INPUT-RELATIONAL VERIFICATION OF TRANSFORMERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Transformers have been driving significant advances across a wide range of applications, yet their deployment in safety-critical domains require rigorous formal verification. Most existing verification techniques are designed to assess perturbation tolerance around individual inputs and are therefore insufficient to ensure model robustness in a global context. To address this limitation, we first propose a self-composition-based approach that enables end-to-end application of local verification techniques. However, due to the complexity of the composed model, this approach turns out to exacerbate the approximation error of local verifiers significantly. In this paper, we show that exploiting cross-execution relationships in intermediate layers can substantially improve verification precision. Specifically, we construct McCormick envelopes for the cross-execution differences of attention-layer and propagate these relational constraints to the output layer, yielding tighter output bounds. We implement this approach in a tool, `ReVerT`, and develop two variants that prioritize efficiency and precision, respectively, to accommodate different verification requirements. We evaluate the approach on CCPP for regression and on Yelp and SST for classification. Across these benchmarks, the proposed methods produce tighter certified output bounds and certify more instances than the naive self-composition and the generic difference propagation, showing that our reasoning of the cross-execution differences is effective for achieving precise input-relational verification of Transformers.

1 INTRODUCTION

Transformers have driven significant advances across a wide range of applications, such as medical decision support, autonomous navigation, unmanned aerial vehicles, industrial monitoring, and financial risk analysis. In these applications, empirical accuracy is only one aspect of reliability: a model may perform well on average while still behaving unpredictably on rare but safety-critical inputs. For high-stakes systems, sampling- or statistics-based certificates are insufficient; instead, we need sound and rigorous guarantees that rule out entire classes of unsafe behavior. Formal verification addresses this need by formally modeling the system and statically analyzing its output range over a given input set, thereby rigorously proving whether the model satisfies the desired properties.

Formal verification has been extensively studied for conventional fully-connected neural networks Kaulen et al. (2025), yet remains much less explored for Transformers. Due to the complexity of Transformer architectures, exact encoding Liao et al. (2023) is not scalable and inevitably compromises the encoding of complex nonlinear operations such as softmax, creating a gap with real Transformer models. By contrast, approximating those operations by linear or polytopic upper/lower bounds is more tractable, and existing approaches Shi et al. (2020); Wei et al. (2023); Zhang et al. (2024); Huang et al. (2026); Liu et al. (2026) have designed different abstractions that soundly bound the non-linear operations, such as dot products and softmax in self-attention.

Despite the rigor and effectiveness of these approaches, their guarantees are limited to safety properties defined over perturbations of individual inputs. However, this is far from sufficient for real-world applications: a medical model should give consistent risk estimates for clinically similar patients; a drone controller should not issue sharply different commands for nearly identical sensor states; and an industrial monitoring model should produce similar predictions for similar operating conditions. These examples require models to behave consistently across closely related inputs. Because the

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

desired guarantee is relational, involving multiple model inputs and their corresponding inferences, the problem cannot be solved by directly applying existing verifiers.

To handle this problem, we first propose a self-composition-based approach: by composing multiple copies of a given model, we turn the relational verification problem into a form that can be handled by applying existing techniques. While this provides a viable solution, the complexity of the composed model exacerbates the approximation error of existing verifiers, which further results in severe imprecision in the verification results and frequent occurrences of false alarms.

Essentially, this naive self-composition solution applies existing verification techniques in an end-to-end manner, which considers only the inputs and outputs of the concerned inferences. In contrast, in this paper, we show that exploiting the cross-execution relationships between intermediate layers across multiple inferences can provide additional information propagated throughout the verification process, and is therefore an effective way to enhance the precision of verification. By incorporating these relationships into the verification procedure, our approach can obtain significantly tighter output bounds, leading to improved verification performance and fewer false alarms.

To realize this, we aim to characterize the differences between corresponding internal operations across multiple inferences and propagate such differences through the layers. This is technically challenging because self-attention involves complex nonlinear operations, and we need to reason about the bounds of their *differences*, rather than simply bounding individual executions. To this end, we design a McCormick envelope that uses linear bounds to over-approximate the differences between corresponding operations, enabling these relational bounds to be propagated through the network to the output layer. Specifically, by reformulating the straightforward expression of the cross-execution difference, we preserve the dependencies over the relational variables, and this turns out to be effective in reducing the approximation error and improving verification precision.

Based on the proposed bounding technique, we devise a tool `ReVerT`, which stands for **RE**lational **VER**ification for **T**ransformers. `ReVerT` has two variants: `ReVerT-F` computes the output bounds efficiently by propagating the bounds through different layers; `ReVerT-P` formulates all linear constraints over different layers as a linear program (LP), leading to higher verification precision. We evaluate the performances of `ReVerT` on three datasets CCPP, Yelp, and SST, ranging from control engineering to NLP. Across these benchmarks, `ReVerT` produces tighter certified output bounds and certifies more instances than the naive self-composition and the generic difference propagation, showing that our approach is effective for achieving precise verification results. In summary,

- We formalize the input-relational verification problem for Transformers and propose self-composition as a solution that enables the application of local verifiers.
- We propose a McCormick envelope for over-approximating the bound differences of non-linear operations, and instantiate two variants including `ReVerT-F` that directly propagates difference bounds, and `ReVerT-P` that employes LP solving.
- We evaluate on CCPP, Yelp, and SST, showing tighter certified output-difference bounds and improved verification success over self-composition and generic difference-propagation baselines.

Related work Neural network verification has been extensively studied in the past decade Katz et al. (2017); Ehlers (2017); Zhang et al. (2018); Singh et al. (2019); Bunel et al. (2020), but remain relatively underexplored for Transformers. Early works such as Shi et al. (2020) and Bonaert et al. (2021) introduce approximation to relax the outputs of non-linear operations, while most later works aim to improve precision and scalability Wei et al. (2023); Zhang et al. (2024); Huang et al. (2026); Liu et al. (2026). Given the growth of the needs for *globally robust* models, input-relational verification, e.g., Banerjee et al. (2024); Suresh et al. (2024); Banerjee & Singh (2024) began to draw attention in the context of classic neural network verification. However, to the best of our knowledge, no existing solution has been proposed for Transformers, while this work aims to fill this gap.

2 Background and Problem Formulation

2.1 Transformer Model

We consider encoder-only Transformers (Vaswani et al., 2017) that have been widely adopted in models such as BERT Devlin et al. (2019). The matrix $H^0 \in \mathbb{R}^{L \times d}$ is an initial hidden representation

transformed from an original input x by an embedding function f_{emb}, where L and d are respectively the sequence length and the hidden dimension. For textual inputs, the rows of H^0 correspond to token embeddings; for tabular or vectorized inputs, they can be viewed as feature tokens. A Transformer T with N encoder layers computes its output $T(x)$, as follows:

$$H^0 = f_{\text{emb}}(x), \qquad H^\ell = f^\ell(H^{\ell-1}) \quad (\ell \in \{1, \ldots, N\}), \qquad T(x) = f_{\text{out}}(H^N). \tag{1}$$

In Eq. 1, f^ℓ is the ℓ-th encoder layer, and f_{out} is the prediction head. H^ℓ represents the output of the ℓ-th encoder layer, while the final layer representation H^N is mapped by the prediction head f_{out} to a vector $T(x)$, for downstream task-specific objectives.

Given $H^{\ell-1}$, the ℓ-th encoder layer f^ℓ consists of a series of transformations to compute H^ℓ:

$$A^\ell = f^\ell_{\text{attn}}(H^{\ell-1}), \quad \widetilde{H}^\ell = f^\ell_{\text{AN},1}(H^{\ell-1}, A^\ell), \quad F^\ell = f^\ell_{\text{ffn}}(\widetilde{H}^\ell),\ H^\ell = f^\ell_{\text{AN},2}(\widetilde{H}^\ell, F^\ell). \tag{2}$$

In Eq. 2, A^ℓ is the attention output, $\widetilde{H}^\ell$ is the hidden state after the first residual Add&Norm block, F^ℓ is the feed-forward output, and H^ℓ is the final output of layer ℓ. The functions $f^\ell_{\text{AN},1}$ and $f^\ell_{\text{AN},2}$ denote the two Add&Norm wrappers. Eq. 2 describes the macro-structure of a Transformer encoder layer. The feed-forward sublayer is made of affine transformations and standard activations; residual connections are additions; and the no-variance LayerNorm abstraction used in our implementation is affine. The distinctive challenge for Transformer verification, compared to classic neural network verification, comes from the attention function f^ℓ_{attn}, so we here expand it explicitly.

For an attention block $f^\ell_{\text{attn}}(H^{\ell-1}) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_h}}\right)V$, let $H^{\ell-1}$ be the input hidden-state matrix to the attention block. The query, key, and value matrices are respectively computed as follows:

$$Q = H^{\ell-1}W_Q, \qquad K = H^{\ell-1}W_K, \qquad V = H^{\ell-1}W_V. \tag{3}$$

In Eq. 3, W_Q, W_K, and W_V are learned projection matrices. Multiplying the hidden state H by these matrices produces queries Q, keys K, and values V. Queries and keys determine which positions attend to which other positions, while values carry the information that will be aggregated.

Given these matrices, the attention block f_{attn} computes its output by three sub-components: attention scores S, probabilities P, and context matrix C, as follows:

$$S = \frac{QK^\top}{\sqrt{d_h}}, \qquad P = \text{softmax}(S), \qquad C = PV, \tag{4}$$

In Eq. 4, S is the score matrix obtained by the dot product operation between Q and K. Each element S_{ij} measures how strongly token position i attends to position j, normalized by a factor $\sqrt{d_h}$ set as the dimension of queries and keys. The row-wise softmax converts each score row into a probability vector P_i, and $C = PV$ computes each context vector by taking weighted sums of value vectors. These three operations pose unique challenges for Transformer verification: $QK^\top$ and PV are bilinear, and softmax introduces normalized nonlinear dependencies across each row.

The final hidden representation is mapped to an output vector $T(x) \in \mathbb{R}^c$ by the prediction head in Eq. 1. For regression, $c = 1$ or a small output dimension. For classification, $T(x)$ is a vector of logits. At this stage, we keep the formulation of verification problems task-agnostic and leave the discussion on task-specific objectives in the experimental setup.

2.2 Problem Formulation

Verification of Transformers has been studied in literature Shi et al. (2020); Liao et al. (2023); Bonaert et al. (2021); Wei et al. (2023); Zhang et al. (2024); Huang et al. (2026); Liu et al. (2026); however, all these approaches are limited to verifying the *local robustness* of models against perturbations. Formally, this requirement can be defined as follows: given a fixed clean input x, for any perturbed input x' that holds $\|x' - x\| \leq \delta_x$ (where δ_x is a radius scalar and $\|\cdot\|$ denotes a distance norm), the output should hold some requirements such as the classification labels for x and x' should be the same. Accordingly, it suffices to solve the problem by computing the reachable region

$$\Omega := \{T(x') \mid \|x' - x\| \leq \delta_x\}$$

of $T(x')$ for all perturbed x' and checking whether Ω overlaps with the region that denotes the violation of output requirement. As directly computing Ω is difficult, most existing techniques pursue

an over-approximated but computationally tractable region $\hat{\Omega} \supseteq \Omega$ as the reachable region of x'. If $\hat{\Omega}$ does not overlap with the violation region, Ω is certainly safe as well. For the details of the construction of $\hat{\Omega}$, we refer the readers to Appendix A; here, we use $\hat{T}$ to denote the approximative projection from x' to $\hat{\Omega}$, i.e., $\hat{\Omega} = \{\hat{T}(x') \mid \|x' - x\| \leq \delta_x\}$.

However, as aforementioned, the local robustness requirement is far from sufficient for verification use cases in real world. Consider the example where a drone controller should not issue sharply different commands for nearly identical sensor states. This requirement does not reason about perturbations to a specific input; rather, it considers a set of relational inputs that result in parallel inferences. Verification of this property requires to reason about the relationship between multiple inferences, and therefore cannot be solved by existing techniques dedicated to local robustness.

We formulate the corresponding verification problem to be the following one.

Definition 1 (Global robustness)**.** *Let $\mathcal{D}$ be a bounded input domain and let $T : \mathcal{D} \to \mathbb{R}^c$ be a Transformer. T is* globally robust *if for any pair of inputs x and x' such that $x \in \mathcal{D}, x' \in \mathcal{D},\ \|x' - x\| \leq \delta_x$, it holds the requirement $\mathsf{dist}(T(x), T(x')) \leq \epsilon_y$, where $\mathsf{dist}(\cdot)$ is a distance measurement, and ϵ_y is a scalar that denotes a tolerance threshold.*

The input-relational verification problem then aims to decide whether a Transformer T satisfies the requirement in Def. 1 under the given input conditions. For regression tasks, $\mathsf{dist}(\cdot)$ can be defined as the absolute difference directly; for classification tasks, we define it as the absolute difference between the selected component of the output vector following Wang et al. (2022). This specification is sufficiently expressive to cover a wide range of applications; its instantiations to regression and classification tasks used in the experiments are described in Appendix G.

3 Self-Composition – Lifting Local Verifiers to Global Contexts

Input-relational verification reasons about multiple inferences that proceed in parallel. By treating each inference as an execution of a separate copy of the original model, we can view these parallel inferences as a single inference over a *composed* model consisting of multiple copies of the original model. This transformation allows us to apply existing verification techniques to global robustness. This aligns with *self-composition* Barthe et al. (2004), a classic approach used to handle k*-safety properties*[1] in program verification. The resulting approach can be formally represented as follows:

$$\max \quad \mathsf{dist}(\hat{y}, \hat{y'}), \qquad \text{s.t.} \quad x \in \mathcal{D}, x' \in \mathcal{D},\ \|x' - x\| \leq \delta_x,\ \hat{y} = \hat{T}(x),\ \hat{y'} = \hat{T}(x') \tag{5}$$

where $\hat{y} = \hat{T}(x)$ and $\hat{y'} = \hat{T}(x')$ are respectively the over-approximation of the outputs of x and x', namely, $T(x) \in \hat{y}$ and $T(x') \in \hat{y'}$. As mentioned in §2.2, $\hat{T}(\cdot)$ implemented a series of techniques that over-approximate the outputs of a Transformer inference $T(\cdot)$. As these techniques often adopt linear constraints to bound the outputs of intermediate layers, Eq. 5 is therefore a linear program (LP). Note that we do not impose more restriction than linearity to $\hat{T}(\cdot)$, therefore Eq. 5 can accommodate most existing Transformer verification approaches, such as Shi et al. (2020).

The program in Eq. 5 essentially asks how large the approximated output difference $\mathsf{dist}(\hat{y}, \hat{y'})$ can be under the input constraints, especially $\|x' - x\| \leq \delta_x$. The abstract outputs $\hat{y}$ and $\hat{y'}$ soundly over-approximate the corresponding outputs of the original model. Therefore, if $\mathsf{dist}(\hat{y}, \hat{y'})$ is at most ϵ_y, no admissible input pair has an output difference exceeding ϵ_y. In this case, the model is globally robust over the specified input domain.

This baseline is sound because each individual copy over-approximates its concrete executions, while the input relation constraints are explicitly incorporated. Its limitation, however, is precision. Although the final LP considers both copies jointly, their abstract executions are still constructed independently. The pairwise relation is imposed at the input and queried through the output objective, but the verifier does not explicitly track how the differences between the two executions evolve through hidden layers and attention. This motivates propagating these differences throughout the network, rather than relating the two executions only after each has been locally abstracted.

[1] Note that, the global robustness property in Def. 1 is essentially a k-safety property.

4 CROSS-EXECUTION BOUNDS PROPAGATION

We introduce our verification approach that, unlike the end-to-end call of existing verifiers, exploits cross-execution dependencies by propagating the difference bounds of intermediate layers. We first introduce the overall framework, and then detail our main technical contribution.

4.1 DIFFERENCE PROPAGATION FRAMEWORK

Similarly to self-composition, `ReVerT` keeps the individual abstractions for the related inferences, but additionally, it also maintains an explicit affine representation of the difference between the two inferences and propagates it through the layers. For each operation, `ReVerT` simultaneously reasons about the corresponding quantities H^ℓ and H'^ℓ in the two inferences, as well as their difference $\Delta H^\ell = H'^\ell - H^\ell$. The purpose of tracking this triple is to ensure that, for every pair of inferences on (x, x'), we can simultaneously bound the hidden quantities H^ℓ, H'^ℓ, and their difference ΔH^ℓ. This allows the final difference bound $\mathsf{dist}(\hat{y}, \hat{y'})$ to be derived by propagating ΔH^ℓ layer by layer, rather than by relating the final outputs only after independently abstracting the two inferences.

Affine layer. For many Transformer operations, this is straightforward. For instance, for linear operations, their outputs (denoted as z, z' and Δz) can be represented as follows:

$$z = H^{\ell-1}W + b,\ z' = H'^{\ell-1}W + b,\ \Delta z = z' - z = (H'^{\ell-1} - H^{\ell-1})W = \Delta H^{\ell-1}W. \tag{6}$$

where $H^{\ell-1}$, $H'^{\ell-1}$, $\Delta H^{\ell-1}$ are the outputs of the prior operations, and W and b respectively represent the weight matrices and bias vectors. Because the network parameters are shared between the two inferences, the bias cancels and the output difference Δz is exactly the same linear map applied to the input difference $\Delta H^{\ell-1}$. This rule applies to embeddings represented as affine maps, query/key/value projections, feed-forward linear layers, attention output projections, and final prediction heads.

Residual connection. This kind can also be handled exactly: if $o = a + b$, then it holds that$\Delta o = \Delta a + \Delta b$, where a and b are the two operands of the residual addition, and Δa and Δb denote their cross-execution differences. Thus, residual addition preserves the difference relation exactly.

Normalization and activation. Under the normalization layers structure used in our evaluated model, following previous works Shi et al. (2020); Bonaert et al. (2021); Zhang et al. (2024); Huang et al. (2026); Liu et al. (2026), layerNorm abstraction are handled according to the affine abstraction witch is also exact; the corresponding formula is provided in Appendix D. Activation differences are handled using standard relational relaxations like Banerjee et al. (2024); Suresh et al. (2024); Banerjee & Singh (2024), as described in Appendix C.

4.2 ATTENTION DIFFERENCE PROPAGATION

This subsection presents our main technical contribution. Attention layers consist of several non-linear operations, including the score product $QK^\top$, the softmax probabilities, and the context product PV. In line with our introduced framework in §4.1, we also need to handle the propagation of their differences through the layers. Based on the results in Shi et al. (2020), softmax and context products can be approximated using dot products, and therefore handling these operations boils down to reasoning about dot-product operations. We thus present our approximation for dot-product differences and defer the details of handling softmax and context product to Appendix E.

Q/K/V differences. Since the query, key, and value projections are affine, their differences are:

$$\Delta Q = \Delta H^{\ell-1}W_Q, \qquad \Delta K = \Delta H^{\ell-1}W_K, \qquad \Delta V = \Delta H^{\ell-1}W_V. \tag{7}$$

where $\Delta H^{\ell-1}$ is hidden-state difference from the prior operations, same as that in Eq. 6, and W_Q, W_K, and W_V are the projection matrices given in Eq. 3. Eq. 7 means that query, key, and value differences are obtained by applying the corresponding projection to $\Delta H^{\ell-1}$; no relaxation is needed.

Score difference ΔS. Let S and S' be the attention-score matrices in the two inferences, which are respectively derived from the query and key matrices Q, K and Q', K' in the respective inference. Given the definition of score in Eq. 4, the score difference can be computed as $\Delta S = \frac{Q'K'^\top - QK^\top}{\sqrt{d_h}}$.

Note that, this results in two bilinear terms $Q'K'$ and $QK^\top$, which are non-linear and therefore require relaxation for propagation in subsequent layers. While it is possible to relax these two terms independently, by respectively applying the planar McCormick envelope for dot products (as given in Appendix A), doing this cannot hold the dependencies between Q' and Q, and between K' and $K^\top$. Consequently, this introduces substantial approximation error, because Q' and Q, for example, become free variables in their respective domain, ignoring the restrictive difference bound between them. To mitigate this issue, we propose the following method based on a reformulated representation of the expression:

$$\Delta S = \underbrace{\frac{Q'K'^\top - QK^\top}{\sqrt{d_h}}}_{\text{naive: dependencies lost}} = \underbrace{\frac{\Delta Q K'^\top + Q\Delta K^\top}{\sqrt{d_h}}}_{\text{ours: dependencies preserved}} \tag{8}$$

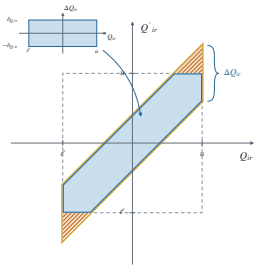


Figure 1: Comparison between the naive expression and our refined one

where $\Delta Q = Q' - Q$ and $\Delta K = K' - K$ are the difference matrices of queries and keys respectively. While Eq. 8 also requires relaxation for bilinear terms, this time the constraints between ΔQ and Q, and between $K'^\top$ and $\Delta K^\top$ are much weaker and therefore significantly reduce the approximation error. This intuition is further illustrated in Fig. 1, where we use ir to index a matrix element. Given $Q_{ir}, Q'_{ir} \in [\ell, u]$ and $|\Delta Q_{ir}| \leq \delta_{Q,ir}$ these conditions derive the valid domain for Q and Q', i.e., the blue hexagon. By the naive expression in Eq. 8, as Q and Q' can vary freely in their respective range, the domain of Q and Q' is essentially abstracted to the dashed box. In contrast, our refined expression in Eq. 8 turns to reason about ΔQ and Q; consequently, it yields the the blue box domain, which corresponds to the orange parallelogram, another abstraction of the hexagon. Compared to the dashed box, the parallelogram is much tighter, and thus brings significant improvement.

To compute the bounds of ΔS, let us consider the representation of Eq. 8 at the element level, namely, each entry ΔS_{ij} of ΔS can be represented as follows:

$$\Delta S_{ij} = \frac{1}{\sqrt{d_h}} \sum_{r=1}^{d_h} (g_{ijr} + m_{ijr}), \qquad \text{where } g_{ijr} = \Delta Q_{ir} K'_{jr},\ m_{ijr} = Q_{ir}\Delta K_{jr}. \tag{9}$$

Here i indexes the query position, j indexes the key position, and r indexes the head dimension. The auxiliary variable g_{ijr} represents one scalar product from the term $\Delta Q K'^\top$, while m_{ijr} represents one scalar product from $Q\Delta K^\top$. This scalar form is the one used to generate constraints, because each bilinear product can be relaxed separately. Each auxiliary variable is relaxed with the McCormick envelope. For $m_{ijr} = Q_{ir}\Delta K_{jr}$, given $Q_{ir} \in [\underline{Q_{ir}}, \overline{Q_{ir}}]$ and $\Delta K_{jr} \in [\underline{\Delta K_{jr}}, \overline{\Delta K_{jr}}]$ obtained from Eq. 7, the product m_{ijr} satisfies the following upper and lower bounds:

$$\begin{aligned} m_{ijr} &\geq \underline{Q_{ir}}\Delta K_{jr} + \underline{\Delta K_{jr}} Q_{ir} - \underline{Q_{ir}}\,\underline{\Delta K_{jr}}, \quad m_{ijr} \geq \overline{Q_{ir}}\Delta K_{jr} + \overline{\Delta K_{jr}} Q_{ir} - \overline{Q_{ir}\Delta K_{jr}}, \\ m_{ijr} &\leq \overline{Q_{ir}}\Delta K_{jr} + \underline{\Delta K_{jr}} Q_{ir} - \overline{Q_{ir}}\,\underline{\Delta K_{jr}}, \quad m_{ijr} \leq \underline{Q_{ir}}\Delta K_{jr} + \overline{\Delta K_{jr}} Q_{ir} - \underline{Q_{ir}}\,\overline{\Delta K_{jr}}. \end{aligned} \tag{10}$$

In Eq. 10, the four inequalities form a convex over-approximation of the saddle $m_{ijr} = Q_{ir}\Delta K_{jr}$ over its rectangle domain. Similarly, we apply this method to other dot-product operations, such as $g_{ijr} = \Delta Q_{ir} K'_{jr}$, and compute their bounds. In our `ReVerT-F`, these product relaxations are converted into one propagated affine lower bound and one propagated affine upper bound. In the `ReVerT-P`, all four inequalities are kept as LP constraints.

Softmax and context differences. Once the score difference has been propagated, the remaining attention operations reuse the same product-difference template. Softmax is decomposed into exponentials, a reciprocal normalizer, and products between numerators and the normalizer; the probability difference is then expressed by applying the product-difference identity to these terms. The context computation is also a product-difference problem because $C = PV$ and $C' = P'V'$. We include the explicit formulas and row-sum constraints in Appendix E. After ΔC is propagated, the attention output projection, residual connection, normalization, and feed-forward block follow the exact and activation rules above.

4.3 Fast and Precise Instantiations

ReVerT-F. `ReVerT-F` is the direct propagation version of our approach. It maintains affine lower and upper bounds for the base execution, the prime execution, and the difference. At each nonlinear or bilinear step, including attention score products, softmax products, and context products, it keeps one lower affine form and one upper affine form for the next layer.

This makes `ReVerT-F` scalable because it does not solve a complex LP. The tradeoff is that once it commits to propagated affine bounds, it discards other constraints that could have been useful.

ReVerT-P. `ReVerT-P` goes through the same difference-propagation pipeline, but it formulates all the bounding constraints as an LP that aims to maximize the output bound. The LP includes exact affine difference equations, ReLU and tanh difference relaxations, McCormick constraints for ΔS, softmax products, and ΔC, softmax row-sum constraints, and consistency constraints tying base, prime, and difference variables together. `ReVerT-P` is therefore the same affine relational abstraction plus solver tightening. This can lead to tighter certified bounds, and the improvement can arise from, e.g., for dot product (see Fig. 1), considering all constraints leads to the use of the hexagon, rather than the parallelogram as done by `ReVerT-F`.

Soundness follows by induction over the Transformer computation graph, since exact affine rules preserve concrete differences and each nonlinear or bilinear rule is a sound over-approximation. We give the formal statement and proof sketch in Appendix H.

5 Experimental Evaluation

We present the experimental evaluation. All our code and data have been made publicly available [2].

5.1 Experiment Settings

Research Questions We evaluate three questions:

- **RQ1 Tightness.** Do `ReVerT-F` and `ReVerT-P` produce smaller certified global robustness bounds than self-composition and simpler difference-propagation variants, and how close are these bounds to empirical feasible-pair lower bounds?
- **RQ2 Scalability.** What runtime overhead do the methods introduce, and how do they scale with Transformer depth?
- **RQ3 Verification Success.** Given a target output threshold, how often can each method certify the property, and what runtime cost does this require?

Benchmarks We evaluate the methods on one regression benchmark and two classification benchmarks using encoder-only Transformers with depths 1, 3, 6, and 12. Dataset, model, and task-objective details are given in Appendix G.

Compared Methods We consider six comparison methods, including five verification methods and an empirical baseline denoted `Sampling`; details can be found in Table 4 in Appendix G. `Sampling` evaluates concrete feasible input pairs on nested finite grids and returns their maximum observed output difference. Prior work does not directly provide a verifier for the global robustness specification of Transformer in Def. 1, but we manage to adapt the local Transformer verifier of Shi et al. (Shi et al., 2020) into the two-execution baseline `SelfComp`, as described in §3. The two naive variants, `Naive-F` and `Naive-P`, derived the naive expression in Eq. 8, serve as two ablation baselines to understand the effectiveness of our refined expression.

Metrics We report tightness, runtime, and certification success. Tightness is the mean taskwise ratio of a method's output-difference bound to a reference: `SelfComp` in Table 1 and `Sampling` in Table 2. Smaller ratios indicate tighter bounds; `Sampling` provides an empirical lower bound. Runtime is wall-clock verification time; lower is better. Certification success is the fraction of evaluated tasks whose sound bound is at most ϵ_y; higher is better. Formal definitions and aggregation rules are given in Appendix G.

[2] `https://anonymous.4open.science/r/global_robustness_verification-8272`

5.2 Main Results

RQ1: Tightness. We compare output-difference bounds in terms of the normalized ratio in Eq. 39. Table 1 compares the verification methods against `SelfComp` on CCPP, Yelp, and SST. Table 2 compares `SelfComp`, `ReVerT-F`, and `ReVerT-P` against `Sampling` on CCPP. Each entry averages ratios for the corresponding method and reference on matched tasks.

Table 1: Mean taskwise certified output bounds normalized by `SelfComp`; smaller is better.

Dataset	Layers	Naive-P	ReVerT-P	Naive-F	ReVerT-F
CCPP	1	0.994	**0.816**	6.773	0.838
	3	0.997	**0.366**	3.242	0.566
	6	0.951	**0.202**	2.868	0.314
	12	0.904	**0.072**	2.372	0.272
Yelp	1	0.976	**0.553**	4.540	4.000
	3	0.938	**0.294**	3.530	2.830
	6	0.898	**0.0687**	3.000	2.270
	12	0.805	**0.00488**	0.973	0.519
SST	1	0.989	**0.678**	4.760	4.320
	3	0.948	**0.183**	3.180	2.610
	6	0.875	**0.257**	4.060	3.380
	12	0.787	**0.0426**	2.080	1.850

Answer to RQ1. `ReVerT-P` has the lowest reported normalized ratio in every dataset/depth setting in Table 1. On CCPP its reported ratio decreases from 0.816 at depth 1 to 0.072 at depth 12, while on Yelp and SST the 12-layer ratios are 0.00488 and 0.0426. Note that the finite-result task sets can differ between methods, these aggregates describe the completed comparisons rather than establishing a taskwise ordering across all methods.

By comparing between `ReVerT-P` and `Naive-P`, we find that `ReVerT-P` always achieves higher precision of the results; for Yelp with 6 or 12 layers, such outperformance is particularly evident. The comparison between `ReVerT-F` and `Naive-F` show similar trends. This clearly demonstrate the usefulness of our refined expression that preserves the dependencies over the cross-execution, and this finally results in a significant improvement of verification effectiveness.

Comparison against `Sampling`. To understand to which extent our approaches over-approximate the output bounds, we compare the three verifiers with a simple but dense `Sampling` method on the same 20 tasks of CCPP within the metric Ratio_m defined in Eq. 39. Each task defines $\mathcal{D}$ by allowing one selected feature to vary by at most 0.2 from its value in a reference input x_0, while fixing the remaining features. Both x and x' range over $\mathcal{D}$ and satisfy $\|x' - x\| \leq \delta_x$, with $\delta_x = 0.05$, as in Def. 1. `Sampling` searches progressively finer uniform grids for the largest output difference among feasible input pairs, aiming to precisely approximate the ground-truth output bounds. Its details are given in Appendix G.

Table 2: Mean taskwise certified output bounds normalized by `Sampling` on CCPP; smaller is better.

Dataset	Layers	SelfComp	ReVerT-F	ReVerT-P
CCPP	1	1.384	1.159	**1.129**
	3	7.481	4.234	**2.738**
	6	52.589	16.513	**10.623**
	12	154.014	40.632	**11.089**

On the common 20 tasks, all methods return valid bounds at depths 1, 3, 6 and 12. `ReVerT-P` has the smallest Ratio_m at every depth. In Table 2, `ReVerT-F` has a smaller reported Ratio_m than `SelfComp` at all four depths.

All reported mean ratios are greater than 1, indicating that the final certificate bounds returned by all over-approximation approaches are larger than those obtained by `Sampling`, which is expected. Moreover, when the number of layers is small, our approaches are very close the ground-truth bounds, which demonstrate the usefulness of our approaches. When the number of layers grows, the gap is indeed enlarged, but this may not simply arise from the over-approximation error, but also the under-approximation error of `Sampling`. By the results, while the bounds by `SelfComp` are pretty imprecise, the precision of our approaches is acceptable, especially `ReVerT-P`.

RQ2: Scalability. We measure runtime and the number of verification jobs returning usable bounds. Fig. 2 shows aggregate CCPP verification time over the processed tasks as model depth increases. Fig. 3 report completed-task counts and per-task runtimes on the three datasets.

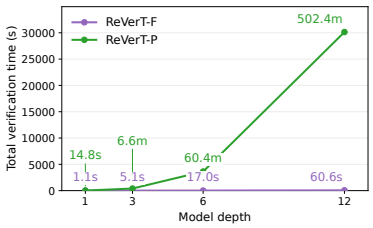


Figure 2: CCPP time cost vs. depth.

Answer to RQ2. The results show the expected precision-efficiency tradeoff. `ReVerT-F` avoids a final expensive LP solving and therefore serves as the scalable instantiation of our framework. Its total CCPP verification time stays around one minute even for the 12-layer model. `ReVerT-P` retains more relational constraints and sends them to the solver, so it is much slower: the total time grows from

seconds or minutes for shallow models to more than 500 minutes for the 12-layer model. This cost allows ReVerT-P to combine constraints across layers and obtain tighter bounds observed in RQ1.

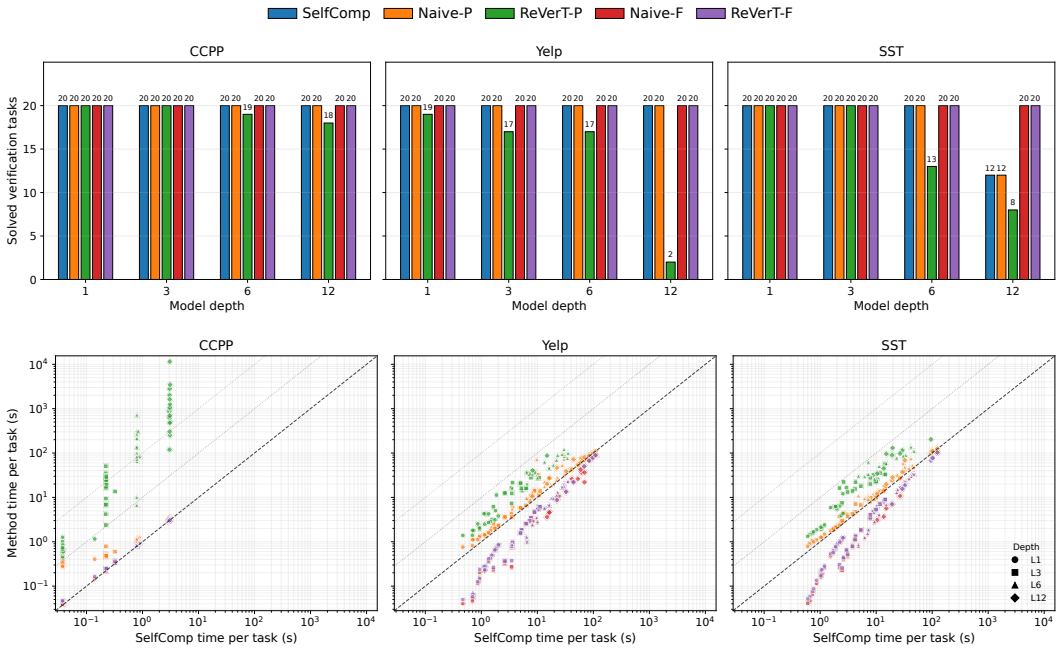


Figure 3: Completed verification tasks and runtime. Top: Completed-task verified counts for the settings in Table 1. Bottom: per-task runtime against SelfComp on a log-log scale; the diagonal indicates equal runtime with SelfComp.

In Fig. 3, the top part shows how many jobs return usable bounds, and the bottom compares each method's runtime with SelfComp on matched tasks. These completion counts measure whether a verification run finishes with a usable result; certification under an output threshold is evaluated separately in RQ3.

RQ3: Verification Success. To answer RQ3, we check whether tighter bounds translate into additional certified tasks under a fixed output tolerance. In a representative CCPP 6-layer setting with $\delta_x = 0.04$, task 2 has bound 0.6899 under ReVerT-F and 0.6497 under ReVerT-P. Therefore, at $\epsilon_y = 0.65$, the Precise configuration certifies all $5/5$ tasks while ReVerT-F certifies $4/5$, with about a $158\times$ runtime cost.

Table 3: RQ3: verification success and runtime on CCPP with $\delta_x = 0.04$. A task is solved when its certified bound is at most ϵ_y. Fast and Precise denote ReVerT-F and ReVerT-P.

Layers	ϵ_y	# solved tasks		Time cost (s)		Slowdown
		ReVerT-F	ReVerT-P	ReVerT-F	ReVerT-P	
6	0.65	4/5	5/5	4.31	680.30	$\approx 158\times$
6	0.66	4/5	5/5	4.31	680.30	$\approx 158\times$
12	0.50	3/5	3/5	15.45	997.24	$\approx 65\times$
12	0.55	4/5	4/5	15.45	997.24	$\approx 65\times$
12	0.60	5/5	5/5	15.45	997.24	$\approx 65\times$

Table 3 gives the results at different output tolerances. The Precise configuration uses the LP-tightened bound when available and the Fast bound otherwise. It certifies all five 6-layer tasks at $\epsilon_y = 0.65$ or 0.66, while Fast certifies four. The 12-layer setting does not separate the certified counts at the tested tolerances, but still illustrates the runtime cost of attempting LP tightening.

6 Conclusion

Input-relational verification for Transformers requires reasoning about relationships between two executions inside attention. We presented affine relational difference propagation, which maintains a dedicated affine difference abstraction from input to output. The method propagates differences exactly through affine components and uses attention-specific difference formulas for scores, softmax probabilities, and context matrices. Its fast and precise instantiations improve certified bounds and verification success over self-composition and generic difference-propagation variants.

AI USE STATEMENT

We used generative AI tools solely to assist with language polishing, translation, and LaTeX formatting during manuscript preparation. These tools were not used for research ideation, theoretical derivations or proofs, method implementation, experimental design, or interpretation of results. The authors reviewed and revised all AI-assisted content to ensure accuracy and consistency with the intended technical meaning. The authors take full responsibility for the final content of this paper, including all claims, proofs, and experimental results.

ETHICS STATEMENT

To the best of our knowledge, this work raises no specific ethical concerns.

This work develops formal verification methods for Transformer models and evaluates them on the CCPP, Yelp, and SST benchmarks. The intended benefit is to support reliable assessment of model behavior under explicitly specified input constraints. The resulting certificates apply only to the bounded input domains, input-pair constraints, output objectives, and tolerances considered in this work, as discussed in Appendix I.

REPRODUCIBILITY STATEMENT

The main text describes the verification methods, with additional derivations in Appendices C, D, and E. Appendix H states the soundness result and provides a proof sketch. Appendix G describes the benchmarks, model configurations, evaluation metrics, and sampling procedure. An anonymous code repository is linked in Section 5.

REFERENCES

Debangshu Banerjee and Gagandeep Singh. Relational dnn verification with cross executional bound refinement. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Debangshu Banerjee, Changming Xu, and Gagandeep Singh. Input-relational verification of deep neural networks. *Proceedings of the ACM on Programming Languages*, 8(PLDI):1–27, 2024.

Gilles Barthe, Pedro R D'argenio, and Tamara Rezk. Secure information flow by self-composition. In *Proceedings. 17th IEEE Computer Security Foundations Workshop, 2004.*, pp. 100–114. IEEE, 2004.

Gregory Bonaert, Dimitar I Dimitrov, Maximilian Baader, and Martin Vechev. Fast and precise certification of transformers. In *Proceedings of the 42nd ACM SIGPLAN international conference on programming language design and implementation*, pp. 466–481, 2021.

Rudy Bunel, Jingyue Lu, Ilker Turkaslan, Philip HS Torr, Pushmeet Kohli, and M Pawan Kumar. Branch and bound for piecewise linear neural network verification. *Journal of Machine Learning Research*, 21(42):1–39, 2020.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

Ruediger Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *Automated Technology for Verification and Analysis: 15th International Symposium, ATVA 2017, Pune, India, October 3–6, 2017, Proceedings 15*, pp. 269–286. Springer, 2017.

Pei Huang, Dennis Wei, Omri Isac, Haoze Wu, Min Wu, and Clark Barrett. Parameterized abstract interpretation for transformer verification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 35500–35508, 2026.

Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I 30*, pp. 97–117. Springer, 2017.

Konstantin Kaulen, Tobias Ladner, Stanley Bak, Christopher Brix, Hai Duong, Thomas Flinkow, Taylor T Johnson, Lukas Koller, Edoardo Manino, ThanhVu H Nguyen, et al. The 6th international verification of neural networks competition (vnn-comp 2025): Summary and results. *arXiv preprint arXiv:2512.19007*, 2025.

Brian Hsuan-Cheng Liao, Chih-Hong Cheng, Hasan Esen, and Alois Knoll. Are transformers more robust? towards exact robustness verification for transformers. In *International Conference on Computer Safety, Reliability, and Security*, pp. 89–103. Springer, 2023.

Hengjie Liu, Zhenya Zhang, and Jianjun Zhao. Precise verification of transformers through relu-catalyzed abstraction refinement. In *International Conference on Computer Aided Verification*, pp. 527–550. Springer, 2026.

Zhouxing Shi, Huan Zhang, Kai-Wei Chang, Minlie Huang, and Cho-Jui Hsieh. Robustness verification for transformers. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=BJxwPJHFwS`.

Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–30, 2019.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pp. 1631–1642, 2013.

Tarun Suresh, Debangshu Banerjee, and Gagandeep Singh. Relational verification leaps forward with rabbit. *Advances in Neural Information Processing Systems*, 37:123328–123352, 2024.

Pinar Tufekci. Prediction of full load electrical power output of a base load operated combined cycle power plant using machine learning methods. *International Journal of Electrical Power & Energy Systems*, 60:126–140, 2014.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Zhilu Wang, Chao Huang, and Qi Zhu. Efficient global robustness certification of neural networks via interleaving twin-network encoding. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1087–1092. IEEE, 2022.

Dennis Wei, Haoze Wu, Min Wu, Pin-Yu Chen, Clark Barrett, and Eitan Farchi. Convex bounds on the softmax function with applications to robustness verification. In *International Conference on Artificial Intelligence and Statistics*, pp. 6853–6878. PMLR, 2023.

Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. *Advances in neural information processing systems*, 31, 2018.

Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28, 2015.

Yunruo Zhang, Lujia Shen, Shanqing Guo, and Shouling Ji. Galileo: general linear relaxation framework for tightening robustness certification of transformers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 21797–21805, 2024.

A LOCAL ROBUSTNESS VERIFICATION FOR TRANSFORMERS

This appendix describes the affine bounds and linear constraints underlying the abstract verifier $\hat{T}$ introduced in §2.2. Projecting these constraints onto the output variables gives the over-approximated reachable region $\hat{\Omega}$. A single-execution Transformer verifier fixes a reference input x_0 and an input region around it. It then constructs a sound over-approximation of all possible outputs for inputs in that region. Let $\mathcal{X}(x_0)$ denote the local input set, which can represent, e.g., an ϵ-ball around x_0. Representative local verifiers propagate affine bounds throughout the Transformer (Shi et al., 2020). For every scalar intermediate value $z_j^\ell(x)$ at layer ℓ, such a verifier maintains bounds of the following form:

$$\underline{z}_j^\ell(x) = a_j^{L,\ell} x + b_j^{L,\ell} \le z_j^\ell(x) \le a_j^{U,\ell} x + b_j^{U,\ell} = \overline{z}_j^\ell(x), \qquad \forall x \in \mathcal{X}(x_0). \tag{11}$$

In Eq. 11, $z_j^\ell(x)$ is the concrete value of the j-th scalar variable in layer ℓ when the input is x. We write affine forms over x by viewing the input tensor as a vector. The coefficients $a_j^{L,\ell}, b_j^{L,\ell}$ define a lower affine bound, and $a_j^{U,\ell}, b_j^{U,\ell}$ define an upper affine bound. Soundness means that the true concrete value lies between these two affine functions for every x in the local input set. This is the layerwise bound-propagation view: once affine bounds are available for one layer, the verifier derives affine bounds for the next layer.

Self-attention is the main difficulty because its score and context computations contain bilinear products, while softmax introduces normalized dependencies. Shi et al. (Shi et al., 2020) provide a representative single-execution linear-relaxation treatment of these operations. For a scalar score term $Q_{ir}K_{jr}$ with $Q_{ir} \in [\underline{Q}_{ir}, \overline{Q}_{ir}]$ and $K_{jr} \in [\underline{K}_{jr}, \overline{K}_{jr}]$, the verifier applies the affine product bounds in Eq. 14. Summing them gives bounds on the score entry S_{ij}:

$$\begin{aligned} S_{ij} &\ge \frac{1}{\sqrt{d_h}} \sum_{r=1}^{d_h} \left(\underline{K}_{jr} Q_{ir} + \underline{Q}_{ir} K_{jr} - \underline{Q}_{ir} \underline{K}_{jr} \right), \\ S_{ij} &\le \frac{1}{\sqrt{d_h}} \sum_{r=1}^{d_h} \left(\overline{K}_{jr} Q_{ir} + \underline{Q}_{ir} K_{jr} - \underline{Q}_{ir} \overline{K}_{jr} \right). \end{aligned} \tag{12}$$

Here, S_{ij} is the input of the subsequent operation of softmax in Eq. 4. The softmax is typically decomposed into exponentials, a reciprocal of the row sum, and products. The context computation PV is treated as another product-and-sum relaxation.

After the last layer, the same propagation gives affine output bounds

$$\underline{y}_k(x) = a_k^{L,\text{out}} x + b_k^{L,\text{out}} \le T_k(x) \le a_k^{U,\text{out}} x + b_k^{U,\text{out}} = \overline{y}_k(x), \qquad k \in \{1, \dots, c\} \tag{13}$$

Here $T_k(x)$ is the k-th output coordinate, and $\underline{y}_k, \overline{y}_k$ are the certified lower and upper affine output bounds. The following subsection details the attention abstraction; further details of the local verifier are given in Shi et al. (2020). These single-execution abstractions can be combined through self-composition, as described in the main text. They do not themselves propagate the hidden cross-execution differences introduced in §4.1.

A.1 INDIVIDUAL ABSTRACTION OF ATTENTION IN SHI ET AL. (2020)

This appendix summarizes the single-execution attention abstraction used by the local Transformer verifier of Shi et al. (Shi et al., 2020). It is the building block denoted by $\mathcal{C}_{\text{indiv}}(x, y)$ in the constraint formulation of self-composition in Appendix F. The important point is that this abstraction bounds one execution of the Transformer at a time. It produces sound affine bounds for Q, K, V, S, P, and C, but it does not introduce variables for the differences between two related executions.

Product relaxation. Self-attention contains products between two scalar variables. Let $a \in [\underline{a}, \overline{a}]$ and $b \in [\underline{b}, \overline{b}]$ be concrete interval bounds obtained by the local verifier. Shi et al. use closed-form affine planes for the product ab:

$$\underline{b}a + \underline{a}b - \underline{a}\underline{b} \le ab \le \overline{b}a + \underline{a}b - \underline{a}\overline{b}. \tag{14}$$

In Eq. 14, the endpoints $\underline{a}, \overline{a}, \underline{b}, \overline{b}$ are scalar interval bounds, while a and b are the variables being multiplied. The symmetric planes obtained by exchanging the roles of a and b are also sound; the above form is the representative one used in Shi et al.'s derivation.

Score bounds. For one attention head, the local verifier first bounds the affine projections

$$Q = HW_Q, \qquad K = HW_K, \qquad V = HW_V. \tag{15}$$

We fix one encoder layer and one attention head, write $H := H^{\ell-1}$ for the attention input, and omit layer and head indices on the projection matrices W_Q, W_K, W_V. Since Eq. 15 is affine, the verifier can obtain affine lower and upper forms for every scalar component of Q, K, and V using the usual linear-layer rule.

The score entry for query position i and key position j is

$$S_{ij} = \frac{1}{\sqrt{d_h}} \sum_{r=1}^{d_h} Q_{ir} K_{jr}. \tag{16}$$

In Eq. 16, d_h is the head dimension. Each scalar product $Q_{ir}K_{jr}$ is relaxed by Eq. 14. Summing the resulting affine lower planes and upper planes gives

$$\underline{S}_{ij}(x) \le S_{ij}(x) \le \overline{S}_{ij}(x), \tag{17}$$

where $\underline{S}_{ij}(x)$ and $\overline{S}_{ij}(x)$ are affine functions of the local input. Eq. 17 is an individual score bound: it tells us where one execution's score can lie, not how two executions' scores differ.

Softmax bounds. For a fixed row i, Shi et al. rewrite softmax using exponentials, a row normalizer, and a reciprocal:

$$E_{ij} = \exp(S_{ij}), \qquad Z_i = \sum_{k=1}^{L} E_{ik}, \qquad R_i = \frac{1}{Z_i}, \qquad P_{ij} = E_{ij} R_i. \tag{18}$$

In Eq. 18, E_{ij} is the unnormalized exponential score, Z_i is the sum of exponentials in row i, R_i is its reciprocal, and P_{ij} is the normalized attention probability.

The exponential is bounded by unary affine relaxations over the current score interval. If $S_{ij} \in [\underline{s}, \overline{s}]$, then the verifier constructs coefficients $\alpha^L_{\exp}, \beta^L_{\exp}, \alpha^U_{\exp}, \beta^U_{\exp}$ such that

$$\alpha^L_{\exp} S_{ij} + \beta^L_{\exp} \le E_{ij} \le \alpha^U_{\exp} S_{ij} + \beta^U_{\exp}. \tag{19}$$

Because $\exp(\cdot)$ is convex, the upper bound in Eq. 19 is the secant line through $(\underline{s}, \exp(\underline{s}))$ and $(\overline{s}, \exp(\overline{s}))$. The lower bound is a tangent line. Shi et al. choose the tangent point so that the lower affine bound remains positive, which is important because the normalizer is later inverted. Substituting the affine score bounds from Eq. 17 into Eq. 19 gives affine bounds on every E_{ij}.

The normalizer Z_i is a sum, so its affine lower and upper bounds are obtained by summing the corresponding bounds on the exponential terms. Since Z_i has a strictly positive lower bound, the reciprocal $R_i = 1/Z_i$ can be bounded over $Z_i \in [\underline{Z}_i, \overline{Z}_i]$:

$$\alpha^L_R Z_i + \beta^L_R \le R_i \le \alpha^U_R Z_i + \beta^U_R. \tag{20}$$

In Eq. 20, the function $1/z$ is convex on $z > 0$, so the lower bound is a tangent line and the upper bound is the secant line through the two interval endpoints. Finally, $P_{ij} = E_{ij} R_i$ is another product, so Eq. 14 gives affine lower and upper bounds

$$\underline{P}_{ij}(x) \le P_{ij}(x) \le \overline{P}_{ij}(x). \tag{21}$$

Context bounds. The context vector is the probability-weighted sum of value vectors:

$$C_{ik} = \sum_{j=1}^{L} P_{ij} V_{jk}. \tag{22}$$

Here k indexes a value feature, while j ranges over key positions. For each scalar product $P_{ij}V_{jk}$, the verifier applies the same product relaxation in Eq. 14, using the current interval and affine bounds for P_{ij} and V_{jk}. Summing over j yields affine context bounds

$$\underline{C}_{ik}(x) \le C_{ik}(x) \le \overline{C}_{ik}(x). \tag{23}$$

Role in self-composition. The constraints summarized above form the attention part of $\mathcal{C}_{\text{indiv}}(x, y)$. In `SelfComp`, we instantiate one copy of these constraints for x and another copy for x', then add the input-pair constraints $\mathcal{C}_{\text{pair}}(x, x')$ defined in Appendix F. This is sound because each copy over-approximates its own Transformer execution. However, the two copies only communicate through the input-domain constraints and the final output objective. They do not propagate hidden relational quantities such as ΔS, ΔP, or ΔC. This is precisely the source of the precision loss addressed by the difference-propagation method in the main text.

B Difference Tracking in Global and Relational Verification

Local verification gives bounds for one execution at a time. To verify a global or input-relational property, however, the verifier must reason about two executions of the same network. A product-network or twin-network view is the natural starting point: one copy processes x, the other processes x', and both copies share weights. If the verifier only keeps the two individual abstractions from Eq. 11, then a hidden difference can only be recovered by subtracting independent ranges:

$$\Delta z_j^\ell = z_j'^\ell - z_j^\ell \in [\underline{z'}_j^\ell - \overline{z}_j^\ell,\ \overline{z'}_j^\ell - \underline{z}_j^\ell]. \tag{24}$$

In Eq. 24, z_j^ℓ and $z_j'^\ell$ are corresponding hidden variables in the two executions. The underlined and overlined quantities in this interval are numerical endpoints obtained over the input domain, rather than the input-dependent affine functions in Eq. 11. The interval is sound, but it can be very loose because the bounds on the two executions were constructed separately. This observation explains why simply lifting a local verifier to two copies is often not precise enough.

ITNE (Wang et al., 2022) and RaVeN/DiffPoly (Banerjee et al., 2024) address this issue by introducing and propagating hidden-layer difference variables. In their setting, the verifier tracks variables of the form

$$\Delta h^\ell = h'^\ell - h^\ell, \tag{25}$$

where h^ℓ and h'^ℓ are corresponding hidden states in the two executions. Here each lowercase h^ℓ is a row-vector representation; the uppercase H^ℓ in the main text collects token representations as matrix rows. Eq. 25 is more than notation: it says that the difference is treated as an internal variable of the abstract network, not merely as a final post-processing quantity.

For affine layers, difference propagation is exact. Using the row-vector convention of Eq. 6, if $h^{\ell+1} = h^\ell W + b$ and $h'^{\ell+1} = h'^\ell W + b$, then

$$\Delta h^{\ell+1} = \Delta h^\ell W. \tag{26}$$

Eq. 26 shows why hidden difference tracking can be much tighter than Eq. 24: the shared bias cancels and the same weight matrix acts directly on the previous-layer difference. For nonlinear activations, ITNE and RaVeN use dedicated relaxations of the relation between the pre-activation difference and the post-activation difference. For example, for an activation σ, one reasons about

$$\Delta r^\ell = \sigma(z^\ell + \Delta z^\ell) - \sigma(z^\ell), \tag{27}$$

where z^ℓ is the base pre-activation, Δz^ℓ is the pre-activation difference, and Δr^ℓ is the post-activation difference. This is the conceptual source of our method: relational verification should propagate differences internally, not compare two independently abstracted outputs at the end.

The remaining gap is Transformer attention. Existing difference-tracking methods mainly handle affine maps and standard activation functions. They do not provide difference transformers for the bilinear score product $QK^\top$, the normalized softmax probabilities, or the probability-value product PV. This paper contributes these missing attention-specific difference rules.

C Activation Difference Rules

The following rules apply coordinatewise to the intermediate quantities in §4.1; z and r denote scalar pre-activation and post-activation values.

For ReLU, we follow the distance-relation view used in relational verifiers. Let $r = \mathrm{ReLU}(z)$ and $r' = \mathrm{ReLU}(z')$, with $z' = z + \Delta z$. We propagate

$$\Delta r = r' - r = \mathrm{ReLU}(z + \Delta z) - \mathrm{ReLU}(z). \tag{28}$$

In Eq. 28, z and z' are the pre-activation values in the two executions, while r and r' are the corresponding post-ReLU values. The equation defines the post-activation difference directly as a function of the base pre-activation and the pre-activation difference. When the activation status of both executions is stable, this relation is exact. If both pre-activations are nonnegative, then $\Delta r = \Delta z$; if both are nonpositive, then $\Delta r = 0$.

In unstable cases, we use an affine distance relaxation. If $\Delta z \in [\underline{\Delta z}, \overline{\Delta z}]$, define

$$l = \min(0, \underline{\Delta z}), \qquad u = \max(0, \overline{\Delta z}).$$

The constants l and u are the lower and upper endpoints of an interval that safely contains both zero and the possible values of Δz. They determine the two linear planes used to relax the ReLU difference. Then

$$\frac{l(u - \Delta z)}{u - l} \leq \Delta r \leq \frac{u(\Delta z - l)}{u - l}. \tag{29}$$

Eq. 29 gives a lower affine bound and an upper affine bound for Δr. Both sides are affine in Δz, so the difference abstraction remains affine.

Tanh difference with parallel affine bounds. For tanh poolers, consider parallel affine lower and upper bounds with shared slope k over a common interval $[\underline{z}, \overline{z}]$ containing both pre-activations:

$$kt + b^L \leq \tanh(t) \leq kt + b^U, \qquad \forall t \in [\underline{z}, \overline{z}]. \tag{30}$$

Both executions use this same pair of bounds. Let $r = \tanh(z)$, $r' = \tanh(z')$, and $\Delta z = z' - z$. Subtracting the base upper bound from the prime lower bound, and the base lower bound from the prime upper bound, gives

$$k\Delta z - (b^U - b^L) \leq \Delta r = r' - r \leq k\Delta z + (b^U - b^L). \tag{31}$$

The intercept gap $b^U - b^L$ is retained because the difference bounds combine opposite sides of the two executions' envelopes. Both resulting bounds remain affine in Δz.

D LayerNorm Difference Rule

This appendix expands the no-variance LayerNorm setting discussed in §4.1 Shi et al. (2020); Bonaert et al. (2021); Zhang et al. (2024); Huang et al. (2026); Liu et al. (2026). Here h denotes one token's hidden vector, and the mean is taken across its hidden features. The normalization has the form

$$\mathrm{LN}_{\mathrm{nv}}(h) = \gamma \odot (h - \mathrm{mean}(h)) + \beta, \tag{32}$$

where γ and β are learned scale and shift parameters, $\odot$ denotes elementwise multiplication, and $\mathrm{mean}(h)$ is the mean over the normalized dimension. This abstraction is affine in h. Therefore, for two executions, the shift parameter β cancels and the difference is

$$\Delta\mathrm{LN}_{\mathrm{nv}} = \gamma \odot (\Delta h - \mathrm{mean}(\Delta h)). \tag{33}$$

Eq. 33 is exact under the no-variance abstraction.

E Softmax and Context Difference Rules

Following §4.1, we propagate the softmax difference explicitly. Fix one attention row and omit its row index; the subscripts i and j below index positions within this fixed row. Decompose

$$E_i = \exp(S_i), \qquad R = \frac{1}{\sum_j E_j}, \qquad P_i = E_i R. \tag{34}$$

In this decomposition, S_i is the i-th score entry in a fixed softmax row, E_i is its exponential numerator, R is the reciprocal of the sum of all exponentials in that row, and P_i is the resulting softmax probability. This decomposition turns softmax into elementary operations that can be relaxed and related across two executions.

The prime execution has E'_i, R', and P'_i. The product difference is

$$\Delta P_i = P'_i - P_i = \Delta E_i R' + E_i \Delta R, \tag{35}$$

where $\Delta E_i = E'_i - E_i$ and $\Delta R = R' - R$. Eq. 35 says that the probability difference can be propagated through the numerator difference ΔE_i and the reciprocal difference ΔR, rather than by subtracting two independent probability intervals. The two products in Eq. 35 are relaxed by applying the McCormick construction illustrated in Eq. 10 to their respective scalar factors.

As with tanh in Appendix C, we construct sound affine lower and upper bounds for the nonlinear difference relations before propagation. For ΔE_i, we form an affine relaxation of the exponential difference in terms of ΔS_i over the score ranges and substitute the propagated affine bounds on ΔS_i. Summing the resulting bounds gives affine bounds on the difference between the two row normalizers. We then apply an affine relaxation of the reciprocal difference over the strictly positive normalizer ranges and substitute the normalizer-difference bounds to obtain affine bounds on ΔR. All substitutions select lower or upper bounds according to the coefficient signs and retain all intercept terms.

Softmax also provides cheap relational equalities:

$$\sum_i P_i = 1, \qquad \sum_i P'_i = 1, \qquad \sum_i \Delta P_i = 0. \tag{36}$$

The first two equalities in Eq. 36 are the standard row-sum constraints for softmax probabilities in the base and prime executions. The third follows by subtracting them, because $\Delta P_i = P'_i - P_i$. It says that entries of a probability-difference row cannot vary independently.

For $C = PV$ and $C' = P'V'$, we use the same product-difference pattern:

$$\Delta C = C' - C = \Delta P V' + P \Delta V. \tag{37}$$

In Eq. 37, C and C' are the context matrices in the two executions. The term $\Delta P V'$ captures how changes in attention probabilities affect the context, while $P\Delta V$ captures how changes in value vectors affect the context. After expanding $\Delta P V'$ and $P\Delta V$ entrywise, each scalar product is relaxed using the same McCormick construction as in Eq. 10, with the corresponding factors and interval endpoints.

F Self-Composition Baseline

The direct local-to-global baseline is self-composition. We build two copies of the Transformer. The base and prime copies receive x and x', respectively. We then apply Shi et al.'s local Transformer verifier independently to the two copies. This construction is included as a comparison baseline; it is not part of our proposed attention-aware relational method.

Let $\mathcal{C}_{\text{indiv}}(x, y)$ encode the single-execution linear relaxation underlying $\hat{T}(x)$ in Eq. 5, together with the chosen scalar output objective. In the LP below, y and y' are relaxed scalar output variables: they range over the scalar values permitted by the abstract output sets $\hat{y} = \hat{T}(x)$ and $\hat{y'} = \hat{T}(x')$, respectively. We use $\Delta y = y' - y$ for their relaxed difference. Concrete executions satisfy these constraints, and their scalar output distance in Def. 1 equals $|y' - y|$. Let $\mathcal{C}_{\text{pair}}(x, x')$ encode the input conditions $x, x' \in \mathcal{D}$ and $\|x' - x\| \leq \delta_x$ from Def. 1. When these input conditions have a linear constraint representation and the scalar objective is linear in the network outputs, the self-composition baseline computes an upper bound on the absolute output difference by solving the following LP:

$$B_{\text{self}} = \max \Delta y \quad \text{s.t.} \quad \mathcal{C}_{\text{pair}}(x, x'), \mathcal{C}_{\text{indiv}}(x, y), \mathcal{C}_{\text{indiv}}(x', y'), \Delta y = y' - y. \tag{38}$$

In Eq. 38, the optimization is over all variables appearing in the constraints, including the auxiliary variables introduced by the linear relaxations. The two copies of $\mathcal{C}_{\text{indiv}}$ constrain the base and prime executions, $\mathcal{C}_{\text{pair}}$ relates the inputs, and the final equality explicitly defines the output difference. Since the pair domain and the two identical abstraction copies are invariant under exchanging the executions, feasible output differences occur in opposite pairs. Hence $\max \Delta y = \max |\Delta y|$ over this feasible set. For this scalar objective, B_{self} is the optimum of the relaxed problem and an upper bound on the true worst-case output distance under the input conditions in Def. 1.

This baseline is sound because both individual copies over-approximate their true executions, and the pair-domain constraints are included. Its limitation is precision. Although the final LP sees both copies, the two abstract executions are still constructed mostly independently. The pair relation is

imposed at the input and queried through the output objective, but the verifier does not explicitly track how the difference between the two executions evolves through hidden layers and attention. This motivates propagating differences inside the network, rather than relating the two executions only after each has already been locally abstracted.

G BENCHMARKS AND MODELS

Table 4: Compared verification and sampling methods.

Method	Meaning	Role
`Sampling`	maximum observed output difference over sampled feasible input pairs	empirical baseline
`SelfComp`	self-composition using Shi et al.'s local verifier plus final LP	main baseline
`Naive-F`	generic affine difference propagation without attention-aware formulas	secondary comparison
`Naive-P`	generic affine difference propagation plus LP tightening	secondary comparison
`ReVerT-F`	attention-aware affine difference propagation without final LP	main method
`ReVerT-P`	attention-aware difference propagation plus LP tightening	main method

We use CCPP (Tufekci, 2014) as a regression benchmark, and Yelp (Zhang et al., 2015) and SST (Socher et al., 2013) as classification benchmarks. CCPP is the Combined Cycle Power Plant dataset, where the task is to predict net electrical power output from operating conditions such as temperature, pressure, humidity, and exhaust vacuum. It is a natural regression benchmark for global robustness because nearby operating states should not lead to unstable output predictions.

For classification benchmarks, we follow the way relational verification work treats classification outputs. ITNE (Wang et al., 2022) evaluates global robustness on MNIST by bounding output variation for selected output coordinates, rather than directly optimizing over the discrete predicted label. RaVeN (Banerjee et al., 2024) evaluates relational classification properties on MNIST and CIFAR10 by encoding output predicates over logits, margins, or worst-case accuracy objectives. Following Def. 1, for Yelp and SST we verify the absolute difference of a selected logit coordinate between the two executions.

We evaluate Transformers with depths 1, 3, 6, and 12. The implementation uses encoder-only models and is built in PyTorch. Unless otherwise stated, all verification experiments are run on a single NVIDIA RTX 5080 GPU with 16GB of GPU memory. Reported runtimes are wall-clock verification times for completed jobs and include the cost of constructing the abstract constraints and solving the corresponding optimization problems. We do not use distributed computation. Full training hyperparameters, feature-token construction details, perturbation radii, software versions, timeouts, and CPU/RAM details will be reported with the final reproducibility package.

Sampling implementation and grid refinement. The comparison in Table 2 covers Transformer depths 1, 3, 6, and 12. At each depth, the same five held-out reference inputs selected with seed 0 and four feature choices per reference input yield 20 CCPP tasks. The models have hidden dimension 64, four attention heads, and intermediate dimension 128.

For each task, `Sampling` evaluates nested uniform grids with $2^q + 1$ points, using $q = 8, 10, 12, 14, 16, 18$. The finest grid contains $262{,}145$ points, spaced approximately 1.526×10^{-6} apart. Grid coordinates are stored as feasible float32 inputs; concrete forward evaluation and the reported verifier runs use float64 arithmetic. A sliding-window search exhaustively computes the maximum output difference over all feasible pairs on each finite grid. This maximum is an empirical lower bound on the continuous-domain worst-case output difference.

For the one-layer subset of 20 tasks, refining from $65{,}537$ to $262{,}145$ grid points increases B_{emp} by 1.12×10^{-7} on average and at most 1.52×10^{-6}; $12/20$ tasks are unchanged at the finest level. This

stabilization describes the sampled values at these resolutions. It does not bound their remaining distance to the continuous optimum, since the search is exhaustive only over feasible pairs on each finite grid.

In the RQ3 configuration of Table 3, `ReVerT-P` uses the LP-tightened bound when available and the `ReVerT-F` bound otherwise.

Metrics We report normalized output-difference bounds, runtime, and verification success. Both RQ1 tables use the same metric: the arithmetic mean of taskwise method-to-reference bound ratios. For a fixed dataset and model depth, each verification task specifies an input domain, an input-pair constraint, and an output objective. For method m, let n_m be the number of valid comparison tasks, indexed by $\tau = 1, \ldots, n_m$. A comparison is valid when the method and reference values are both finite and the reference value is positive. Let $B_m(\tau)$ and $B_{\mathrm{ref}}(\tau)$ denote the method and reference values for the same task τ. We report

$$\mathrm{Ratio}_m = \frac{1}{n_m} \sum_{\tau=1}^{n_m} \frac{B_m(\tau)}{B_{\mathrm{ref}}(\tau)}. \tag{39}$$

Table 1 uses `SelfComp` as the reference. Table 2 uses `Sampling`. For each task τ, `Sampling` evaluates a finite grid in $\mathcal{D}$ and reports $B_{\mathrm{emp}}(\tau)$, the largest output difference $\mathsf{dist}(\Delta y)$ among sampled input pairs. Since these pairs form a subset of $\mathcal{P}$, this value is an empirical lower bound on the true worst-case output difference. For any verification method m with a sound upper bound,

$$B_{\mathrm{emp}}(\tau) \le B^\star(\tau) \le B_m(\tau). \tag{40}$$

Smaller ratios indicate tighter verifier bounds under the same reference. A ratio below 1 in Table 1 improves on `SelfComp`; a ratio closer to 1 from above in Table 2 narrows the observed lower–upper bracket. The valid comparison tasks and their count n_m can differ between methods; no ratio is reported when $n_m = 0$. A sampled violation is a concrete counterexample, while the absence of one does not certify the continuous-domain property.

For verification success, we use the output tolerance ϵ_y from Def. 1. Let n be the total number of evaluated tasks, indexed by $\tau = 1, \ldots, n$. The indicator $\mathbb{I}[B_m(\tau) \le \epsilon_y]$ equals 1 when method m returns a sound bound satisfying this inequality, and 0 otherwise, including when no bound is returned. The success rate is

$$\mathrm{Succ}_m(\epsilon_y) = \frac{1}{n} \sum_{\tau=1}^{n} \mathbb{I}[B_m(\tau) \le \epsilon_y]\,. \tag{41}$$

Timeouts and failed runs count as uncertified tasks and remain in the denominator.

H SOUNDNESS

Theorem 1 (Soundness)**.** *For a scalar output objective considered here, let B be the bound returned by `ReVerT-F` or `ReVerT-P` for the input conditions and output distance in Def. 1. Then, for all $x, x' \in \mathcal{D}$,*

$$\|x' - x\| \le \delta_x \quad \Longrightarrow \quad \mathsf{dist}(T(x), T(x')) \le B.$$

Proof sketch. The proof is by induction over the Transformer computation graph. The input abstraction contains all input pairs satisfying the conditions in Def. 1 by construction. For affine maps, the difference rule is exact because shared affine biases cancel and shared weights act directly on the previous difference. For activation functions, the relaxation encloses every concrete pair of base and prime activations over the current abstract domain. For attention scores, Eq. 8 is an exact algebraic identity before relaxing its bilinear terms, and McCormick envelopes soundly over-approximate each introduced product over its current interval bounds. The softmax and context rules in Appendix E use the same product-difference argument, together with sound scalar relaxations for exponential and reciprocal terms. Thus every concrete pair of executions remains inside the maintained base, prime, and difference abstractions after each operation. For the scalar objectives considered here, the absolute output difference equals the output distance in Def. 1. Evaluating the propagated affine

bounds in ReVerT-F, or bounding the same objective over the retained constraints in ReVerT-P, therefore gives an upper bound B on that distance for every admissible input pair. Consequently, $B \leq \epsilon_y$ certifies the property; a larger upper bound alone does not establish a violating input pair.

I LIMITATIONS

The guarantees in this paper apply only to input pairs in the bounded domain $\mathcal{D}$ that satisfy the input-distance constraint in Def. 1. They should not be interpreted as robustness over the entire natural input space. A certificate applies to these input conditions, the selected output distance, and the tolerance ϵ_y.

The dense-grid experiment provides feasible witnesses and an empirical lower bound only. Without a continuous-domain discretization argument, failure to find a violating grid pair is not a robustness certificate; the formal guarantee continues to come from the verifier's upper bound.

The main technical tradeoff is precision versus scalability. ReVerT-F is lightweight but may lose dependencies when affine bounds are propagated directly through softmax and attention products. ReVerT-P retains more relational constraints and can yield tighter certified bounds, but its LP formulation can be expensive. Some verification jobs may fail because of solver difficulty, timeout, or out-of-memory errors, especially as depth and constraint count increase. As stated in the experimental metrics, failed jobs are not included in the matched bound-ratio table.

Finally, the experimental evaluation covers one regression benchmark and two classification benchmarks. These datasets are useful for isolating the verification behavior of the proposed method, but they do not cover all domains where Transformer robustness is important. Future work should evaluate broader architectures, larger models, additional relational specifications, and more realistic discrete perturbation models for language tasks.