Abstract

A syntactic language model (SLM) incrementally generates a sentence with its syntactic tree in a left-to-right manner. We present Generative Pretrained Structured Transformers (GPST), an unsupervised SLM at scale capable of being pre-trained from scratch on raw texts with high parallelism. GPST circumvents the limitations of previous SLMs such as relying on gold trees and sequential training. It consists of two components, a usual SLM supervised by a uni-directional language modeling loss, and an additional composition model, which induces syntactic parse trees and computes constituent representations, supervised by a bi-directional language modeling loss. We propose a representation surrogate to enable joint parallel training of the two models in a hard-EM fashion. We pre-train GPST on OpenWebText, a corpus with 9 billion tokens, and demonstrate the superiority of GPST over GPT-2 with a comparable size in numerous tasks covering both language understanding and language generation. Meanwhile, GPST also significantly outperforms existing unsupervised SLMs on left-to-right grammar induction, while holding a substantial acceleration on training.

1 Introduction

Pre-training a Transformer architecture (Vaswani et al., 2017) as a large language model has dominated the field of natural language processing (NLP) (Devlin et al., 2019; Liu et al., 2019; Radford et al., 2018, 2019; Brown et al., 2020; Ouyang et al., 2022). While Transformer language models have exhibited remarkable performance over various downstream NLP tasks (Bang et al., 2023), the recursive compositions behind language are represented in an implicit and entangled form. In contrast, human language understanding exhibits explicit composition decisions, as exemplified by the garden path sentence (Dynel, 2009) “Time flies like an arrow; Fruit flies like a banana”, where distinct syntactic configurations yield vastly divergent meanings. In addition, human infants acquire such compositional capability without supervision (Saffran et al., 1996). These phenomena motivate us to explore an unsupervised approach to learning explicit compositions in language modeling.

A typical approach to achieve language modeling with explicit composition is to model the joint distribution of words and a syntactic tree within the framework of syntactic language models (SLMs) (Dyer et al., 2016). Though there has been a long line of research on SLMs, they are rarely exploited as the backbone in state-of-the-art language modeling due to poor scalability. Recent Transformer-based SLMs (Sartrak et al., 2022; Murty et al., 2023) require annotated parse trees or supervised parsers as structural supervision, leading to limited training data scales and domain adaptation issues (McClosky et al., 2006). On the other hand, in unsupervised SLMs (Kim et al., 2019b; Shen et al., 2019a), non-terminal constituents are composed from their sub-constituents sequentially in a left-to-right manner, resulting in data dependencies that impede training parallelism.

In this work, we aim to pre-train an SLM at scale on raw texts. To this end, we propose Generative Pretrained Structured Transformer (GPST), an unsupervised SLM with the Transformer architecture as a backbone. A common practice in existing unsupervised SLMs is to learn structures by a uni-directional language modeling loss (LM loss). However, we empirically find such an asymmetric loss with only right-to-left feedback results in branching biases in the induced parse trees. Based
on the insight, we propose two components in GPST, a composition model performing structural learning supervised by a bi-directional LM loss, and a generative model for uni-directional syntactic language modeling. Specifically, we train the GPST in a fashion similar to hard-EM (Liang et al., 2017): in E-step, the composition model runs a pruned deep inside-outside encoder to induce a parse tree and compute inside and outside representations of constituents simultaneously within logarithmic steps (Hu et al., 2024); while in M-step, we update all parameters of GPST by minimizing both the bi-directional (reconstructing the sentence from outside representations) and uni-directional LM loss given the induced tree. The key in the M-step lies in using the inside representations of constituents computed by the composition model as a surrogate of inputs for the generative model, which enjoys two advantages. First, the representations of all constituents pre-computed in the E-step can be simultaneously fed into the generative model, which breaks the data dependencies and facilitates training parallelism. Second, with these representations participating in generation, the uni-directional LM loss in the M-step could be back-propagated to not only the generative model but the composition model used in the E-step as well.

In experiments, we pre-train GPSTs with sizes comparable to those of GPT-2$_{\text{small}}$ and GPT-2$_{\text{medium}}$ on OpenWebText (Gokaslan and Cohen, 2019) (~9 billion tokens), and evaluate the models on various tasks including language understanding, language generation, and grammar induction. GPST demonstrates an approximately 60-fold training acceleration and over 15% absolute increase in left-to-right grammar induction in comparison with existing unsupervised SLMs. Meanwhile, GPST also shows advantages over GPT-2 across almost all language understanding/generation benchmarks. GPST provides constituent-level interfaces that are not inherently possessed by the conventional Transformer-based language models, and thus exhibits great potential to enhance interpretability (Hu et al., 2023), support multi-modality (Wan et al., 2022), and improve dense retrieval in the future. Our contributions are three-fold:

- We propose an SLM consisting of a composition model in addition to a generative model, which can be trained without gold trees via a novel approach akin to hard-EM.
- We propose a representation surrogate to enable joint parallel training of all components.
- To the best of our knowledge, GPST is the first unsupervised SLM able to be pre-trained from scratch on billions of tokens and surpass GPT-2 on various benchmarks. The experimental results demonstrate the potential of GPST as a backbone for large language models.

2 Related work

Syntactic Language Models. There have been extensive studies on syntactic language modeling (Baker, 1979; Jelinek and Lafferty, 1991; Chelba, 1997; Chelba and Jelinek, 2000; Vinyals et al., 2015; Charniak et al., 2016; Dyer et al., 2016; Qian et al., 2021; Yoshida and Oseki, 2022), in which words and constituent symbols are mixed up and generated in a left-to-right manner. Recent works (Sartran et al., 2022; Murty et al., 2023) utilize Transformers to parameterize action probability distributions, but relies on annotated parse trees or parsers trained on gold trees as structural guidance. Besides, unsupervised SLMs are also explored, by differentiable structured hidden layers (Kim et al., 2017; Shen et al., 2018, 2019a; DuSell and Chiang, 2024), reinforcement learning approaches (Yogatama et al., 2017), or variational approximations (Li et al., 2019; Kim et al., 2019b). Normally, these unsupervised models are trained in a sequential manner. Our model follows a similar generation paradigm, but has stark differences in model architecture and training approach.

Composition Models. A composition model transforms text encoding into a combinatorial optimization problem, learns and searches for the optimal structure, and encodes the text in a bottom-up manner along a binary tree via a composition function recursively. Maillard et al. (2019) proposes a CKY-like (Cocke, 1969; Kasami, 1966; Younger, 1967) encoder, in which high-level constituents are soft-weighted over composed representations of its sub-constituents. Drozdov et al. (2019) proposes a deep inside-outside encoder (Baker, 1979; Lari and Young, 1990), enabling the model to learn underlying structures via an auto-encoding objective. Recently, a series of studies (Hu et al., 2021, 2022, 2024) have been conducted to reduce the neural inside encoder complexity from cubic to linear. Our SLM is built on top of state-of-the-art composition modeling techniques, in which we achieve unsupervised learning and enhance training parallelism by taking advantage of the pruned inside-outside encoder (Hu et al., 2024).
3 Methodology

Given a sentence \( x = [x_1, x_2, \ldots, x_n] \) with \( x_i \) from a vocabulary \( V \) (1 \( \leq i \leq n \)), our goal is to train an SLM without gold trees that can simultaneously generate \( x \) and its syntactic structure. We first introduce the generative architecture of GPST, and then elaborate on how to perform training and inference with the model.

3.1 Generative Model

GPST generates a sentence and its parse tree from left to right via two types of actions, GEN and COMP, along with a stack (Dyer et al., 2015) to maintain partially completed sub-trees during generation. GEN generates a word \( x \) and pushes its embedding onto the stack. We denote such an action as GEN(\( x \)), with \( x \in V \). COMP pops the top two elements off the stack, computes their composed representation, and pushes it back to the stack. A major difference in model architecture between GPST and existing unsupervised SLMs, such as URNNG (Kim et al., 2019b), is that GPST makes good use of the architecture of Transformers to parameterize the action probabilities and thus hidden states from previous actions can be directly accessed via self-attention during generation.

Figure 1 illustrates the generative process of GPST. The generative model comprises type layers and token layers, both consisting of multi-layered Transformers. Let us denote the stack at step \( t \) as \( S_t \), with \( S_t^0 \) and \( S_t^1 \) representing the top two elements, respectively. Initially, \( S_t^0 \) is set to the embedding of the beginning-of-sentence token (i.e., \( \text{<bos>} \) in Figure 1). At each step \( t \), \( S_t^0 \) along with a position ID \( w_t \) is fed into the type layers, yielding a hidden state \( h_t \), which is then utilized to predict the next action type \( y_t \):

- If \( y_t = 0 \) (COMP), we set \( w_{t+1} \) as \( w_t \), pop off \( S_t^0 \) and \( S_t^1 \) from \( S_t \), and compose them using a composition function. The composed representation is then pushed back into the stack. In such a case, action \( a_t \) at time step \( t \) is set to COMP.
- If \( y_t = 1 \) (GEN), we set \( w_{t+1} \) as \( w_t + 1 \), feed \( h_t \) to the subsequent token layers, and get an output state \( g_{w_t} \) that is used to generate \( x_{w_t+1} \). In such a case, we have \( a_t = \text{GEN}(x_{w_t+1}) \).

Suppose that \( a_{xy} \) is the action sequence to generate a sentence \( x \) and its parse tree \( y \), then the joint distribution of \( x \) and \( y \) can be formulated as:

\[
p(x, y) = p(a_{xy}) = \prod_{t} p(a_t | a_{<t}),
\]

where \( p(a_t | a_{<t}) \) is computed by:

\[
p(\text{COMP} | a_{<t}) = p(y_t = 0 | a_{<t}),
\]

\[
p(\text{GEN}(x_{w_t+1}) | a_{<t}) = p(x_{w_t+1} | y_t = 1, a_{<t}) p(y_t = 1 | a_{<t}),
\]

\[
p(x_{w_t+1} | y_t = 1, a_{<t}) = \text{softmax}(\text{MLP}_y(g_{w_t})),
\]

\[
p(y_t | a_{<t}) = \text{softmax}(\text{MLP}_y(h_t)).
\]

MLP\( y(\cdot) \) and MLP\( x(\cdot) \) convert inputs to a 2-dim vector and a \(|V|\)-dim vector, respectively. By predicting action types through shallow layers and tokens through deep layers, we can keep the total computational cost close to that of vanilla Transformers.

3.2 Unsupervised Training

How to train an unsupervised SLM effectively and efficiently has always been a challenge. Existing methods suffer from two issues: asymmetric feedback and inability to train in parallel. The former arises from the uni-directional LM loss, and the latter stems from the inherent data dependency of each composition step on the representations of its sub-constituents from previous steps. We tackle both issues with an approach similar to hard-EM. In E-step, we employ a composition model to induce a parse tree through a pruned deep inside-outside encoder. In M-step, we update both the composition model and the generative model by optimizing a joint objective based on the induced tree. The composition and generative model are connected by sharing the same composition function. Below we present details of the two steps and explain how they tackle the issues mentioned above.

**E-step.** During the E-step, the composition model searches for the best parse tree and composes representations through a deep inside-outside
We then compute each internal span representation $e_{i,j}$ as a weighted average over all possible pairs of constituents$^3$: $${i, j}^{k}_p = f_o(i, i_k, i_{k+1}, i_{j}), \quad \omega_{i, j}^k = \phi_o(i, i_k, i_{k+1}, i_{j}),$$ $${\bar{w}}_{i, j}^k = \frac{\exp(\omega_{i, j}^k)}{\sum_{k'=1}^{2^n} \exp(\omega_{i, j}^{k'})}, \quad i_{1, 2} = \sum_{k=1}^{2^n} \bar{w}_{i, j}^k \omega_{i, j}^k.$$ 

in which $f_o$ and $\phi_o$ are formulated in Appendix A.2. An illustration to compute $i_{1, 3}$ is given in Figure 2(a). Analogously, the outside pass computes each outside representation $\alpha_{i, j}$ in a top-down manner based on bi-directional information outside span $(i, j)$. To accelerate computation, we use the pruned deep inside-outside encoder (Hu et al., 2024) which achieves linear space complexity and approximately logarithmic parallel time complexity. The details of the algorithm and the complete outside pass are presented in Appendix A.1.

Note that for a given span $(i, j)$, the best split-point is $k$ with the highest $\bar{w}_{i, j}^k$. Thus, to derive a parse tree, we can recursively select the best split-points top-down starting from the root span $(1, n)$.

The outside representations of tokens can be used to define an auto-encoding loss (i.e., predicting each token from its outside representation) for the composition model, which is optimized in the M-step:

$$L_{ae} = -\frac{1}{n} \sum_{i=1}^{n} \log \frac{\exp(\alpha_{i, j}^o)}{\sum_{k=1}^{2^n} \exp(\alpha_{i, j}^{o, k})},$$

where $e_k$ is the embedding of the $k$-th token in the vocabulary. As the auto-encoding loss provides feedback to each token representation from both sides of the token, the asymmetric feedback issue is addressed.

**M-step.** With the induced tree $y$, we update the parameters of the composition model and the generative model in a joint manner. Denote the sequence of node spans in post-order as $[(i_0, j_0), (i_1, j_1), \ldots, (i_{2n-1}, j_{2n-1})]$. The action sequence can be formulated as:

$$a_t = \begin{cases} \text{COMP} & \text{if } i_t < j_t \\ \text{GEN}(z_{i_t}) & \text{otherwise} \end{cases}$$

An auto-regression loss can be defined as:

$$L_{ar} = -\log p(x, y) = -\frac{1}{2n-1} \sum_{t=0}^{2n-1} \log p(a_t|a_{<t}).$$

However, even though the action sequence is given, there are still two challenges. First, there are data dependencies within the inputs for the generative model as mentioned earlier and shown in Figure 2(b), which impedes parallel training. Second, there are no feedforwards from the composition model to the generative model, so the two models are disconnected and hence cannot be trained jointly. A key insight to tackle these issues is to use the internal span representations $i_{1, j}$ as surrogates$^4$ of the inputs $x_{i, j}$ to the generative model as depicted in Figure 2(b)(ii). As the internal span representations are already computed in the E-step, they can be fed into Transformers seamlessly all at once to fully leverage the parallel training ability of the architecture. Moreover, replacing $x$ with $i$ enables the representations computed by the composition model to participate in the generative model, thus

---

$^3$A nuance between ours and the original DIORA is that we use the single-step compatibility score $\omega_{i, j}^k$ to estimate the split probability. We empirically find modeling in this way results in better grammar induction performance.

$^4$It is an approximation because $i_{1, j}$ is soft-weighted over all $i_{1, k}$ with the best split-point at $k$. Though $x_{i, j}$ is composed using the same composition function and with the same split-point $k$ during inference, it is in a one-hot manner. However, our experimental results indicate that such an approximation approach has minimal impact on actual inference.
the two models are connected and can be jointly optimized via the auto-regression loss. Note that internal span representations do not contain any information outside spans, so there is no information leakage in the uni-directional generative model.

The final training loss for the M-step combines the auto-regression and auto-encoding losses as:

$$\mathcal{L} = \mathcal{L}_{ae} + \mathcal{L}_{ar}.$$  

We empirically find that the combined loss leads to left-branching bias in parse trees induced by the composition model that is not observed when training with $\mathcal{L}_{ae}$ alone. A possible reason is that left-leaning trees provide more left-side context for each step during generation and thus are reinforced in learning. To tackle the issue, we stop gradient propagation from $\mathcal{L}_{ar}$ to $a_{i,j}$, which means only gradients from $\mathcal{L}_{ae}$ are allowed to be back-propagated along $a_{i,j}$. Note that other variables like $i_{i,j}$ still receive gradients from $\mathcal{L}_{ar}$.

### 3.3 Inference

The action space of $\text{GEN}(x)$ is much larger than that of $\text{COMP}$, leading to an imbalance between their probabilities. Stern et al. (2017) point out that during beam search decoding, hypotheses in a beam should be grouped by the length of generated tokens instead of action history, which they refer to as “word-level search”. However, their approach does not guarantee that the top-k next words searched are the optimal ones. To address the issue, we propose an improved word-level search tailored for our generation paradigm. The core idea is to guarantee that all hypotheses in a beam have the same number of $\text{GEN}(x)$ actions. Beams satisfying the condition are marked as $\text{sync}$ and otherwise $\text{sync}$. Below we depict the entire word-level search process through an example shown in Figure 3:

1. Starting with a $\text{sync}$ beam, e.g., $A, B, C$ and $(A, B), D$, we estimate the probability distribution of the next action for each hypothesis within it. For each possible action, we compute the probability of the resulting new hypothesis as the product of the probabilities of the current hypothesis and the action. The new hypotheses are pooled and ranked, and the top-k are retained (e.g., $A, (B, C)$ and $(A, B), D, E$).
2. If the current beam contains hypotheses with the last step being $\text{COMP}$, e.g., $A, (B, C)$, we continue to explore their next actions, update their probabilities, pool them with other hypotheses in the beam, and rank the top-k, until all the top-k hypotheses have $\text{GEN}(x)$ as their last action.
3. End generation upon reaching the length limit or producing an end token; otherwise, go back to step 1.

This method is also applicable to top-k random sampling (Fan et al., 2018), or parsing with a given input sentence by simply setting the probabilities of all $\text{GEN}(x)$ actions to zeros except for the correct next token.

### 4 Experiments

To fairly compare GPST and GPT-2, we pre-train both models from scratch on the same corpus with the same setups and comparable parameter sizes. Evaluation is conducted on various language understanding/generation tasks. Besides, we also evaluate GPST on grammar induction to verify to what extent the induced parse trees are consistent with human annotation.

**Pre-training Corpus.** We pre-train models on WikiText-103 (Merity et al., 2017) and OpenWebText (Gokaslan and Cohen, 2019), where the two datasets contain 116 million tokens and 9 billion tokens, respectively. The context window size in pre-training is set to 1024. When a context involves more than one complete sentence, parse trees are induced for each sentence separately.

**Hyper-parameters.** Following GPT-2 (Radford et al., 2019), we use 768/1024-dimensional embeddings, a vocabulary size of 30522, 3072/4096-dimensional hidden layer representations, and
12/16 attention heads for the generative models of GPST_{small} and GPST_{medium}, respectively. To align with Transformer layer counts in GPT-2, we configure GPST_{small} with 3 type layers and 9 token layers, and GPST_{medium} with 3 type layers and 21 token layers, respectively. We set the input dimension of the composition model to 256/512, and the number of Transformer layers used in the composition function and decomposition function to 4 and 1, corresponding to the small and medium setups. To compare with GPT-2 under the same parameter sizes, we provide GPT-2_{small,13} and GPT-2_{medium,25} as additional baselines, representing models with 13 and 25 layers, respectively. The token embeddings are down-scaled before being fed into the composition model, and the constituent representations are up-scaled before being fed into GPST. All models are trained on 8 A100 GPUs with a learning rate of 5e-5/1e-4, 8 x 32 x 1024 tokens per step, 5 billion and 15 billion total training tokens for WikiText-103 and OpenWebText, respectively.

### 4.1 Understanding Tasks

**Dataset.** We evaluate GPST on the GLUE benchmark (Wang et al., 2018), which collects tasks covering a broad range of natural language understanding (NLU) domains.

**Evaluation Settings.** We borrow and minimally modify the fine-tuning paradigm from Radford et al. (2018). Details are described in Appendix A.3. As the whole sentence is given, the composition model is utilized to induce the best tree and compose constituent representations as described in the E-step. The constituent representations in the induced tree are gathered in post-order as inputs for the generative model. We derive two additional baselines GPST_{w/o surrogate} and GPST_{w/o grad.stop} for ablation study. In GPST_{w/o surrogate}, all constituent representations of non-terminals are replaced by embeddings of a placeholder COMP as in Transformer Grammars (Sartran et al., 2022), and thus there is no interaction between the composition model and the generative model (i.e., they are separately optimized). In GPST_{w/o grad.stop}, partial gradient stopping is disabled to study the impact of left-leaning trees on downstream tasks. We run three rounds of fine-tuning with different seeds and report average results (accuracy by default) on the validation sets.

### Results and Discussions

Table 1 reports the results on the GLUE benchmark. GPST significantly outperforms GPT-2 in both small and medium setups. We find that GPST_{w/o grad.stop} and GPST_{w/o surrogate} underperform GPST, but are still better than GPT-2. The performance drop of GPST_{w/o grad.stop} indicates that poor structures compromise the performance of downstream tasks. GPST_{w/o surrogate} is better than GPT-2, implying that as long as induced syntactic structures are utilized, simply replacing non-terminal representations with COMP embeddings is also helpful. It is, however, worse than GPST, demonstrating that computing representations of non-terminal constituents via an explicit composition function further benefits language understanding. One more interesting thing we find is that GPST consistently and most significantly outperforms baselines on the RTE task. One possible explanation is that certain relationships in RTE are predicated on negation words such as “not”, which generally affects high-level semantics through compositions with other phrases. Explicit
syntactic composition modeling contributes to a better representation of such cases.

### 4.2 Generation Tasks

#### 4.2.1 Abstractive Summarization

**Datasets.** We conduct experiments on three summarization datasets: BBC extreme (XSum) (Narayan et al., 2018), CNN and DailyMail (Nallapati et al., 2016), and Gigaword (Napoles et al., 2012) to assess the performance of GPST in terms of language generation abilities. Statistics of the datasets are presented in Table 6 in Appendix A.4.

**Evaluation Settings.** For XSum and CNN/DailyMail, we truncate the documents and their summaries to 900 and 100 tokens respectively, and concatenate them with short prompt Summary:. For Gigaword, the truncating thresholds of documents and summaries are set to 400 and 120 respectively, following the settings of Rothe et al. (2020). Considering the complexity of the generation task, we primarily evaluate the models pre-trained on OpenWebText. More details are described in Appendix A.5. We apply the word-level search described in §3.3 to top-k random sampling for GPST, except for models with w/o sync which only uses naive action-level beam search. ROUGE (Lin and Hovy, 2003) is employed as the evaluation metric.

#### 4.2.2 Syntactic Generalization

**Datasets.** The syntactic generalization task (Hu et al., 2020) collects 34 test suites to assess syntactic generalizability of the models. The test suites are grouped into 6 circuits: Agreement (Agr.), Center Embedding (C.E), Garden-Path Effects (G.P.E), Cross Syntactic Expectation (C.S.E), Licensing (L.cs.) and Long-Distance Dependencies (L.D.D.).

**Evaluation Settings.** We evaluate models on syntactic generalization test suites by comparing surprisals (Hale, 2001) without fine-tuning, as required by Hu et al. (2020). Surprisal: \( S(w|C) = - \log_2 p(w|C) \) is defined as negative log conditional probabilities of a sub-sentence \( w \) given the left-side context \( C \). In detail, when we apply word-level search with beam size \( b \) to do left-to-right parsing with a given input, we temporarily store \( b \) best hypotheses with their probability \( p(x_{<t}, y_{<n(t)}) \) at each token position \( t \), in which \( y_{<n(t)} \) refers to the current latent structure before generating \( x_t \). We marginalize \( y_{<n(t)} \) out of \( p(x_{<t}, y_{<n(t)}) \) by summing up all the probabilities of the \( b \) best hypotheses. Finally, we obtain the surprisal of a sub-sentence with starting position \( s \) and ending position \( e \) as \( S(w|C) = - \log p(x_{<e}) + \log p(x_{s-1}) \).

To align with Murty et al. (2023) and Sartran et al. (2022), we set beam size \( b \) to 300.

#### 4.2.3 Results and Discussions

Table 2 and 3 report the results of summarization and syntactic generalization tasks. Overall, the performance of GPST is comparable to GPT, with a slight advantage. One possible reason why the advantage of GPST on generalization tasks is not as significant as that on GLUE is the discrepancies between training and inference. During training, the constituent representations are computed via the inside algorithm, where the representations are soft-weighed over composed representations of valid sub-constituents. However, during inference, constituent representations are composed of the top two elements in the stack, which is a one-hot version of the inside algorithm. This issue could potentially be resolved using a hard inside-outside algorithm (Drozdov et al., 2020), which we may explore in our future work. Despite the discrepancies, our performance still slightly surpasses that of GPT-2, which adequately demonstrates the potential of GPST in generation tasks. One more interesting thing is that GPST\textsubscript{medium} even outperforms baselines with gold trees in the syntactic generalization task, and the results of all GPSTs maintain a lead on Garden-Path Effect. Note that the results have a large variance due to the relatively small size of the evaluation set, e.g., GPT\textsubscript{medium} even underperforms GPT\textsubscript{small}. However, the results still imply that unsupervised syntactic LMs have reached a critical point where they can surpass approaches reliant on gold trees.

### 4.3 Grammar Induction

**Baselines & Dataset.** We select baselines that report unsupervised left-to-right parsing results: Neural variational (NV) approaches (Li et al., 2019) and PRPN (Shen et al., 2018). For reference, we also select some baselines performing parsing requiring whole sentence visible: URNNG (Kim et al., 2019b), C-PCFG (Kim et al., 2019a), DIORA (Drozdov et al., 2019), ON-LSTM (Shen et al., 2019b), Fast-R2D2 (Hu et al., 2022) and GPST using the deep inside algorithm. We report their performance on PTB (Marcus et al., 1993). Besides, we also report results of GPST\textsubscript{w/o grad.stop} and GPST\textsubscript{w/o surrogate} for checking the gains from partial
Table 2: Abstractive summarization results.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>GPT-2small</td>
<td>31.66 10.91 25.16 22.58</td>
<td>37.18 15.23 25.59 26.00</td>
<td>33.13 15.27 30.85 26.42</td>
<td>32.71 15.12 31.28 26.68</td>
<td>33.19 15.27 30.91 26.46</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPT-2small,13</td>
<td>32.45 1.1 35.52</td>
<td>37.18 15.23 25.59 26.00</td>
<td>33.13 15.27 30.85 26.42</td>
<td>32.71 15.12 31.28 26.68</td>
<td>33.19 15.27 30.91 26.46</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPT-2small-w/o sync</td>
<td>32.45 1.1 35.52</td>
<td>37.18 15.23 25.59 26.00</td>
<td>33.13 15.27 30.85 26.42</td>
<td>32.71 15.12 31.28 26.68</td>
<td>33.19 15.27 30.91 26.46</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPT-2small-w/o grad.stop</td>
<td>32.45 1.1 35.52</td>
<td>37.18 15.23 25.59 26.00</td>
<td>33.13 15.27 30.85 26.42</td>
<td>32.71 15.12 31.28 26.68</td>
<td>33.19 15.27 30.91 26.46</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPT-2small-w/o surrogate</td>
<td>32.45 1.1 35.52</td>
<td>37.18 15.23 25.59 26.00</td>
<td>33.13 15.27 30.85 26.42</td>
<td>32.71 15.12 31.28 26.68</td>
<td>33.19 15.27 30.91 26.46</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Syntactic generalization results. For reference, we list the results of models with gold trees from Sartran et al. (2022) and Murty et al. (2023).

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>WikiText-103</td>
<td>86.41</td>
<td>94.64</td>
<td>95.04</td>
<td>85.71</td>
<td>79.38</td>
<td>82.95</td>
<td></td>
</tr>
<tr>
<td>PRPN</td>
<td>78.95</td>
<td>87.50</td>
<td>82.22</td>
<td>79.83</td>
<td>71.58</td>
<td>83.29</td>
<td></td>
</tr>
<tr>
<td>GPST</td>
<td>77.19</td>
<td>85.71</td>
<td>94.54</td>
<td>96.74</td>
<td>68.95</td>
<td>72.38</td>
<td></td>
</tr>
<tr>
<td>GPST small</td>
<td>64.91</td>
<td>94.64</td>
<td>86.41</td>
<td>98.91</td>
<td>73.42</td>
<td>79.38</td>
<td></td>
</tr>
<tr>
<td>GPST medium</td>
<td>85.96</td>
<td>85.71</td>
<td>95.04</td>
<td>94.57</td>
<td>83.68</td>
<td>78.17</td>
<td></td>
</tr>
</tbody>
</table>

For Reference (Models with gold trees)

| TG         | 69.57 | 88.43 | 90.43  | 95.67  | 78.11 | 77.93  |
| Pushdown Layers | 79.00 | 92.00 | 84.21  | 100.00 | 77.78 | 77.58  |

4.4 Training Efficiency

Finally, we conduct a fair comparison of training efficiency with other unsupervised SLMs. We keep the model sizes and memory usage comparable and the training tokens the same. We report their
time consumption in Table 5, from which we can observe the huge advantage of GPST over the baselines in terms of efficiency.

<table>
<thead>
<tr>
<th></th>
<th>#param.</th>
<th>sentence length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>128</td>
</tr>
<tr>
<td>GPST</td>
<td>24M</td>
<td>1x</td>
</tr>
<tr>
<td>URNNG</td>
<td>23M</td>
<td>130.6x</td>
</tr>
<tr>
<td>OM</td>
<td>28M</td>
<td>2.0x</td>
</tr>
</tbody>
</table>

Table 5: Training acceleration on the same number of tokens.

5 Conclusion

In this paper, we propose an unsupervised approach to train GPST at scale efficiently. A key insight of our work is to guide the left-to-right structural learning with symmetric supervision such as an auto-encoding loss, which can receive feedback from both sides. A key technical contribution is that we propose the representation surrogate which enables joint training of all components in parallel. Besides, the composition model of GPST can be regarded as an enhancement to the conventional embedding layer, which provides context-invariant embeddings of various granularities beyond token embeddings. Our experiment results show the superiority of GPST on language understanding, generation, and left-to-right grammar induction, which demonstrate the potential of GPST as a foundational architecture for large language models.

6 Limitation

Despite GPST achieving a multiple-fold acceleration compared to previous syntactic language models, it still requires 1.5 to 5 times the training time compared to vanilla GPTs. The more layers there are in the type/token layers, the lower the overall time multiplier becomes. The additional training time comes from the composition model which only accounts for one-tenth of the overall model parameters. Due to the unpredictable memory overhead at each step, many fragmented memories are generated, resulting in PyTorch having to spend extra time cleaning up the memory cache periodically. Meanwhile, our implementation is quite naive, without any operator fusion or hardware-aware implementation. Thus there should be multiple potential ways to further reduce the time consumption of the composition model in the future.

7 Acknowledgement

This work was supported by Ant Group through the CCF-Ant Research Fund.
References


John Cocke. 1969. **Programming Languages and Their Compilers: Preliminary Notes.** New York University, USA.


Brian DuSell and David Chiang. 2024. **Stack attention: Improving the ability of transformers to model hierarchical patterns.** In *The Twelfth International Conference on Learning Representations*.


A Appendix

A.1 Pruned inside-outside algorithm

Fast-R2D2 (Hu et al., 2022) introduces a pruned variant of the inside algorithm that reduces its complexity from \(O(n^3)\) to \(O(n)\) in both space and time. Building on this, ReCAT (Hu et al., 2024) extends the pruning method to the inside-outside algorithm, and further enables it to complete in approximate \(\log n\) steps, whose key idea is to prune out unnecessary cells in the chart-table and encode cells in different rows simultaneously. The main idea of the pruning process is to decide which two spans should be merged at each step during the inside pass and prune out cells that would break the non-splittable span. An unsupervised top-down parser is applied to determine the merge order of spans. Given a sentence \(x = [x_1, x_2, \ldots, x_n]\), the top-down parser assigns each split point a score \(v_i\) s.t. \(1 \leq i \leq n - 1\), and recursively split the sentence into two in the descending order of the scores shown in Figure 4 (a). Hence, the reverse order of the split points could be used to decide which cells to merge. Specifically, the pruned inside-outside algorithm works as follows:

0. Prepare merge batches according to the height of merge points in the induced tree, with the lowest merge points in the first batch, as illustrated in Figure 4(b).

1. Merge each pair of adjacent cells into one according to the current merge batch. For example, in Figure 5(b), at merge point 1, we merge \(x_1\) and \(x_2\) into \(x_{1:2}\); at merge point 5, we merge \(x_5\) and \(x_6\) into \(x_{5:6}\).

2. Remove all conflicting cells that would break the now non-splittable span from Step 1, e.g., the dark cells in Figure 5(c), and reorganize the chart table much like in the Tetris game as in (d).

3. Encode the cells that just descend to height \(m\) and record their valid splits in \(K\), e.g., the cells highlighted with stripes in Figure 5(d) with valid splits \{2, 3\} for span (1, 4) and \{3, 4\} span (3, 6). Go back to Step 1 until no blank cells are left.

Therefore, the entire inside process can be completed within steps equal to the height of the tree. Using the valid splits \(K\) recorded for each cell during the pruning process, we now have the new inside state transition equation as:

\[
\bar{a}_{i,j}^k = \phi_a(\{i, k, k+1, j\}) \quad \bar{w}_{i,j}^k = \frac{\exp(\bar{a}_{i,j}^k)}{\sum_{k' \in K_{i,j}} \exp(\bar{a}_{i,j}^{k'})}, \quad h_{i,j} = \sum_{k \in K_{i,j}} \bar{w}_{i,j}^k \bar{h}_{i,j}^k
\]

where \(K_{i,j}\) is the valid splits set for span \((i, j)\). According to \(K\), we can obtain a mapping from a span to its immediate sub-spans. By reversing such mapping, we get a mapping from a span to its valid immediate parent spans denoted as \(P\), which records the non-overlapping endpoint \(k\) in the parent span \((i, k)\) or \((k, j)\) for a given span \((i, j)\).

Thus, for the outside pass, we have:

\[
\hat{o}_{i,j}^k = \begin{cases} f_\beta(\hat{o}_{i,k}, i_{j+1,k}) & \text{if } k > j \\ f_\beta(\hat{o}_{i,j}, k_{i-1}) & \text{if } k < i \end{cases} \quad \hat{w}_{i,j}^k = \frac{\exp(\hat{h}_{i,j}^k)}{\sum_{k' \in P_{i,j}} \exp(\hat{h}_{i,j}^{k'})}, \quad \hat{h}_{i,j}^k = \sum_{k \in P_{i,j}} \hat{w}_{i,j}^k \hat{h}_{i,j}^k
\]

We optimize the top-down parser jointly at the M-step with GPST. Given the parse tree \(y\) induced at the E-step, we maximize \(p(y|x; \Theta)\) for the top-down parser, whose parameters are denoted as \(\Theta\). As shown in Figure 4(a), at \(t\) step, the span corresponding to a given split \(a_t\) is determined, which is denoted as \((i^t, j^t)\). Thus we can minimize the negative log-likelihood of the parser as follows:

\[
\mathcal{L}_p = -\log p(y|x; \Theta) = -\sum_{t=1}^{n-1} \log p(a_t|x; \Theta).
\]

We notice that the steps to finish the pruned inside-outside algorithm depend on the highest tree in a batch, thus any extremely skewed tree may result in a significant increase in time consumption. A straightforward approach to reduce the maximum height of parse trees in a batch is to introduce a height penalty. During the inside pass, the weighted tree height of span \((i, j)\) could be computed as:

\[
\bar{h}_{i,j} = \max(h_{i,k} + h_{k+1,j}) + 1, \quad h_{i,j} = \sum_{k \in K_{i,j}} \bar{w}_{i,j} \bar{h}_{i,j}
\]

To minimize the impact of height penalties on grammar induction, we set a threshold \(H_{thr}\), which is 15 by default. Only trees that exceed this threshold will be affected.

\[
\mathcal{L}_h = \frac{1}{n} \max(h_{1:n} - H_{thr}, 0)
\]

Thus the final auto-encoding objective is:

\[
\mathcal{L}^{ne}_a = \mathcal{L}_{ae} + \mathcal{L}_h + \mathcal{L}_p
\]
Figure 4: Fast encoding follows the order given by a top-down parser, with the merging order \( M \) being the reverse order of the split point sequence \( A \). \( x_i \) denotes the \( i \)th token in a sentence of length 6. Numbers in \( A \) and \( M \) denote the indices of the split/merge point between tokens. \( v_j \) denotes the split score of \( j \)th split point, predicted by the top-down parser.

Figure 5: The initial step of encoding in \( O(\log n) \) steps. The numbers in blue correspond to the indices of the split points introduced in Figure 4.

A.2 Composition function and score function

![Model illustrations for the composition and decomposition functions.](image)

We borrow the idea from Hu et al. (2021) to use Transformers as the backbone of the composition function \( f_{\alpha} \). As shown in Figure 6(a), composition function \( f_{\alpha} \) takes left/right constituent representations \( \ell_{i,k}/\ell_{k+1,j} \) along with their role embeddings \([\text{LEFT}]/[\text{RIGHT}]\) into N-layered Transformers as inputs, passes the summation of their corresponding outputs through a layer normalization layer to get the composed representation. Decomposition function \( f_{\beta} \) works analogously as shown in Figure 6(b) and (c), with \([\text{PRT}] \) as the role embedding for parents.

We define the score function \( \phi_{\alpha} \) as:

\[
\phi_{\alpha}(l, r) = \text{MLP}_{\alpha}^l(l)^T \text{MLP}_{\alpha}^r(r)/\sqrt{d}
\]

where \( l \) and \( r \) are representations for left/right constituents. \( \text{MLP}_{\alpha}^l \) and \( \text{MLP}_{\alpha}^r \) are used to capture syntactic features from the left and right inputs, which convert inputs to \( d \)-dimensional vectors. Analogously, \( \phi_{\beta} \) is defined as:

\[
\phi_{\beta}(p, c) = \text{MLP}_{\beta}^c(p)^T \text{MLP}_{\beta}^p(c)/\sqrt{d}
\]

\[
\phi_{\beta}(p, r) = \text{MLP}_{\beta}^r(p)^T \text{MLP}_{\beta}^p(r)/\sqrt{d}
\]

where \( p \) is the outside representation of a parent. \( \text{MLP}_{\beta}^c, \text{MLP}_{\beta}^r, \) and \( \text{MLP}_{\beta}^p \) are used to capture features from left/right children and parents respectively.

A.3 Glue fine-tuning

In detail, we append a CLS token after the input sequence and then feed the hidden states of the CLS tokens to a linear layer as the logits for classification. An additional cross-entropy loss along with the pre-training objective is used during fine-tuning.

A.4 Summarization dataset statistics

BBC extreme (XSum) comprises 204k document-summary pairs for single-sentence summarization of long documents. CNN and DailyMail (CNN/DailyMail) contains 287k training pairs, each consisting of a document annotated with highlights. Gigaword focuses on sentence summarization with 3.8M sentence-summary training pairs conversely. We organize the statistics in Table 6.

A.5 Summarization fine-tuning

We fine-tune for 15 epochs with a batch size of 16 on XSum and CNN/DailyMail datasets. For Gigaword, we fine-tune for 10 epochs with a batch size of 16.
size of 64. Top-k random sampling with $k = 2$ is used as the basic inference method as suggested in GPT-2 (Radford et al., 2019).

### A.6 Comparison of parameter sizes

<table>
<thead>
<tr>
<th>model</th>
<th>MLP</th>
<th>QKV linear</th>
<th>score linear</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>comp. func.</td>
<td>256<em>1024</em>2</td>
<td>256<em>256</em>3</td>
<td>256<em>128</em>2</td>
<td>786,432</td>
</tr>
<tr>
<td>GPT-2</td>
<td>768<em>3072</em>2</td>
<td>768<em>768</em>3</td>
<td>—</td>
<td>6,488,064</td>
</tr>
</tbody>
</table>

Table 7: The comparison of parameter size of a single layer in the composition function of GPST$_{small}$ and GPT-2$_{small}$.

### A.7 Author Contributions Statement

We list our contributions below:

- Xiang Hu: proposing the unsupervised training approach, implementing GPST, pre-training and decoding algorithms, and paper writing.
- Pengyu Ji: GPST fine-tuning, experiments code, running experiments, and paper writing.

### A.8 Case studies

Please refer to the following pages.
<table>
<thead>
<tr>
<th>System</th>
<th>Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left to right</td>
<td>Skipper’s said the merger will help finance remodeling and future growth</td>
</tr>
<tr>
<td>Left to right w/o sync</td>
<td>Skipper’s said the merger will help finance remodeling and future growth</td>
</tr>
<tr>
<td>Inside</td>
<td>Skipper’s said the merger will help finance remodeling and future growth</td>
</tr>
<tr>
<td>Gold</td>
<td>Skipper’s said the merger will help finance remodeling and future growth</td>
</tr>
<tr>
<td>Left to right</td>
<td>Yesterday the stock market’s influence at first created nervousness</td>
</tr>
<tr>
<td>Left to right w/o sync</td>
<td>Yesterday the stock market’s influence at first created nervousness</td>
</tr>
<tr>
<td>Inside</td>
<td>Yesterday the stock market’s influence at first created nervousness</td>
</tr>
<tr>
<td>Gold</td>
<td>Yesterday the stock market’s influence at first created nervousness</td>
</tr>
<tr>
<td>Left to right</td>
<td>And yesterday the top performing industry group was oil field equipment issues</td>
</tr>
<tr>
<td>Left to right w/o sync</td>
<td>And yesterday the top performing industry group was oil field equipment issues</td>
</tr>
<tr>
<td>Inside</td>
<td>And yesterday the top performing industry group was oil field equipment issues</td>
</tr>
<tr>
<td>Gold</td>
<td>And yesterday the top performing industry group was oil field equipment issues</td>
</tr>
<tr>
<td>System</td>
<td>Tree</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>Left to right</td>
<td>All told the federal government already guarantees more than 900 billion of mortgages</td>
</tr>
<tr>
<td>Left to right w/o sync</td>
<td>All told the federal government already guarantees more than 900 billion of mortgages</td>
</tr>
<tr>
<td>Inside</td>
<td>All told the federal government already guarantees more than 900 billion of mortgages</td>
</tr>
<tr>
<td>Gold</td>
<td>All told the federal government already guarantees more than 900 billion of mortgages</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>