

Beyond Sequence Order: Syntax-Informed Positional Embeddings for Transformers

Haris Riaz and Hyungji Kim and Mihai Surdeanu

Department of Computer Science

University of Arizona

Tucson, AZ, USA



Abstract

Positional embeddings (PE) in Transformers encode token distance and order but are largely agnostic to *syntactic structure*. We introduce **Syntax-informed Positional Embeddings (SiPE)**, which learns a lightweight syntactic prior from dependency parses during pretraining and injects it across all three dominant PE families (absolute, relative, rotary), for both encoders and decoders, leaving self-attention and the rest of the architecture untouched. We isolate *where* and *how* the prior should enter the model, and find it depends on the architecture: for autoregressive decoders that use relative PE, the prior is strongest when coupled multiplicatively with the relative-position term of the attention score, outperforming injection into the input embeddings, into self-attention, or into the positional and attention terms jointly—while for encoders it is best added directly to the input embeddings, composing with each encoder’s native positional mechanism. We find that models pre-trained with SiPE improve on the SyntaxGym benchmark by up to 10.3% while simultaneously reducing perplexity by 9.0% over a base model with no syntactic supervision—a metric nearly every existing syntax-injection method instead degrades. Crucially, these gains extend beyond syntactic generalization: SiPE also improves real-world language understanding, raising scores on the GLUE benchmark by up to 8.2% over a model trained without it. Unlike existing syntactic language models that marginalize over many parses at inference or discard syntax at runtime, SiPE conditions on a single parse, establishing a new Pareto frontier between syntactic supervision and inference cost.

1 Introduction

The transformer architecture (Vaswani et al., 2023) forms the foundation of modern LLMs. While

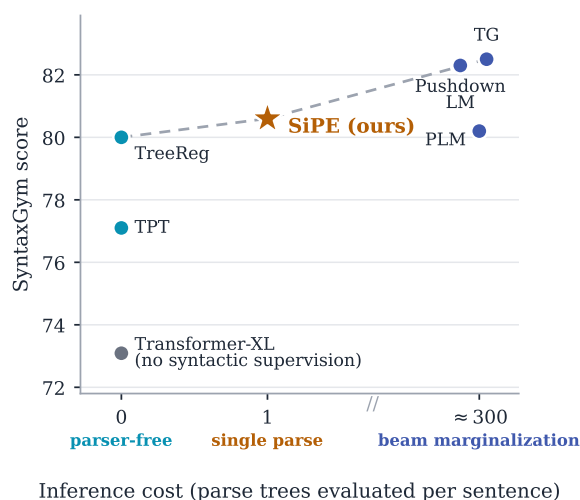


Figure 1: **Our Syntax-Informed Positional Embeddings (SiPE) move the Pareto frontier between syntactic supervision and inference cost.** Each point plots a method’s SyntaxGym score (Table 2) against the number of parse trees it evaluates per sentence at inference. Joint syntactic LMs (TG (Sartran et al., 2022), Pushdown LM (Murty et al., 2023), PLM (Qian et al., 2021)) recover $p(x)$ by marginalizing over ≈ 300 candidate parses; parser-free methods (TreeReg (Nandi et al., 2025), Tree-Planted Transformers (Yoshida et al., 2024)) inject syntax only during training and discard the parser at runtime, making inference cheap but sacrificing some syntactic knowledge. SiPE conditions on a single parse, retaining most of the benefit of full marginalization at 1/300th of its inference cost.

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

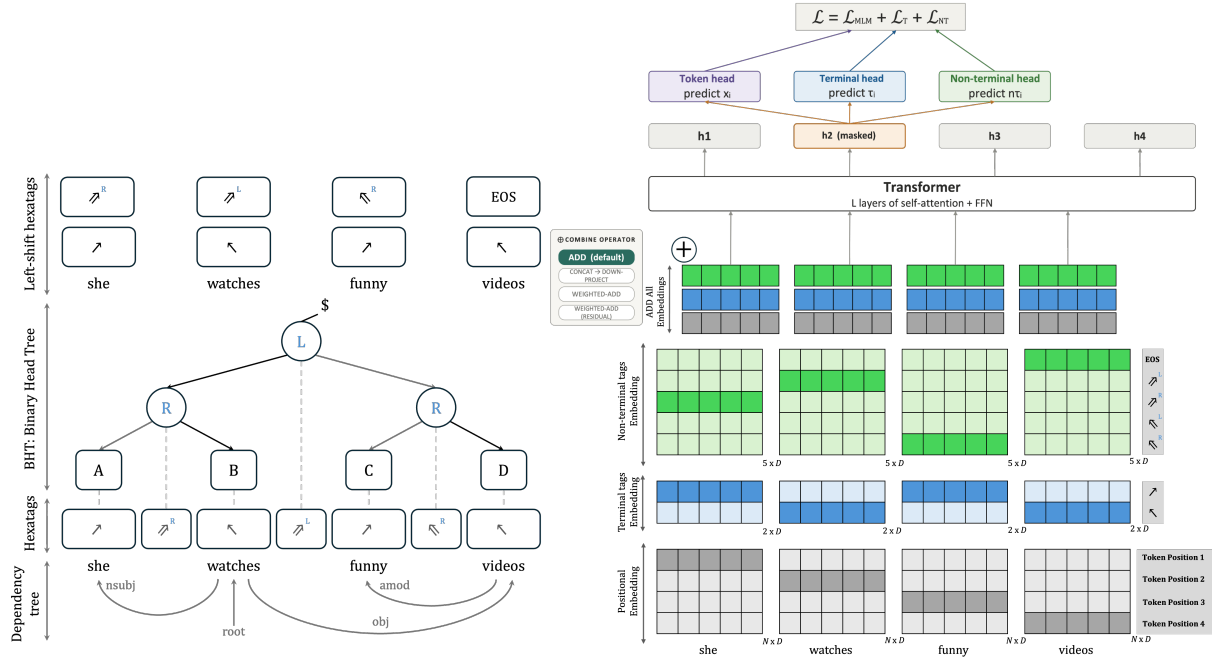


Figure 2: **Left:** Dependency arcs, their binary head tree (BHT), and left-shifted hexatags for “she watches funny videos.” **Right:** Input-pathway SiPE for absolute positional embeddings. At each token’s first subword (subsection B.1), rows from the token ($|V| \times D$; omitted), non-terminal ($5 \times D$), terminal ($2 \times D$), and position ($N \times D$) embedding tables are combined by $\oplus$ into $\mathbf{x}_p$ (Equation 2); addition is the default, with concatenation, weighted addition, and skip-connection addition ablated in Appendix G. The Transformer produces $\mathbf{h}_i$, from which color-matched heads predict the masked token and its two tags, giving $\mathcal{L} = \mathcal{L}_{MLM} + \mathcal{L}_T + \mathcal{L}_{NT}$ (one masked position shown). For decoders, next-token prediction replaces MLM: the token head predicts $i+1$ at every position, while tag heads operate only at first subwords and predict the next tagged position (subsection B.1).

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 modifier (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 or 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., 2021). 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). A broader discussion of related work appears in Appendix A. The main contributions of our paper are:

(1) We introduce **Syntax-informed 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 (§2), 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 on out-of-domain evaluation), indicating the prior generalizes beyond its pre-training distribution (Table 3). The same models also improve on GLUE (Wang et al., 2019) at both small and large scale (tables Table 4, 5 and 6).

(4) For autoregressive decoders like Transformer-XL, injecting through the positional pathway alone is the strongest recipe: it achieves the best SyntaxGym (Hu et al., 2020) score of any configuration of our method (80.60, +10.3 relative improvement over the no-syntax baseline) while cutting perplexity by 9.0% (relative), and drives a significant +8.2% relative improvement on GLUE for the same model (Table 6).

(5) As shown in Figure 1, prior syntactic LMs sit at two extremes: parsing at inference (expensive marginalization over many parse trees) or only during training (discarding syntax 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 most prior syntactic LMs which jointly model syntax and language, SiPE leaves downstream usage unchanged—models architecturally remain compatible with continual pre-training, supervised fine-tuning and reinforcement learning. 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 2).

2 Approach

Both training and inference operate on a hexatagged sequence: the input is first tagged by our parser (subsection B.1). Encoders train with the standard MLM objective; decoder training is standard next-token prediction, as each token carries its own tags and truncating the sequence at any prefix leaves the retained tags intact.

¹Dependency 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 C. Implemented in-house with a multitask-learning encoder, two per-token tag heads: <https://clulab.org/processors/>

2.1 Hexatagging the Input

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.

We discuss the relationship between this formulation and past work on syntax-aware language modeling in Appendix A.

2.2 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. The hexatagging procedure of subsection 2.1 yields two embedding tables, one per tag type, composable with the PE mechanism under any of these operations. At position p they contribute:

$$m_p \cdot (\mathbf{E}_{\tau_p}^T + \mathbf{E}_{\nu_p}^N) \quad (1)$$

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 B.1). Tables are randomly initialized and trained jointly with the LM objective. Throughout the paper, every occurrence of $\mathbf{E}_{\tau_p}^T + \mathbf{E}_{\nu_p}^N$ is implicitly multiplied by m_p ; we omit the mask from later equations for readability, so continuation subwords contribute no tag term anywhere.

2.3 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:

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

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 conflict².

2.4 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) (hereon referred to as TXL), which is one of the earliest “LLM-like” architectures carrying a sinusoidal relative-position encoding directly in the attention score and is a common syntactic-LM baseline (Sartan 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. Notably, this content-conditioned

²However, the input embedding is not the only place in the transformer where we can inject these priors; see appendix G and figure 9.

relative-position term remains competitive at frontier scale: the concurrent 975B-parameter open-weights model Inkling (Thinking Machines Lab, 2026) abandons RoPE for a relative positional bias of the same query–offset-embedding form, citing better quality and length extrapolation. Because **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.

2.4.1 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.

2.4.2 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. Using a layer-specific projection $\mathbf{W}_E^{(\ell)}$, we map the tag at key j into head n ’s subspace and take its inner product with the *same* effective query $\mathbf{q}_{i,n}^{(\ell)} + \mathbf{v}_n$ that appears in the position term **BD** (Eq. 3):

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

where $[\cdot]_n$ selects head n ’s d_{head} -dimensional block and 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*?” (**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 **AC** untouched:

$$\tilde{A}_{i,j,n}^{(\ell)} = \mathbf{AC}_{i,j,n}^{(\ell)} + (1 + c_{i,j,n}^{(\ell)}) \cdot \mathbf{BD}_{i,j,n}^{(\ell)} \quad (6)$$

In our syntactic evaluation, we observe the multiplicative form (Eq. 6) to outperform all other approaches (refer to experiments in table 2 and ablations in table 7, appendix B.2): its correction term $c \cdot \mathbf{BD}$ *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.

2.4.3 Fully-Disentangled Injection

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

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

Now $c_{i,j,n}^{(\ell)}$ depends on the query and the tag but not on $\mathbf{BD}_{i,j,n}^{(\ell)}$, 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 therefore drops to nearly zero when the query does not attend to the offset (i.e., when $\mathbf{BD} \approx 0$). In our experiments, we find that this ungated form is weaker (refer to Table 2 and ablations in Table 8, appendix B.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.

2.4.4 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, (Strubell et al., 2018; Murty et al., 2023; Zhao et al., 2024; Omote et al., 2019; Xie et al., 2021; Yoshida et al., 2024), we propose our hexatag-derived bias applied to attention logits and the attention output, leaving the input residual,

³We initialize $W_E^{(\ell)}$ small, so $c \approx 0$ and $\tilde{A} \approx A$ at initialization: training starts from the vanilla Transformer-XL score and learns the syntactic correction into it.

$\mathbf{AC}$, and the positional pathway $\mathbf{BD}$ untouched ($\mathbf{h}_p^{(0)} = \mathbf{e}_p^{\text{tok}}$).

The mechanism follows 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 $\mathbf{W}_K^{\text{tag},(\ell)}$ to form a key bias that is added to the attention score $\tilde{A}_{i,j,n}^{(\ell)}$, and through $\mathbf{W}_V^{\text{tag},(\ell)}$ to form a value bias that is added to the attended output $\mathbf{z}_{i,n}^{(\ell)}$:

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

$$\mathbf{z}_{i,n}^{(\ell)} = \sum_j \alpha_{i,j,n}^{(\ell)} \left(\mathbf{v}_{j,n}^{(\ell)} + [\mathbf{W}_V^{\text{tag},(\ell)}(\mathbf{E}_{\tau_j}^T + \mathbf{E}_{\nu_j}^N)]_n \right) \quad (9)$$

where $\alpha_{i,j,n}^{(\ell)}$ are the softmax weights, and $[\cdot]_n$ selects multi-head attention head n 's block. Two things differ from the multiplicative alignment score $c_{i,j,n}^{(\ell)}$ of Eq. 5. First, that score was built from the effective query $\mathbf{q}_{i,n}^{(\ell)} + \mathbf{v}_n$ and scaled the position term $\mathbf{BD}$, whereas this key bias uses the plain query $\mathbf{q}_{i,n}^{(\ell)}$ and is added directly to the logit. Second, it is keyed on the tag at the single key position j rather than on a relative pair (i, j) , which keeps the added cost linear (rather than quadratic) in sequence length. The projections may be learned per layer (the (ℓ) superscript) or tied to a single pair shared across all layers; in our experiments we find the former (per-layer variant) outperforms the latter (shared variant) on syntactic generalization tasks, both on its own and when combined with the input-side injection of Eq. 4 (Appendix B.4).

2.4.5 Injecting Syntax into Position and Attention Simultaneously

The final variant pairs our two strongest single-site injections: SiPE on the positional pathway (the multiplicative coupling, Eq. 6) and the per-layer attention-side biases. Combining the two, however, hurts: the joint variant scores below either injection strategy used on its own (Table 2, Appendix B.4). *Injecting the same hexatag signal as both a positional and attentional bias is thus redundant rather than complementary*; for Transformer-XL, the prior is best supplied once, through the positional pathway. Having established *where* and *how* in the score the prior should be combined, we next ask *which layer* it should enter from: at least for the decoder model in our experiments, **syntax is best infused from layer 1 onward** (Appendix D).

2.5 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:

$$S_{ij} = \underbrace{Q_i^c K_j^{c\top}}_{S_{ij}^{\text{con}}} + \underbrace{Q_i^c K_{\delta(i,j)}^r + K_j^c Q_{\delta(i,j)}^r}_{S_{ij}^{\text{pos}}}, \quad (10)$$

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} .

2.5.1 Input-Side Injection

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

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

The prior propagates through $\mathbf{W}_Q$ and $\mathbf{W}_K$ into the content terms of the attention score, while the relative-position mechanisms (its bucketing, lookup table, and projection matrices) are 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. More details, including how the prior propagates to all three score terms, is given in Appendix B.5.

2.5.2 Position-Pathway Injection (PP-SiPE)

Mirroring our Transformer-XL injection, we do not add the prior to the input embeddings (the token representations enter the network unchanged) and instead route it only into the relative-position term S_{ij}^{pos} of the attention score, leaving the content term S_{ij}^{con} 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:

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

$$\tilde{S}_{ij} = S_{ij}^{\text{con}} + (1 + c_{ij}) S_{ij}^{\text{pos}}. \quad (13)$$

The coefficient c_{ij} 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

2.6 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 et al., 2025; Yang et al., 2025; Kamath et al., 2025) delivers positional information through rotary positional 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.

2.6.1 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.

2.6.2 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

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

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

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

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

The effective angle between query m and key j at frequency t is then $(m - j) \omega_t + (\Delta\theta_{m,t} - \Delta\theta_{j,t})$,

⁴ $W_E^{(\ell)}$ is initialized small so that $c_{ij} \approx 0$ and $\tilde{S}_{ij} \approx S_{ij}$ at initialization; training starts from the unmodified DeBERTa score and learns the syntactic correction into it.

<i>BLiMP Example</i>	
✓	The keys to the cabinet are on the table.
✗	The keys to the cabinet is on the table.
<i>SyntaxGym Example</i>	
✓	The farmer near the clerks knows many people.
✗	The farmer near the clerks know many people.

Table 1: Examples from the BLiMP dataset (top) and SyntaxGym (bottom). In both datasets, $p(\checkmark) > p(\times)$, but in BLiMP this probability is computed for the complete sentence, whereas in SyntaxGym it is computed only over the region of interest.

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. Since the tag enters only as a rotation angle, it leaves the query and key magnitudes unchanged and perturbs only their direction (Appendix B.6).

3 Experimental Results

3.1 Experimental Settings and Datasets

Prior work injecting syntax into Transformers focused mainly on *intrinsic* evaluation—directly testing syntactic understanding (Sartran et al., 2022; Xie et al., 2021; Omote et al., 2019; Yoshida and Oseki, 2022; Zhao et al., 2024). 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 1). 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. For our models, test sentences are hex-

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

Model	BLiMP ↑	Δ BLiMP	PPL ↓	Δ PPL	Syntax Gym ↑	Δ SyntaxGym
<i>No Inductive Bias</i>						
Transformer-XL (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
<i>Constituency Parsing</i>						
PLM (GPT-2) (Qian et al., 2021)	75.10	↓ -0.27	29.80	↓ -59.96	80.20	↑ +9.73
Transformer Grammar (Transformer-XL) (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
<i>Tree-Planted Transformers (TPT) (Yoshida et al., 2024)</i>						
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
<i>Syntax-Informed Embeddings with Hexatagging (Ours)</i>						
<i>SiPE (positional pathway)</i>						
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
<i>Attention Bias</i>						
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
<i>SiPE + Attention Bias</i>						
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 2: Language modeling and syntax evaluation of decoder LMs; all our SiPE variants (Ours) use Transformer-XL as the decoder model, and each baseline’s model is noted in parentheses. BLiMP and SyntaxGym are higher-is-better, while PPL (perplexity, computed on BLLIP-LG test split (Charniak et al., 2000)) is lower-is-better. Δ columns report relative improvement or degradation against vanillaTransformer-XL (tokens). Best scores for each dataset are **bolded**.

atagged once by the parser before scoring, so our perplexity is conditioned on this single fixed parse, $p(x \mid \hat{T}(x))$, whereas joint syntactic LMs report a marginal $p(x)$ approximated by summing over many candidate parses.⁶

In a second set of experiments, we pretrain three encoders: RoBERTa, DeBERTa, ModernBERT (Liu et al., 2019; He et al., 2023; Warner

⁶This is the trade-off SiPE makes by design: a single parse at inference instead of expensive marginalization (Figure 1).

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

Table 3: BLiMP accuracy results for models trained with SiPE. For each model family we report its most performant GLUE variant: input-pathway SiPE injection for the encoders and positional-pathway SiPE injection for the Transformer-XL decoder; the OOD experiments likewise use the most performant GLUE variant of each family. For encoders, we use PLL-based accuracy. Δ : relative change w.r.t. the corresponding base model. OOD section: models pre-trained on WikiText, then continuously pre-trained on BLLIP-LG (Charniak et al., 2000). $\uparrow$ improvement, $\downarrow$ degradation. **Bold**: best relative improvement with our method.

et al., 2024) and Transformer-XL on a 50M-token, offline-hexatagged slice of WikiText-103 (Merity et al., 2016).⁷ 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; for the Transformer-XL decoder we instead use sentence log-likelihood (the sum of next-token log-probabilities). Accuracy is scored against BLiMP’s gold labels.

Full pre-training hyperparameters for the BLLIP-LG and WikiText-103 experiments are reported in Appendix F.1.

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

⁷Our goal is to study syntactic embeddings on real downstream tasks, so we pretrain on WikiText and then fine-tune on GLUE. For encoders, we also re-use the same checkpoints for syntactic evaluation as a separate BLLIP-LG pretraining run with its own hyperparameter tuning was beyond our academic budget.

fine-tune all four backbones on each GLUE task using the settings reported in Appendix F.2; results appear in Tables 4, 5, and 6.

3.2 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 vanilla token baseline) with perplexity cut from 18.63 to 16.95 (Table 2). This carries through to GLUE, improving the TXL macro average from 68.17 to 73.78, a dramatic +8.2% relative gain (Table 6). 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, cutting perplexity by roughly a quarter against the strongest such baseline (16.95 vs. 22.30 for TreeReg) and by more than half against the Tree-Planted Transformer variants (16.95 vs. 45.5–47.7) while matching or exceeding their syntactic generalization.

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 outperforms the base model on GLUE (Table 4); 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 3)—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.⁸

⁸The 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 2 (PLM, Transformer Grammars, TreeReg, most TPT variants) likewise fall below the vanilla token baseline on BLiMP.

Task	RoBERTa (base)		DeBERTa (base)			ModernBERT (base)		
	Base	+SiPE	Base	+SiPE	+PP-SiPE	Base	+SiPE	+PP-SiPE
CoLA	31.41 (± 0.02)	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)	68.98 (± 0.37)

Table 4: 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)		ModernBERT (large)	
	Base	+SiPE	Base	+SiPE
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 5: GLUE evaluation results for large encoder models.

4 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

Task	Transformer-XL	
	Base	+PP-SiPE
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 6: 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.

intrinsic and extrinsic evaluations and multiple encoder and decoder transformer variants.

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 model. 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 the requirements of model 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 G, Figures 7 and 8), 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 8) show that no single setting dominates across all tasks.

Perhaps the biggest limitation, which we plan to address in future work, is fast autoregressive text generation with the decoder. Because the injected prior is conditioned on hexatags, generating each new token requires re-tagging the sequence produced so far. Our hexatagger is fast, so this is feasible in principle, but it is not compatible with standard KV-caching: as the sequence grows, the parser may revise the tags of earlier tokens, changing their injected representations and invalidating the cached keys and values. Efficient incremental decoding under a per-step syntactic prior is therefore an open research-engineering problem, and the direction we consider most important for future work.

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 sizes increase.

Acknowledgments

This work used Jetstream2 (Hancock et al., 2021) at Indiana University through allocations CIS260698 and CIS250722 from the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) program (Boerner et al., 2023). We gratefully acknowledge the U.S. National Science Foundation’s ACCESS program for providing the computational resources that supported the experiments in this work.

Mihai Surdeanu declares a financial interest in lum.ai. This interest has been properly disclosed to the University of Arizona Institutional Review Committee and is managed in accordance with its conflict of interest policies.

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*.
- {Timothy J.} Boerner, Stephen Deems, {Thomas R.} Furlani, {Shelley L.} Knuth, and John Towns. 2023. [Access: Advancing innovation: Nsf’s advanced cyberinfrastructure coordination ecosystem: Services & support](#). In *PEARC 2023 - Computing for the common good*, PEARC 2023 - Computing for the common good: Practice and Experience in Advanced Research Computing, pages 173–176, United States. Association for Computing Machinery. Publisher Copyright: © 2023 Owner/Author.; 2023 Practice and Experience in Advanced Research Computing, PEARC 2023 ; Conference date: 23-07-2023 Through 27-07-2023.
- 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, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai,

- and 181 others. 2025. [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](#). *Preprint*, arXiv:2010.07792.
- David Y. Hancock, Jeremy Fischer, John Michael Lowe, Winona Snapp-Childs, Marlon Pierce, Suresh Marru, J. Eric Coulter, Matthew Vaughn, Brian Beck, Nirav Merchant, Edwin Skidmore, and Gwen Jacobs. 2021. [Jetstream2: Accelerating cloud computing via jetstream](#). In *Practice and Experience in Advanced Research Computing 2021: Evolution Across All Dimensions*, PEARC '21, New York, NY, USA. Association for Computing Machinery.
- 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 impacts 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 Kingsbury and Martha Palmer. 2002. [From Tree-Bank to PropBank](#). In *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC'02)*, Las Palmas, Canary Islands - Spain. European Language Resources Association (ELRA).
- 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](#). *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*, page 2699–2712. 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.
- Emma Strubell, Patrick Verga, Daniel Andor, David Weiss, and Andrew McCallum. 2018. [Linguistically-informed self-attention for semantic role labeling](#). In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 5027–5038, Brussels, Belgium. Association for Computational Linguistics.

- 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.
- Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019. [BERT rediscovers the classical NLP pipeline](#). In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4593–4601, Florence, Italy. Association for Computational Linguistics.
- Thinking Machines Lab. 2026. [Inkling: Our open-weights model](#). Blog post; model weights at <https://huggingface.co/thinkingmachineslab/Inkling>.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. [Attention is all you need](#). *Preprint*, arXiv:1706.03762.
- 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. 2021. [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](#). *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 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 attention masks, adding parameters and training complexity (Xie et al., 2021; 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 2.2).

B Preliminaries

B.1 Subword-Level Tag Injection

Hexatag annotations are produced at the word level, whereas the language model operates on subword tokens: the tokenizer may split a word w_i into one or more subwords $s_{i,1}, \dots, s_{i,k_i}$. We must therefore specify how each word-level tag pair is aligned 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 $s_{i,2}, \dots, s_{i,k_i}$ 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 the

following mask:

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

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

Assigning the same hexatag to all k_i subwords would contribute k_i loss terms per word and overweight words that fragment into many pieces. We leave this alternative 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* tagged 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; wherever a token is masked, its injected tag embedding is masked along with it (the m_p term is zeroed), so the auxiliary head must infer the tag from surrounding context rather than copy it from its own input.

B.2 SiPE Position-Pathway Design Choices for Transformer-XL

The multiplicative position-pathway injection of section 2.4 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 the dimension $W_E^{(\ell)}$ projects the tag into before its inner product with

the query forms the alignment score c : either a single attention head’s d_{head} -dimensional subspace (a small map that places the tag in the same space as that head’s query, so c is formed by the same query that produces **BD** and lands on the same per-head scale; the one projected vector is then shared across all heads), or the full model dimension d_{model} (a larger map whose output is split into n_{head} blocks, giving each head its own projected tag). Table 7 reports all four combinations.

Sharing	Projection	BLiMP $\uparrow$	SG $\uparrow$
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.

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 works better with d_{model} , whereas the fully-disentangled coupling of Appendix B.3 works better in 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.

B.3 Fully-Disentangled Syntax Injection Design Choices for Transformer-XL

For the fully-disentangled variant of section 2.4.3, we evaluate the same two design axes as for the multiplicative position-pathway variant: layer sharing and projection target. Table 8 reports all four combinations.

The best disentangled configuration is per-layer projection into the head subspace (74.72 BLiMP,

Sharing	Projection	BLiMP $\uparrow$	SG $\uparrow$
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.

78.72 SG; Table 8). In terms of syntactic generalization performance, it outperforms input-side injection (73.82 BLiMP, 76.97 SG) but lags behind the multiplicative coupling (§2.4.2). We thus find that the least to most performant design choices for entangling the syntactic prior are: input-side < fully disentangled < multiplicative.

B.4 Attention-Side and Combined Injection Design Choices for Transformer-XL

Relation to the Shaw formulation: The relationship to Shaw et al. (2018) is given in the main-text construction (Eqs. 8–9). Two implementation details are omitted there: a position-validity mask m_j zeroes the bias wherever the tag label is -100 , and because we set $\text{mem_len} = 0$, the bias spans exactly the intra-segment keys and requires no padding for memory positions.

Sharing across layers. We compare a single pair of $W_K^{\text{tag}}, W_V^{\text{tag}}$ shared across layers with independent projections in each layer. Per-layer projections perform better on the syntactic evaluations, both alone and when combined with input-side injection, so all reported attention-side variants use the per-layer configuration.

B.5 DeBERTa-v3: Disentangled Relative Positional Embeddings and Tag Injection

Background: disentangled attention. Expanding the grouped position term in Eq. 10, the score for a query at position i and key at position j decomposes into a content-to-content (c2c), a content-to-position (c2p), and a position-to-content (p2c)

term,

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

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.⁹

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:

$$\mathbf{q}_i = \mathbf{W}_Q \left(\mathbf{e}_i^{\text{tok}} + \mathbf{e}_i^{\text{seg}} + m_i \cdot (\mathbf{E}_{\tau_i}^T + \mathbf{E}_{\nu_i}^N) \right), \quad (19)$$

$$\mathbf{k}_j = \mathbf{W}_K \left(\mathbf{e}_j^{\text{tok}} + \mathbf{e}_j^{\text{seg}} + m_j \cdot (\mathbf{E}_{\tau_j}^T + \mathbf{E}_{\nu_j}^N) \right). \quad (20)$$

Eqs. 18–20 show explicitly that the tag priors propagate through the content projections into c2c, c2p, and p2c, while the bucketing function, relative-position table, and $\mathbf{W}_{c2p}, \mathbf{W}_{p2c}$ projections remain unchanged.

Position-pathway injection: In the position-pathway variant of section 2.5.2, the factor $(1 + c_{ij})$ multiplies c2p and p2c while c2c remains unchanged. Thus the expanded form makes clear that DeBERTa applies the syntactic coefficient to two relative-position terms rather than Transformer-XL’s single BD term.

B.6 ModernBERT: Rotary Injection Details

Identity at initialization. With $W_\delta^{(\ell)} \rightarrow \mathbf{0}$ we have $\Delta\theta \rightarrow 0$ in Eq. 14, so Eqs. 15–16 reduce to plain RoPE: training starts from an unmodified ModernBERT, exactly as in the Transformer-XL and DeBERTa variants.

⁹The 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. $\{+9, \dots, +15\}$ might collapse into a single bucket — so the table stays small while preserving fine-grained discrimination near the diagonal.

C Time Complexity

SiPE adds two embedding tables, $\mathbf{E}^T \in \mathbb{R}^{2 \times d}$ and $\mathbf{E}^N \in \mathbb{R}^{5 \times d}$, totaling $7d$ parameters per pathway ($\sim 5\text{K}$ at $d = 768$, $\sim 7\text{K}$ at $d = 1024$), well under 0.01% of the LM’s parameters.¹⁰ Lookup and addition cost $\mathcal{O}(Ld)$ per sequence. Position-pathway variants add tag–query interactions with $\mathcal{O}(L^2d)$ complexity, matching self-attention’s asymptotic cost. A single parser pass supplies the inference-time hexatags with overhead linear in the input length.¹

D Where Should Syntax Enter? A Layerwise Injection Study

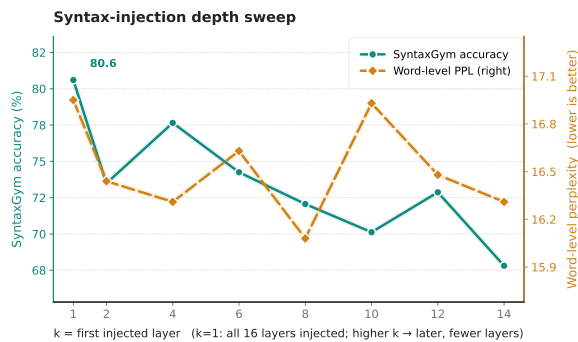


Figure 3: Layerwise SiPE injection sweep on Transformer-XL. Injecting from layer $k=1$ (all layers) is best; skipping the first layer already drops SyntaxGym accuracy sharply, and later entry points degrade syntactic generalization further.

To understand at which depth syntax should interact with semantics, we sweep the entry point of the positional pathway SiPE bias (PP-SiPE) in Transformer-XL. For a model with N layers, we inject the bias from layer k onwards: layers $1, \dots, k-1$ use the default positional information, while layers $k, \dots, N$ 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

¹⁰For comparison, a single attention layer in RoBERTa-base contains $\sim 2.4\text{M}$ parameters; the priors add less than 0.3% of one layer’s parameters.

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.

Complementing this view of *where* syntax should enter, Appendix E examines *how* the injected prior manifests in attention on BLiMP object–verb agreement: the encoders redistribute verb→object attention to mirror syntactic adjacency, whereas Transformer-XL surfaces the prior only weakly in its attention maps and instead converts it into the largest downstream semantic (GLUE) gains of any model we train.

E Analyzing Attention Patterns on Object–Verb Agreement Tasks

A natural desideratum for SiPE-pretrained models is that they should up-weight attention between tokens that are linearly distant but syntactically adjacent (distance 1 in the dependency tree). We probe this on the BLiMP *Causative* split for three model architectures spanning the three positional encoding families: RoBERTa-base (absolute), ModernBERT-base (rotary), and Transformer-XL (relative), each against its SiPE counterpart. Verb→object attention is averaged across all heads; the layers we average over differ by architecture. For Transformer-XL, our layerwise injection study (Appendix D) shows that syntax matters at *every* layer, so we average over all of its layers. We have not run the corresponding sweep for the encoders, so we instead follow Tenney et al. (2019), who find that “BERT’s intermediate layers encode a rich hierarchy of linguistic information, with surface features at the bottom, syntactic features in the middle and semantic features at the top,” and therefore average over each encoder’s middle layer band.¹¹

Across all three families, SiPE attains higher overall BLiMP accuracy than the corresponding base model. We therefore inspect 20 minimal pairs for which SiPE is correct and the base model is not. The two encoders consistently redirect verb→object attention toward the syntactically relevant object, whereas Transformer-XL shows the same direction much more weakly (Figures 4–6).

¹¹Layers 5–8 for the 12-layer RoBERTa-base and layers 9–12 for the 22-layer ModernBERT-base; all layers 1–16 for the Transformer-XL we pre-train.

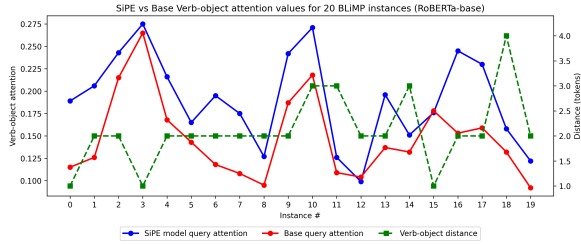


Figure 4: Verb→object attention on BLiMP (Warstadt et al., 2023) for RoBERTa-base vs. RoBERTa-base+SiPE (input pathway injection), both pre-trained on the 50M-token WikiText slice (subsection 3.1). On the subset of *Causative* examples where SiPE assigns higher PLL to the grammatical sentence but the base model does not, verb→object attention is higher under SiPE in 90% of cases (Appendix E).

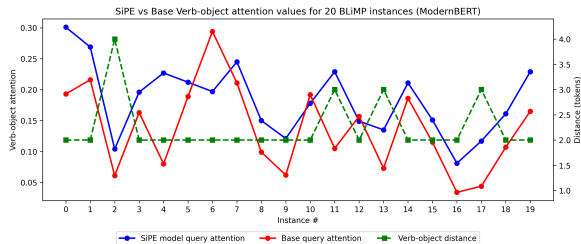


Figure 5: Verb→object attention on BLiMP for ModernBERT-base vs. ModernBERT-base+SiPE (input pathway injection), both pre-trained on the 50M-token WikiText slice (subsection 3.1). On *Causative* examples where SiPE is correct and the base model is not, the SiPE model assigns higher verb→object attention in 17/20 (85%) cases, mirroring the RoBERTa pattern despite ModernBERT’s rotary positional encoding.

In over 80% of the inspected examples the object is nonadjacent to the verb, so the encoder pattern cannot be explained by a preference for neighboring tokens.

The weaker Transformer-XL attention effect is notable because the decoder still obtains the largest downstream gain in our experiments (Table 6). This suggests that its use of the syntactic signal extends beyond merely redistributing attention weights: the decoder may instead route more of this information toward richer semantic representations that support downstream understanding, unlike the clearer attention-level pattern observed in the encoders.

F Hyperparameters

F.1 Pre-training Hyperparameters

For each model family and size, the baseline and SiPE variants use identical pre-training and fine-tuning hyperparameters. Both Transformer-XL settings use a 16-layer, 8-head backbone with hidden size 1024.

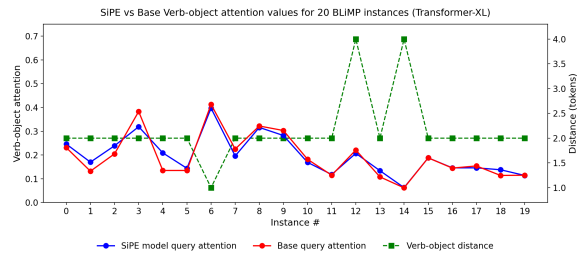


Figure 6: Verb→object attention on BLiMP for Transformer-XL vs. Transformer-XL+SiPE (positional pathway injection), both pre-trained on the 50M-token WikiText slice (subsection 3.1), averaged over all 16 layers. On *Causative* examples where SiPE assigns higher sentence log-likelihood to the grammatical sentence but the base model does not, SiPE places higher verb→object attention in 11/20 (55%) cases: the same direction as the encoders (Figs. 4 and 5), but a much weaker majority. The green dashed line (right axis) marks the verb→object surface distance.

Corpus	Training steps/epochs	Batch/acc.	LR	Warmup	WD/drop.
BLiP-LG	4 epochs	64/1	1.5e−4	8k	0/0
WikiText-103	50M tokens	8/4	2.5e−4	4k	0.01/0.1

Table 11: Transformer-XL pre-training hyperparameters. Batch/acc. denotes per-device batch size and gradient accumulation; WD/drop. denotes weight decay and dropout.

Model	Steps	Batch/acc.	LR	Warmup	WD	MLM
RoBERTa-base	1M	16/4	2e−5	20k	0.01	0.15
RoBERTa-large	250k	16/4	2e−5	20k	0.01	0.15
DeBERTa-v3-base	300k	4/1	2e−5	0	0.01	0.15
ModernBERT-base/large	1M	4/1	2e−5	0	0.01	0.30

Table 12: Encoder pre-training hyperparameters. Batch/acc. denotes per-device batch size and gradient accumulation.

F.2 GLUE Fine-tuning Hyperparameters

Table 13 reports the principal fine-tuning hyperparameters used for each backbone.

	LR	WD	Epochs	Batch
<i>RoBERTa-base/large & DeBERTa-v3-base/large</i>				
all tasks	1e−5	0.1	20	32
<i>Transformer-XL</i>				
all tasks	1e−5	0.1	20	16
<i>ModernBERT-base/large (per task)</i>				
CoLA	8e−5	1e−6	20	32
SST-2	8e−5	1e−5	20	32
MRPC	5e−5	5e−6	20	32
STS-B	8e−5	5e−6	20	32
QQP	5e−5	5e−6	20	32
MNLI	5e−5	5e−6	20	32
QNLI	8e−5	5e−6	20	32
RTE	5e−5	1e−5	20	32

Table 13: GLUE fine-tuning hyperparameters. Only ModernBERT’s learning rate and weight decay vary by task.

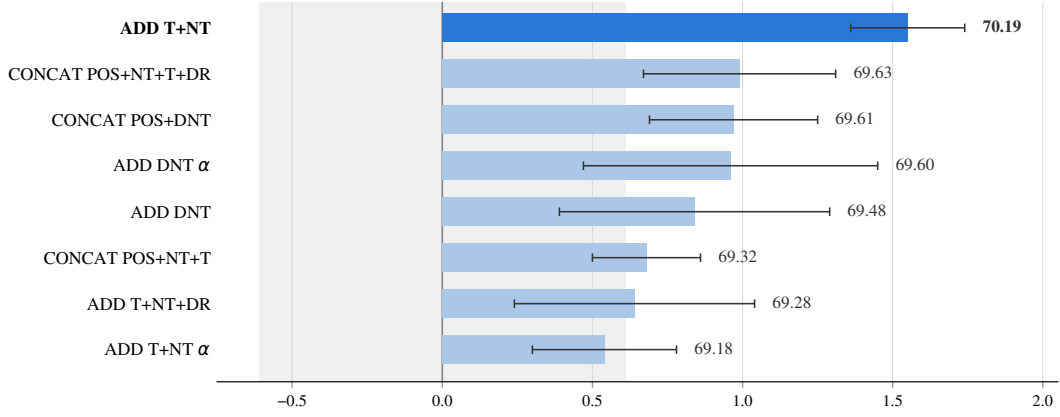


Figure 7: Average GLUE performance across prior-injection variants for RoBERTa-base, shown as the improvement over the no-prior baseline (68.64 ± 0.61 , shaded band at zero); bar-end labels give absolute scores and whiskers denote one standard deviation over 3 seeds. The strongest downstream performance is achieved by directly adding terminal and non-terminal prior embeddings to the absolute positional embedding model (**ADD T+NT**), outperforming concatenation, weighted addition, and variants using full dependency labels (the residual-injection variant is evaluated separately in Table 9). In these experiments, we set $\alpha = 0.5$.

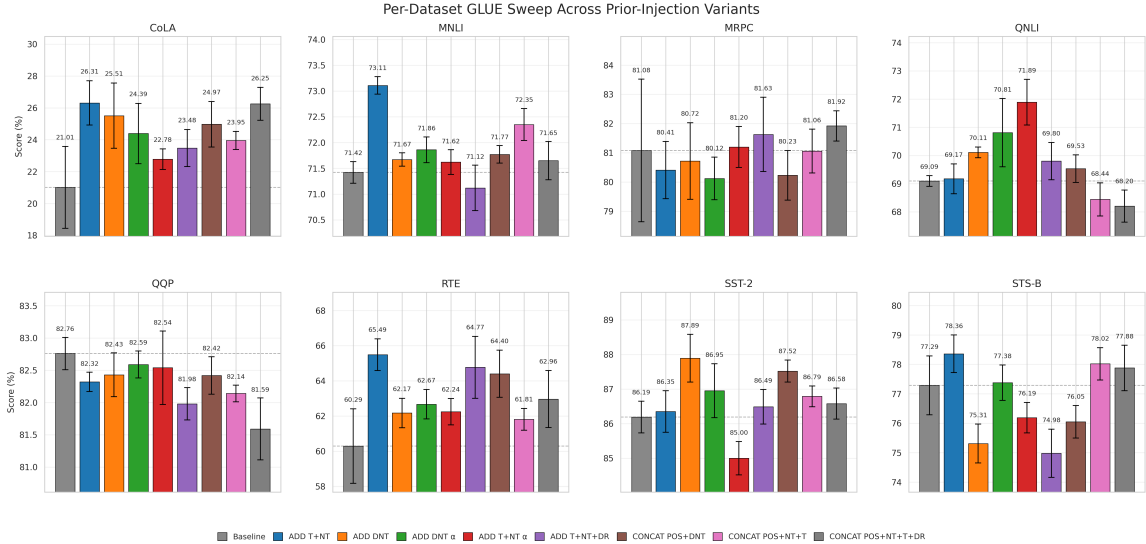


Figure 8: GLUE performance averaged across 3 random seeds **per dataset** under various prior-injection variants for RoBERTa-base. The strongest downstream performance is achieved by directly adding terminal and non-terminal prior embeddings to the absolute positional embedding model, outperforming concatenation, weighted addition, and variants using full dependency labels (the residual-injection variant is evaluated separately in Table 9).

G Alternative Embedding Combination Strategies

Throughout the paper, each token carries just two hexatag priors: a terminal tag $\mathbf{t}_i = \mathbf{E}_{r_i}^T$ and a non-terminal tag $\mathbf{n}_i = \mathbf{E}_{\nu_i}^N$, read from small learned tables ($2 \times D$ for the terminals, $5 \times D$ for the non-terminals). But is this the richest syntactic signal we could inject? A natural alternative is the token’s full *dependency-relation* label (DEPREL; 40 labels such as *nsubj* and *doobj*), which would come from a much larger $40 \times D$ table, giving

a per-token embedding deprel_i . This opens two questions we study together: whether the richer DEPREL prior helps, and, once it is combined with the terminal and non-terminal priors, *how* all of these vectors should be fused with the token and positional embeddings, by addition, concatenation, a learned interpolation weight, or injection deeper into the network.

We explore both questions as a small neural-architecture search over injection strategies. To keep it tractable we run the search on the simplest positional scheme, absolute positional embeddings,

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

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

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.

using RoBERTa-base as the encoder (the case illustrated in Figure 9). The strategies we explore are collected in Table 14 and split into two studies at different pretraining scales.

Fusion mechanism (1M steps). We first fix the priors to $\mathbf{t}_i$ and $\mathbf{n}_i$ and vary only *how* they enter the model, pretraining RoBERTa-base for 1M steps (top block of Table 14, Table 9). The simplest option wins: **ADD_T_NT**, which just adds the two priors to the input embedding with no extra parameters, attains the best macro-average. **CONCAT_T_NT** is the weakest (with large drops on QNLI and CoLA), and injection into the model’s residual (skip) connection (**ADD_T_NT (Residual Connection)**) yields only modest gains. Replacing the plain addition with a learned linear interpolation between the input embedding and the prior, **ADD_T_NT (α)**: $\mathbf{x}'_i = \alpha \mathbf{x}_i + (1 - \alpha)(\mathbf{t}_i + \mathbf{n}_i)$,

never improves over **ADD_T_NT** at any α initialization (Table 10). Adding capacity or depth to the fusion, therefore, does not help; direct addition at the embedding layer is both the cheapest and the strongest.

Adding dependency relations (500k steps). We then enrich the prior itself: we embed each token’s DEPREL label and combine it with the terminal and non-terminal priors through the bottom-block strategies of Table 14. Because this study is more expensive, every variant, including a re-run baseline and **ADD_T_NT**, is pretrained for 500k steps for an apples-to-apples comparison (Figures 7, 8). Every prior-injection variant improves over the no-prior baseline on macro-average, yet **ADD_T_NT** still attains the highest mean. Layering DEPRELS on top of $\mathbf{t}_i + \mathbf{n}_i$ (**ADD_T_NT_DR**, **CONCAT_POS_NT_T_DR**) does not close the

Strategy	Injected input representation $\mathbf{x}'_i$	Description
<i>Fusion mechanism (terminal + non-terminal tags)</i>		
ADD_T_NT	$\mathbf{x}'_i = \mathbf{x}_i + \mathbf{t}_i + \mathbf{n}_i$	Add both priors to the input embedding (default; no extra parameters).
CONCAT_T_NT	$\mathbf{x}'_i = \mathbf{W} [\mathbf{x}_i; \mathbf{t}_i; \mathbf{n}_i]$	Concatenate with the input embedding, then down-project to D .
ADD_T_NT (Residual Connection)	$\mathbf{h}_i^{(1)} = \text{LN}(\mathbf{x}_i + \mathbf{t}_i + \mathbf{n}_i + \text{Attn}(\mathbf{x})_i)$	Add the priors to the skip connection of the first attention sublayer.
ADD_T_NT (α)	$\mathbf{x}'_i = \alpha \mathbf{x}_i + (1 - \alpha)(\mathbf{t}_i + \mathbf{n}_i)$	Linearly interpolate the input embedding with the (terminal + non-terminal) prior; learned α initialized in $\{0.1, 0.5, 0.9\}$.
<i>Adding dependency relations (deprel_i, 40 labels)</i>		
ADD_T_NT_DR	$\mathbf{x}'_i = \mathbf{x}_i + \mathbf{t}_i + \mathbf{n}_i + \text{deprel}_i$	Add all three priors to the input embedding.
CONCAT_POS_NT_T_DR	$\mathbf{x}'_i = \mathbf{W} [\mathbf{x}_i; \mathbf{n}_i; \mathbf{t}_i; \text{deprel}_i]$	Concatenate all priors with the input embedding, then down-project.
ADD_DNT	$\mathbf{x}'_i = \mathbf{x}_i + \text{deprel}_i + \mathbf{n}_i$	Swap the terminal tag for the DEPREL.
ADD_DNT (α)	$\mathbf{x}'_i = \alpha \mathbf{x}_i + (1 - \alpha)(\text{deprel}_i + \mathbf{n}_i)$	Interpolated counterpart of ADD_DNT ; learned α initialized in $\{0.1, 0.5, 0.9\}$.
CONCAT_POS_DNT	$\mathbf{x}'_i = \mathbf{W} [\mathbf{x}_i; \text{deprel}_i; \mathbf{n}_i]$	Concatenate (DEPREL replacing the terminal), then down-project.

Table 14: Injection strategies we explore for absolute positional embeddings on RoBERTa-base. $\mathbf{x}_i = \mathbf{e}_i + \mathbf{p}_i$ is the standard input embedding (token $\mathbf{e}_i$ plus positional $\mathbf{p}_i$), $\mathbf{x}'_i$ the representation actually fed to the encoder, and $\mathbf{W}$ a learned projection back to the model dimension D . $\mathbf{t}_i/\mathbf{n}_i$ are the terminal/non-terminal hexatag priors and deprel_i the dependency-relation prior. For the residual variant we write the first attention sublayer explicitly, $\mathbf{h}^{(1)} = \text{LN}(\mathbf{x} + \text{Attn}(\mathbf{x}))$: the priors are added on the skip branch, so unlike **ADD_T_NT** they do not enter the self-attention computation itself. The top block fuses only the terminal and non-terminal tags; the bottom block additionally brings in DEPRELS.

gap, and swapping the terminal tag for the DEPREL is no better, whether added directly (**ADD_DNT**), concatenated (**CONCAT_POS_DNT**), or interpolated through a learned weight, **ADD_DNT (α)**: $\mathbf{x}'_i = \alpha \mathbf{x}_i + (1 - \alpha)(\text{deprel}_i + \mathbf{n}_i)$. Richer syntactic signal, at least in this form, does not translate into stronger downstream performance.

Overall finding and limitations. Across these experiments, **ADD_T_NT**—the direct addition of terminal and non-terminal priors at the input embedding—is the strongest injection strategy for absolute positional embeddings on the RoBERTa encoder. Two points are worth noting here. First, the per-dataset results (Figure 8) show that no single variant is optimal on every task: the relative ordering of strategies varies across GLUE tasks, and **ADD_T_NT** is strongest in the macro-average rather than uniformly. Second, our search covers only the absolute-PE encoder. The optimal injection strategy for relative (DeBERTa-v3, Transformer-XL) and rotary (ModernBERT) positional schemes, as well as under larger pretraining budgets, remains open. We therefore adopt **ADD_T_NT** as a simple and robust default and

leave a systematic study of injection strategies across positional schemes to future work.

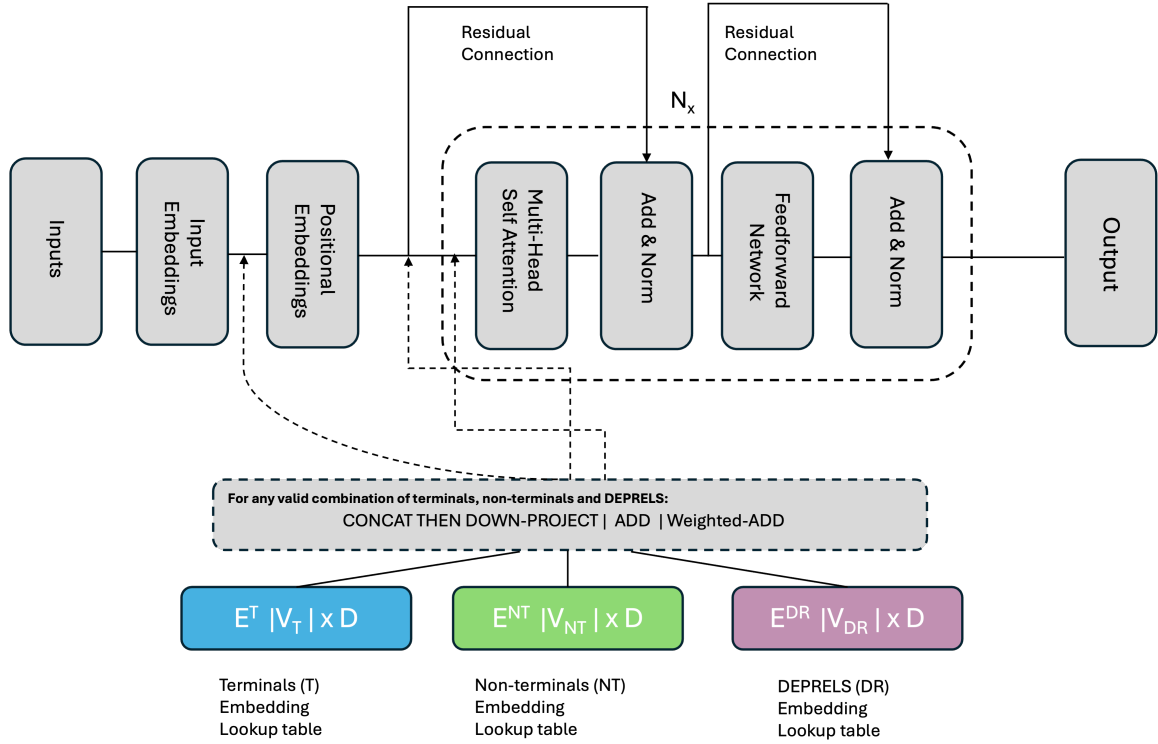


Figure 9: **Injection points for syntactic priors in a Transformer encoder.** We study where and how to inject hexatag-derived priors (terminal t_i , non-terminal n_i , and optionally dependency-relation labels deprel_i) into RoBERTa-base. Three architectural locations are considered: (i) at the *input 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 $\mathbf{W}_Q, \mathbf{W}_K$.