Beyond Sequence Order: Syntax-Informed Positional Embeddings for Transformers

Anonymous ACL submission

Abstract

Positional embeddings (PE) in Transformers encode token distance and order but are largely agnostic to *syntactic structure*. We introduce **S**yntax-**i**nformed **P**ositional **E**mbeddings (**SiPE**), which learn a lightweight syntactic prior from dependency parses during pretraining and inject it into the positional pathway of any PE scheme—absolute, relative, or rotary—for both encoders and decoders, leaving self-attention and the rest of the Transformer untouched. Our central finding is that biasing the positional pathway alone is the strongest recipe: it outperforms injecting the same bias into self-attention, or into both sites jointly, across syntactic generalization and language understanding benchmarks. We further isolate *where* and *how* the prior should enter the model, finding it is best combined directly with positional information rather than first with token embeddings. SiPE gives consistent gains—e.g., $+10.3$ points on SyntaxGym and 8.2% relative improvement on GLUE over a baseline decoder LM, alongside lower perplexity, while incurring overhead that is only linear in sequence length. Unlike prior syntactic LMs that marginalize over many parses at inference, SiPE conditions on a single parse, retaining explicit syntax at runtime at low cost.

1 Introduction

The transformer architecture (Vaswani et al., 2017) forms the foundation of modern LLMs. While the impact of LLMs is substantial, several linguistic simplifications were introduced in the underlying architecture to improve scalability. One key choice was the use of positional embeddings (PE) to encode each token's location within a sequence. While this approach preserved computational efficiency and scalability, it reduced positional information to a simple ordering mechanism—sufficient to indicate *where* tokens appear, but insufficient to capture the *syntactic relations* that form the scaffolding for downstream semantics. That is, without explicitly modeling syntactic compositionality, semantic frames (Baker et al., 1998; Kingsbury and Palmer, 2002) are more likely to be extracted incorrectly, increasing the risk of misinterpretation for LLM-based agents. Consider, for example, this hypothetical agent request: *"Please move this large file to another folder."* To execute the requested action correctly, the *move* predicate must be associated with its arguments: the object to be moved (*file*) and the destination (*folder*). Absolute positional encodings (Devlin et al., 2019; Liu et al., 2019), provide no signal to link this predicate to its arguments. Relative positional encodings, e.g., DeBERTa (He et al., 2021) and Transformer-XL (Dai et al., 2019), weaken this signal. For instance, the relative distance between *move* and *folder* is large (six words), even though a direct `oblique` syntactic dependency connects the verb to its mod-

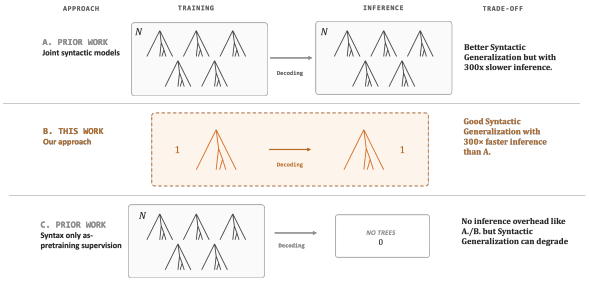

Figure 1: **Our syntax-informed positional embeddings (SiPE) move the Pareto frontier between syntactic supervision and inference cost.** Prior work sits at two extremes: *Type A* syntactic LMs (TG (Sartran et al., 2022), DTG (Zhao et al., 2024), pushdown LMs (Murty et al., 2023)) parse at inference, gaining syntactic competence but paying for marginalization over parses ($\sim$300 trees); *Type C* methods (TreeReg (Nandi et al., 2025), tree-planted transformers (Yoshida et al., 2024)) inject syntax only in training and discard the parser, cheapening inference at the cost of syntactic knowledge. SiPE is a middle ground: a single lightweight tree guides decoding without marginalization, avoiding Type A's cost while outperforming Type C on most syntactic benchmarks.

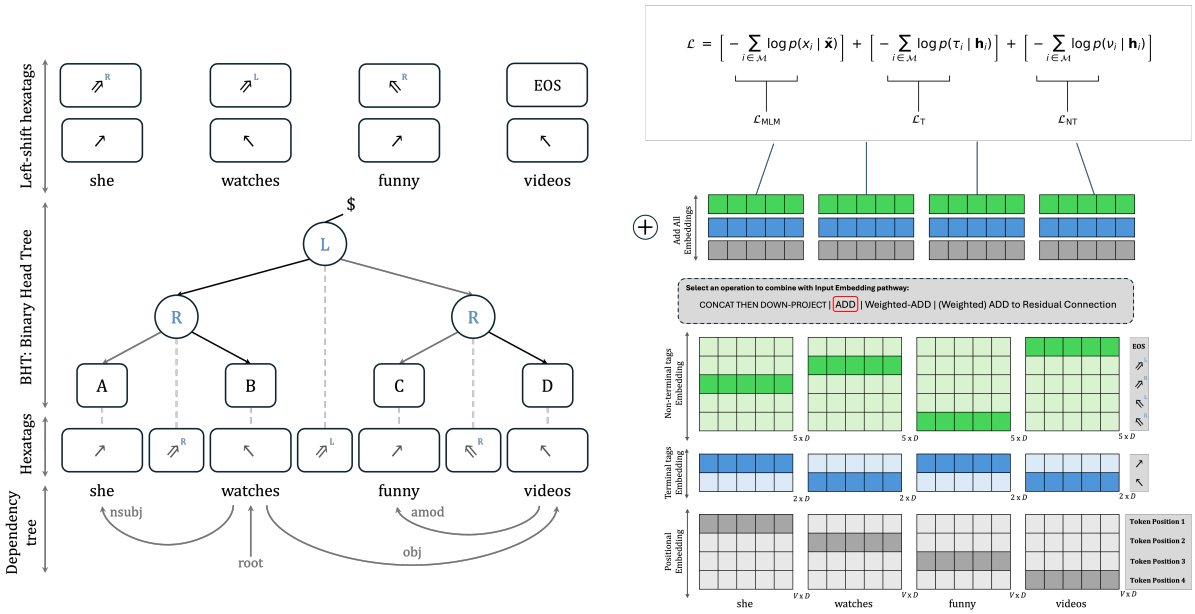

Figure 2: **Left:** From bottom to top, the figure shows the correspondence between dependency arcs, binary head tree (BHT), and hexatags (left-shifted) for the sentence "she watches funny videos." **Right:** Our prior-injection method, where each token combines positional, terminal, and non-terminal tag embeddings (lookup tables of dimension D) via an injection operation (ADD, CONCAT, or Weighted-ADD), and the hexatag embeddings are learned through an auxiliary tag-prediction objective alongside the primary language-modeling loss (MLM for encoders, next token prediction for decoders); see subsection 3.1.

ifier (De Marneffe et al., 2021).

Partially due to this shallower representation, language models (LMs) still exhibit weak compositional generalization (Guo et al., 2020). Prior work injects hierarchical inductive bias by constraining self-attention using constituency/dependency structure, often via quadratic token-token interactions and hard-coded attention masks (Sartran et al., 2022; Zhao et al., 2024; Murty et al., 2023; Qian et al., 2021; Xie et al., 2021a). This is a complex process that requires considerable changes in the transformer architecture and which increases the attention mechanism's computational overhead (e.g., Pushdown LMs (Murty et al., 2023) maintain a recursively updated stack over the sequence that reshapes the attention pattern at every step).

The contributions of our paper are:

(1) We introduce *syntactic positional embeddings* (SiPE) that augment a model's positional pathway with a lightweight prior derived from linearized dependency trees—coarse directional indicators relating each token to its syntactic governor (§3), learned during pre-training via an auxiliary indicator-prediction objective alongside the LM loss.

(2) SiPE composes with all three dominant positional encoding families—absolute (Liu et al., 2019), relative (He et al., 2021; Dai et al., 2019), and rotary (Warner et al., 2024)—without modifying self-attention or any other Transformer component.

(3) For encoders, SiPE improves BLiMP (Warstadt et al., 2023) syntactic generalization across all three families (up to $+2.3\%$ relative for ModernBERT), with gains compounding under continued pre-training on BLLIP-LG ($+3\%$ for DeBERTa-v3 OOD), indicating the prior generalizes beyond its pre-training distribution (Table 1). The same models also improve on GLUE (Wang et al., 2019) at both small and large scale (tables Table 2, 3 and 4).

(4) For autoregressive modeling on Transformer-XL, injecting through the positional pathway alone is the strongest single recipe: it achieves the best SyntaxGym (Hu et al., 2020) score of any configuration of our method (80.60, $+10.3$ over the no-syntax baseline) while cutting perplexity by 9.0%, and drives a dramatic $+8.2\%$ relative improvement on GLUE for the same model (Table 4).

(5) As shown in Figure 1, prior syntactic LMs sit at two extremes: parsing at inference (expensive marginalization over parses) or only at training (discarding structure at runtime, with weaker syntactic generalization). SiPE moves the Pareto frontier

between these by conditioning on a *single* parse at inference. The injection of syntactic information adds no asymptotic cost over self-attention,[1] and unlike training-only methods, SiPE leaves downstream usage unchanged—models remain compatible with continual pre-training, instruction tuning, SFT, RLHF/RLVR, and task fine-tuning. Empirically we outperform parser-free approaches—Tree-Planted Transformers (Yoshida et al., 2024) and TreeReg (Nandi et al., 2025)—on both BLLIP-LG perplexity and SyntaxGym (Table 5).

2 Related Work, Background, and Notations

Prior work shows that pretraining on formal languages or injecting tree-based signals can improve syntactic generalization and data efficiency (Hu et al., 2025). However, most methods encode syntax by constraining self-attention via pairwise token interactions and hard-coded (often dynamically updated) attention masks, adding parameters and training complexity (Xie et al., 2021a; Sartran et al., 2022; Qian et al., 2021; Murty et al., 2023). We instead sidestep this by encoding dependency structure as two lightweight prior vectors derived from *Hexatagging*, and apply these priors to the positional embedding pathway. Modern Transformers encode position through three dominant schemes: *absolute* embeddings added to the input residual (Liu et al., 2019), *relative* embeddings indexed by offset $i - j$ inside attention (He et al., 2023; Dai et al., 2019), and *rotary* embeddings that rotate queries and keys by position-dependent angles (Su et al., 2023). We show that our method seamlessly composes with all three PE types (subsection 3.1).

Hexatagging. In this work, we leverage *Hexatagger* (Amini et al., 2023), a "parsing-as-tagging" dependency parser that assigns each token two types of discrete syntactic labels (analogous to position or subword IDs): (a) the position of its syntactic governor (which can further be decomposed into two subtypes) and (b) the dependency relation. This formulation can be mapped to simple embedding lookup tables that can be linearly combined with token embeddings without modifying self-attention. Hexatagger maps a projective dependency tree to a *binary head tree* (BHT, a special form of constituency tree) by binarizing the structure and labeling each internal node with L/R to indicate whether the span head lies in the left or right subtree, then linearizes the BHT via an in-order (left-corner) traversal. Each visited node yields a tag encoding its attachment direction (left vs. right child), and for non-terminals, the head-direction (L vs. R). This yields a fixed projective tag inventory with terminal tags $\nearrow$ and $\nwarrow$, plus non-terminal tags $\nearrow^{L}$, $\nearrow^{R}$, $\nwarrow^{L}$, and $\nwarrow^{R}$ (Figure 2, left). This avoids unbounded (length-dependent) tag sets whose cardinality grows with the input length; instead each token's tag can be predicted independently with a linear classifier. In our implementation, we further append a dedicated EOS non-terminal (via a left shift of between-token non-terminals) to obtain one non-terminal label per token.[1] In this work, we omit dependency relation labels, finding that terminal/non-terminal hexatags alone provide sufficient syntactic inductive bias to improve downstream language modeling performance.

3 Approach

3.1 Deriving a Positional Syntactic Bias

An inductive bias can enter the PE layer in three ways (Figure 2): (a) *entangled with the token embedding*, applied to the input before any positional or attention information is introduced; (b) *composed with the default PE scheme* under some operation; or (c) *disentangled*, a separate additive term in the attention score that modulates each head. Linearizing the binary head tree into discrete tags (the *hexatagging* procedure of section 2) yields two embedding tables, one per tag type, composable with the PE mechanism under any of these operations. At position p they contribute

$$m_p \cdot \left(\mathbf{E}^{T}_{\tau_p} + \mathbf{E}^{N}_{\nu_p}\right), \tag{1}$$

where $\mathbf{E}^{T} \in \mathbb{R}^{|\mathcal{T}|\times d}$ and $\mathbf{E}^{N} \in \mathbb{R}^{|\mathcal{N}|\times d}$ are learned tables ($|\mathcal{T}|$=2 terminals, $|\mathcal{N}|$=5 non-terminals), and $m_p \in \{0, 1\}$ is the first-subword mask, equal to 1 only at each word's first subword so the hexatag is injected once per word (subsection A.1). Tables are randomly initialized and trained jointly with the LM objective.

[1] Dependency tags are predicted in parallel by a lightweight `DeBERTa-v3-base` classifier whose overhead on top of the encoder's is linear in input length; see Appendix B. Implemented in-house: shared encoder, two per-token tag heads. Repository URL hidden for anonymity.

3.2 RoBERTa: Absolute Positional Embeddings

We begin with the simplest case. Adopting strategy (a), hexatag priors are injected directly into the input embedding, which combines a learned absolute position vector $\mathbf{p}_p$ with the token, segment, and tag terms:

$$\mathbf{x}_p = \mathbf{e}_p^{\text{tok}} + \mathbf{p}_p + \mathbf{e}_p^{\text{seg}} + m_p \cdot \left(\mathbf{E}_{\tau_p}^T + \mathbf{E}_{\nu_p}^N\right). \quad (2)$$

The tag term sits at the same level as $\mathbf{p}_p$, so the model learns to attend to a richer composite input without architectural conflict. **Moreover, we find that syntax should be infused from layer 1**, see appendix C[2].

3.3 Transformer-XL: Syntactic Priors in an Autoregressive Decoder with Relative PE

To test prior injection on autoregressive decoders, we use Transformer-XL (Dai et al., 2019) (TXL), which carries a sinusoidal relative-position encoding directly into the attention score and is a common syntactic-LM baseline (Sartran et al., 2022; Zhao et al., 2024). Following prior work, we disable cross-segment caching (so relative positions apply only within the current window) and replace the adaptive softmax with a tied linear projection, making TXL a purely causal decoder. For a query at position i and key at position j in head n of layer ℓ, the attention score splits into a content term and a position term:

$$A_{i,j,n}^{(\ell)} = \underbrace{\langle \mathbf{q}_{i,n}^{(\ell)} + \mathbf{u}_n,\ \mathbf{k}_{j,n}^{(\ell)} \rangle}_{\mathbf{AC}_{i,j,n}^{(\ell)}\ \text{(content)}} + \underbrace{\langle \mathbf{q}_{i,n}^{(\ell)} + \mathbf{v}_n,\ \mathbf{W}_R\, \mathbf{r}_{i-j,n} \rangle}_{\mathbf{BD}_{i,j,n}^{(\ell)}\ \text{(position)}} \quad (3)$$

where $\mathbf{r}_{i-j,n}$ is the sinusoidal vector for offset $i-j$, $\mathbf{W}_R$ projects it into head n's d_{head}-dimensional subspace, and $\mathbf{u}_n, \mathbf{v}_n$ are the learned content and position bias vectors. Because $\mathbf{BD}$ depends only on $i-j$, it carries no information about *which* tokens occupy positions i and j or their syntactic role. We supply that missing signal from the hexatag embeddings $\mathbf{E}_{\tau_j}^T + \mathbf{E}_{\nu_j}^N$ and study where in the score it should enter, across five injection sites treated in turn below.

3.3.1 Input-side injection (ADD)

The simplest approach adds the tag embeddings to the token embedding, so the prior enters the input residual at layer 0 and reaches the score only indirectly, through the content projections $\mathbf{W}_Q, \mathbf{W}_K$ that build the query and key:

$$\mathbf{h}_p^{(0)} = \mathbf{e}_p^{\text{tok}} + m_p\,(\mathbf{E}_{\tau_p}^T + \mathbf{E}_{\nu_p}^N), \quad (4)$$

with $m_p \in \{0, 1\}$ masking tag-free positions. The relative-position encoding itself is untouched; the prior entangles with lexical content rather than with position.

3.3.2 Position-pathway injection (PP-SiPE)

The approach which we find works best is to leave the input residual clean and route the tag straight into the position term. We project the tag at key j into the subspace of head n's of multi-head attention with a layer-specific $W_E^{(\ell)}$, then take its inner product with the *same* effective query $\mathbf{q}_{i,n}^{(\ell)} + \mathbf{v}_n$ that already drives $\mathbf{BD}$:

$$c_{i,j,n}^{(\ell)} = \frac{1}{\sqrt{d_{\text{head}}}} \left\langle \mathbf{q}_{i,n}^{(\ell)} + \mathbf{v}_n,\ \left[W_E^{(\ell)}(\mathbf{E}_{\tau_j}^T + \mathbf{E}_{\nu_j}^N)\right]_n \right\rangle \quad (5)$$

where $[\cdot]_n$ selects head n's d_{head}-dimensional block and where where $\mathbf{E}_{\tau_j}^T$ and $\mathbf{E}_{\nu_j}^N$ are the terminal and nonterminal hexatag embeddings at key j. Eqs. 3 and 5 are deliberately parallel: the *one* effective query $\mathbf{q}_{i,n}^{(\ell)} + \mathbf{v}_n$ poses two questions — "how well do I align with this *offset*?" ($\mathbf{BD}$) and "how well do I align with this *tag*?" (c). The second is therefore a syntactic counterpart of the first, on the same per-head scale. We couple it to the position term multiplicatively, leaving the content term $\mathbf{AC}$ untouched:

$$\boxed{\widetilde{A}_{i,j,n}^{(\ell)} = \mathbf{AC}_{i,j,n}^{(\ell)} + c_{i,j,n}^{(\ell)} \cdot \mathbf{BD}_{i,j,n}^{(\ell)}} \quad (6)$$

In our syntactic evaluation, we observe the multiplicative form (Eq. 6) to outperform all other approaches (refer to experiments in table 5 and ablations in Table 7, appendix A.2): it *gates by offset alignment*, applying the syntactic adjustment in proportion to how strongly the query already attends to that offset. Where the query-offset alignment is strong ($|\mathbf{BD}|$ large), the tag exerts a large pull; where it is near zero, the tag exerts almost none, so syntax modulates an existing positional preference rather than acting in isolation.[3] The offset embedding $\mathbf{r}_{i-j,n}$ itself is never modified.

[2] However, the input embedding is not the only place in the transformer where we can inject these priors; see appendix E and figure 6

[3] In practice we use $\widetilde{A} = \mathbf{AC} + (1 + c)\,\mathbf{BD}$; this gives $c \approx 0$ at initialization, so $\widetilde{A} \approx A$ and training starts from the vanilla Transformer-XL rather than a zeroed position term.

3.3.3 Fully-disentangled injection

A third option completely disentangles the tag from *both* pathways. Rather than folding c into $\mathbf{BD}$, we add it as an independent third component of the logit, alongside the content and position terms and on the standard attention scale:

$$\widetilde{A}^{(\ell)}_{i,j,n} = \mathbf{AC}^{(\ell)}_{i,j,n} + \mathbf{BD}^{(\ell)}_{i,j,n} + c^{(\ell)}_{i,j,n}. \quad (7)$$

Now $c^{(\ell)}_{i,j,n}$ depends on the query and the tag but not on $\mathbf{BD}^{(\ell)}_{i,j,n}$, so its size is fixed once the tag is chosen: a query at i adds the same syntactic bias toward key j whether or not it already attends to the offset $i{-}j$. This is the opposite of the multiplicative coupling (Eq. 6), where that same bias is scaled by $\mathbf{BD}$ and vanishes when the query ignores the offset. In our experiments, we find that the ungated form is weaker (refer to Table 5 and ablations in Table 8, appendix A.3) than the multiplicatively entangled bias, which indicates that the bias from SiPE helps most when it augments an attention preference the position pathway has already established, rather than when it acts on its own.

3.3.4 Injecting Syntactic Bias In Self-Attention

Motivated by nearly all prior work which imposes syntactic constraints on attention weights as either a hard or soft bias, (Murty et al., 2023; Zhao et al., 2024; Omote et al., 2019; Xie et al., 2021a; Yoshida et al., 2024), we propose our hexatag-derived bias applied to attention logits and the attention output, leaving the input residual, $\mathbf{AC}$, and the positional pathway $\mathbf{BD}$ untouched ($\mathbf{h}^{(0)}_p = \mathbf{e}^{\text{tok}}_p$).

The mechanism follows Shaw et al. (Shaw et al., 2018), who augment attention with learned key-side and value-side vectors. We project the summed hexatag embedding at key j two ways: through $\mathbf{W}^{\text{tag},(\ell)}_K$ to form a key bias that is added to the attention score $\widetilde{A}^{(\ell)}_{i,j,n}$, and through $\mathbf{W}^{\text{tag},(\ell)}_V$ to form a value bias that is added to the attended output $\mathbf{z}^{(\ell)}_{i,n}$:

$$\widetilde{A}^{(\ell)}_{i,j,n} = A^{(\ell)}_{i,j,n} + \tfrac{1}{\sqrt{d_{\text{head}}}}\langle \mathbf{q}^{(\ell)}_{i,n}, [\mathbf{W}^{\text{tag},(\ell)}_K(\mathbf{E}^T_{\tau_j} + \mathbf{E}^N_{\nu_j})]_n\rangle \quad (8)$$

$$\mathbf{z}^{(\ell)}_{i,n} = \sum_j \alpha^{(\ell)}_{i,j,n}\Big(\mathbf{v}^{(\ell)}_{j,n} + [\mathbf{W}^{\text{tag},(\ell)}_V(\mathbf{E}^T_{\tau_j} + \mathbf{E}^N_{\nu_j})]_n\Big) \quad (9)$$

where $\alpha^{(\ell)}_{i,j,n}$ are the softmax weights, and $[\,\cdot\,]_n$ selects head n's block. Unlike the score-side gate of Eq. 5, the key bias uses the bare query $\mathbf{q}^{(\ell)}_{i,n}$, and the bias is keyed on the tag at position j alone rather than on a relative pair (i, j), which keeps the added cost linear in sequence length. The projections may be learned per layer (the (ℓ) superscript) or tied to a single pair shared across all layers; the per-layer form is stronger, outperforming the shared form both on its own and when combined with the input-side injection of Eq. 4 (Appendix A.4).

3.3.5 Injecting syntax into position and attention simultaneously

The final variant pairs our two strongest single-site injections: SiPE on the position side (the multiplicative coupling, Eq. 6) and the per-layer attention-side biases. Routing the prior through both at once does not help: the pairing scores below each one applied alone (Table 5, Appendix A.4). Injecting the same hexatag signal through two pathways is thus redundant rather than complementary; the prior is best supplied once, through the position pathway.

3.4 DeBERTa-v3: injecting the prior under disentangled relative attention

DeBERTa-v3 (He et al., 2023) carries no input-level position embedding; positional information enters only at the attention score, through *disentangled relative attention*. For a query at i and key at j, the score splits into a content term and two relative-position terms:

$$S_{ij} = \underbrace{Q^c_i K^{c\top}_j}_{S^{\text{con}}_{ij}} + \underbrace{Q^c_i K^{r\top}_{\delta(i,j)} + K^c_j Q^{r\top}_{\delta(i,j)}}_{S^{\text{pos}}_{ij}}, \quad (10)$$

where $\delta(i, j)$ is the bucketed relative offset and Q^c, K^c are the content query and key. As with the relative-PE models above, we consider two injection sites: the input residual, and the position term S^{pos}.

3.4.1 Input-side injection

Because DeBERTa has no absolute position embedding, the tag prior becomes the only positionally-localized signal at the input layer:

$$\mathbf{h}^{(0)}_p = \mathbf{e}^{\text{tok}}_p + \mathbf{e}^{\text{seg}}_p + m_p \cdot \big(\mathbf{E}^T_{\tau_p} + \mathbf{E}^N_{\nu_p}\big). \quad (11)$$

The prior propagates through $\mathbf{W}_Q$ and $\mathbf{W}_K$ into the content-bearing terms of the score, while the relative-position machinery (its bucketing, lookup table, and projection matrices) is left untouched.

Tag priors and relative position thus interact *only through the content stream*, which is what makes the injection portable across positional-encoding families. The full algebra, including how the prior reaches all three score terms, is given in Appendix A.5.

3.4.2 Position-pathway injection (PP-SiPE)

Mirroring our Transformer-XL injection, we instead leave the residual clean and route the prior into the position term S^{pos}_{ij}, leaving the content term S^{con}_{ij} untouched. A per-layer projection $W_E^{(\ell)}$ maps the summed hexatag embedding at key j into head space, and we form a tag–query alignment against the content query:

$$c_{ij} = \frac{1}{\sqrt{d_h}} \langle Q_i^c,\, W_E^{(\ell)}(\mathbf{E}^T_{\tau_j} + \mathbf{E}^N_{\nu_j}) \rangle, \quad (12)$$

$$\tilde{S}_{ij} = S^{\text{con}}_{ij} + c_{ij}\, S^{\text{pos}}_{ij}. \quad (13)$$

The gate scales the relative-position score in proportion to how strongly the query aligns with the key's syntactic tag, so syntax sharpens an existing positional preference rather than acting on its own, exactly as in the multiplicative Transformer-XL variant (Eq. 6).[4]

3.5 ModernBERT: injecting the prior into rotary position encoding

ModernBERT (Warner et al., 2024) like many other Large language models (Dubey et al., 2024; DeepSeek-AI, 2024; Yang et al., 2025; Kamath et al., 2025) delivers positional information through rotary position embeddings (RoPE) (Su et al., 2023), which rotate the content query and key by position-dependent angles *after* the content projections, so that relative-position dependence emerges from the geometry of two absolute rotations. Each head dimension is split into frequency pairs indexed by t; pair t is rotated by $\theta_{p,t} = p\,\omega_t$ at absolute position p, with frequency ω_t and a 2×2 rotation $R(\cdot)$. Because RoPE carries no input-level position term, we again consider two injection sites: the input residual, and the rotation angle itself.

3.5.1 Input-side injection

Since RoPE has no input-level position embedding, the tag prior is added at the input residual exactly as in DeBERTa (Eq. 11, without the segment term, which ModernBERT does not use), and propagates into $\mathbf{q}_p, \mathbf{k}_p$ before rotation; the rotary mechanism itself is untouched.

3.5.2 Position-pathway injection (PP-SiPE)

Mirroring the Transformer-XL and DeBERTa injections, we can also route the prior into the position pathway instead of the input embedding, which for RoPE is the rotation angle. Using the summed hexatag embedding at position p, a per-layer projection $W_\delta^{(\ell)}$ produces a per-frequency phase offset

$$\Delta\theta_{p,t} = \big(W_\delta^{(\ell)}(\mathbf{E}^T_{\tau_p} + \mathbf{E}^N_{\nu_p})\big)_t, \quad (14)$$

which is added to the rotary angle *before* rotation, for both the query at position m and the key at position j:

$$\tilde{q}_{m,t} = R\big(m\,\omega_t + \Delta\theta_{m,t}\big)\, q_{m,t}, \quad (15)$$

$$\tilde{k}_{j,t} = R\big(j\,\omega_t + \Delta\theta_{j,t}\big)\, k_{j,t}. \quad (16)$$

The effective angle between query m and key j at frequency t is then $(m-j)\,\omega_t + (\Delta\theta_{m,t} - \Delta\theta_{j,t})$, so the tag enters as a syntactic phase shift on top of the positional one: the rotation that already encodes relative distance is nudged by how the two positions' hexatags relate. Because the perturbation enters as a rotation it is norm-preserving, leaving the query and key magnitudes unchanged (for more details refer to appendix A.6).

Model	Acc. (%)	Δ (%)
Our models on WikiText		
RoBERTa-base	70.68	–
RoBERTa + SiPE (Input pathway)	71.68	↑ +1.41
DeBERTa-base	67.40	–
DeBERTa + SiPE (Input pathway)	68.93	↑ **+2.27**
ModernBERT-base	49.94	–
ModernBERT + SiPE (Input pathway)	51.10	↑ +2.32
Transformer-XL	53.56	–
Transformer-XL + SiPE (Positional Pathway)	53.44	↑ -0.22
OOD Eval: WikiText → BLLIP-LG		
RoBERTa-base	71.29	–
RoBERTa + SiPE (Input pathway)	72.62	↑ +1.87
DeBERTa-base	69.01	–
DeBERTa-base + SiPE (Input pathway)	71.12	↑ +3.06
ModernBERT-base	51.25	–
ModernBERT-base + SiPE (Input pathway)	53.41	↑ +4.21

Table 1: BLiMP accuracy results comparing encoders trained with SiPE (input pathway) with prior approaches. For encoders, we use PLL-based accuracy. Δ: relative change w.r.t. the corresponding base encoder model. OOD section: models pre-trained on WikiText, then continuously pre-trained on BLLIP-LG (Charniak et al., 2000). ↑ improvement, ↓ degradation. **Bold**: best relative improvement with our method.

4 Experimental Results

4.1 Experimental Settings and Datasets

Prior work injecting syntax into Transformers focused mainly on *intrinsic* evaluation—directly test-

[4] In practice we use $\tilde{S}^{\text{pos}}_{ij} = (1 + c_{ij})\, S^{\text{pos}}_{ij}$, with $W_E^{(\ell)}$ initialized small so that $c_{ij} \approx 0$ at initialization and $\tilde{S}_{ij} \approx S_{ij}$; training therefore starts from the unmodified DeBERTa score.

Task	RoBERTa (base)	+SiPE (input pathway)	DeBERTa (base)	+SiPE (input pathway)	+SiPE (positional pathway)	ModernBERT (base)	+SiPE (input pathway)	+SiPE (positional pathway)
CoLA	31.41(±2.02)	**31.77**(±0.71)	**27.22**(±2.37)	26.09(±2.14)	24.97(±2.02)	**25.00**(±0.83)	24.45(±1.85)	23.26(±2.83)
SST-2	**87.23**(±0.79)	87.22(±0.52)	87.31(±0.30)	**87.88**(±0.54)	87.50(±0.19)	85.09(±1.01)	**85.86**(±0.79)	84.25(±1.38)
QQP	83.24(±0.21)	**83.36**(±0.11)	84.02(±0.31)	**84.95**(±0.15)	84.88(±0.13)	**83.53**(±0.13)	83.09(±0.45)	82.45(±0.94)
QNLI	77.65(±0.50)	**78.24**(±0.27)	68.07(±0.08)	**68.88**(±0.24)	67.94(±0.29)	62.51(±1.01)	61.83(±0.70)	**65.79**(±0.39)
MNLI	73.03(±0.33)	**74.05**(±0.17)	71.25(±0.14)	**71.85**(±0.14)	71.74(±0.30)	71.29(±0.03)	**72.24**(±0.35)	71.15(±0.11)
RTE	**65.76**(±0.74)	64.66(±0.56)	62.94(±1.23)	63.06(±1.73)	**64.14**(±1.68)	62.21(±0.45)	**63.06**(±2.13)	63.90(±1.02)
STSB	79.11(±0.30)	**81.41**(±0.13)	76.20(±0.56)	**76.98**(±0.24)	76.33(±1.32)	79.67(±0.30)	**81.21**(±0.78)	79.49(±0.06)
MRPC	80.96(±0.44)	**82.02**(±0.85)	80.49(±0.84)	81.11(±0.76)	**83.45**(±0.71)	79.74(±1.43)	**82.82**(±1.08)	81.56(±1.07)
Macro	72.30(±0.33)	**72.84**(±0.42)	69.69(±0.25)	70.10(±0.39)	**70.12**(±0.22)	68.63(±0.08)	**69.32**(±0.11)	69.02(±0.37)

Table 2: GLUE evaluation of baseline encoders vs. SiPE augmentation, over 3 seeds (std in parentheses). Baselines span three PE schemes: RoBERTa-base (absolute), DeBERTa-v3-base (relative), ModernBERT-base (RoPE). SiPE is injected via two routes: input pathway (into token embeddings) and positional pathway (into positional information; relative/RoPE only). Best variant per task in **bold**. SiPE improves every architecture on the macro average—RoBERTa $+0.75\%$, DeBERTa-v3 $+0.59\%$, ModernBERT $+1.0\%$ relative.

Task	RoBERTa (large)	+SiPE (input pathway)	ModernBERT (large)	+SiPE (input pathway)
CoLA	**30.71**(±0.51)	30.36(±0.88)	**25.06**(±1.21)	24.58(±0.87)
SST-2	**88.95**(±0.35)	87.50(±0.09)	84.44(±0.52)	**85.47**(±0.36)
QQP	84.31(±0.18)	**85.10**(±0.24)	82.54(±0.39)	**82.55**(±0.55)
QNLI	80.18(±0.18)	**82.68**(±0.33)	61.99(±0.52)	**62.11**(±0.41)
MNLI	**74.72**(±0.10)	74.43(±0.31)	72.03(±0.53)	**72.36**(±0.22)
RTE	**65.94**(±0.45)	64.62(±1.06)	**63.54**(±0.29)	61.49(±2.27)
STSB	81.89(±0.59)	**82.34**(±0.38)	79.03(±0.41)	**80.05**(±0.35)
MRPC	80.92(±1.00)	**83.70**(±1.15)	81.86(±1.24)	**82.00**(±0.76)
Macro	73.45(±0.16)	**73.84**(±0.13)	68.81(±0.13)	**68.83**(±0.35)

We did not pre-train DeBERTa-large with SiPE (input pathway/positional pathway), nor ModernBERT-large with SiPE (positional pathway), due to limited academic compute and budget constraints.

Table 3: GLUE evaluation for large-scale encoder models.

Task	Transformer-XL (base)	+SiPE (positional pathway)
CoLA	13.80(±0.26)	**26.26**(±0.22)
SST-2	84.14(±0.11)	**88.80**(±0.44)
QQP	83.83(±0.12)	**85.64**(±0.21)
QNLI	76.00(±0.26)	**82.28**(±0.29)
MNLI	68.07(±0.32)	**75.22**(±0.14)
RTE	63.42(±1.70)	**67.51**(±0.29)
STSB	76.90(±0.17)	**81.80**(±0.12)
MRPC	79.20(±0.21)	**82.75**(±0.26)
Macro	68.17(±0.17)	**73.78**(±0.05)

Table 4: GLUE results for the decoder-only Transformer-XL (TXL) baseline and its SiPE (positional pathway) variant (blue tint); best per task in bold. SiPE improves every task and gains +8.2% relative on the macro average (68.17→73.78), showing the positional pathway best augments TXL's relative positional encoding.

ing syntactic understanding (Sartran et al., 2022; Xie et al., 2021a; Omote et al., 2019; Xie et al., 2021b; Yoshida and Oseki, 2022; Zhao et al., 2024; Murty et al., 2023). We argue these gains should also be assessed *extrinsically*, on downstream tasks where syntax is indirectly useful, and organize our experiments accordingly.

Intrinsic evaluation: We test syntactic understanding on BLiMP (Warstadt et al., 2023) and SyntaxGym (Hu et al., 2020), both targeted minimal-pair benchmarks contrasting grammaticality (Table 6). BLiMP requires assigning higher probability to the grammatical sentence overall, whereas SyntaxGym compares probabilities only in the critical region where the ungrammaticality occurs.

Following prior work (Zhao et al., 2024; Sartran et al., 2022), we pretrain Transformer-XL on BLLIP-LG (Charniak et al., 2000) with their hyperparameters, using the training splits of Hu et al. (2020).[5] We report BLiMP accuracy via sentence log-likelihood, SyntaxGym via the standard suite-level voting protocol, and BLLIP-LG test perplexity to confirm language-modeling ability is preserved.

In a second set of experiments, we pretrain three encoders: RoBERTa, DeBERTa, ModernBERT (Liu et al., 2019; He et al., 2023; Warner et al., 2024) and Transformer-XL for 1M steps on a 500M-token, offline-hexatagged slice of WikiText-103 (Merity et al., 2016).[6] We evaluate these models on a hexatagged BLiMP (Warstadt et al., 2023): using encoders, for each minimal pair we prefer the sentence with higher pseudo-log-likelihood (PLL) (Salazar et al., 2020) under the masked-LM objective, scoring accuracy against BLiMP's gold labels.

[5]We do *not* use the dataset's syntactic annotations; we hexatag the sentences with an in-house parser (URL hidden for blind review).

[6]Our goal is to study syntactic embeddings at scale on real downstream tasks, so we pretrain for GLUE fine-tuning and reuse the same checkpoints for syntactic evaluation; a separate BLLIP-LG pretraining run with its own hyperparameter tuning was beyond our academic budget.

Model	BLiMP ↑	Δ BLiMP	PPL ↓	Δ PPL	Syntax Gym ↑	Δ SyntaxGym
No Inductive Bias						
TXL (tokens)	75.30	–	18.63	–	73.09	–
GPT-2 (tokens) (Radford et al., 2019)	72.20	↓ -4.12	21.60	↓ -15.94	71.90	↓ -1.63
Constituency Parsing						
PLM (TXL) (Qian et al., 2021)	75.10	↓ -0.27	29.80	↓ -59.96	80.20	↑ +9.73
Transformer Grammar (TXL) (Sartran et al., 2022)	73.50	↓ -2.39	18.40	↑ +1.23	**82.50**	↑ **+12.87**
Pushdown LM (GPT-2) (Murty et al., 2023)	**75.60**	↑ **+0.40**	19.90	↓ -6.82	82.30	↑ +12.60
TreeReg (GPT-2) (Nandi et al., 2025)	74.80	↓ -0.66	22.30	↓ -19.70	80.00	↑ +9.45
Tree-Planted Transformers (TPT) (Yoshida et al., 2024)						
TPT [zero]	—	—	47.50	↓ -154.97	71.70	↓ -1.90
TPT [rand.]	—	—	47.40	↓ -154.43	69.00	↓ -5.60
TPT [seq.]	—	—	47.30	↓ -153.89	70.10	↓ -4.09
TPT [dep.]	—	—	47.70	↓ -156.04	77.10	↑ +5.49
TPT [cons.]	—	—	45.50	↓ -144.23	75.80	↑ +3.71
TPT [bin.]	—	—	45.60	↓ -144.77	73.00	↓ -0.12
Syntax-Informed Embeddings with Hexatagging (Ours)						
SiPE (positional pathway)						
Input-side Injection	73.82	↓ -1.97	16.16	↑ +13.26	76.97	↑ +5.31
Position-pathway Injection	74.01	↓ -1.71	16.95	↑ +9.02	80.60	↑ +10.29
Fully-disentangled Injection	74.72	↓ -0.77	16.66	↑ +10.57	78.72	↑ +7.70
Attention Bias						
Projections shared across layers	73.60	↓ -2.26	16.73	↑ +10.20	76.35	↑ +4.46
Projections Learned Per-Layer	74.66	↓ -0.85	16.24	↑ +12.83	78.10	↑ +6.86
SiPE + Attention Bias						
Input-side Injection	74.16	↓ -1.51	**15.89**	↑ **+14.71**	78.47	↑ +7.36
Position-pathway Injection	74.44	↓ -1.14	16.52	↑ **+11.32**	77.97	↑ +6.68

Table 5: Language modeling and syntax evaluation. BLiMP and SyntaxGym are higher-is-better, while PPL (perplexity, computed on BLLIP-LG test split (Charniak et al., 2000)) is lower-is-better. Delta columns report relative improvement or degradation against TXL (tokens). Best scores for each dataset are **bolded**.

Extrinsic evaluation: To assess the downstream impact of injecting syntactic information directly in the transformer architecture, we carry out experiments on the GLUE benchmark (Wang et al., 2019), which contains a suite of real-world NLP tasks. We finetune our three encoder models as well as Transformer-XL (which were all previously pre-trained with SiPE on Wikitext) on each task using standard GLUE hyperparameters for each model (refer to Tables 2, 3 and 4 for results) and table 11 in appendix D.

4.2 Key Takeaways

Positional-pathway injection is the strongest recipe for a relative-PE decoder. On Transformer-XL, injecting the SiPE prior into the positional pathway gives our best decoder result—SyntaxGym 80.60 ($+10.29$ over the token baseline) with perplexity cut from 18.63 to 16.95 (Table 5). This carries through to GLUE, improving the TXL macro average from 68.17 to 73.78, a dramatic $+8.2\%$ relative gain (Table 4). Using only a single parse tree at inference, SiPE also surpasses parser-free approaches, TreeReg (Nandi et al., 2025) and all Tree-Planted Transformer variants (Yoshida et al., 2024), on both SyntaxGym and BLLIP-LG perplexity, roughly halving perplexity against the strongest such baselines (e.g., 16.95 vs. 22.30 for TreeReg) while matching or exceeding their syntactic generalization.

BLIMP Example
✓ The keys to the cabinet **are** on the table.
✗ The keys to the cabinet **is** on the table.
SyntaxGym Example
✓ The farmer near the clerks **knows** many people.
✗ The farmer near the clerks **know** many people.

Table 6: Examples from the BLIMP dataset (top) and SyntaxGym (bottom). In both datasets, $p(✓) > p(✗)$, but in BLIMP this probability is computed for the complete sentence, whereas in SyntaxGym it is computed only over the region of interest.

For encoders, the simpler input-embedding injection is better, with gains that strengthen out of distribution. Across all three encoder families—absolute (RoBERTa), disentangled-relative (DeBERTa-v3), and rotary (ModernBERT), input-pathway SiPE consistently beats the base model on GLUE (Table 2); the positional-pathway route also helps for the relative and rotary schemes but yields slightly smaller gains, so for encoders mixing the prior directly with input embeddings is sufficient and most reliable. All three encoders likewise improve on BLiMP—$+1.41\%$ (RoBERTa), $+2.27\%$ (DeBERTa-v3), $+2.32\%$ (ModernBERT) (Table 1)—and these gains grow under continued pre-training on BLLIP-LG after WikiText, rising to $+1.87\%$, $+3.06\%$, and $+4.21\%$ respectively, indicating the prior generalizes beyond its original distribution rather than overfitting. [7]

5 Conclusion

We proposed a simple strategy to infuse syntactic information in any transformer architecture. Our proposed modification is lightweight, adding only $O(1,000)$ parameters per model via an additive prior embedding, yet it yields reliable gains in both intrinsic and extrinsic evaluations and multiple encoder and decoder transformer variants.

[7] The small BLiMP drops are expected: BLiMP scores the *full* sentence, so language-modeling ability dominates over the ungrammatical span alone (Zhao et al., 2024). Most baselines in Table 5 (PLM, Transformer Grammars, TreeReg, most TPT variants) likewise fall below the token baseline on BLiMP.

Limitations

Our approach assumes access to hexatag annotations at inference time: each input sequence must be tagged by a dependency parser before being passed to the syntax-infused encoder. While this adds a preprocessing step, the tagger is lightweight and fast in practice (we use `DeBERTa-v3-base` as the hexatagger, 184M parameters) and runs efficiently relative to encoder pre-training and downstream finetuning. The approach also inherits the hexatagger's errors, so its benefit may be smaller in domains where the parser is less accurate.

We consider only coarse-grained syntactic priors derived from directional terminal and non-terminal tags. Preliminary pre-training experiments incorporating full dependency-relation labels (DEPREL) did not yield meaningful gains on GLUE (Appendix E, Figures 4 and 5), but we did not investigate this further.

Our study of where to inject the prior is broad but not exhaustive. We study the main injection sites and coupling choices for each positional scheme, but under a fixed academic compute budget we could not pre-train every conceivable combination, particularly for the relative and rotary attention-side variants, where the design space is large. The configurations we report are therefore the strongest we found rather than a guaranteed global optimum, and our per-dataset results with RoBERTa (Figure 5) show that no single setting dominates across all tasks.

Finally, our experiments are limited to small models, English text, and the pre-training budgets used here. Whether the gains hold at larger scale, in other languages, or under substantially longer pre-training, especially with billion parameter LLMs remains to be seen; its unclear how the relative benefit of an explicit syntactic prior will scale as model and data scale increase.

References

Afra Amini, Tianyu Liu, and Ryan Cotterell. 2023. Hexatagging: Projective dependency parsing as tagging. *Preprint*, arXiv:2306.05477.

Collin F Baker, Charles J Fillmore, and John B Lowe. 1998. The berkeley framenet project. In *COLING 1998 Volume 1: The 17th International Conference on Computational Linguistics*.

Eugene Charniak, Don Blaheta, Niyu Ge, Keith Hall, John Hale, and Mark Johnson. 2000. BLLIP 1987-89 WSJ corpus release 1. LDC2000T43.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. 2019. Transformer-xl: Attentive language models beyond a fixed-length context. *Preprint*, arXiv:1901.02860.

Marie-Catherine De Marneffe, Christopher D Manning, Joakim Nivre, and Daniel Zeman. 2021. Universal dependencies. *Computational linguistics*, 47(2):255–308.

DeepSeek-AI. 2024. Deepseek-v3 technical report. *Preprint*, arXiv:2412.19437.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. *Preprint*, arXiv:1810.04805.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony S. Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, and 510 others. 2024. The llama 3 herd of models.

Yinuo Guo, Zeqi Lin, Jian-Guang Lou, and Dongmei Zhang. 2020. Hierarchical poset decoding for compositional generalization in language. *Advances in Neural Information Processing Systems*, 33:6913–6924.

Pengcheng He, Jianfeng Gao, and Weizhu Chen. 2023. Debertav3: Improving deberta using electra-style pre-training with gradient-disentangled embedding sharing. *Preprint*, arXiv:2111.09543.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2021. Deberta: Decoding-enhanced bert with disentangled attention. *Preprint*, arXiv:2006.03654.

Jennifer Hu, Jon Gauthier, Peng Qian, Ethan Wilcox, and Roger P. Levy. 2020. A systematic assessment of syntactic generalization in neural language models. *Preprint*, arXiv:2005.03692.

Michael Y. Hu, Jackson Petty, Chuan Shi, William Merrill, and Tal Linzen. 2025. Between circuits and chomsky: Pre-pretraining on formal languages imparts linguistic biases. *Preprint*, arXiv:2502.19249.

Gemma Team Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ram'e, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean-Bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, Gael Liu, and 191 others. 2025. Gemma 3 technical report. *ArXiv*, abs/2503.19786.

Paul R Kingsbury and Martha Palmer. 2002. From treebank to propbank. In *Proceedings of the Language Resources and Evaluation Conference (LREC)*, pages 1989–1993.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *Preprint*, arXiv:1907.11692.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*.

Shikhar Murty, Pratyusha Sharma, Jacob Andreas, and Christopher D. Manning. 2023. Pushdown layers: Encoding recursive structure in transformer language models. *Preprint*, arXiv:2310.19089.

Ananjan Nandi, Christopher D. Manning, and Shikhar Murty. 2025. Sneaking syntax into transformer language models with tree regularization. *Preprint*, arXiv:2411.18885.

Yutaro Omote, Akihiro Tamura, and Takashi Ninomiya. 2019. Dependency-based relative positional encoding for transformer nmt. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2019)*, pages 854–861, Varna, Bulgaria. INCOMA Ltd.

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.

Julian Salazar, Davis Liang, Toan Q. Nguyen, and Katrin Kirchhoff. 2020. Masked language model scoring. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics.

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

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-attention with relative position representations. *Preprint*, arXiv:1803.02155.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. 2023. Roformer: Enhanced transformer with rotary position embedding. *Preprint*, arXiv:2104.09864.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. Glue: A multi-task benchmark and analysis platform for natural language understanding. *Preprint*, arXiv:1804.07461.

Benjamin Warner, Antoine Chaffin, Benjamin Clavié, Orion Weller, Oskar Hallström, Said Taghadouini, Alexis Gallagher, Raja Biswas, Faisal Ladhak, Tom Aarsen, Nathan Cooper, Griffin Adams, Jeremy Howard, and Iacopo Poli. 2024. Smarter, better, faster, longer: A modern bidirectional encoder for fast, memory efficient, and long context finetuning and inference. *Preprint*, arXiv:2412.13663.

Alex Warstadt, Alicia Parrish, Haokun Liu, Anhad Mohananey, Wei Peng, Sheng-Fu Wang, and Samuel R. Bowman. 2023. Blimp: The benchmark of linguistic minimal pairs for english. *Preprint*, arXiv:1912.00582.

Yikuan Xie, Wenyong Wang, Mingqian Du, and Qing He. 2021a. Transformer with syntactic position encoding for machine translation. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2021)*, pages 1536–1544, Held Online. INCOMA Ltd.

Yikuan Xie, Wenyong Wang, Mingqian Du, and Qing He. 2021b. Transformer with syntactic position encoding for machine translation. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2021)*, pages 1536–1544, Held Online. INCOMA Ltd.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Ryo Yoshida and Yohei Oseki. 2022. Composition, attention, or both? In *Findings of the Association for Computational Linguistics: EMNLP 2022*, page 5822–5834. Association for Computational Linguistics.

Ryo Yoshida, Taiga Someya, and Yohei Oseki. 2024. Tree-planted transformers: Unidirectional transformer language models with implicit syntactic supervision. In *Findings of the Association for Computational Linguistics ACL 2024*, page 5120–5134. Association for Computational Linguistics.

Yida Zhao, Chao Lou, and Kewei Tu. 2024. Dependency transformer grammars: Integrating dependency structures into transformer language models. *Preprint*, arXiv:2407.17406.

A Preliminaries

A.1 Subword-level tag injection

Hexatag annotations are produced at the word level: every word w_i in a sentence carries a terminal tag

t_i and a nonterminal tag n_i (drawn from small vocabularies $\mathcal{T}$ and $\mathcal{N}$, respectively, where $|T| = 2$ and $|N| = 5$). The language model, however, operates on subword tokens: the tokenizer may split w_i into one or more subwords $s_{i,1}, \ldots, s_{i,k_i}$. We must therefore specify how a word-level hexatag is associated with its corresponding subword positions.

We adopt a *assign hexatag to first-subword only strategy*: the tag pair (t_i, n_i) is attached to the position of the first subword $s_{i,1}$, and the remaining subwords $s_{i,2}, \ldots, s_{i,k_i}$ receive no tag (in our implementation this is equivalent to receiving a tag ID of -100). Concretely, for each subword position p in the flattened input sequence, we define a tag-bearing mask:

$$m_p = \begin{cases} 1 & \text{if } p \text{ is the first subword} \\ & \text{of some word } w_i, \\ 0 & \text{otherwise.} \end{cases} \tag{17}$$

At subword positions with $m_p = 1$, the prior contributes a learned tag embedding to augment the subword embedding; at positions with $m_p = 0$ (continuation subwords), no prior is added, and the subword embedding is used as-is. Tag information still reaches continuation subwords, but only indirectly, through self-attention in subsequent layers.

This design has two practical benefits: (i) The total tag embedding signal injected per word is invariant to its subword length k_i, avoiding over-amplification for words that break into many subwords

(ii) it gives a clean one-to-one correspondence between words and tag-bearing subword positions, which is required by the auxiliary tag-prediction objective: this objective is a per-position classification head that predicts the (terminal, nonterminal) tag pair, and it only contributes loss at positions where $m_p = 1$.

The *assign hexatag to first-subword only* is also the simpler choice here. The natural alternative would be to assign the same hexatag to all k_i subwords of a word, which would require supervising all pieces, contributing k_i loss terms per word and thus over-weighting words that fragment into many subwords. We leave this variant to future work.

The indicator m_p thus controls both prior injection and supervision, giving each word exactly one tag signal at the input and one prediction at the output. The auxiliary objective takes one of two concrete forms in our experiments. In autoregressive models, the head at position p predicts the tags (τ_{p+1}, ν_{p+1}) of the *next* tag-bearing position, teacher-forced from gold left context. In masked language models, the head at position p predicts (τ_p, ν_p) at masked positions only.

The remainder of this section details the design choices and trade-offs for each injection strategy we study. These fall into a few families: adding the syntactic bias to the input embeddings; entangling it directly with the positional encoding; keeping it disentangled and adding it as a separate term to the attention score; injecting it as a separate attention-side bias; and combining any of these. We examine these strategies across four backbones that span the major positional-encoding schemes: RoBERTa (encoder, absolute positional embeddings), DeBERTa-v3 (encoder, relative positional embeddings), ModernBERT (encoder, rotary positional embeddings), and Transformer-XL (autoregressive decoder, relative positional embeddings).

A.2 SiPE position-pathway design choices for Transformer-XL

The multiplicative position-pathway injection of §3.3 has two design axes. **Layer sharing** controls whether a single projection W_E is reused across all layers or each layer learns its own $W_E^{(\ell)}$. **Projection target** controls where $W_E^{(\ell)}$ sends the tag before it meets the query: either into a single attention head's d_{head}-dimensional subspace (a small map that keeps the tag correction on the same per-head geometry as **BD**), or into the full model dimension d_{model} (a larger map whose output is reshaped into n_{head} per-head blocks, so every head reads from one shared tag representation). Table 7 reports all four combinations.

Sharing	Projection	BLiMP ↑	SG ↑
Shared	d_{head}	73.95	78.80
Per-layer	d_{head}	74.15	75.84
Shared	d_{model}	73.95	75.09
Per-layer	d_{model}	**74.01**	**80.60**

Table 7: Multiplicative position-pathway injection across the two design axes: layer sharing (shared vs. per-layer W_E) and projection target (d_{head} head subspace vs. d_{model} full dimension). Per-layer projection into the full model dimension gives the best average across BLiMP and SyntaxGym (SG); this is the configuration we adopt as SiPE (positional pathway).

For Transformer-XL, the best configuration we find is the per-layer projection into the full model dimension (74.01 BLiMP, 80.60 SG). Neither axis

dominates alone: the gain comes from their pairing. Per-layer projections matter because the tag correction is scaled by each layer's own **BD**, so a separate $W_E^{(\ell)}$ lets each layer specialize how syntax modulates its positional preference. The full-dimension target matters because it gives the tag a richer map than a single head's subspace allows. Notably, the better projection target flips with the coupling: the multiplicative coupling here prefers d_{model}, whereas the fully-disentangled coupling of Appendix A.3 prefers the d_{head} head subspace. The best target therefore depends on how the prior is coupled to the position pathway, not on the projection axis in isolation.

Stepping back to the coupling itself: multiplicative coupling scales the position term by the tag alignment (Eq. 6), while the disentangled form adds the alignment as a standalone logit term (§3.3.3, Eq. 7). The multiplicative variants are the strongest overall, but the ordering is informative: disentangled injection already beats plain input-side injection (Eq. 4), and multiplicative coupling beats both. Entangling the prior with position thus helps progressively more as the coupling becomes gated by the query's existing offset alignment, so that syntax sharpens a positional preference the model has already formed rather than adding an unconditional bias.

A.3 Fully-disentangled syntax injection design choices for Transformer-XL

The fully-disentangled variant of §3.3.3 adds the tag–query alignment c to the attention logit as a standalone term (Eq. 7), leaving both the content term **AC** and the position term **BD** exactly as in standard Transformer-XL. It is governed by the same two projection axes as the gated variant of Appendix A.2: $W_E^{(\ell)}$ may map the tag into the full model dimension (then reshaped into per-head blocks) or directly into a single head's d_{head}-dimensional subspace, and it may be shared across all layers or learned per layer. Since neither axis has an obvious right answer a priori, we ran all four combinations and report them in Table 8.

The best disentangled configuration is the per-layer projection into the head subspace, which keeps the tag correction on the same per-head geometry as **BD** while letting each layer learn its own tag-to-logit map. This setting is in fact stronger than input-side injection on both metrics (74.72 vs. 73.82 BLiMP, 78.72 vs. 76.97 SG; Table 8), so

Sharing	Projection	BLiMP ↑	SG ↑
Shared	d_{head}	74.17	77.72
Shared	d_{model}	75.03	76.10
Per-layer	d_{model}	73.51	77.20
Per-layer	d_{head}	**74.72**	**78.72**

Table 8: Fully-disentangled injection across the two projection axes: layer sharing (a single W_E shared across all layers vs. a per-layer $W_E^{(\ell)}$) and projection target (mapping the tag into a single head's d_{head}-dimensional subspace vs. the full model dimension d_{model}, reshaped into per-head blocks). Per-layer projection into the head subspace gives the best average across BLiMP and SyntaxGym (SG); this is the configuration reported in the main results.

adding the prior as a separate logit term is a genuine improvement over placing it in the input residual. What it does not match is the multiplicative coupling of §3.3.2, which remains the strongest instantiation of our method for Transformer-XL. This ordering — input-side < disentangled < multiplicative — shows the prior helps most when *gated* by the query's offset alignment. As a flat logit term, the disentangled bias applies equally at every position: better than diffusing the prior through the residual, but weaker than scaling it by **BD**, where syntax can sharpen a positional preference the model has already formed.

A.4 Attention-side and combined injection design choices for Transformer-XL

Relation to the Shaw formulation. The key and value biases of Eqs. 8–9 adapt Shaw et al. (Shaw et al., 2018), who add learned per-pair vectors $\mathbf{a}_{ij}^K$ and $\mathbf{a}_{ij}^V$ to the key-side logit and the value-side aggregation, $e_{ij} \mathrel{+}= \mathbf{q}_i^\top \mathbf{a}_{ij}^K$ and $\mathbf{z}_i \mathrel{+}= \sum_j \alpha_{ij} \mathbf{a}_{ij}^V$. We change two things. First, our bias is indexed by the key position j alone rather than by the relative pair (i, j), so the tag at j contributes a single per-key vector that every query sees; this is what keeps the added cost linear in sequence length rather than quadratic. Second, the bias is conditioned on hexatag identity rather than relative offset, and is learned jointly with the language model rather than read from a separately trained parser. The terminal and nonterminal tag embeddings are projected separately and summed, and a position-validity mask m_j zeroes the contribution wherever the tag label is `-100`. The key-side query is the bare projected query $\mathbf{q}_i^{(\ell)}$, not $\mathbf{q}_i^{(\ell)} + \mathbf{u}_n$ as in the content term, so the bias is not routed through the global content

bias of Dai et al. With mem_len $= 0$ the bias applies over the same key range as within-segment attention, so no padding for memory positions is needed.

This places our design between the dependency-distance bias of Omote et al. (Omote et al., 2019) and the root-to-node path embedding of Xie et al. (Xie et al., 2021b). Both put syntactic structure inside attention rather than in the residual stream, but our bias is keyed on hexatag identity rather than a tree-distance proxy, is a single per-key term rather than a pairwise one, and is applied in a causally masked unidirectional language model rather than a bidirectional encoder.

Sharing across layers. The one design axis is whether $\mathbf{W}_K^{\text{tag}}$ and $\mathbf{W}_V^{\text{tag}}$ are shared across all layers or learned per layer. In the shared form a single pair is reused at every layer; in the per-layer form each of the L layers owns its own pair, giving L independent key and value maps and letting each layer specialize its tag-to-attention routing, in the same way the content projections $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V$ already specialize per layer. The per-layer form is the stronger of the two on syntactic evaluations, both for the attention-side injection on its own and for its combination with the input-side injection of Eq. 4, so we use it wherever the attention-side biases appear.

A.5 DeBERTa-v3: disentangled relative positional embeddings and tag injection

DeBERTa-v3 (He et al., 2023) does not use absolute positional embeddings at the input level. Instead it delivers positional information through *disentangled relative attention*: each pair of positions (i, j) contributes three distinct terms to the attention score, only one of which depends on token content alone. This raises a natural question for our setting: where do tag priors enter when there is no input-level position embedding to sit alongside?

Background: disentangled attention. For a query at position i and a key at position j, the score decomposes into a content-to-content (c2c), a content-to-position (c2p), and a position-to-content (p2c) term,

$$A_{ij} = \underbrace{\mathbf{q}_i^\top \mathbf{k}_j}_{\text{c2c}} + \underbrace{\mathbf{q}_i^\top \mathbf{W}_{c2p}\, \mathbf{r}_{\delta(i,j)}}_{\text{c2p}} + \underbrace{\mathbf{k}_j^\top \mathbf{W}_{p2c}\, \mathbf{r}_{\delta(j,i)}}_{\text{p2c}}, \tag{18}$$

where $\mathbf{q}_i = \mathbf{W}_Q \mathbf{h}_i^{(0)}$ and $\mathbf{k}_j = \mathbf{W}_K \mathbf{h}_j^{(0)}$. The relative-position embeddings $\mathbf{r}_{\delta(\cdot,\cdot)}$ are pulled from a learned table indexed by bucketed signed offsets, and $\mathbf{W}_{c2p}, \mathbf{W}_{p2c}$ project content vectors into this relative-position subspace.[8]

Input-side injection: where priors enter, and where they do not. Under input-side injection (Eq. 11) each input vector carries the summed tag prior, so the queries and keys carry it into the score through the content stream:

$$\mathbf{q}_i = \mathbf{W}_Q\Big(\mathbf{e}_i^{\text{tok}} + \mathbf{e}_i^{\text{seg}} + m_i \cdot \big(\mathbf{E}^T_{\tau_i} + \mathbf{E}^N_{\nu_i}\big)\Big), \tag{19}$$

$$\mathbf{k}_j = \mathbf{W}_K\Big(\mathbf{e}_j^{\text{tok}} + \mathbf{e}_j^{\text{seg}} + m_j \cdot \big(\mathbf{E}^T_{\tau_j} + \mathbf{E}^N_{\nu_j}\big)\Big). \tag{20}$$

Reading Eqs. 18–20 together makes the design visible. The tag priors sit inside $\mathbf{h}_i^{(0)}$ and $\mathbf{h}_j^{(0)}$ and therefore propagate through $\mathbf{W}_Q, \mathbf{W}_K$ into all three score terms via the content projections. The relative-position table, by contrast, appears only in c2p and p2c, and is purely a function of the offset. The two streams enter through entirely different machinery and meet only at the additive combination in A_{ij}; the relative-position machinery itself — the bucketing, the $\mathbf{r}_\delta$ table, and the $\mathbf{W}_{c2p}, \mathbf{W}_{p2c}$ projections — is left entirely unmodified. **Tag priors interact with relative position through the content stream alone**, which is what makes the design portable across positional schemes. The tag tables $\mathbf{E}^T, \mathbf{E}^N$ are constructed as for RoBERTa: uniformly initialized, masked by the first-subword indicator m_p, and added inside the input layer norm.

Position-pathway injection. The position-pathway variant of §3.4.2 leaves the input clean and instead gates the two relative-position terms (the c2p and p2c terms, grouped as S^{pos} in Eq. 10) by the tag–query alignment c_{ij} of Eq. 12, leaving the content-to-content term untouched. The gate uses the content query Q_i^c and the per-layer projection $W_E^{(\ell)}$ of the summed hexatag embedding; the relative-position table and the $\mathbf{W}_{c2p}, \mathbf{W}_{p2c}$ projections are again left unmodified. This is the direct DeBERTa analogue of the multiplicative

[8] The buckets δ collapse pairs of positions with similar offsets into shared rows of the relative-position table, following (Shaw et al., 2018): small offsets $\{-3, -2, -1, 0, 1, 2, 3\}$ each receive their own bucket, while larger offsets are progressively merged — e.g. $\{+9, \ldots, +15\}$ might collapse into a single bucket — so the table stays small while preserving fine-grained discrimination near the diagonal.

Transformer-XL injection: in both cases the prior modulates the position pathway multiplicatively and the content pathway is untouched, with the only structural difference being that DeBERTa's position pathway carries two terms (c2p and p2c) rather than Transformer-XL's single **BD**.

A.6 ModernBERT: injecting the prior into rotary position encoding

ModernBERT (Warner et al., 2024) delivers positional information through rotary position embeddings (RoPE) (Su et al., 2023), which rotate the content query and key by position-dependent angles *after* the content projections, so that relative-position dependence emerges from the geometry of two absolute rotations. Each head dimension is split into frequency pairs indexed by t; pair t is rotated by $\theta_{p,t} = p\,\omega_t$ at absolute position p, with frequency ω_t and a $2{\times}2$ rotation $R(\cdot)$. Because RoPE carries no input-level position term, we again consider two injection sites: the input residual, and the rotation angle itself.

A.6.1 Input-side injection

Since RoPE has no input-level position embedding, the tag prior is added at the input residual exactly as in DeBERTa (Eq. 11, without the segment term, which ModernBERT does not use), and propagates into $\mathbf{q}_p, \mathbf{k}_p$ before rotation; the rotary mechanism itself is untouched. The unifying observation across the three encoders is that input-level injection commutes with whatever positional mechanism the model already uses, whether absolute (RoBERTa), relative (DeBERTa), or rotary (ModernBERT).

A.6.2 Position-pathway injection (SiPE)

Mirroring the Transformer-XL and DeBERTa injections, we instead leave the residual clean and route the prior into the position pathway, which for RoPE is the rotation angle. Using the summed hexatag embedding at position p, a per-layer projection $W_\delta^{(\ell)}$ produces a per-frequency phase offset

$$\Delta\theta_{p,t} \;=\; \big(W_\delta^{(\ell)}(\mathbf{E}^T_{\tau_p} + \mathbf{E}^N_{\nu_p})\big)_t, \qquad (21)$$

which is added to the rotary angle *before* rotation, for both the query at position m and the key at position j:

$$\tilde{q}_{m,t} \;=\; R\big(m\,\omega_t + \Delta\theta_{m,t}\big)\, q_{m,t}, \qquad (22)$$

$$\tilde{k}_{j,t} \;=\; R\big(j\,\omega_t + \Delta\theta_{j,t}\big)\, k_{j,t}. \qquad (23)$$

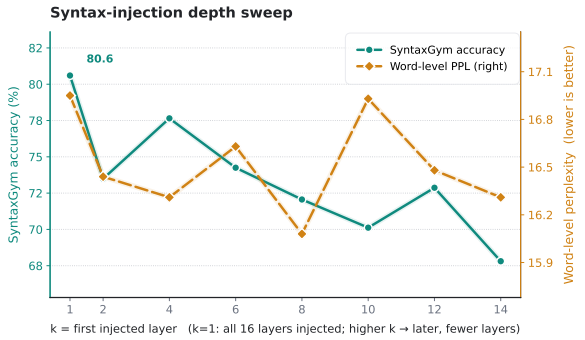

Figure 3: Layerwise SiPE injection sweep on Transformer-XL. Injecting from layer $k{=}1$ (all layers) is best; skipping the first layer already drops SyntaxGym accuracy sharply, and later entry points degrade syntactic generalization further.

The effective angle between query m and key j at frequency t is then $(m-j)\,\omega_t + (\Delta\theta_{m,t} - \Delta\theta_{j,t})$, so the tag enters as a syntactic phase shift on top of the positional one: the rotation that already encodes relative distance is nudged by how the two positions' hexatags relate. With $W_\delta^{(\ell)} \to \mathbf{0}$ we have $\Delta\theta \to 0$ and recover plain RoPE (identity-at-initialization). Because the perturbation enters as a rotation it is norm-preserving, leaving the query and key magnitudes unchanged.[9]

B Time Complexity

SiPE adds two small embedding tables for terminal and non-terminal tags, $\mathbf{E}^T \in \mathbb{R}^{2\times d}$ and $\mathbf{E}^N \in \mathbb{R}^{5\times d}$, totaling $7d$ parameters per pathway ($\sim$5K at $d = 768$, $\sim$7K at $d = 1024$), well under 0.01% of the LM's total parameters[10]. Runtime is correspondingly cheap: for a sequence of length L, each injection is one $\mathcal{O}(d)$ lookup per position, contributing $\mathcal{O}(L{\cdot}d)$ to a forward pass already dominated by the $\mathcal{O}(L^2 \cdot d)$ cost of self-attention. At inference, hexatags are produced by a lightweight `DeBERTa-v3-base` parser, which adds linear overhead on top the encoder's runtime,[1] so total is additive over self-attention rather than multiplicative, a favorable trade against parser-free methods that bake an approximate static structural bias into training.

[9] The phase offset requires the eager attention path; the fused RoPE kernel cannot accept a per-token, per-frequency angle.

[10] For comparison, a single attention layer in RoBERTa-base contains $\sim$2.4M parameters; the priors add less than 0.3% of a single layer's parameters.

C Where Should Syntax Enter? A Layerwise Injection Study

To understand at which depth syntax should interact with semantics, we sweep the entry point of the SiPE bias in Transformer-XL, which couples multiplicatively into the positional pathway. For a model with N layers, we inject the bias from layer k onwards: layers $1, \ldots, k-1$ use the default positional information, while layers $k, \ldots, N$ receive the SiPE bias. We vary $k \in \{2, 4, 6, 8, 10, 12, 14\}$ and compare injecting syntactic information in every layer (from k=1 onwards), measuring syntactic generalization (SyntaxGym) and word-level perplexity on BLLIP_{LG} (Figure 3).

Two findings emerge. First, syntax is most impactful at the very first layer: full injection (k=1) is clearly best, and skipping just the first layer (k=2) already causes a sharp drop in syntactic generalization: SyntaxGym falls from 80.6 to 73.5. Second, and more broadly, syntactic information matters most in the lower layers of a decoder like Transformer-XL: injecting it in later layers yields consistently weaker syntactic generalization. Together, these results indicate that for autoregressive models the positional pathway should carry syntactic structure from the earliest layer onward, rather than being introduced in later layers.

D Hyperparameters

Table 11 reports the task-specific fine-tuning hyperparameters used for each backbone. Unless noted otherwise, all models use AdamW with $(\beta_1, \beta_2) = (0.9, 0.999)$ and $\epsilon = 10^{-6}$.

E Alternative Embedding Combination Strategies

Throughout the paper the terminal and non-terminal priors are the hexatag embedding-table rows for a token's tags, $\mathbf{t}_i = \mathbf{E}^T_{\tau_i}$ and $\mathbf{n}_i = \mathbf{E}^N_{\nu_i}$, added directly to each token's input embedding alongside the standard positional embedding $\mathbf{p}_i$; we call this strategy **ADD_T_NT**. Beyond this default (and as shown in figure 6), we explored a range of alternative fusion strategies, organized into two studies run at different pretraining scales.

Fusion mechanism (1M steps). Pretraining `RoBERTa-base` for 1M steps, we compared three ways of combining the priors with the input (Table 9): addition at the embedding layer (**ADD_T_NT**); concatenation with a learned down-projection, $\tilde{\mathbf{p}}_i = W\,[\mathbf{p}_i; \mathbf{t}_i; \mathbf{n}_i]$ (**CONCAT_T_NT**); and injecting $(\mathbf{t}_i + \mathbf{n}_i)$ into the residual stream before the first self-attention sublayer (**ADD_T_NT (Residual)**). We also tested a learned interpolation gate $\alpha \in \{0.1, 0.5, 0.9\}$ scaling the prior contribution (Table 10). The simplest variant worked best: addition produced the best macro-average, concatenation was worst (with large drops on QNLI and CoLA), residual injection gave only modest gains, and no α initialization improved over the parameter-free additive variant.

Adding dependency relations (500k steps). We then asked whether richer syntactic signal helps. We embedded each token's dependency-relation label (DEPREL; 40 labels, e.g. `nsubj`, `dobj`) and combined it with our priors via several fusion strategies, denoted with a leading **D** or trailing **DR**. Because this sweep was more expensive, all variants—including a re-run baseline and **ADD_T_NT**—were pretrained for 500k steps for an apples-to-apples comparison (Figures 4, 5). Every prior-injection variant outperformed the baseline on macro-average, but **ADD_T_NT** attained the highest mean. Adding DEPRELS on top of $\mathbf{t}_i + \mathbf{n}_i$ (**ADD_T_NT_DR**, **CONCAT_POS_NT_T_DR**) did not recover the gap, and replacing terminals with DEPRELS (**ADD_DNT**, **CONCAT_POS_DNT**) likewise underperformed.

Caveat. The per-dataset breakdown (Figure 5) tempers this story: no single variant dominates everywhere, and the ranking reshuffles substantially across tasks. **ADD_T_NT** is the best on average in our setup, but the optimal way to combine syntactic priors with positional information remains open, particularly for other positional schemes (rotary or relative) or larger pretraining budgets, which we do not study. We therefore present **ADD_T_NT** as a strong, simple default rather than a definitive answer.

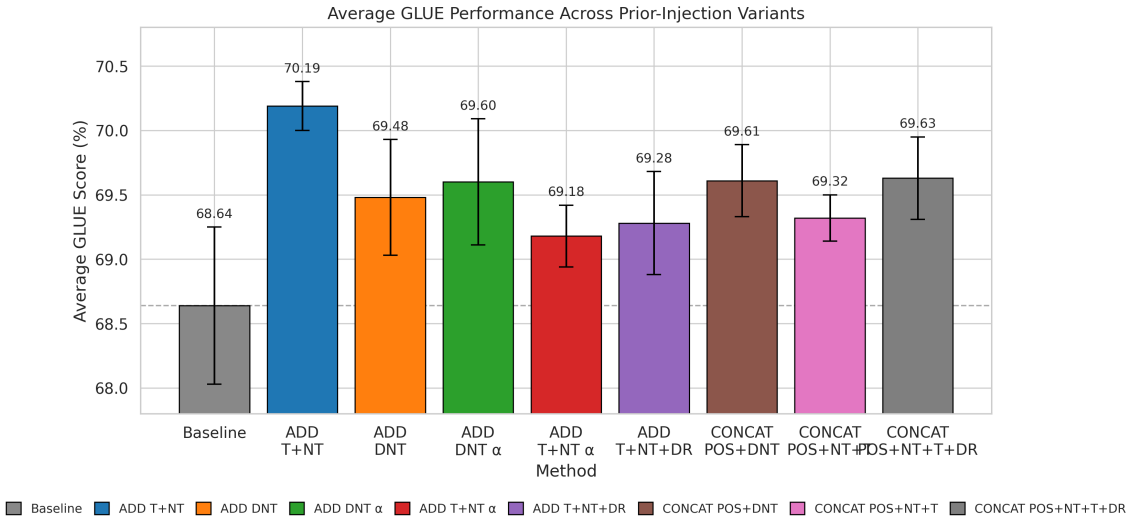

Figure 4: Average GLUE performance across prior-injection variants for RoBERTa-base. The strongest downstream performance is achieved by directly adding terminal and non-terminal prior embeddings to the absolute positional embedding model, outperforming concatenation, weighted addition, residual-stream addition, and variants using full dependency labels. In these experiments, we set $\alpha = 0.5$.

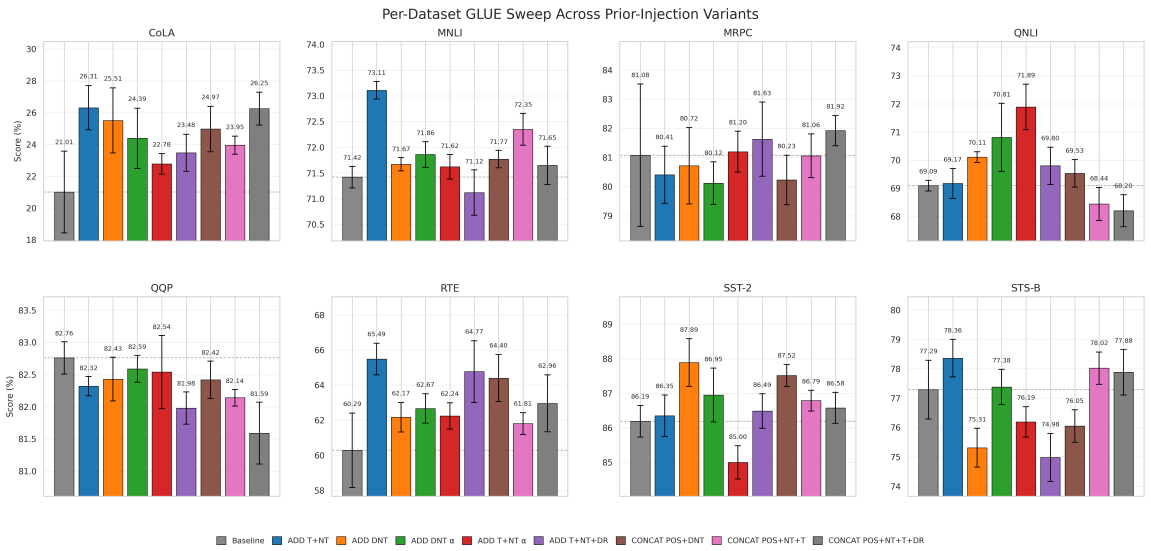

Figure 5: GLUE performance averaged across 3 random seeds **per dataset** under various prior-injection variants for RoBERTa-base. The strongest downstream performance is achieved by directly adding terminal and non-terminal prior embeddings to the absolute positional embedding model, outperforming concatenation, weighted addition, residual-stream addition, and variants using full dependency labels.

Task	Baseline	ADD_T_NT	CONCAT_T_NT	ADD_T_NT (Residual)
CoLA	31.41 (±2.02)	31.77 (±0.71) ↑	26.85 (±1.06) ↓	32.35 (±0.79) ↑
SST-2	87.23 (±0.79)	87.22 (±0.52) ↓	87.27 (±0.35) ↑	87.42 (±0.19) ↑
QQP	83.24 (±0.21)	83.36 (±0.11) ↑	82.60 (±0.22) ↓	83.13 (±0.15) ↓
QNLI	77.65 (±0.50)	78.24 (±0.27) ↑	70.79 (±0.70) ↓	75.47 (±0.54) ↓
MNLI	73.03 (±0.33)	74.05 (±0.17) ↑	72.89 (±0.31) ↓	73.90 (±0.14) ↑
RTE	65.76 (±0.74)	64.66 (±0.56) ↓	64.74 (±1.45) ↓	65.46 (±0.45) ↓
STSB	79.11 (±0.30)	81.41 (±0.13) ↑	79.36 (±0.26) ↑	80.41 (±0.47) ↑
MRPC	80.96 (±0.44)	82.02 (±0.85) ↑	82.76 (±0.72) ↑	81.06 (±0.96) ↑
Average	**72.30**	**72.84** ↑	**70.91** ↓	**72.40** ↑

Table 9: RoBERTa performance on GLUE under alternative strategies for combining syntactic priors. **ADD_T_NT** adds the terminal prior $\mathbf{t}_i$ and non-terminal prior $\mathbf{n}_i$ directly to the token representation (i.e., additive fusion at the embedding layer). **CONCAT_T_NT** concatenates the original embedding (including standard positional information) with $\mathbf{t}_i$ and $\mathbf{n}_i$, then applies a learned linear down-projection to the model dimension before feeding the encoder. **ADD_T_NT (Residual connection)** injects $(\mathbf{t}_i + \mathbf{n}_i)$ into the residual stream before the multi-head self-attention sublayer. Results are averaged over 3 random seeds. Green arrows indicate improvements over the baseline and red arrows indicate degradations. Overall, **ADD_T_NT** yields the strongest macro-average (72.84%), while concatenation is the weakest configuration (70.91%), with particularly large drops on QNLI and CoLA.

Task	Baseline	α = 0.1	α = 0.5	α = 0.9
CoLA	31.41 (±2.02)	29.40 (±1.62) ↓	29.89 (±0.71) ↓	30.37 (±0.54) ↓
SST-2	87.23 (±0.79)	86.93 (±0.34) ↓	86.81 (±0.29) ↓	88.13 (±0.06) ↑
QQP	83.24 (±0.21)	83.04 (±0.08) ↓	83.01 (±0.11) ↓	83.66 (±0.48) ↑
QNLI	77.65 (±0.50)	77.08 (±0.73) ↓	73.72 (±0.22) ↓	76.45 (±2.00) ↓
MNLI	73.03 (±0.33)	72.58 (±0.02) ↓	73.50 (±0.37) ↑	73.07 (±0.05) ↑
RTE	65.76 (±0.74)	66.25 (±0.18) ↑	64.98 (±0.62) ↓	66.07 (±1.44) ↑
STSB	79.11 (±0.30)	78.38 (±0.03) ↓	79.64 (±0.51) ↑	78.25 (±0.14) ↓
MRPC	80.96 (±0.44)	82.77 (±0.09) ↑	81.28 (±1.51) ↑	80.42 (±0.04) ↓
Average	**72.30**	**72.05** ↓	**71.60** ↓	**72.05** ↓

Table 10: Effect of initializing the interpolation gate α for RoBERTa under the **ADD_T_NT** setting. We initialize $\alpha \in \{0.1, 0.5, 0.9\}$ and allow it to be updated during pretraining via backpropagation; all results report downstream GLUE performance after pretraining. Despite task-level variability (notably on QNLI), none of the α initializations improves upon the baseline on average, and all are weaker than the simple additive fusion.

Model	Hyperparam	GLUE Task							
		CoLA	SST-2	MRPC	STS-B	QQP	MNLI	QNLI	RTE
RoBERTa-base/large	LR	1×10^{-5}							
	WD	0.1							
	Epochs	20							
DeBERTa-v3-base/large	LR	1×10^{-5}							
	WD	0.1							
	Epochs	20							
ModernBERT-base/large	LR	8e−5	8e−5	5e−5	8e−5	5e−5	5e−5	8e−5	5e−5
	WD	1e−6	1e−5	5e−6	5e−6	5e−6	5e−6	5e−6	1e−5
	Epochs	20							

Table 11: GLUE fine-tuning hyperparameters per backbone and task. RoBERTa and DeBERTa-v3 use identical settings across all tasks; only ModernBERT's learning rate and weight decay vary by task. All models use AdamW with $(\beta_1, \beta_2) = (0.9, 0.999)$ and $\epsilon = 10^{-6}$. LR denotes learning rate; WD denotes weight decay.

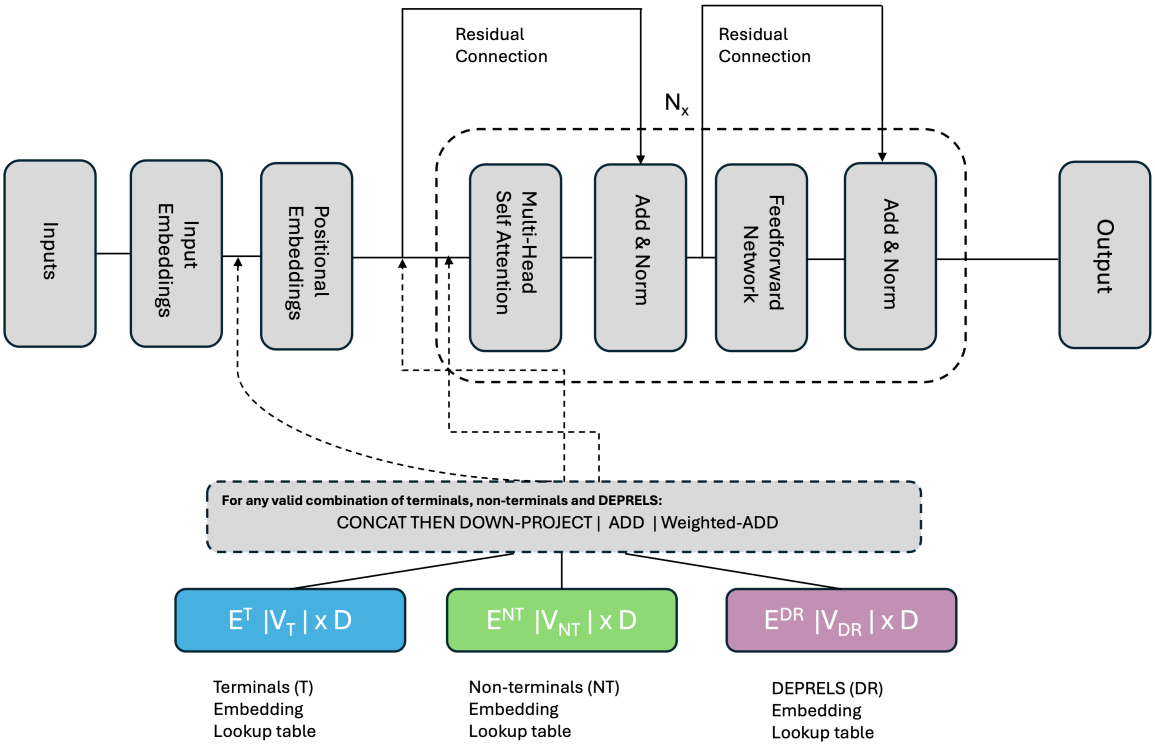

Figure 6: **Injection points for syntactic priors in a Transformer encoder.** We study where and how to inject hexatag-derived priors (terminal $\mathbf{t}_i$, non-terminal $\mathbf{n}_i$, and optionally dependency-relation labels $\mathbf{d}_i$) into RoBERTa-base. Three architectural locations are considered: (i) at the *input embedding*, alongside the token, segment, and positional embeddings; (ii) at the *residual (skip) connection* entering the first self-attention sublayer; and (iii) implicitly in the attention computation, by injecting at the input and propagating through $\mathbf{W}_Q, \mathbf{W}_K$.