↑	Log Joint Probability ↑
RA	89.43	**-3.27**
FTR	92.64	-3.35
CTR	**94.1**	-3.31

Table 3: The F1 score of dependencies and average log joint probability of the test set when training with different representing strategies

the graph in our Graph-Infused layer, which helps to improve the generation ability greatly. Theoretically, although FTR is able to update graph earilier than CTR and more informative word representation than RA, FTR behave worst.

5 Related Work

There has been a line of studies about leveraging recursive linguistic structural information for sequential language modeling. RNNGs (Dyer et al., 2016), known as syntactic LMs, jointly model the syntactic structure and words by integrating top-down transition-based constituency parsing into a recursive neural network, while more recent studies (Qian et al., 2021; Yoshida and Oseki, 2022; Sartran et al., 2022) have applied this approach to the Transformer architecture. They explicitly model the syntactic tree along with words by imposing hard constraints over attention masks to simulate the shift/compose operations in transition-based parsing, which serves as a bottleneck in information gathering to force better representation learning of compositions. Hu et al. (2024) further explores an unsupervised training framework for constituency-based syntactic LMs, showing the potential of training syntactic LMs at scale.

In addition to constituency-based models mentioned above, studies on neural models based on dependency tree structures (Buys and Blunsom, 2015; Mirowski and Vlachos, 2015), which is another important form of syntax, also achieve improved syntactic generalization performance. A recent example is Dependency Transformer Grammars (Zhao et al., 2024), which employs a constrained attention pattern similar to Sartran et al. (2022) to encourage head-dependent representation learning.

Both constituency-based and dependency-based studies discussed above incorporate the inductive biases of symbolic structures into the self-attention mechanism by regulating the attention masks dynamically. Some other studies focus on adapting the self-attention modules, or combining the two (Wang et al., 2019; Peng et al., 2019; Deshpande and Narasimhan, 2020; Murty et al., 2023). Our work also follows these conventions of adaptation, modifying the self-attention module by incorporating dependency graph feature representations without changing the input or output space of Transformer LMs.

This line of models shows considerable performance in generalizing syntactic information via recursion as tree structures. However, most of these studies focus solely on trees rather than a more general and flexible form — graphs. One notable

work (Prange et al., 2022) proposes a model that exploits information from both syntactic and semantic graphs. However, this work only introduces graph-informed language modeling without actually modeling the explicit symbolic structure; that is, gold syntax and semantics are needed for both training and test-time inference of the model. Semantic graphs are also employed to guide the model in other research fields such as machine translation and visual tasks, but these studies directly apply the gold signals for model augmentation from semantic graphs instead of encoding the graphs into the model (Aue et al., 2004; Ke et al., 2024). GiLT differs from these models as we model graphs in the Transformer LM, and we can incrementally build a graph along with the next token prediction without graph supervision during inference.

6 Conclusion

We propose GiLT, a novel type of syntactic language models that incorporates dependency graphs — a more general and flexible form of linguistic structural information compared with traditional syntactic tree structures - into Transformers. GiLT predicts dependencies jointly with token generation, and moderate attention scores through features extracted from building up dependency graphs. Our experiments show that GiLT surpasses base language models and other syntactical language models in terms of syntactic generalization, achieving these gains without inserting extra tokens and with minimal impact on perplexity. Additionally, pretrained LM finetuned with Graph-Infused Layers can also improve performance on several downstream tasks.

For future work, we plan to explore the potential of the feature tape for jointly modeling multiple types of parse graphs. This presents a significant challenge for both effective training and efficient inference. Furthermore, we consider unsupervised training for GiLT as another promising direction.

Limitations

During inference, we rely on beam search to estimate the marginalized probability, which can only provide the lower bound. Although our action space is constant and independent of the sequence length, beam search remains computationally expensive.

Additionally, the discussion in Appendix B suggests that the performance limitations observed are primarily due to the underutilization of tree properties in our graph-based modeling approach. This insight highlights the potential for further research to focus on better integrating the inherent properties of graphs, such as the presence of multiple heads, to improve the model's overall performance and effectiveness.

References

Anthony Aue, Arul Menezes, Bob Moore, Chris Quirk, and Eric Ringger. 2004. Statistical machine translation using labeled semantic dependency graphs. In *Proceedings of the 10th Conference on Theoretical and Methodological Issues in Machine Translation of Natural Languages*, Baltimore, Maryland.

Jan Buys and Phil Blunsom. 2015. Generative incremental dependency parsing with neural networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 863–869, Beijing, China. Association for Computational Linguistics.

Eugene Charniak, Don Blaheta, Niyu Ge, Keith Hall, John Hale, and Mark Johnson. 2000. BLLIP 1987-89 WSJ Corpus Release 1.

Do Kook Choe and Eugene Charniak. 2016. Parsing as language modeling. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2331–2336, Austin, Texas. Association for Computational Linguistics.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988, Florence, Italy. Association for Computational Linguistics.

Ameet Deshpande and Karthik Narasimhan. 2020. Guiding attention for self-supervised learning with transformers. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4676–4686, Online. Association for Computational Linguistics.

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. In *International Conference on Learning Representations*.

Timothy Dozat and Christopher D. Manning. 2018. Simpler but more accurate semantic dependency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 484–490, Melbourne, Australia. Association for Computational Linguistics.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. 2016. Recurrent neural network grammars. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 199–209, San Diego, California. Association for Computational Linguistics.

Dan Flickinger. 2000. On building a more effcient grammar by exploiting types. *Nat. Lang. Eng.*, 6(1):15–28.

Daniel Flickinger, Yi Zhang, and Valia Kordoni. 2012. Deepbank: A dynamically annotated treebank of the wall street journal. In *Proceedings of the Eleventh International Workshop on Treebanks and Linguistic Theories. International Workshop on Treebanks and Linguistic Theories (TLT-11), 11th, November 30-December 1, Lisbon, Portugal*, pages 85–96. Edições Colibri.

Jan Hajic, Eva Hajicová, Jarmila Panevová, Petr Sgall, Ondřej Bojar, Silvie Cinková, Eva Fučíková, Marie Mikulová, Petr Pajas, Jan Popelka, Jiří Semecký, Jana Šindlerová, Jan Štěpánek, Josef Toman, Zdenka Urešová, and Zdeněk Žabokrtský. 2012. Announcing prague czech-english dependency treebank 2.0. In *International Conference on Language Resources and Evaluation*.

Jennifer Hu, Jon Gauthier, Peng Qian, Ethan Wilcox, and Roger Levy. 2020. A systematic assessment of syntactic generalization in neural language models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1725–1744, Online. Association for Computational Linguistics.

Xiang Hu, Pengyu Ji, Qingyang Zhu, Wei Wu, and Kewei Tu. 2024. Generative pretrained structured transformers: Unsupervised syntactic language models at scale. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2640–2657, Bangkok, Thailand. Association for Computational Linguistics.

Junlong Ke, Zichen Wen, Yechenhao Yang, Chenhang Cui, Yazhou Ren, Xiaorong Pu, and Lifang He. 2024. Integrating vision-language semantic graphs in multi-view clustering. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24*, pages 4273–4281. International Joint Conferences on Artificial Intelligence Organization. Main Track.

Yoon Kim, Alexander Rush, Lei Yu, Adhiguna Kuncoro, Chris Dyer, and Gábor Melis. 2019. Unsupervised recurrent neural network grammars. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1105–1117, Minneapolis, Minnesota. Association for Computational Linguistics.

Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium. Association for Computational Linguistics.

Piotr Mirowski and Andreas Vlachos. 2015. Dependency recurrent neural language models for sentence completion. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 511–517, Beijing, China. Association for Computational Linguistics.

Yusuke Miyao. 2006. *From linguistic theory to syntactic analysis : corpus-oriented grammar development and feature forest model*. Ph.D. thesis, University of Tokyo. Unpublished doctoral dissertation.

Shikhar Murty, Pratyusha Sharma, Jacob Andreas, and Christopher Manning. 2023. Pushdown layers: Encoding recursive structure in transformer language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3233–3247, Singapore. Association for Computational Linguistics.

Hiroshi Noji and Yohei Oseki. 2021. Effective batching for recurrent neural network grammars. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 4340–4352, Online. Association for Computational Linguistics.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Silvie Cinková, Dan Flickinger, Jan Hajič, and Zdeňka Urešová. 2015. SemEval 2015 task 18: Broad-coverage semantic dependency parsing. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 915–926, Denver, Colorado. Association for Computational Linguistics.

Martha Palmer, Daniel Gildea, and Paul Kingsbury. 2005. The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1):71–106.

Hao Peng, Roy Schwartz, and Noah A. Smith. 2019. PaLM: A hybrid parser and language model. 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)*, pages 3644–3651, Hong Kong, China. Association for Computational Linguistics.

Jakob Prange, Nathan Schneider, and Lingpeng Kong. 2022. Linguistic frameworks go toe-to-toe at neuro-symbolic language modeling. In *Proceedings of the 2022 Conference of the North American Chapter of*

the Association for Computational Linguistics: Human Language Technologies, pages 4375–4391, Seattle, United States. Association for Computational Linguistics.

Peng Qian, Tahira Naseem, Roger Levy, and Ramón Fernandez Astudillo. 2021. Structural guidance for transformer language models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3735–3745, Online. Association for Computational Linguistics.

Laurent Sartran, Samuel Barrett, Adhiguna Kuncoro, Miloš Stanojević, Phil Blunsom, and Chris Dyer. 2022. Transformer grammars: Augmenting transformer language models with syntactic inductive biases at scale. *Transactions of the Association for Computational Linguistics*, 10:1423–1439.

Mitchell Stern, Daniel Fried, and Dan Klein. 2017. Effective inference for generative neural parsing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1695–1700, Copenhagen, Denmark. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium. Association for Computational Linguistics.

Xinyu Wang, Yong Jiang, Nguyen Bach, Tao Wang, Zhongqiang Huang, Fei Huang, and Kewei Tu. 2021. Automated concatenation of embeddings for structured prediction. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2643–2660, Online. Association for Computational Linguistics.

Yaushian Wang, Hung-Yi Lee, and Yun-Nung Chen. 2019. Tree transformer: Integrating tree structures into self-attention. 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)*, pages 1061–1070, Hong Kong, China. Association for Computational Linguistics.

Alex Warstadt, Alicia Parrish, Haokun Liu, Anhad Mohananey, Wei Peng, Sheng-Fu Wang, and Samuel R. Bowman. 2020. BLiMP: The benchmark of linguistic minimal pairs for English. *Transactions of the Association for Computational Linguistics*, 8:377–392.

Ryo Yoshida and Yohei Oseki. 2022. Composition, attention, or both? In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 5822–5834, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Yida Zhao, Chao Lou, and Kewei Tu. 2024. Dependency transformer grammars: Integrating dependency structures into transformer language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1543–1556, Bangkok, Thailand. Association for Computational Linguistics.

A Other Experimeatal Details

Hyperparamter for finetuning We use a batch size of 64 and a fixed learning rate of 3e-6. We choose the best model based on performance on the validation set. We use the following prompts to convert text classification task into language modeling:

- **RTE**: We utilize the following prompt: *Sentence1:{s_1}; Sentence2:{s_2}; Label:{l}.* $l \in \{0, 1\}$ for input sentence pair (s_1, s_2)
- **MRPC**: Given input sentence pair (s_1, s_2), we construct the prompt: *Sentence1:{s_1}; Sentence2:{s_2}; Label:{l}.* $l \in \{\text{inequivalent, equivalent}\}$.
- **SST2**: Given string s and label l, prompt is: *Sentence1:{s_1}; Sentiment:{l}.* $l \in \{0, 1\}$.
- **STS-B**: Given the sentence pair (s_1, s_2), we create the prompt *Sentence1:{s_1}; Sentence2:{s_2}; Score:*. We use the final hidden states to train a linear regression model, training jointly with LM.

Since our pretrained TXL is sentence-level, we use semicolons instead of periods.

Computational costs We use PyTorch version 2.7.0 for all experiments. For language modeling experiment, we spent one NVIDIA A6000 GPU for each training, which lasted about 50 hours. For finetuning experiment, we spent one NVIDIA H800 GPU for each training, which lasted less than 1 hour for each task.

B Discussion on different parsing

By analyzing metrics in Table 1, we discover that the order of performance from high to low in perplexity is: PSD, DM, PAS and DP. It also roughly conforms to this order in other metrics.

SDP Dataset	Avg. Dependencies	PPL
PSD	16.9	14.9
DM	18.8	14.9
PAS	24.6	15.1
DP	24.2	15.6

Table 4: average number of dependencies per sentence in different SDP dataset based on BLLIP-LG and reported perplexity of each model from Section 4.1.

We calculate the average number of dependencies in the graphs and report the results in Table 4. We can surprisingly find that the fewer dependencies we need to establish, the better performance we will get. This is likely because fewer dependencies result in less noise we obtained from silver parse graphs, and the simpler graphs are probably easier to model. Although PAS has more dependencies than DP, GiLT-PAS performs better than GiLT-DP.

Performance degradation on the DP dataset is not unexpected, as the parse graphs for DP are essentially trees. We get rid of the property of trees which is unable to leverage their unique properties. This suggests that while GiLT is able to handle more dependencies in graphs with relatively minor performance degradation, it has limitations in effectively utilizing tree structures, a specific type of graph.