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 Transformer language models. Unlike most previous work, GiLT does 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 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 have been deemed essential in traditional natural language processing are absent from the model design and training process of Transformer LMs.

Over the past decade, a number of researchers 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). Empirically, they achieve stronger syntactic generalization compared with standard Transformer LMs while retaining competitive language modeling performance.

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 inserting additional tree-building operations into the input and output sequence, leading to longer sequence lengths and higher computational cost, and making it harder to finetune a pretrained LM into a syntactic LM. An exception is Pushdown Layers (Murty et al., 2023), which leverages syntactic trees to guide attention computation without changing the LM's symbol space.

In this paper, we propose the Graph-Infused Layer Transformer LM (GiLT) that addresses the above-mentioned limitations in integrating linguistic structures into Transformer LMs. GiLT is based on *dependency graphs* that subsume both syntactic dependency trees and *semantic* dependency graphs, thus extending syntactic LM research beyond syntax. Inspired by Pushdown Layers (Murty et al., 2023), GiLT incrementally constructs dependency graphs without changing the symbol space of the underlying LMs, and modulates attention scores with features computed from graph attributes such as node degrees, depths and distances.

Experimental results show that GiLT achieves gains in syntactic generalization over baselines with almost no degradation in perplexity on language modeling. Furthermore, GiLT finetuned

from a pretrained GPT2 achieves better performance on downstream tasks compared with the original pretrained GPT2, suggesting that the Graph-Infused layer is a competitive alternative to standard self-attention.

In summary, our contributions are as follows:

- We propose *Graph-Infused Layers*, leveraging dependency graphs to enhance LMs by our novel *graph-based feature tapes* without modifying the input or output space.
- Comprehensive experiments on language modeling, syntactic evaluation and finetuning on text classification demonstrate competitive perplexity, improved syntactic generalization and language understanding. Ablation study on feature tapes shows the importance of each part and test on generation speed illustrates the advantage of not requiring extra tokens.

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 builds constituency syntactic trees and modulates attention scores based on the constituency trees. Unlike other syntactic LMs, it does not change the symbol 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* $\mathbf{t}_i$ the depths of all the tokens that are already generated in the partially built constituency tree.

Pushdown-LM then augments self-attention with stack tape $\mathbf{t}_i$:

$$\tilde{\alpha}^l_{ij} = [\mathbf{h}^l_j + \mathbf{d}^l_{ij}]^\top \mathbf{W}_k^\top \mathbf{W}_q \mathbf{h}^l_i \qquad (1)$$

where $\tilde{\alpha}^l_{ij}$ is the attention score before softmax assigned to the j-th token from the i-th token at layer l, $\mathbf{h}^l_j$ is the hidden state of j-th token at the l-th attention block, $\mathbf{d}^l_{ij}$ is the embedding of the depth of the j-th token recorded in $\mathbf{t}_i$, and $\mathbf{W}_k$ and $\mathbf{W}_q$ are learnable query and key matrices in self-attention. In this way, structural information from the constituency tree is implicitly introduced into self-attention computation and thus influences the decoding of the underlying LM.

2.2 Semantic Dependency Graphs

A semantic dependency graph forms as a directed acyclic graph instead of a tree. The dependencies 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) are 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 dependencies that incrementally construct dependency graphs over the sentences. We first score possible dependencies that link the current word to previous words (Section 3.1), then update the dependency graph based on the scoring (Section 3.2), and utilize the *graph-based feature tapes* (Section 3.3), which characterize generated tokens in the graph, to modulate attention computation (Section 3.4).

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 with a biaffine mechanism. Since a word may correspond to multiple tokens, we first define the word-level representations that serve as input to the biaffine module.

Suppose a word w_i is tokenized into m tokens with input embeddings $\{\mathbf{x}_k, \cdots, \mathbf{x}_{k+m-1}\}$ and corresponding hidden states $\{\mathbf{h}^l_k, \cdots, \mathbf{h}^l_{k+m-1}\} \subseteq \mathbb{R}^d$ from all layers $l = 1, \ldots, L$. We define its word-level representation $\mathbf{o}_i \in \mathbb{R}^{3d}$ by concatenating three components: (i) the hidden state from the middle layer, $\mathbf{h}^{L/2}_{k-1}$; (ii) the hidden state from the penultimate layer, $\mathbf{h}^{L-1}_{k-1}$; (iii) the input embed-

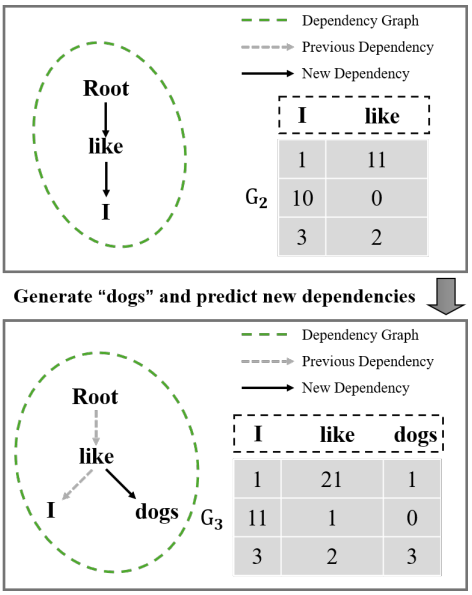

Figure 1: Illustration of how the feature tape is recomputed when generating a sentence and constructing its dependency graph. Rows in G_2 and G_3 from top to bottom correspond to Degree, Distance and Depth, respectively. We set $m_{in} = 1$ and $m_{out} = 10$ for this example. As *dogs* is predicted, one dependency is added to the graph.

ding of the first token, $\mathbf{x}_k$, which provides direct lexical information about the word. According to the assumption in Murty et al. (2023), since $\mathbf{h}_{k-1}^{L/2}$ and $\mathbf{h}_{k-1}^{L-1}$ are hidden states from sufficiently deep layers used to predict the k-th token, they capture useful information about the k-th token rather than the $(k-1)$-th token. We do not use the final-layer hidden states to reserve them exclusively focused on next token prediction. Note that we do not use input embeddings and hidden states computed after $\mathbf{x}_k$ so that we can predict all dependencies of w_i before processing $\mathbf{x}_k$, thus being able to infuse structural information from the dependency graph to the hidden states of the tokens in w_i.

We follow the biaffine parsing approach (Dozat and Manning, 2018) to compute the probability 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:

$$\begin{aligned} \tilde{\mathbf{o}}_i^{par} &= \mathrm{MLP}_{par}^2(\mathrm{MLP}_{par}^1(\mathbf{o}_i) + \mathbf{pe}_{ii}) \\ \tilde{\mathbf{o}}_j^{chd} &= \mathrm{MLP}_{chd}^2(\mathrm{MLP}_{chd}^1(\mathbf{o}_j) + \mathbf{pe}_{ij}) \\ p_{ij} &= \sigma(\tilde{\mathbf{o}}_i^{par\top}\mathbf{W}_p\tilde{\mathbf{o}}_j^{chd}) \end{aligned} \tag{2}$$

where $\mathbf{o}_i$ is the word representation of w_i as defined above, $\mathbf{W}_{par} \in \mathbb{R}^{d\times d}$ is a learnable matrix, $\mathrm{MLP}_{par/chd}^{1/2}$ denotes the first/second MLP for computing parent/child representations $\tilde{\mathbf{o}}_i^{par/chd} \in \mathbb{R}^d$, σ denotes the sigmoid function, and $\mathbf{pe}_{ij}$ denotes the positional embedding which is a sum of the sinusoid encoding of $|i-j|$ and the embedding of graph-based feature tape G_i (see Section 3.3).

3.2 Graph Update

Given dependency probabilities $\{p_{ij}, p_{ji}, p_{ii}\}$ where $j \in \{0, \cdots, i-1\}$ for all possible dependencies with regard to the i-th word w_i, a straightforward method to greedily update the dependency graph is to add any dependency whose probability exceeds 0.5. However, this becomes computationally intractable when we employ beam search of dependency graphs (Section 3.5) because of the exponentially large search space. To address this issue, we consider a restricted subspace of dependency graphs by using a two-step method as follows. For w_i:

(i) We predict the number of dependencies $c_i \in \{0, 1, \cdots, C\}$, where C is a constant upper bound:

$$\begin{aligned} \mathbf{s} &= \sum_{j=0}^{i} \tilde{\mathbf{o}}_i^{par} \odot \mathbf{W}_s\tilde{\mathbf{o}}_j^{chd} + \sum_{j=0}^{i-1} \tilde{\mathbf{o}}_j^{par} \odot \mathbf{W}_s\tilde{\mathbf{o}}_i^{chd} \\ \boldsymbol{\pi}_i &:= [\pi_i^0, \pi_i^1, \cdots, \pi_i^C]^T \\ &= \mathrm{softmax}\left(\mathbf{W}_a^\top\left(\frac{\mathbf{s}}{\sqrt{2i-1}}\right) + \mathbf{b}_a\right) \end{aligned} \tag{3}$$

where $\odot$ denotes the Hadamard product operation, $\mathbf{W}_a \in \mathbb{R}^{(C+1)\times d}$, $\mathbf{W}_s \in \mathbb{R}^{d\times d}$ and $\mathbf{b}_a \in \mathbb{R}^{C+1}$ are learnable parameters, $\boldsymbol{\pi}_i \in \mathbb{R}^{C+1}$ is the probability distribution over $\{0, 1, \cdots, C\}$. To normalize the variance, we scale $\mathbf{s}$ by $\sqrt{2i-1}$.

(ii) We either choose the most probable value of c_i or sample its value from $\boldsymbol{\pi}_i$. Then we pick c_i highest-scoring dependencies and add them to the dependency graph. This two-step method reduces the search space from exponential to linear for each step in beam search.

3.3 Feature Extraction

Given the input sequence $x_{<k}$, where x_k is the first token of the i-th word w_i as defined in Section 3.1, we extract features from the partially constructed dependency graph and form a graph-based feature tape $G_k = [g_{1k}, g_{2k}, \cdots, g_{kk}] \in \mathbb{N}^{3\times k}$ for the token x_k. Note that the graph is word-level, but G_k corresponds to a token. Therefore, for any token i, j belongs to the same word, $g_{ik} = g_{jk}$ in G_k.

The feature tape involves three graph-based features: (i) degree, an attribute for each word; (ii)

distance, measuring connectivity between words; (iii) depth, reflecting the global structure of the graph.

Degree. The degree of a word refers to the number of its incoming and outgoing dependencies, denoted as c_{in} and c_{out} respectively. According to the definition, $c_{in} + c_{out}$ is the degree for each word, but empirically, we discover that weighted summation achieves better performance (see section 4.4): we assign weight $m_{in} \in \mathbb{Z}^+$ to in-degree and $m_{out} \in \mathbb{Z}^+$ to out-degree, where $0 < m_{in} < m_{out}$, and set the degree as $m_{out}c_{out} + m_{in}c_{in}$.

Distance. The distance from word w_i to word w_j is computed by finding the weighted shortest path on the current dependency graph. When traversing a dependency its direction, we weight it by m_{out}; against the direction, we use m_{in}. Thereby encoding dependency direction information into the distance measure. Intuitively, the distance measures the relevance of the two words. Specifically, distance recorded in g_{1k} is measured from the word of token x_k to the word of token x_1.

Depth. The dependency graphs used in our work are all rooted. We define the depth of a word to be the length of the shortest path to the root plus one in the undirected backbone of the dependency graph. Since the dependency graph is partial, a word may be disconnected from the root and we set its depth to be 0. We can compute the depths of all the words with breadth-first search starting from the root. A visited flag ensures that each word is processed exactly once.

In Figure 1, we illustrate the feature tapes when generating an example sentence.

3.4 Computing Attention Scores

We incorporate information in the graph-based feature tape G_k into the Transformer LM by modifying the self-attention module. Specifically, we first map $G_k \in \mathbb{N}^{3\times k}$ via a learned embedding layer onto a global embedding $\tilde{\mathbf{e}}_k \in \mathbb{R}^{3\times k\times \tilde{d}}$. For each Transformer layer l, we apply a linear projection f_l for feature fusion, that is, $\mathbf{e}^l_k = f_l(\tilde{\mathbf{e}}_k) \in \mathbb{R}^{k\times d}$. For each token position $j \in \{0, 1, \cdots, k\}$, the corresponding fused graph feature $\mathbf{e}^l_{kj} \in \mathbb{R}^d$ is directly added to its key in attention computation,

$$\tilde{\alpha}^l_{kj} = [\mathbf{h}^l_j + \mathbf{e}^l_{kj}]^\top \mathbf{W}^\top_k \mathbf{W}_q \mathbf{h}^l_k \tag{4}$$

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

$$\begin{aligned}\tilde{\alpha}^l_{kj} &= {\mathbf{h}^l_j}^\top \mathbf{W}^\top_{k,c} \mathbf{W}_q \mathbf{h}^l_k \\ &+ (\mathbf{r}_{kj} + \mathbf{e}^l_{kj})^\top \mathbf{W}^\top_{k,r} \mathbf{W}_q \mathbf{h}^l_j \\ &+ u^\top \mathbf{W}_{k,c} \mathbf{h}^l_k + v^\top \mathbf{W}_{k,r}(\mathbf{r}_{kj} + \mathbf{e}^l_{kj}),\end{aligned} \tag{5}$$

where $\mathbf{r}_{kj}$ is a vector with sinusoid encoding of $|k-j|$, $W_{k,c}$ and $W_{k,r}$ are key matrix for respectively extracting content and relative representation, u and v are learnable bias vectors.

3.5 Training and Inference

Training. Given a corpus of strings annotated with dependency graphs, we precompute graph-based feature tape G_k for each prefix $x_{\leq k}$ based on the ground-truth dependencies over $x_{\leq k}$. After this preprocessing step, 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 and M words and its ground-truth dependency graph, we derive a sequence of one-hot ground-truth vectors of length M $\{\hat{\boldsymbol{\pi}}_1, \cdots, \hat{\boldsymbol{\pi}}_M\}$ indicating the number of dependencies of each word, and a matrix $\hat{p}$ where $\hat{p}_{ij} = 1$ iff. there is a dependency from word i to word j, the training loss function is defined as follows:

$$\begin{aligned}\mathcal{L} &= \alpha \frac{1}{N} \sum_{i=1}^{N} \mathrm{CE}(x_i, \hat{x}_i) \\ &+ \beta \frac{1}{M^2} \sum_{j=1}^{M} \sum_{k=1}^{M} \mathrm{BCE}(p_{jk}, \hat{p}_{jk}) \\ &+ \gamma \frac{1}{M} \sum_{k=1}^{M} \mathrm{CE}(\boldsymbol{\pi}_k, \hat{\boldsymbol{\pi}}_k)\end{aligned} \tag{6}$$

where α, β, γ are constant coefficients, p_{jk} and $\boldsymbol{\pi}_k$ are from Eq. 2 & 3 respectively.

Inference. GiLT jointly generates a string x and its dependency graph y. As discussed in Section 3.2, GiLT considers only a subspace Y of dependency graphs and models uncertainty over the subspace with uncertainty over numbers of dependencies of the words in x. Therefore, the joint probability $p(x, y)$ is computed as follows.

$$\begin{aligned}p(x, y) &= \prod_{k=1}^{N} p(x_k | x_{<k}; G_{k-1}(y)) \\ &\times \prod_{j=1}^{M} p(c_j(y) | x_{\leq z_j}; G_{z_j}(y))\end{aligned} \tag{7}$$

where N and M are the numbers of tokens and words in x, $c_j(y)$ is the number of dependencies of the j-th word in graph y, z_j is the index of the last token within the j-th word, $G_k(y)$ is the feature tape for token x_k based on the subgraph of y containing only the words corresponding to the first k tokens.

Computing the marginal $p(x) = \sum_{y \in Y} p(x, y)$ is computationally intractable due to the huge space of all possible graphs. Following Murty et al. (2023), we approximate it by marginalizing (summing) over a relatively small set of dependency graphs produced via beam search (i.e., retaining multiple most likely values of c_j and their corresponding dependencies for every beam). The approximated marginal probability 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 preprocessing. 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 dependency graphs with unlabeled dependencies by parsing the dataset with ACE (Wang et al., 2021). Since a dependency tree can be seen as a special case of a dependency graph, 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 SentencePiece (Kudo and Richardson, 2018). We follow Murty et al. (2023) and model each sentence independently.

Setup. We evaluate the perplexity (PPL) of the models on the BLLIP-LG dataset. We train a 16-layer TXL (Dai et al., 2019) language model of 252M parameters as the baseline. We also reimplement Pushdown-LM (Murty et al., 2023) based on the code base of TXL for fair comparison. We use PSD, DM and PAS dependency 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		PPL ↓	10%BLiMP ↑	SG ↑
Models that add structural tokens to inputs				
PLM		29.8♠	75.1♣	76.9
PLM-Mask		49.1♠	75.3♣	74.7
TG		18.4	73.5♣	79.6
DTG		14.9	<u>76.1</u>♣	<u>81.2</u>
Models that do not add extra tokens to inputs				
TXL (baseline)		14.8	75.1	72.1
TXL-Large (334M)		14.7	75.4	71.8
Pushdown-LM		<u>**14.6**</u>	74.0	74.6
Ours	GiLT-DP	15.5	<u>**76.1**</u>	72.3
	GiLT-PAS	15.0	73.0	75.7
	GiLT-DM	14.9	74.9	76.3
	GiLT-PSD	14.9	75.7	**79.7**

Table 1: Results on language modeling and syntactic generalization. The best values over models that do not add extra tokens are **bold**. Overall best values are <u>underlined</u>. All PPL results except for that of TXL are approximated upper bounds. SG scores are computed without the "other" suite. ♠ denotes that the PPL is taken from the original paper. ♣ denotes that the result is evaluated with the full BLiMP dataset reported in the original paper.

We set the hyperparameters for GiLT with m_{in} = 1, m_{out} = 10, $\alpha = 1$, $\beta = 0.2$, $\gamma = 0.2$, $\tilde{d} = 256$, and $d = 1024$. This results in 268M parameters for Transformer and 54M parameters for the modules described in Section 3.1-3.4 in GiLT. For TXL, the same configuration of hyperparameters (model size, dropout, learning rate schedulers) is used as in Zhao et al. (2024). For perplexity computation, we apply beam search of dependency graphs with the same beam size b of 300 following Pushdown-LM (Murty et al., 2023) to estimate the perplexity upper bound of GiLT. To account for the additional parameters in GiLT in comparison with the baseline, we also train a TXL-Large with 22 layers (6 more than the base TXL, resulting in 334M parameters), to investigate the impact of scaling up model sizes alone.

Result. We report the PPL of all models in Table 1. Some syntactic language models, such as PLM and TG, introduce syntactic inductive bias at the cost of language modeling performance. On the other hand, 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). The three GiLT models based on de-

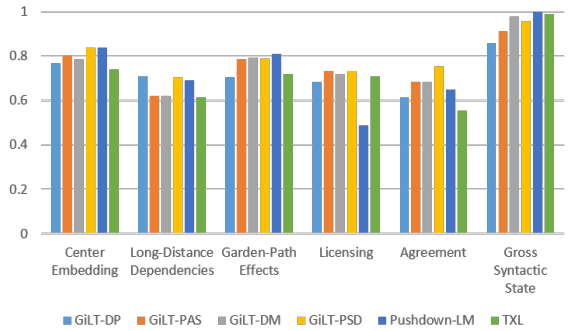

Figure 2: Scores on the 6 circuits of the SG test suites among models without extra tokens.

pendency graphs maintain PPL values comparable to both Pushdown-LM and the baseline. In contrast, GiLT-DP exhibits a higher PPL, highlighting the limitations of tree structures compared to more flexible graph-based 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 baseline models as in Section 4.1. For BLiMP, models are evaluated on their ability to assign higher probability to grammatical sentences than to their minimally altered ungrammatical counterparts. The reported BLiMP score is the percentage of such pairs where the model succeeds, i.e., where the grammatical sentence receives a higher probability. Due to limited computational resources, we evaluate on a 10% subset of the BLiMP dataset, selecting every tenth example (i.e., the 1st, 11th, 21st, etc.). We find that doing this only leads to very small perturbations to the scores (e.g., 0.2 for TXL).

The SG test suites include seven syntactic phenomenon classes. For each suite, the model is evaluated on its ability to satisfy a predefined inequality concerning the probability of generating a target span. We report the per-suite satisfaction percentage rate (i.e., the fraction of inequalities that hold) and then compute the SG score as the macro-average of six rates except for the "other" suite, as it contains only a single sentence, does not correspond to any specific syntactic phenomenon, and disproportionately influences the macro-average.

For 10%BLiMP and SG, we use beam search of dependency graphs to both compute marginal probability $p(x)$ and conditional probability $p(x_t|x_{<t})$.

Result. The results are presented in Table 1. Although TXL-Large has more parameters than our models, its improvements are marginal, indicating that simply scaling up TXL without syntactic inductive bias fails to improve syntactic generalization. GiLT-PSD outperforms most models in both tests, surpassing the baseline by 0.6 points in 10%BLiMP and 7.6 points in SG. GiLT-DP achieves the best 10%BLiMP performance, matching that of DTG. In contrast, Pushdown-LM exhibits better SG performance but worse 10%BLiMP performance than the baseline.

4.3 Finetuning on Pretrained LM

Since GiLT does not change the symbol space of the Transformer LM, it can be finetuned from any pretrained language model on any datasets annotated with dependency graphs to introduce syntactic inductive bias. We therefore evaluate GiLT by starting from a pretrained GPT-2 model and finetuning it on downstream tasks.

Setup. We use the pretrained GPT2-medium (355M) (Radford et al., 2019) as the base model. We create GiLT-GPT2 by replacing its last 12 vanilla transformer layers with our Graph-Infused layers and finetune GiLT-GPT2 on BLLIP-LG. We evaluate the language understanding ability of GiLT-GPT2 on four downstream text classification tasks from GLUE (Wang et al., 2018): RTE, SST2, MRPC and STS-B. Each task is transformed into a language modeling task via prompting (details are provided in Appendix A). We use GiLT-GPT2 to parse each prompt and then finetune the model with the parsed dependency graph on text classification. For fair comparison, we also perform the same workflow for vanilla GPT2-medium, i.e., finetuning on BLLIP-LG and then on task data.

Result. In Table 2, we report F1 scores for SST2 and RTE, accuracy/F1 for MRPC, and Pearson/Spearman correlation for STS-B. It can be seen that GiLT-GPT2 wins on more tasks than GPT2, implying generally enhanced language understanding capabilities. It is also worth noting that our parsing

Model	RTE ↑	SST2 ↑	MRPC ↑	STS-B ↑
GPT2	64.1	**94.84**	80.75/86.29	**84.2/83.6**
GiLT-GPT2	**65.7**	**94.84**	**80.93/86.43**	82.5/81.6

Table 2: Results of finetuned models on downstream tasks.

Model	PPL ↓	10%BLiMP ↑	SG ↑
GiLT-PSD	14.9	75.7	**79.7**
-degree	14.9	**76.5**	75.3
-depth	14.9	76.4	73.8
-distance	14.9	74.5	75.1
-weighted degree	14.9	74.8	77.7
-weighted distance	14.9	76.2	77.0

Table 3: The results of the ablation study.

Model	Beam Size	Tokens/s ↑	Max Cuda Memory ↓
TXL	1	52.6	2.8
DTG	1	24.4	2.9
GiLT	1	43.0	3.6
DTG	20	5.4	6.4
GiLT	20	24.7	4.2
DTG	100	0.9	22.7
GiLT	100	9.0	6.7
DTG	300	/	>48.0
GiLT	300	3.5	12.9

Table 4: The results of the generation speed test.

process uses a modest beam size of 20 (compared to 300 in language modeling), yet achieves good task performance.

4.4 Ablation Study

We design five controlled ablations: (1) –degree: removal of degree from the feature tape, (2) –depth: removal of depth, (3) –distance: removal of distance, (4) unweighted degree: removing the degree weighting coefficients m_{in} and m_{out}, and (5) unweighted distance: removing the distance weighting coefficients m_{in} and m_{out}. Each ablation setting is re-trained from scratch. We use GiLT-PSD as the base model and evaluate on both PPL, SG and 10%BLiMP as in previous sections.

As shown in Table 3, the PPLs of the six settings are at the same level, but their syntactic generalization performances can be quite different. Notably, the –degree, –depth and unweighted distance model exhibit better 10%BLiMP than the GiLT-PSD base model, but they yield significantly lower SG scores. The other two ablation settings degrade on both 10%BLiMP and SG. The ablation study indicates that each feature captures a distinct linguistic aspect and their combination leads to more robust overall performance.

4.5 Efficiency Comparison

We have observed the strong performance of DTG in Table 1. However, DTG introduces additional structural tokens, which slows down inference. We measure the speed of DTG and our model on a small set of sentences when using beam search of dependency graphs with the same beam size b. We also measure the greedy decoding speed of TXL as a baseline, which does not require beam search of sentence structures and can be seen as using a beam size of 1. It can be seen that GiLT is slightly slower than TXL when $b = 1$, showing that the low consumption of our extra module. Compares with DTG, our GiLT remains significantly faster and more memory efficient. As b increases, the efficiency degrades and GPU memory grows, yet both worsen markedly more slowly for GiLT than DTG. Note that when $b = 300$, we were unable to complete DTG's inference on our NVIDIA A6000 GPU due to excessive memory demands.

4.6 Case Study

We obtain the attention scores of both GiLT and TXL when inputting "`Writing long reports every week is boring`" and visualize it in Figure 3. TXL consistently assigns a large proportion of attention to the most recent noun and fails to identify the subject of this sentence. GiLT correctly focuses on "`Writing`" when the input is "`is`" since "`Writing`" is the governing word of the subject. When inputting "`week`", GiLT assigns attention scores more evenly to "`long reports`" and "`every week`" besides the attention sink, since both phrases modify "`Writing`".

5 Related Work

There have been studies about leveraging recursive linguistic structural (symbolic) information for sequential language modeling. For syntactic LMs with neural architectures, RNNGs (Dyer et al., 2016), jointly model the syntactic structure and words by integrating top-down transition-based constituency parsing into a recursive neural network, while recent studies (Qian et al., 2021; Yoshida and Oseki, 2022; Sartran et al., 2022) have applied this approach to Transformers, which explicitly model a syntactic tree along with words by imposing hard constraints over attention masks to simulate the shift/compose operations in transition-based parsing. Hu et al. (2024) further explores an unsupervised training framework for constituency-

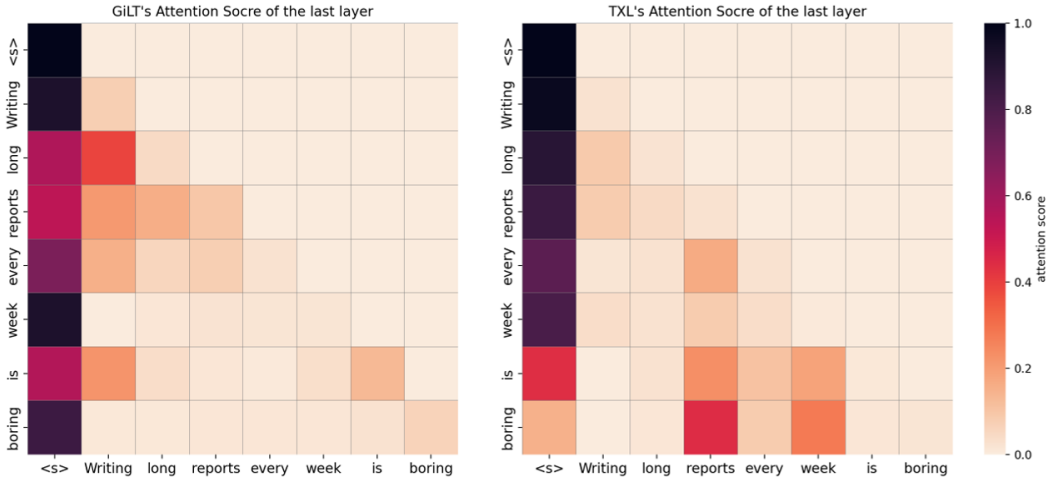

Figure 3: Visualization of attention scores of the first head in the last layer of GiLT (left) and TXL (right) given the input "`Writing long reports every week is boring.`".

based syntactic LMs, showing the potential of training syntactic LMs at scale.

In addition to constituency-based models mentioned above, studies on those based on dependency tree structures (Buys and Blunsom, 2015; Mirowski and Vlachos, 2015), 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 incorporate the inductive bias of symbolic structures into the self-attention mechanism by regulating the attention masks dynamically. Some other studies also focus on adapting the self-attention modules, or both (Wang et al., 2019; Peng et al., 2019; Deshpande and Narasimhan, 2020; Murty et al., 2023), whereas our work follows the conventions of adaptation, modifying the self-attention module by incorporating dependency graph feature representations without changing the input or output space of Transformer LMs.

These models show 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, it only introduces graph-informed language modeling without actually modeling the explicit symbolic structure: 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 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 jointly predicts tokens and dependencies, incrementally constructing a dependency graph and using features extracted from it to modulate attention scores. Experiments show that GiLT achieves enhanced syntactic generalization without introducing extra tokens and with minimal impact on perplexity. Additionally, finetuning GiLT from pretrained language models also improves language understanding performance on several downstream tasks. These results demonstrate GiLT can effectively construct dependency graphs of generated sentences and and extract their structural information to serve as inductive bias for language modeling. Our future work is discussed in Appendix C.

Limitations

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

Additionally, the discussion in Appendix B suggests that the performance limitations observed on GiLT-DP are primarily due to the under-utilization 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.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. Technical report, OpenAI.

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 5e-5. 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.

Computational costs We use PyTorch version 2.7.0 for all experiments. For language modeling experiments, we spent one NVIDIA A6000 GPU for each training, which lasted about 50 hours. For finetuning experiments, 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 in perplexity of models trained on different datasets can be listed high to low as: 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	15.0
DP	24.2	15.5

Table 5: 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 5. 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 dependency graphs, and the simpler graphs are probably easier to model.

The exception is GiLT-PAS with better performance than GiLT-DP when PAS has average dependencies more than DP. Performance degradation on the DP dataset is not unexpected, as the dependency graphs for DP are essentially trees: we lessen the constraints in the models, hence our model for dependency trees are weaker to leverage the unique recursive properties of trees. 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.

C Future work

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