The Credibility Transformer

Ronald Richman* Salvatore Scognamiglio † Mario V. Wüthrich‡

Version of September 26, 2024

Abstract

Inspired by the large success of Transformers in Large Language Models, these architectures are increasingly applied to tabular data. This is achieved by embedding tabular data into low-dimensional Euclidean spaces resulting in similar structures as time-series data. We introduce a novel credibility mechanism to this Transformer architecture. This credibility mechanism is based on a special token that should be seen as an encoder that consists of a credibility weighted average of prior information and observation based information. We demonstrate that this novel credibility mechanism is very beneficial to stabilize training, and our Credibility Transformer leads to predictive models that are superior to state-of-the-art deep learning models.

Keywords. Transformer, credibility, tabular data, feature-engineering, entity embedding.

1 Introduction

Feed-forward neural networks (FNNs) provide state-of-the-art deep learning regression models for actuarial pricing. FNNs can be seen as extensions of generalized linear models (GLMs), taking covariates as inputs to these FNNs, feature-engineering these covariates through several hidden FNN layers, and then using these feature-engineered covariates as inputs to a GLM. Advantages of FNNs over classical GLMs are that they are able to find functional forms and interactions in the covariates that cannot easily be captured by GLMs, and which typically require the modeler to have specific deeper insights into the data generation process. Since these specific deeper insights are not always readily available, FNNs may support the modeler in finding such structure and insight.

Taking inspiration from the recent huge success of large language models (LLMs), the natural question arises whether there are network architectures other than FNNs that share more similarity with LLMs and which can further improve predictive performance of neural networks in actuarial pricing. LLMs are based on the Transformer architecture which has been invented by Vaswani et al. [31]. The Transformer architecture is based on attention layers which are special network modules that allow covariate components to communicate with each other. Specifically, one may think of each covariate component receiving a so-called query and a key, and the attention mechanism tries to find queries and keys of different covariate components that match

*Old Mutual Insure and University of the Witwatersrand, ronald.richman@ominsure.co.za

†Department of Management and Quantitative Sciences, University of Naples "Parthenope", salvatore.scognamiglio@uniparthenope.it

‡RiskLab, Department of Mathematics, ETH Zurich, mario.wuethrich@math.ethz.ch

in order to send forward a positive or negative signal. For example, in the case of pricing motor insurance for car drivers, young drivers may have a key labeled 'risky', and car brands may have queries, for instance, sports cars trying to find the age group of risky drivers. Having a match of a query and a key then leads to an increase of the expected claim frequency, which accounts for a corresponding interaction of these two covariates in the regression function. This idea of keys, queries and values, which is central to Transformer models, originates in the information retrieval literature, where a search term (the query) is matched to relevant documents (the keys), the contents of which provide the information the user is looking for (the values).

Transformer architectures have been introduced for natural language data (i.e., time-series data); see Vaswani et al. [31]. A main question is whether there is a similar beneficial way to use Transformers and attention layers on tabular input data, which lack the same time-series structure. The main tool for applying Transformers to tabular data is the so-called entity embedding mechanism. Embedding all covariates into a low-dimensional Euclidean space, one receives embedded tabular covariates that share similar features as time-series data. The first attempt at building this type of model is found in Huang et al. [19]. These authors proposed a model embedding only categorical components of tabular data, applying a Transformer to these, and then concatenating the Transformer outputs with the numerical input variables for further processing this input information by a FNN. This TabTransformer model of Huang et al. [19] has been applied by Kuo–Richman [21], who found that this model provides a small advantage over FNN architectures for predicting severity of flood claims. A more comprehensive approach has recently been proposed by Gorshniy et al. [14], the feature tokenizer (FT) Transformer architecture, which embeds numerical components of tabular data as well. This proposal has been considered in the actuarial literature by Brauer [2]. Thus, technically there is no difficulty in using FT-Transformers on tabular data. However, in view of the numerical results obtained in Brauer [2], there does not seem to be any specific benefit in using a FT-Transformer architecture for tabular data in terms of predictive performance. This is precisely the starting point of this research. Namely, we are going to modify the FT-Transformer architecture by a novel credibility mechanism. Using this additional credibility mechanism we find Transformer-based network architectures that outperform the networks that have been exploited so far on a specific motor insurance dataset. This gives clear evidence that Transformers can also be very beneficial on tabular input data if network architectures are designed in a sophisticated manner.

Our main idea is to add a special token to the classical Transformer architecture, inspired by the Bidirectional Encoder Representations from Transformers (BERT) architecture of Devlin et al. [8]. This special token (called the CLS token by Devlin et al. [8]) is used to encode the covariate information gained from the attention mechanism in a lower dimensional representation. Through the training process described later, one is able think of this token as playing the role of prior information consisting of the portfolio mean in a Bayesian credibility sense. This prior information-based predictor is then combined in a linear credibility fashion (in the abstract embedding space of the model) with a second predictor that plays the role of observed information in Bayesian credibility theory. Specifically, we consider a credibility-based average of the two sets of information, and this provides us with a similar credibility structure as, e.g., the classical linear credibility formula of Bühlmann–Straub [4]. This motivates the use of the term *Credibility Transformer* for our proposal. In other words, the Credibility Transformer does not only try to find matches between keys and queries, but it also weighs this information

by a credibility mechanism. As noted above, in this work, we make two main connections to the linear credibility formula: first, when training the model, we force the special CLS token to provide prior information to the model using a credibility factor, and second, we interpret the attention mechanism within the Transformer as a linear credibility formula between prior information contained in the CLS token and the rest of the covariate information.
Moreover, we exploit special fitting strategies of these Credibility Transformers since training Transformer architectures needs a sort of tempered learning to not get trapped in too early stopping decisions. For this we adapt the `NormFormer` proposal of Shleifer et al. [30], which applies a special Transformer pre-training that can cope with different gradient magnitudes in stochastic gradient descent training. We verify that this proposal of Shleifer et al. [30] is also beneficial in our Credibility Transformer architecture resulting in superior performance compared to plain-vanilla trained architectures. Building on this initial exploration, we then augment the Credibility Transformer with several advances from the LLM and deep learning literature, producing a deep and multi-head attention version of the initial Credibility Transformer architecture. Furthermore, we implement the concept of Gated Linear Units (GLU) of Dauphin et al. [6] to select more important covariate components, and we exploit the Piecewise Linear Encoding (PLE) of Gorshniy et al. [14] which should be thought of as a more informative embedding, especially adapted for numerical covariates, than its one-hot encoded counterpart. It is more informative because PLE preserves the ordinal relation of continuous covariates, while enabling subsequent network layers to produce a multidimensional embedding for the numerical data. This *improved deep Credibility Transformer* equips us with regression models that have an excellent predictive performance. Remarkably, we show that the Credibility Transformer approach can improve a state-of-the-art deep Transformer model applied to a non-life pricing problem. Finally, we examine the explainability of the Credibility Transformer approach by exploring a fitted model.

Organization of this manuscript. This paper is organized as follows. Section 2 describes the architecture of the Credibility Transformer in detail, including the input tokenization process, the Transformer layer with its attention mechanism, and our novel credibility mechanism. Section 3 presents a real data example using the French motor third party liability (MTPL) claims counts dataset, demonstrating the implementation and performance of the Credibility Transformer. Section 4 explores improvements to the Credibility Transformer, incorporating insights from LLMs and recent advances in deep learning. Section 5 discusses the insights that can be gained from a fitted Credibility Transformer model. Finally, Section 6 concludes by summarizing our contribution and findings and discusses potential future research directions.

2 The Credibility Transformer

This section describes the architecture of the Credibility Transformer. The reader should have the classical Transformer architecture of Vaswani et al. [31] in mind, with one essential difference, namely, the entire set of relevant covariate information is going to be encoded into additional classify (CLS) tokens, and these CLS tokens are going to be combined with a credibility mechanism. For this we extend the classical Transformer architecture of Vaswani et al. [31] by CLS tokens. CLS tokens have been introduced by Devlin et al. [8] in the language model Bidirectional Encoder Representations from Transformers (BERT). In BERT, these tokens are used to

summarize the probability of one sentence following an other one, therefore they have used the term 'classify token'. In our architecture, this terminology may be a bit misleading because our CLS tokens will just be real numbers not relating to any classification probabilities, but they will rather present biases in regression models. Nevertheless, we keep the term CLS token to emphasize its structural analogy to BERT.

2.1 Construction of the input tensor

We first describe the input pre-processing. This input pre-processing tokenizes all input variables (features, covariates) by embedding them into a low dimensional Euclidean space. These embeddings are complemented by positional embeddings and CLS tokens. We describe this in detail in the next three subsections. The main difference to the classical Transformer of Vaswani et al. [31] concerns the last step of adding the CLS tokens for encoding the information from the attention mechanism, this feature is described in Section 2.1.3, below.

2.1.1 Feature tokenizer

We start by describing the tokenization of the covariates (input features). Assume we have T_1 categorical covariates $(x_t)_{t=1}^{T_1}$ and T_2 continuous covariates $(x_t)_{t=T_1+1}^{T} \in \mathbb{R}^{T_2}$; we set $T = T_1 + T_2$ for the total number of covariate components in input $\boldsymbol{x} = (x_t)_{t=1}^{T}$.
We bring these categorical and continuous input variables into the same structure by applying entity embedding $(\boldsymbol{e}_t^{\mathrm{EE}})_{t=1}^{T_1}$ to the categorical variables and FNN embedding $(\boldsymbol{e}_t^{\mathrm{FNN}})_{t=T_1+1}^{T}$ to the continuous variables; this pre-processing step is called *feature tokenizer*. Entity embedding has been introduced in the context of natural language processing (NLP) by Brébisson et al. [3] and Guo–Berkhahn [16], and it has been introduced to the actuarial community by Richman [24, 25] and Delong–Kozak [7]. The main purpose of entity embedding is to represent (high-cardinality) nominal features by low dimensional Euclidean embeddings such that proximity in these low dimensional Euclidean space means similarity w.r.t. the prediction task at hand. Additionally, we also embed the continuous input variables into the same low dimensional Euclidean space such that all input variables have the same structure. This approach has been considered in Gorshniy et al. [14], and it is used by Brauer [2] in the actuarial literature.
The embeddings are defined as follows. For the entity embeddings of the categorical covariates we select mappings

$$\boldsymbol{e}_t^{\mathrm{EE}} : \{a_1, \ldots, a_{n_t}\} \to \mathbb{R}^b, \qquad x_t \mapsto \boldsymbol{e}_t^{\mathrm{EE}}(x_t),$$

where $a_1, \ldots, a_{n_t}$ are the different levels of the t-th categorical covariate x_t, and $b \in \mathbb{N}$ is the fixed, pre-chosen embedding dimension; this embedding dimension b is a hyperparameter that needs to be selected by the modeler. Importantly, for our Credibility Transformer all embeddings must have the same embedding dimension b (possibly enlarged, as we do later, by concatenating other vectors to these embeddings); this embedding dimension b is called the *model dimension* of the Transformer model.
Each of these categorical entity embeddings involves bn_t embedding weights (parameters), thus, we have $\sum_{t=1}^{T_1} bn_t$ parameters in total from the embedding of the categorical covariates $(x_t)_{t=1}^{T_1}$.
For the continuous input variables we select fully-connected FNN architectures $\boldsymbol{z}_t^{(2:1)} = \boldsymbol{z}_t^{(2)} \circ \boldsymbol{z}_t^{(1)}$ of depth 2 being composed of two FNN layers

$$\boldsymbol{z}_t^{(1)} : \mathbb{R} \to \mathbb{R}^b \qquad \text{and} \qquad \boldsymbol{z}_t^{(2)} : \mathbb{R}^b \to \mathbb{R}^b. \tag{2.1}$$

We select such a FNN architecture $\boldsymbol{z}_t^{(2:1)}$ for each continuous covariate component x_t, $T_1 + 1 \leq t \leq T$. Each of these continuous FNN embeddings $\boldsymbol{z}_t^{(2:1)}$ involves $2b + b(b+1)$ network weights (including the bias parameters). This gives us $T_2(2b+b(b+1))$ parameters for the embeddings of the continuous covariates $(x_t)_{t=T_1+1}^T$; for more technical details and the notation used for FNNs we refer to Wüthrich–Merz [33, Chapter 7].
Concatenating and reshaping these categorical and continuous covariate embeddings equips us with the *raw input tensor*

$$\boldsymbol{x} = (x_t)_{t=1}^T \;\mapsto\; \boldsymbol{x}_{1:T}^{\circ} = \left[\boldsymbol{e}_1^{\mathrm{EE}}(x_1), \ldots, \boldsymbol{e}_{T_1}^{\mathrm{EE}}(x_{T_1}), \boldsymbol{z}_{T_1+1}^{(2:1)}(x_{T_1+1}), \ldots, \boldsymbol{z}_T^{(2:1)}(x_T)\right]^\top \;\in\; \mathbb{R}^{T\times b}.$$

This raw input tensor $\boldsymbol{x}_{1:T}^{\circ}$ involves the following number of parameters to be selected/fitted

$$\varrho^{\mathrm{input}} = b\sum_{t=1}^{T_1} n_t + T_2(2b + b(b+1)).$$

Having the input in this raw tensor structure $\boldsymbol{x}_{1:T}^{\circ} \in \mathbb{R}^{T\times b}$ allows us to employ attention layers and Transformers to further process this input data.

2.1.2 Positional encoding

Since attention layers do not have a natural notion of position and/or time in the tensor information, we complement the raw tensor $\boldsymbol{x}_{1:T}^{\circ}$ by a b-dimensional positional encoding; see Vaswani et al. [31] for positional encoding in the context of NLP. However, we choose a simpler learned positional encoding compared to the sine-cosine encoding used by Vaswani et al. [31], as this simpler version is sufficient for our purpose of learning on tabular data which do not have the natural ordering exploited in the original Transformer architecture.
The positional encoding can be achieved by another embedding that considers the positions $t \in \{1, \ldots, T\}$ in a b-dimensional representation

$$\boldsymbol{e}^{\mathrm{pos}} : \{1, \ldots, T\} \to \mathbb{R}^b, \qquad t \mapsto \boldsymbol{e}^{\mathrm{pos}}(t).$$

This positional encoding involves another $\varrho^{\mathrm{position}} = Tb$ parameters. We concatenate this positional encoding with the raw input tensor providing us with the *input tensor*

$$\begin{aligned}
\boldsymbol{x}_{1:T} &= \begin{bmatrix} \boldsymbol{x}_{1:T}^{\circ} & \begin{pmatrix} \boldsymbol{e}^{\mathrm{pos}}(1)^\top \\ \vdots \\ \boldsymbol{e}^{\mathrm{pos}}(T)^\top \end{pmatrix} \end{bmatrix} \\
&= \begin{bmatrix} \boldsymbol{e}_1^{\mathrm{EE}}(x_1) & \cdots & \boldsymbol{e}_{T_1}^{\mathrm{EE}}(x_{T_1}) & \boldsymbol{z}_{T_1+1}^{(2:1)}(x_{T_1+1}) & \cdots & \boldsymbol{z}_T^{(2:1)}(x_T) \\ \boldsymbol{e}^{\mathrm{pos}}(1) & \cdots & \boldsymbol{e}^{\mathrm{pos}}(T_1) & \boldsymbol{e}^{\mathrm{pos}}(T_1+1) & \cdots & \boldsymbol{e}^{\mathrm{pos}}(T) \end{bmatrix}^\top \;\in\; \mathbb{R}^{T\times 2b}.
\end{aligned}$$

We emphasize that all components of this input tensor $\boldsymbol{x}_{1:T}$ originate from a specific input being either a covariate x_t or the position t of that covariate, i.e., it contains information about the individual instances. In the next subsection, we are going to add an additional variable, the CLS token, to this input tensor, but this additional variable does not originate from a covariate variable with a specific meaning.

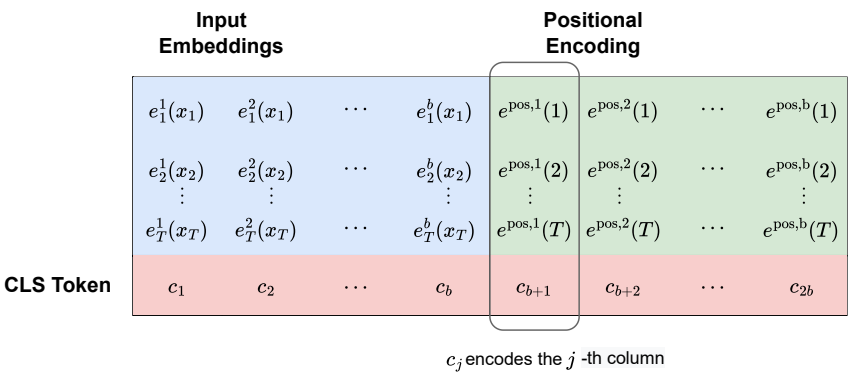

Figure 1: A diagram showing the augmented input tensor $\boldsymbol{x}^+_{1:T+1}$ consisting of input and positional encoding and the CLS token, moreover, it shows how the scalar components comprising the CLS token encode the columns of the input tensor $\boldsymbol{x}_{1:T}$.

2.1.3 Classify (CLS) token

We extend the input tensor $\boldsymbol{x}_{1:T}$ by an additional component, the CLS token. This is the step that differs from the classical Transformer proposal of Vaswani et al. [31]. The purpose of the CLS token is to encode every column $1 \leq j \leq 2b$ of the input tensor $\boldsymbol{x}_{1:T} \in \mathbb{R}^{T\times 2b}$ into a single variable. This then gives us the *augmented input tensor*

$$\begin{aligned}\boldsymbol{x}^+_{1:T+1} &= \begin{bmatrix}\boldsymbol{x}_{1:T}\\ \boldsymbol{c}^\top\end{bmatrix} \\ &= \begin{bmatrix}\boldsymbol{e}^{\text{EE}}_1(x_1) & \cdots & \boldsymbol{e}^{\text{EE}}_{T_1}(x_{T_1}) & \boldsymbol{z}^{(2:1)}_{T_1+1}(x_{T_1+1}) & \cdots & \boldsymbol{z}^{(2:1)}_{T}(x_{T}) & \boldsymbol{c}_1\\ \boldsymbol{e}^{\text{pos}}(1) & \cdots & \boldsymbol{e}^{\text{pos}}(T_1) & \boldsymbol{e}^{\text{pos}}(T_1+1) & \cdots & \boldsymbol{e}^{\text{pos}}(T) & \boldsymbol{c}_2\end{bmatrix}^\top \in \mathbb{R}^{(T+1)\times 2b}.\end{aligned} \tag{2.2}$$

where $\boldsymbol{c} = (\boldsymbol{c}_1^\top, \boldsymbol{c}_2^\top)^\top = (c_1, \ldots, c_{2b})^\top \in \mathbb{R}^{2b}$ denote the CLS tokens. Each of the scalars $c_j \in \mathbb{R}$ comprising the CLS tokens, $1 \leq j \leq 2b$, will encode one column of the input tensor $\boldsymbol{x}_{1:T} \in \mathbb{R}^{T\times 2b}$, i.e., it will provide a one-dimensional projection of the corresponding j-th T-dimensional vector to a scalar $c_j \in \mathbb{R}$; in text recognition models, one may imagine that the input tensor $\boldsymbol{x}_{1:T}$ consists of T embeddings of size $2b$, and the j-th dimension of each token embedding is summarized into a real-valued token c_j. For further processing, only the information contained in the CLS token $\boldsymbol{c}$ will be forwarded to make predictions, as it reflects a compressed (encoded) version of the entire tensor information (after training of course). We illustrate the augmented input tensor $\boldsymbol{x}^+_{1:T+1}$ in Figure 1.

2.2 Credibility Transformer layer

2.2.1 Transformer architecture

The CLS token augmented input tensor $\boldsymbol{x}^+_{1:T+1} \in \mathbb{R}^{(T+1)\times 2b}$ is first processed through a normalization layer (not indicated in our notation, but highlighted in Table 1, below) before entering the Transformer architecture. Whereas several previous works in the actuarial literature have used batch-normalization [20], Transformer models usually use layer-normalization [1], which is what is utilized here.

We briefly describe the crucial modules of Transformers that are relevant for our Credibility Transformer architecture; for a more detailed description (and illustration) we refer to Vaswani et al. [31] and Richman–Wüthrich [28, Section 3.6].
Transformers are based on attention layers, and attention layers consist of queries $\boldsymbol{q}_t$, keys $\boldsymbol{k}_t$ and values $\boldsymbol{v}_t$, $1 \leq t \leq T+1$, given by the FNN layer transformed inputs

$$\begin{aligned} \boldsymbol{k}_t &= \phi\left(\boldsymbol{b}_K + W_K x_t^+\right) \in \mathbb{R}^{2b}, \\ \boldsymbol{q}_t &= \phi\left(\boldsymbol{b}_Q + W_Q x_t^+\right) \in \mathbb{R}^{2b}, \\ \boldsymbol{v}_t &= \phi\left(\boldsymbol{b}_V + W_V x_t^+\right) \in \mathbb{R}^{2b}, \end{aligned} \tag{2.3}$$

with weight matrices $W_K, W_Q, W_V \in \mathbb{R}^{2b \times 2b}$, biases $\boldsymbol{b}_K, \boldsymbol{b}_Q, \boldsymbol{b}_V \in \mathbb{R}^{2b}$, and where the activation function $\phi : \mathbb{R} \to \mathbb{R}$ is applied element-wise. Since the weight matrices and biases are assumed to be t-independent, we apply the same FNN transformation (2.3) in a time-distributed manner to all components $1 \leq t \leq T+1$ of the augmented input tensor $\boldsymbol{x}^+_{1:T+1}$. This provides us with tensors

$$\begin{aligned} K &= K(\boldsymbol{x}^+_{1:T+1}) = [\boldsymbol{k}_1, \ldots, \boldsymbol{k}_{T+1}]^\top \in \mathbb{R}^{(T+1)\times 2b}, \\ Q &= Q(\boldsymbol{x}^+_{1:T+1}) = [\boldsymbol{q}_1, \ldots, \boldsymbol{q}_{T+1}]^\top \in \mathbb{R}^{(T+1)\times 2b}, \\ V &= V(\boldsymbol{x}^+_{1:T+1}) = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{T+1}]^\top \in \mathbb{R}^{(T+1)\times 2b}. \end{aligned}$$

The keys K and queries Q are used to construct the attention weight matrix A by applying the $\mathsf{softmax}$ function to all rows (in a row-wise manner) in the following matrix

$$A = A(\boldsymbol{x}^+_{1:T+1}) = \mathsf{softmax}\left(\frac{QK^\top}{\sqrt{2b}}\right) \in \mathbb{R}^{(T+1)\times(T+1)}, \tag{2.4}$$

where the $\mathsf{softmax}$ operation is defined for each row as

$$\mathsf{softmax}(\boldsymbol{z})_j = \frac{\exp(z_j)}{\sum_{k=1}^{T+1} \exp(z_k)}, \quad \text{for } j = 1, \ldots, T+1, \tag{2.5}$$

with $\boldsymbol{z}$ being a row of the matrix $QK^\top/\sqrt{2b}$.
Finally, the *attention head* of the Transformer is received by the matrix multiplication

$$H = H(\boldsymbol{x}^+_{1:T+1}) = AV \in \mathbb{R}^{(T+1)\times 2b}. \tag{2.6}$$

This function encodes the augmented input tensor $\boldsymbol{x}^+_{1:T+1} \in \mathbb{R}^{(T+1)\times 2b}$ in the attention head $H(\boldsymbol{x}^+_{1:T+1}) \in \mathbb{R}^{(T+1)\times 2b}$ of the same dimension. Essentially, this attention mechanism is a weighting scheme reflected by the attention weights $A(\boldsymbol{x}^+_{1:T+1})$ applied to the values $V(\boldsymbol{x}^+_{1:T+1}) = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{T+1}]^\top$. For our first application of the Transformer model, we only use one attention head, we modify this to a multi-head attention in Section 4, below.
A Transformer layer is then obtained by adding the attention head to the augmented input tensor resulting in a so-called skip-connection transformation

$$\boldsymbol{x}^+_{1:T+1} \;\mapsto\; \boldsymbol{z}^{\text{skip1}}(\boldsymbol{x}^+_{1:T+1}) = \boldsymbol{x}^+_{1:T+1} + H(\boldsymbol{x}^+_{1:T+1}) \in \mathbb{R}^{(T+1)\times 2b}. \tag{2.7}$$

Typically, this transformed input is further processed by a time-distributed normalization layer $\boldsymbol{z}^{\text{norm}}$, drop-out layers $\boldsymbol{z}^{\text{drop}}$ and post-processing time-distributed FNN layers $\boldsymbol{z}^{\text{t-FNN}}$ in combination with skip connections $\boldsymbol{z}^{\text{skip}}$; for full details we refer to Vaswani et al. [31]. In our

architecture, we process the output $\boldsymbol{z}^{\text{skip1}}(\boldsymbol{x}^{+}_{1:T+1})$ of (2.7) further by applying the composed transformations

$$\begin{aligned}\boldsymbol{z}^{\text{trans}}(\boldsymbol{x}^{+}_{1:T+1}) &= \boldsymbol{z}^{\text{skip1}}(\boldsymbol{x}^{+}_{1:T+1}) \\ &+ \left(\boldsymbol{z}^{\text{norm2}} \circ \boldsymbol{z}^{\text{drop2}} \circ \boldsymbol{z}^{\text{t-FNN2}} \circ \boldsymbol{z}^{\text{drop1}} \circ \boldsymbol{z}^{\text{t-FNN1}} \circ \boldsymbol{z}^{\text{norm1}}\right)\left(\boldsymbol{z}^{\text{skip1}}(\boldsymbol{x}^{+}_{1:T+1})\right).\end{aligned} \tag{2.8}$$

Thus, we first normalize $\boldsymbol{z}^{\text{norm1}}$, then apply a time-distributed FNN layer followed by drop-out $\boldsymbol{z}^{\text{drop1}} \circ \boldsymbol{z}^{\text{t-FNN1}}$, and a second time-distributed FNN layer followed by drop-out $\boldsymbol{z}^{\text{drop2}} \circ \boldsymbol{z}^{\text{t-FNN2}}$, and finally, we normalize again $\boldsymbol{z}^{\text{norm2}}$. In a second skip-connection transformation we aggregate input and output of this transformation. This Transformer architecture outputs a tensor shape $\mathbb{R}^{(T+1)\times 2b}$. We remark that the drop-out layers are only used during model fitting, to prevent the architecture from in-sample overfitting.

While the Transformer architecture is quite complex, we further remark that the model presented here is now quite standard in the machine learning literature. We provide a diagram for ease of understanding the various components of the Transformer in Figure 12 in the appendix.

2.2.2 Credibility mechanism

The remaining steps differ from Vaswani et al. [31], and here, we take advantage of the integrated CLS tokens to complement this Transformer architecture $\boldsymbol{z}^{\text{trans}}(\cdot)$ by a credibility mechanism. For this we recall that the augmented input tensor $\boldsymbol{x}^{+}_{1:T+1}$ considers the (embedded) covariates $\boldsymbol{x} = (x_t)_{t=1}^{T}$ and their positional encodings $(\boldsymbol{e}^{\text{pos}}(t))_{t=1}^{T}$ in the first T components, and it is complemented by the CLS tokens $\boldsymbol{c} = \boldsymbol{x}^{+}_{T+1} \in \mathbb{R}^{2b}$ in the $(T+1)$-st component of the augmented input tensor $\boldsymbol{x}^{+}_{1:T+1}$, see (2.2).

At this stage, i.e., before processing the CLS tokens using the Transformer, the CLS tokens do not carry any covariate specific information. Note that this is because before we have applied the Transformer layer, these CLS tokens $\boldsymbol{c} = \boldsymbol{x}^{+}_{T+1}$ have not interacted with the covariates; and they are going to play the role of a global prior parameter in the following credibility consideration. For this purpose, we will extract the CLS token before the Transformer processing. After the Transformer processing, these CLS tokens have interacted with the covariates through the attention mechanism (2.4)-(2.6), and we are going to extract them a second time after this interaction, providing an embedded covariate summary.

The first version of the CLS token is extracted from the value matrix $V = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{T+1}]^{\top}$. Importantly, to get this CLS token to be in the same embedding space as the outputs of the Transformer model, we need to process this first version of the CLS token with exactly the same operations (with the same weights) as in the Transformer (2.8), except that we do not need to apply the attention mechanism nor time-distribute the layers. This provides us with the following prior information

$$\boldsymbol{c}^{\text{prior}} = \left(\boldsymbol{z}^{\text{norm2}} \circ \boldsymbol{z}^{\text{drop2}} \circ \boldsymbol{z}^{\text{FNN2}} \circ \boldsymbol{z}^{\text{drop1}} \circ \boldsymbol{z}^{\text{FNN1}} \circ \boldsymbol{z}^{\text{norm1}}\right)(\boldsymbol{v}_{T+1}) \ \in \ \mathbb{R}^{2b}. \tag{2.9}$$

Second, we extract the CLS token after processing through the Transformer (2.8), providing us with the tokenized information of the covariates $\boldsymbol{x}$ and their positional embeddings

$$\boldsymbol{c}^{\text{trans}} = \boldsymbol{z}^{\text{trans}}_{T+1}(\boldsymbol{x}^{+}_{1:T+1}) \ \in \ \mathbb{R}^{2b}, \tag{2.10}$$

i.e., this is the $(T+1)$-st row of the Transformer output (2.8). In this case, the CLS token has had the attention mechanism applied, and we emphasize that this version of the CLS token now contains a summary of the covariate information.
These two tokens (2.9) and (2.10) give us two different representations/predictors for the responses, the former representing only prior information, and the latter, prior information augmented by the covariates. We will use both of these tokens to make predictions from the Credibility Transformer by assigning weights to these representations.
This is done by selecting a fixed probability weight $\alpha \in (0,1)$ and then sampling during gradient descent training independent Bernoulli random variables $Z \sim \text{Bernoulli}(\alpha)$ that encode which CLS token is forwarded to the rest of the network to make predictions, that is, we forward the CLS token information according to

$$\boldsymbol{c}^{\text{cred}} = Z\, \boldsymbol{c}^{\text{trans}} + (1-Z)\, \boldsymbol{c}^{\text{prior}} \;\in\; \mathbb{R}^{2b}. \tag{2.11}$$

Thus, in $\alpha \cdot 100\%$ of the gradient descent steps we forward the Transformer processed CLS token $\boldsymbol{c}^{\text{trans}}$ which has interacted with the covariates, and in $(1-\alpha)\cdot 100\%$ of the gradient descent steps we forward the prior value CLS token $\boldsymbol{c}^{\text{prior}}$. This can be seen as assigning a credibility of α to the Transformer token $\boldsymbol{c}^{\text{trans}}$ and the complementary credibility of $1-\alpha$ to the prior value $\boldsymbol{c}^{\text{prior}}$ of that CLS token, to receive reasonable network parameters during gradient descent training. This mechanism (2.11) is only applied during training, and for forecasting we set $Z \equiv 1$. The probability $\alpha \in (0,1)$ is treated as a hyperparameter that can be optimized by a grid search. We selected it bigger than $1/2$, because we would like to put more emphasis on the tokenized covariate information.

2.2.3 Rationale of credibility weighted CLS token

We explain the rationale of the proposed credibility weighted CLS token $\boldsymbol{c}^{\text{cred}}$. There are two credibility mechanisms involved, a more obvious one that involves the hyperparameter $\alpha \in (0,1)$, but, also, there is a second credibility mechanism involved in the CLS token which learns an optimal credibility weight. This latter credibility mechanism arises from the prior information learned by the CLS token as a result of training the network with the hyperparameter α. We now discuss these two credibility mechanisms.
First, when training the Credibility Transformer, the CLS token $\boldsymbol{c}^{\text{cred}}$ will randomly be set to be either the token $\boldsymbol{c}^{\text{trans}}$, including covariate information, or the token $\boldsymbol{c}^{\text{prior}}$, encoding only prior information. In the latter case, the best prediction (in the sense of minimizing a deviance based loss function) that can be made without covariate information will be the portfolio mean. Thus, in $(1-\alpha)\cdot 100\%$ of the training iterations of the Credibility Transformer, we will encourage the CLS token to incorporate prior information in the form of predicting the portfolio average. In the remaining iterations of the training, the CLS token will be trained on the covariate information to make more precise predictions than can be made using the portfolio average. In this sense, we can consider the CLS token $\boldsymbol{v}_{T+1}$ before processing, i.e., as given in(2.9), representing the portfolio average in the embedding space of the rest of the tokens.
We now dive deeper into the attention mechanism as it relates to the CLS token. Let $\boldsymbol{a}_{T+1} = (a_{T+1,1}, \ldots, a_{T+1,T+1})^\top \in \mathbb{R}^{T+1}$ denote the $(T+1)$-st row of the attention weight matrix A defined in (2.4). This row corresponds to the attention weights for the CLS token that forwards

the extracted information. By using the softmax function, we have normalization

$$\sum_{j=1}^{T+1} a_{T+1,j} = 1.$$

We can interpret the last element of this vector, $a_{T+1,T+1}$, as the probability assigned to the CLS token itself, and we set

$$P = a_{T+1,T+1} \ \in \ (0,1).$$

Consequently, the remaining probability $1-P$ is distributed among the covariate information

$$\sum_{j=1}^{T} a_{T+1,j} = 1 - P.$$

This formulation allows us to interpret the attention mechanism in a way that is analogous to a linear credibility formula. Specifically, the attention output for the CLS token can be expressed as

$$\boldsymbol{v}^{\text{trans}} = P\, \boldsymbol{v}_{T+1} + (1-P)\, \boldsymbol{v}^{\text{covariate}}, \tag{2.12}$$

where $\boldsymbol{v}^{\text{trans}}$ is the CLS embedding processed by the attention mechanism, in fact, this is the $(T+1)$-st row of the attention head H given in (2.6), and the covariate information is a weighted sum of all values that consider the covariate information

$$\boldsymbol{v}^{\text{covariate}} = \sum_{j=1}^{T} \frac{a_{T+1,j}}{1-P}\, \boldsymbol{v}_j. \tag{2.13}$$

This formulation demonstrates how the attention mechanism for the CLS token can be interpreted as a credibility weighted average of the CLS token's own information (representing collective experience) and the information from the covariates (representing individual experience), see Figure 2. Essentially, this is a Bühlmann–Straub [4] type linear credibility formula, or a time-dynamic version thereof, with the credibility weights in this context learned and being input-dependent; this is similar to Wüthrich [32, Chapter 5].

2.3 Decoding of the tokenized information

In the previous two sections we have encoded the covariate input $\boldsymbol{x}$ in two steps to a tokenized variable $\boldsymbol{c}^{\text{cred}}$:

$$\boldsymbol{x} \qquad \mapsto \qquad \boldsymbol{x}^{+}_{1:T+1} = \boldsymbol{x}^{+}_{1:T+1}(\boldsymbol{x}) \qquad \mapsto \qquad \boldsymbol{c}^{\text{cred}} = \boldsymbol{c}^{\text{cred}}(\boldsymbol{x}).$$

The final step is to decode this tokenized variable $\boldsymbol{c}^{\text{cred}} = \boldsymbol{c}^{\text{cred}}(\boldsymbol{x})$ so that it is suitable to predict a response variable Y. This decoder is problem-specific. Because our example below corresponds to a one-dimensional single-task prediction problem, we use a plain-vanilla FNN with a single output component to decode the credibilitized token $\boldsymbol{c}^{\text{cred}}$. This results in the decoder

$$\boldsymbol{z}^{(2:1)} : \mathbb{R}^{2b} \to \mathbb{R}, \qquad \boldsymbol{c}^{\text{cred}}(\boldsymbol{x}) \ \mapsto \ \boldsymbol{z}^{(2:1)}(\boldsymbol{c}^{\text{cred}}(\boldsymbol{x})),$$

where $\boldsymbol{z}^{(2:1)}$ is a shallow FNN that maps from $\mathbb{R}^{2b}$ to $\mathbb{R}$. Since our responses Y is a non-negative random variable, we are going to model claim counts, we add the exponential output activation, which all together results in the *Credibility Transformer* (CT)

$$\boldsymbol{x} \ \mapsto \ \mu^{\text{CT}}(\boldsymbol{x}) = \exp\left\{\boldsymbol{z}^{(2:1)}(\boldsymbol{c}^{\text{cred}}(\boldsymbol{x}))\right\} \ > \ 0. \tag{2.14}$$

The next section presents a real data example to exemplify the use of the Credibility Transformer.

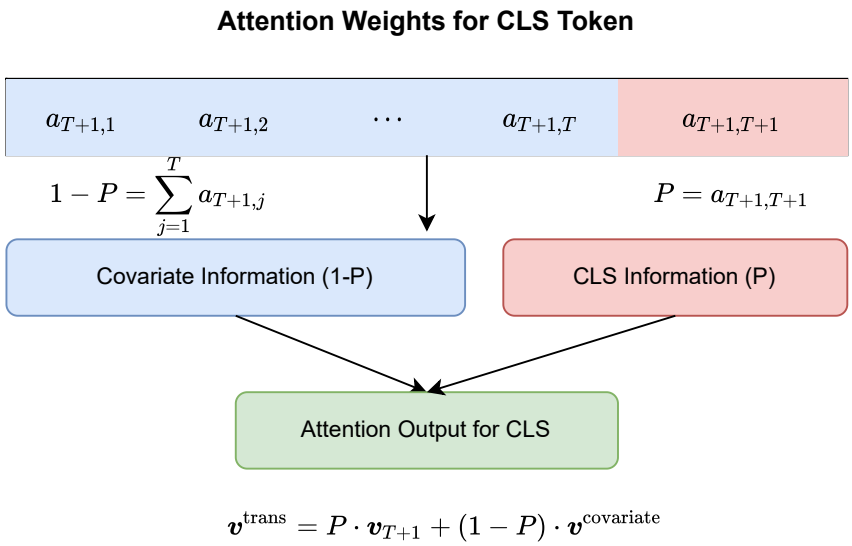

Figure 2: Diagram of how the attention mechanism can be viewed as a linear credibility formula.

3 Real data example

We benchmark the Credibility Transformer on the commonly used French motor third party liability (MTPL) claims counts dataset of Dutang et al. [9].

3.1 Description of data

We perform the data pre-processing exactly as in Wüthrich–Merz [33], so that all our results can directly be benchmarked by the results in that reference, and they are also directly comparable to the ones in Brauer [2, Table 2]. For a detailed description of the French MTPL dataset we refer to Wüthrich–Merz [33, Section 13.1]; and the data cleaning and pre-processing is described in [33, Listings 13.1 and 5.2]. We use the identical split into learning and test datasets as in [33].[1] The learning dataset consists of 610,206 instances and the test dataset of 67,801 instances; we refer to [33, Section 5.2.4]. This partition provides an aggregated time exposure of 322,392 calendar years on the learning dataset, and 35,967 calendar years on the test dataset. There are 23,738 claim events on the learning data, providing an empirical frequency of $\widehat{\lambda} = 7.35\%$, i.e., on average we observe an accident every $13.58 = 1/\widehat{\lambda}$ calendar years. This is a commonly observed expected frequency in European MTPL insurance. On the test dataset we have 2,645 claim events providing almost the same empirical frequency as on the learning dataset.

3.2 Input tokenizer

There are $T_1 = 4$ categorical covariates with numbers of levels $n_t = 6, 2, 11$ and 22, and there are $T_2 = 5$ continuous covariates, providing $T = T_1 + T_2 = 9$ covariates. For embedding and tokenizing these input covariates we select an embedding dimension of $b = 5$. This provides us with $\varrho^{\text{input}} = 405$ weights from the feature tokenizer giving us the raw input tensor $\boldsymbol{x}^{\circ}_{1:9} \in \mathbb{R}^{9\times 5}$.

[1] The cleaned data can be downloaded from `https://people.math.ethz.ch/~wueth/Lecture/freMTPL2freq.rda`, and for the partition into learning and test data we use the random number generator under R version 3.5.0.

For the continuous input variable tokenizer we select the linear activation function in the first FNN layers $\boldsymbol{z}_t^{(1)}$ and the hyperbolic tangent activation function in the second FNN layers $\boldsymbol{z}_t^{(2)}$.

3.3 Description of the selected Credibility Transformer architecture

Table 1 summarizes the Credibility Transformer architecture used.

Module	Variable/layer	# Weights
Feature tokenizer (raw input tensor)	$\boldsymbol{x}_{1:9}^{\circ}$	405
Positional encoding	$\boldsymbol{e}_{1:9}^{\text{pos}}$	45
CLS tokens	$\boldsymbol{c}$	10
Time-distributed normalization layer	$\boldsymbol{z}^{\text{norm}}$	20
Credibility Transformer	$\boldsymbol{c}^{\text{cred}}$	1,073
FNN decoder	$\boldsymbol{z}^{(2:1)}$	193

Table 1: Credibility Transformer architecture used on the French MTPL dataset.

The total number of weights that need to be fitted is 1,746. The Credibility Transformer uses 1,073 weights, these are 330 weights from the time-distributed layers providing the keys $\boldsymbol{k}_t$, queries $\boldsymbol{q}_t$ and values $\boldsymbol{v}_t$, 682 neurons from the two time-distributed FNN layers $\boldsymbol{z}^{\text{t}-\text{FNN1}}$ and $\boldsymbol{z}^{\text{t}-\text{FNN2}}$ having 32 and $2b = 10$ neurons, respectively, and the remaining weights are from the normalization layers. The decoder $\boldsymbol{z}^{(2:1)}$ uses 193 weights coming from a first FNN layer having 16 neurons and the output layer, resulting in the one-dimensional real-valued positive predictor (2.14). In all hidden layers we use the Gaussian error linear unit (GELU) activation function that takes the form $x \in \mathbb{R} \mapsto x\Phi(x)$ with the standard Gaussian distribution Φ. The credibility parameter is set to $\alpha = 90\%$, this is an optimal value found by a grid search. For the drop-out layers we choose a drop-out rate of 1%. Both the credibility mechanism and the drop-out is only used during network fitting, and for prediction we set $\alpha = 1$ and the drop-out rate to zero.

3.4 Gradient descent network fitting: 1st version

3.4.1 Plain-vanilla gradient descent fitting

The Poisson deviance loss is used for claims counts model fitting. Minimizing the Poisson deviance loss is equivalent to maximum likelihood estimation (MLE) under a Poisson assumption. The average Poisson deviance loss is given by, see Wüthrich–Merz [33, formula (5.28)],

$$\frac{2}{n}\sum_{i=1}^{n} v_i\mu^{\text{CT}}(\boldsymbol{x}_i) - Y_i - Y_i \log\left(\frac{v_i\mu^{\text{CT}}(\boldsymbol{x}_i)}{Y_i}\right) \;\geq\; 0, \tag{3.1}$$

where Y_i are the observed numbers of claims on policy i, $v_i > 0$ is the time exposure of policy i, and $\mu^{\text{CT}}(\boldsymbol{x}_i)$ is the estimated expected claims frequency of that policy received from the Credibility Transformer (2.14). The sum runs over all instances (insurance policies) $1 \leq i \leq n$ that are included in the learning data. Note that the Poisson deviance loss is a strictly consistent loss function for mean estimation which is an important property that selected loss functions should possess for mean estimation; see Gneiting–Raftery [12] and Gneiting [11].

In our first fitting approach we use the `nadam` version of stochastic gradient descent (SGD) with its pre-selected parametrization implemented in the `keras` package [5]. For SGD we use a batch

size of $2^{10} = 1,024$ instances, and since neural networks are prone to overfitting, we exploit an early stopping strategy by partitioning the learning data at random into a training dataset and a validation dataset at a ratio of 9:1. We use the standard `callback` of `keras` [5] using its pre-selected parametrization to retrieve the learned weights with the smallest validation loss. The optimal network weights are found after roughly 150 SGD epochs.
Neural network fitting involves several elements of randomness such as the (random) initialization of the SGD algorithm; see Wüthrich–Merz [33, Section 7.4.4]. To improve the robustness of the results we always run 20 SGD fittings with different random initializations, and the reported results correspond to averaged losses over the 20 different SGD fittings, in round brackets we state the standard deviation of the losses over these 20 SGD fittings.
Finally, in Richman–Wüthrich [26] we have seen that one can substantially improve predictive performance by ensembling over different SGD runs. On average it takes 10 to 20 SGD fittings to get good ensemble predictors, see Wüthrich–Merz [33, Figure 7.19]. For this reason, in the last step of our procedure, we consider the ensemble predictor over the 20 different SGD runs. From our results it can be verified that ensembling significantly improves predictive models.

3.4.2 Results of 1st fitting approach

The results are given in Table 2 (rows `nadam`). These results of the Credibility Transformer are benchmarked with the ones in Wüthrich–Merz [33, Tables 7.4-7.7 and 7.9] and Brauer [2, Tables 2 and 4]. We conclude that the Credibility Transformer clearly outperforms any of these other proposals out-of-sample, even the ensembled plain-vanilla FNN model is not much better than a single run of the Credibility Transformer. By building the ensemble predictor of the Credibility Transformers we receive a significantly better model providing an out-of-sample average Poisson deviance loss of $23.717 \cdot 10^{-2}$, see Table 2. The best ensemble predictor in Brauer [2, Table 4] called $\text{CAFFT}_{\text{def}}$ has an average Poisson deviance loss of $23.726 \cdot 10^{-2}$.

Model	# Param.	In-sample Poisson loss		Out-of-sample Poisson loss	
Poisson null	1	25.213		25.445	
Poisson GLM3	50	24.084		24.102	
Poisson plain-vanilla FNN	792	23.728	(± 0.026)	23.819	(± 0.017)
Ensemble Poisson plain-vanilla FNN	792	23.691		23.783	
Credibility Transformer: `nadam`	1,746	23.648	(± 0.071)	23.796	(± 0.037)
Ensemble Credibility Transformer: `nadam`	1,746	23.565		23.717	
Credibility Transformer: `NormFormer`	1,746	23.641	(± 0.053)	23.788	(± 0.040)
Ensemble Credibility Transformer: `NormFormer`	1,746	23.562		23.711	

Table 2: Number of parameters, in-sample and out-of-sample Poisson deviance losses (units are in 10^{-2}); benchmark models in the upper part of the table are taken from [33, Table 7.9].

3.5 NormFormer gradient descent fitting: 2nd version

Standard SGD optimizers such as `nadam` with a standard early stopping `callback` do not work particularly well on Transformers because of gradient magnitudes mismatches, typically gradients on earlier layers are much bigger than gradients at later layers; see Shleifer et al. [30].

Therefore, we exploit the `NormFormer` proposed by Shleifer et al. [30] in our 2nd fitting attempt with the `adam` version of SGD with a learning rate of 0.002 and $\beta_2 = 0.98$; reducing this parameter of `adam` has been suggested, e.g., in Zhai et al. [34]. The results are given in the lower part of Table 2. We observe a slight improvement in prediction accuracy, the Credibility Transformer ensemble having a smaller out-of-sample average Poisson deviance loss of $23.711 \cdot 10^{-2}$.

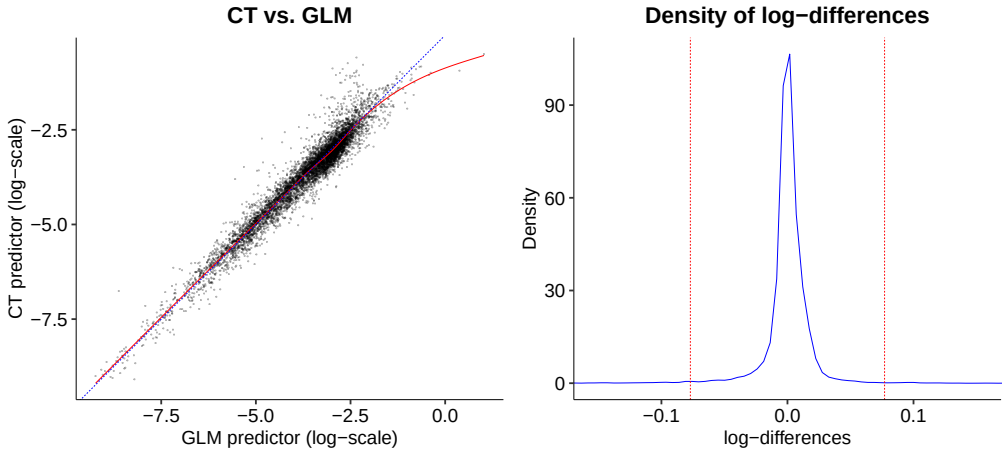

Figure 3: (lhs) Scatterplot of predictions from one run of the Credibility Transformer (CT) vs. GLM predictions; (rhs) density of log-differences of the two predictors, the vertical dotted lines correspond to 2 empirical standard deviations.

Figure 3 (lhs) gives a scatterplot that compares one run of the Credibility Transformer predictions (`NormFormer` SGD fitting) against the GLM predictions (out-of-sample). The red line gives a GAM smoother. On average the two predictors are rather similar, except in the tails. However, the black cloud of predictors has individual points that substantially differ from the darkgray diagonal line, indicating that there are bigger differences on an individual insurance policy scale. This is verified by the density plot of the individual log-differences of the predictors on the right-hand side of Figure 3.

Figure 4 compares the two ensemble Credibility Transformer predictions obtained from the `NormFormer` SGD fitting procedure and the `nadam` fitting procedure, the former slightly outperforming the latter in terms of out-of-sample loss. The figure shows that the individual predictions lie fairly much on the diagonal line saying that there are not any bigger individual differences. This supports robustness of the fitting procedure.

We emphasize that the credibility mechanism (2.11) is only applied during SGD training. For prediction, we turn this credibility mechanism off by setting $\alpha = 100\%$, resulting in weights $Z \equiv 1$ in (2.11), thus, only the observation based CLS token $\boldsymbol{c}^{\text{trans}}$ is considered for prediction. We may ask whether the network has also learned any structure for the prior CLS token $\boldsymbol{c}^{\text{prior}}$. This can be checked by setting $\alpha = 0\%$, resulting in weights $Z \equiv 0$ in (2.11). Using the resulting predictor we receive a constant prediction of $\widehat{\lambda} = 7.35\%$, i.e., the homogeneous model without considering any covariate information. Of course, this is expected by the design of the network architecture. However, it is still worth to verify this property to ensure that the network architecture works as expected. If we set $Z \equiv 90\%$ to the same rate as used for network fitting we receive a bigger out-of-sample loss of $23.727 \cdot 10^{-2}$ not supporting credibility robustification

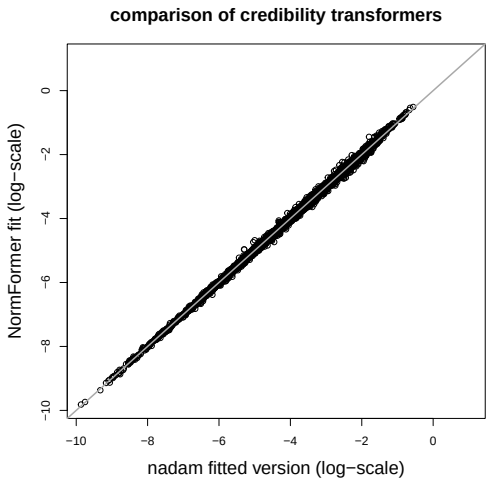

Figure 4: Scatterplot of ensemble Credibility Transformer predictions: `NormFormer` SGD fitting against `nadam` fitting.

of the predictor, once properly trained.

4 Improving the Credibility Transformer

We now work to further improve the Credibility Transformer. The following modifications can be categorized into three main areas. First, we augment the Transformer with a multi-head attention and utilize a deep version of the architecture. Second, we use insights from the LLM literature to more flexibly handle inputs to the model by using gated layers. Third, we modify the inputs to the network by using a differentiable continuous covariate embedding layer based on a novel idea from Gorishny et al. [14]. Through applying these changes, we achieve a very strong out-of-sample performance of the Credibility Transformer. We start by discussing each of these changes and then return to demonstrate the performance of the improved architecture.

4.1 Multi-head attention and deep Transformer architecture

4.1.1 Multi-head attention

In Section 2.2.1, we have described a Transformer model with a single attention head, see (2.6). An important extension of this simpler model is *multi-head attention* (MHA), which applies the attention mechanism (2.4)-(2.6) multiple independent copies of the query, key and value projections of the input data. This allows the model to attend to information from different learned representations. After learning these multiple representations, these are projected back to the original dimension of the model, which is $2b$ in our case.
Formally, we chose the number of attention heads $M \in \mathbb{N}$. The MHA is defined as

$$H^{\text{MHA}}(\boldsymbol{x}^{+}_{1:T+1}) = \text{Concat}(H_1, \ldots, H_M)W^O \;\in\; \mathbb{R}^{(T+1)\times 2b}, \tag{4.1}$$

where we concatenate multiple attention heads to a tensor

$$\text{Concat}(H_1, \ldots, H_M) = [H_1, H_2, \ldots, H_M] \;\in\; \mathbb{R}^{(T+1)\times(Md)},$$

with attention heads $H_m \in \mathbb{R}^{(T+1)\times d}$ of dimension $d \in \mathbb{N}$, for $1 \leq m \leq M$, and output projection matrix $W^O \in \mathbb{R}^{(Md)\times 2b}$. Typically, one chooses $Md = 2b$, in other words, the total dimension of the model is chosen for Md, and it is proportionally allocated to each attention head.
For each head, $1 \leq m \leq M$, we use the same formulation as above for the attention operation, with attention head specific keys, queries and values

$$\begin{aligned}
H_m &= A_m V_m \in \mathbb{R}^{(T+1)\times d}, \\
A_m &= \mathsf{softmax}\left(\frac{Q_m K_m^\top}{\sqrt{d}}\right) \in \mathbb{R}^{(T+1)\times(T+1)}, \\
K_m &= \phi\left(\boldsymbol{b}_K^{(m)} + \boldsymbol{x}_{1:T+1}^{+} W_K^{(m)}\right) \in \mathbb{R}^{(T+1)\times d}, \\
Q_m &= \phi\left(\boldsymbol{b}_Q^{(m)} + \boldsymbol{x}_{1:T+1}^{+} W_Q^{(m)}\right) \in \mathbb{R}^{(T+1)\times d}, \\
V_m &= \phi\left(\boldsymbol{b}_V^{(m)} + \boldsymbol{x}_{1:T+1}^{+} W_V^{(m)}\right) \in \mathbb{R}^{(T+1)\times d}.
\end{aligned}$$

At this point, we have described a standard implementation of a MHA. To improve the stability of fitting this model we again follow Shleifer et al. [30] by adding a multiplicative scaling coefficient to each head before it enters into (4.1), i.e., we update this to

$$H^{\mathrm{MHA}}(\boldsymbol{x}_{1:T+1}^{+}) = \mathrm{Concat}(\alpha_1 H_1, \ldots, \alpha_M H_M) W^O \in \mathbb{R}^{(T+1)\times 2b}, \tag{4.2}$$

where $\alpha_m \in (0, 1]$, for $1 \leq m \leq M$. During training, we apply dropout to α_m to improve the out-of-sample performance of the network. After this MHA modification, the rest of the Transformer layer used here is the same as in Sections 2.2.1 and 2.2.2.

4.1.2 Deep Credibility Transformer

We use multiple Transformer layers composed together to create a deep version of the model applied above, assuming that we have Transformer layers $1 \leq \ell \leq L$, for $L \in \mathbb{N}$. To compose these layers, while retaining the ability to use the credibility mechanism 2.11, we need to modify the inputs to the Transformer layers. The first layer of these is exactly the same as presented above, producing outputs $\boldsymbol{z}^{\mathrm{trans},1}(\boldsymbol{x}_{1:T+1}^{+})$, including the CLS token $\boldsymbol{c}^{\mathrm{trans},1}$, which could be stripped out if needed at this point, and $\boldsymbol{c}^{\mathrm{prior},1}$, where we have added an upper index ^{1} to show that this is the first Transformer layer with its outputs.
For the second and subsequent layers, $2 \leq \ell \leq L$, we input $\boldsymbol{z}^{\mathrm{trans},\ell-1}$ and $\boldsymbol{c}^{\mathrm{prior},\ell-1}$ recursively into layer specific versions of (2.8)

$$\begin{aligned}
\boldsymbol{z}^{\mathrm{trans},\ell}(\boldsymbol{x}_{1:T+1}^{+}) &= \boldsymbol{z}^{\mathrm{skip1},\ell}(\boldsymbol{z}^{\mathrm{trans},\ell-1}(\boldsymbol{x}_{1:T+1}^{+})) \\
&\quad + \left(\boldsymbol{z}^{\mathrm{norm2},\ell} \circ \boldsymbol{z}^{\mathrm{drop2},\ell} \circ \boldsymbol{z}^{\mathrm{t-FNN2},\ell} \circ \boldsymbol{z}^{\mathrm{drop1},\ell} \circ \boldsymbol{z}^{\mathrm{t-FNN1},\ell} \circ \boldsymbol{z}^{\mathrm{norm1},\ell}\right) \\
&\quad \circ \left(\boldsymbol{z}^{\mathrm{skip1},\ell}(\boldsymbol{z}^{\mathrm{trans},\ell-1}(\boldsymbol{x}_{1:T+1}^{+}))\right) \in \mathbb{R}^{(T+1)\times 2b},
\end{aligned} \tag{4.3}$$

and (2.9)[2]

$$\boldsymbol{c}^{\mathrm{prior},\ell} = \left(\boldsymbol{z}^{\mathrm{norm2},\ell} \circ \boldsymbol{z}^{\mathrm{drop2},\ell} \circ \boldsymbol{z}^{\mathrm{FNN2},\ell} \circ \boldsymbol{z}^{\mathrm{drop1},\ell} \circ \boldsymbol{z}^{\mathrm{FNN1},\ell} \circ \boldsymbol{z}^{\mathrm{norm1},\ell}\right)\left(\boldsymbol{c}^{\mathrm{prior},\ell-1}\right) \in \mathbb{R}^{2b}. \tag{4.4}$$

[2]For notational convenience, we have not included the head-specific key, query and value projections of $\boldsymbol{c}^{\mathrm{prior},\ell-1}$ to derive $\boldsymbol{c}^{\mathrm{prior},\ell}$, nonetheless, these are also performed.

Finally, the credibility weighting between the outputs is performed

$$\boldsymbol{c}^{\text{cred,L}} = Z\,\boldsymbol{c}^{\text{trans,L}} + (1-Z)\,\boldsymbol{c}^{\text{prior,L}} \;\in\; \mathbb{R}^{2b}. \tag{4.5}$$

The same rationale of linear credibility as given in Section 2.2.3 applies here too; in each attention head (a projection of) the prior information in the CLS token is reweighted with the (projected) covariate information being processed there. For an illustration, see Figure 13 in the appendix.

4.2 Gated layers

The model presented to this point uses a standard FNN to process the outputs of the attention mechanism. Modern LLMs usually rely on a more advanced mechanism incorporating a gating principle for the FNN; this has been introduced to the LLM literature by Dauphin et al. [6], with the concept of a Gated Linear Unit (GLU). The main difference between the usual FNNs and a GLU is that not all of the inputs to a FNN have equal importance, and, indeed, some of the less important inputs should be down-weighted or even removed from the computations performed by the network. A GLU accomplishes this by adding a second FNN with a sigmoid activation function - which produces outputs between 0 and 1 - to the usual FNN, and then multiplies these by an (element-wise) Hadamard product $\odot$, i.e.,

$$\boldsymbol{z}^{\text{GLU}}(\boldsymbol{x}) = \boldsymbol{z}^{\text{FNN}_{\text{sigmoid}}}(\boldsymbol{x}) \odot \boldsymbol{z}^{\text{FNN}_{\text{linear}}}(\boldsymbol{x}), \tag{4.6}$$

where the GLU layer $\boldsymbol{z}^{\text{GLU}}$ is the Hadamard product of a standard affine projection $\boldsymbol{z}^{\text{FNN}_{\text{linear}}}$ and a FNN layer with sigmoid activation $\boldsymbol{z}^{\text{FNN}_{\text{sigmoid}}}$. Beyond this simple formulation of the GLU, modern LLMs usually follow Shazeer [29], who replaces the sigmoid activation in (4.6) with a different activation function. Especially popular is to replace the sigmoid function with a so-called sigmoid linear unit (SiLU) activation function, due to Elfwing et al. [10], defined as

$$\text{SiLU}(x) = x \odot \sigma(x).$$

The SiLU-GLU layer, abbreviated to SwiGLU for Swish GLU, see Shazeer [29], is defined as

$$\boldsymbol{z}^{\text{SwiGLU}}(\boldsymbol{x}) = \boldsymbol{z}^{\text{FNN}_{\text{linear}}}(\boldsymbol{x}) \odot \boldsymbol{z}^{\text{FNN}_{\text{SiLU}}}(\boldsymbol{x}).$$

In our implementation, we replace the first FNNs in equations (4.3) and (4.4) with these SwiGLU layers. This modification allows one for more complex interactions between features and improves the model's ability to down-weight less important components of the input data.

4.3 Improving the continuous covariate embedding

The final major modification to the Credibility Transformer is to improve the continuous covariate embedding (2.1) by replacing the simple two-layer FNN with a more advanced approach. We start with the Piecewise Linear Encoding (PLE) of Gorishny et al. [13], which encodes continuous covariates using a numerical approach that, in principle, is similar to an extension of one-hot encoding to continuous variables. The main idea of PLE is to split the range of each continuous covariate into bins, and then encode the covariate value based on which bin it falls into. This provides a more expressive representation compared to the original scalar values.
Formally, for the t-th continuous covariate of the T_2 continuous covariates available in the dataset, we partition its range of values into $B_t \in \mathbb{N}$ disjoint intervals - which are called *bins* -

$\mathfrak{B}_t^j = [b_t^{j-1}, b_t^j)$, for $1 \leq j \leq B_t$, and with $b_t^{j-1} < b_t^j$. The PLE encoding of covariate value x_t is then defined by the vector

$$\mathrm{PLE}_t(x_t) = \left(e_1^{\mathrm{PLE}}, \ldots, e_{B_t}^{\mathrm{PLE}}\right)^\top \in \mathbb{R}^{B_t},$$

where for $1 \leq j \leq B_t$

$$e_j^{\mathrm{PLE}} = e_j^{\mathrm{PLE}}(x_t) = \frac{x_t - b_t^{j-1}}{b_t^j - b_t^{j-1}} \mathbb{1}_{\{x_t \in \mathfrak{B}_t^j\}} + \mathbb{1}_{\{x_t \geq b_t^j\}}.$$

This encoding gives a B_t-dimensional vector $\mathrm{PLE}_t(x_t)$, which has components 1 as long as $x_t \geq b_t^j$, for $x_t \in \mathfrak{B}_t^j$ we interpolate linearly, and for $x_t < b_t^{j-1}$ the components of $\mathrm{PLE}_t(x_t)$ are zero. This encoding has several desirable properties, among which, we note that it provides a more expressive yet lossless representation of the original scalar values that preserves the ordinal nature of continuous covariates. In the original proposal of Gorishny et al. [13], the bin boundaries are determined either using quantiles of the observed values of the covariate components x_t or they are derived from a decision tree. Differently to these approaches, we introduce a differentiable version of PLE that allows for end-to-end training of the bin boundaries. This approach enables the model to learn optimal bin placements for each continuous covariate during the training process. The key idea is to parameterize the bin boundaries indirectly through trainable weights that define the length of each bin. The bin boundaries $(b_t^j)_{j=0}^{B_t}$ are computed as

$$b_t^j = s_t + \sum_{k=0}^{j} \delta_t^k,$$

where $s_t \in \mathbb{R}$ is a given (fixed) starting value, and $\delta_t^k > 0$ are learnable bin lengths. To ensure numerical stability and enforce non-negative bin widths, we parameterize the bin lengths using their logarithms

$$\delta_t^k = \exp(\log(\delta_t^k)).$$

The logged bin lengths $\log(\delta_t^k)$ are initialized randomly or from pre-computed initial bins using quantiles. Finally, to handle potential numerical issues and to ensure a minimum bin length, we introduce a threshold $\epsilon > 0$

$$\delta_t^k \leftarrow \delta_t^k \mathbb{1}_{\{\delta_t^k \geq \epsilon\}}$$

Thus, if the size of the learned bin is less than ϵ, we collapse the bin into the previous one. This differentiable PLE allows the model to adapt the bin boundaries during training, which, taken together with the next part, allows more effective feature representations to be learned.

To use the PLE in the Transformer model, following Gorishny et al. [13], we add a trainable FNN, $\boldsymbol{z}_t^{\mathrm{FNN}}$, with a hyperbolic tangent activation after the PLE to produce the final feature embedding

$$x_t \mapsto \boldsymbol{e}_t^{\mathrm{NE}}(x_t) = \left(\boldsymbol{z}_t^{\mathrm{FNN}} \circ \mathrm{PLE}_t\right)(x_t) \in \mathbb{R}^b,$$

where $\boldsymbol{e}_t^{\mathrm{NE}}$ is the numerical embedding (NE) from the PLE layer. We expect that the PLE followed by a hyperbolic tangent layer is more effective than a simple two-layer FNN for continuous covariate embeddings since, unlike a FNN which applies a global transformation to its inputs, this numerical embedding captures local patterns within different intervals of the covariate's range. This is because the FNN with a hyperbolic tangent activation expands the information contained in the PLE into a fixed-size embedding $\boldsymbol{e}_t^{\mathrm{NE}} \in \mathbb{R}^b$, which varies linearly within bins but non-linearly across different bins. A diagram of the differentiable PLE is shown in Figure 5,

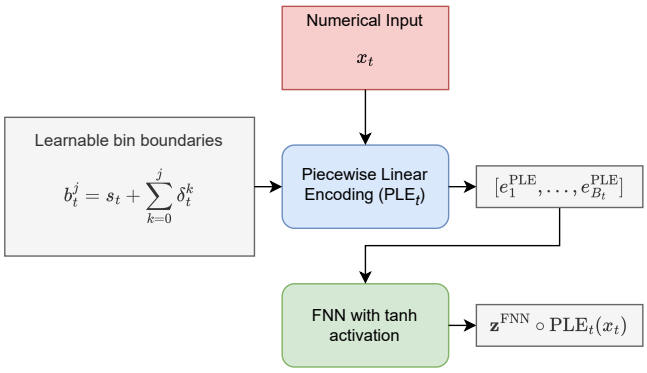

Figure 5: Visualization of differentiable Piecewise Linear Encoding (PLE) for a numerical feature.

4.4 Other modifications

Here, we briefly mention some of the other less complex modifications made to the Credibility Transformer. Two of these are inspired by Holzmüller et al. [18]. First, continuous inputs to the network are scaled by subtracting the median of the data and then dividing by the interquartile range; this provides inputs that are more robust to outliers. Second, we allow for learned feature selection by scaling each embedding by a constant in the range $(0, 1]$ before these enter the Transformer. We initialize all FNN layers that are followed by the GeLU activation function using the scheme of He et al. [17] and use the `adamW` optimizer of Loshchilov–Hutter [23], with weight decay set to 0.02. Finally, we set the $\beta_2 = 0.95$ in the optimizer, which is a best practice to stabilize optimization with Transformer architectures that use large batches and versions of the `adam` optimizer, see Zhai et al. [34].

4.5 Results of the improved deep Credibility Transformer

The improved version of the Credibility Transformer was fit to the same split of the French MTPL dataset as above. Compared to the details discussed in Section 3, we follow the same approach, except that we fit on batches of size $2^{12} = 4096$ instances. We set the number of attention heads in each layer $M = 2$ and use three transformer layers, i.e., we use a deep Transformer architecture. Moreover, we set $b = 40$, in other words, we use a model that is eight times wider than the one used above. Since this results in a very large number of parameters to optimize - approximately 320,000 - it becomes infeasible to fit these models without utilizing Graphics Processing Units (GPUs); the approach taken was to utilize a cloud computing service to fit these models. Each training run of the model takes about 7 minutes[3], i.e, fitting such a large and complex model is entirely feasible using GPUs. We show the in-sample and out-of-sample Poisson deviance losses for each credibility parameter in Table 3, where we evaluate the model's performance for different values of the credibility parameter α, ranging from 90% to 100%. Recall that setting the credibility parameter to $\alpha = 100\%$ corresponds to the plain-vanilla Transformer training approach without using the credibility mechanism for training.
From Table 3 we observe that the different credibility parameters $\alpha \in \{90\%, 95\%, 98\%, 100\%\}$

[3]This was done on an L4 GPU on the Google Colab service.

Model	In-sample Poisson loss	Out-of-sample Poisson loss
Ensemble Poisson plain-vanilla FNN	23.691	23.783
Ensemble Credibility Transformer (best-performing)	23.562	23.711
Improved Credibility Transformer with $\alpha = 90\%$	23.533 ($\pm$ 0.058)	23.670 ($\pm$ 0.031)
Ensemble Credibility Transformer ($\alpha = 90\%$)	23.454	23.587
Improved Credibility Transformer with $\alpha = 95\%$	23.557 ($\pm$ 0.058)	23.676 ($\pm$ 0.027)
Ensemble Credibility Transformer ($\alpha = 95\%$)	23.465	23.593
Improved Credibility Transformer with $\alpha = 98\%$	23.544 ($\pm$ 0.042)	23.670 ($\pm$ 0.032)
Ensemble Credibility Transformer ($\alpha = 98\%$)	23.460	23.577
Improved Credibility Transformer with $\alpha = 100\%$	23.535 ($\pm$ 0.051)	23.689 ($\pm$ 0.044)
Ensemble Credibility Transformer ($\alpha = 100\%$)	23.447	23.607

Table 3: In-sample and out-of-sample Poisson deviance losses (units are 10^{-2}) for each credibility parameter applied to the improved deep Credibility Transformer of Section 4; ensembling results are shown for each credibility parameter.

lead to similar in-sample Poisson deviance losses, and they are relatively stable, with minor fluctuations. Specifically, the in-sample losses are all very close to $23.544 \cdot 10^{-2}$. The standard deviations are also comparable, indicating consistent performance across different runs.
When examining the out-of-sample Poisson deviance losses, we notice a slightly different pattern. The losses slightly decrease as the credibility parameter increases from 90% to 98%, reaching the lowest value at $\alpha = 98\%$ with an average loss of $23.670 \cdot 10^{-2}$ (standard deviation ± 0.032). Setting $\alpha = 98\%$ yields the best out-of-sample performance among the individual models, suggesting that incorporating the credibility mechanism enhances the model's generalization ability. Comparing the models with and without the credibility mechanism, we find that the models with $\alpha < 100\%$ generally outperform the plain-vanilla Transformer model ($\alpha = 100\%$) in out-of-sample predictions. This indicates that the credibility mechanism effectively leverages prior information, improving the predictive accuracy on unseen data.
The benefits of ensemble modeling through averaging are evident from the results, i.e., even with a state-of-the-art Transformer model, significant gains can be made through ensembling. This is a somewhat surprising result, since the usual practice with large Transformer-based models is usually to use only one model. The ensemble models consistently achieve lower Poisson deviance losses compared to their individual counterparts. For example, the ensembling architecture with $\alpha = 98\%$ attains the lowest out-of-sample loss of $23.577 \cdot 10^{-2}$, outperforming both the individual models and the plain-vanilla Transformer ensemble approach.
Moreover, in comparison to the original Credibility Transformer presented earlier, the improved version shows substantial performance gains. The best-performing model with $\alpha = 98\%$ achieves an out-of-sample loss of $23.577 \cdot 10^{-2}$, which is a significant improvement over the original model's performance of $23.711 \cdot 10^{-2}$. We observe that the model's performance appears to be somewhat sensitive to the credibility parameter, with optimal performance achieved at $\alpha = 98\%$. This suggests that while the credibility mechanism is beneficial, it is helpful to fine-tune this parameter for optimal results.
We also observe that the out-of-sample standard deviation of the Poisson deviance loss is somewhat higher in the plain-vanilla training approach, i.e., when the credibility parameter is set

to 100%. Thus, we see that the proposed credibility approach helps to stabilize the training process.

While the improved model's complexity necessitates the use of GPUs, the performance gains likely justify the increased computational requirements for many practical applications in insurance pricing and risk assessment. However, practitioners should carefully consider the trade-offs between model complexity, performance, and interpretability in their specific contexts.

In summary, the improved deep Credibility Transformer with a credibility parameter α slightly less than 100%, i.e., incorporating the credibility regularization, achieves superior predictive performance. We thus conclude that even in a state-of-the-art Transformer model, as presented here, incorporating the credibility mechanism can improve the out-of-sample performance of the model.

Figure 6 (lhs) gives a scatterplot that compares the original Credibility Transformer predictions (`NormFormer` fitting) against the improved deep Credibility Transformer predictions (out-of-sample). Similar to the above, the red line gives a local GAM smoother. We see that, on average the two predictors are rather similar, in this case, even in the tails. We further see there are bigger differences on an individual insurance policy scale; nonetheless, the density plot of the individual log-differences of the predictors on the right-hand side of Figure 6 shows that these differences are smaller than the differences between the GLM and Credibility Transformer shown in Figure 3.

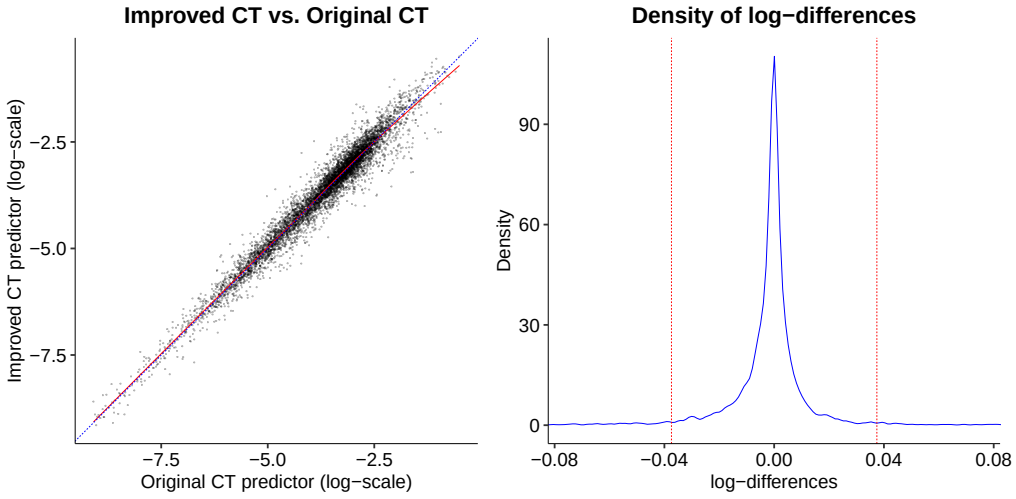

Figure 6: (lhs) Scatterplot of predictions from one run of the (original) Credibility Transformer vs. one run of the improved deep Credibility Transformer; (rhs) density of log-differences of the two predictors, the vertical dotted lines correspond to 2 empirical standard deviations.

5 Exploring the Credibility Transformer predictions

The Credibility Transformer architecture provides rich information about how the model predictions are formed, in particular, we refer to Section 2.2.3 for a discussion of how the attention operation that updates the CLS token can be seen as a linear credibility formula. We focus on the single-head variations of Section 2 for simplicity; with some effort, the same analysis can be

produced for the improved deep Credibility Transformer, after aggregating the attention outputs across heads and layers. Here, we select one trained Credibility Transformer model to explore the workings of this model and note that we have selected a model that performs similarly to the out-of-sample Poisson deviance loss reported in Table 2 for the `NormFormer` variation.
We start by examine the mean values of the attention outputs for the CLS token, $(a_{T+1,j})_{j=1}^{T+1}$. These mean values are produced by taking the average over the test dataset used for assessing the model's performance. In Figure 7 (lhs), we show that the variable to which the model attends the most is the `BonusMalus` score, followed by `VehAge`, `VehBrand` and `Area` code. The CLS token, which we recall is calibrated to produce the portfolio average frequency, has an attention probability of about 6.5% on average, i.e., with reference to equation (2.12), $P = a_{T+1,T+1}$ is comparably low, implying that the complement probability $1 - P$, which is the weight given to the covariates, is relatively high. This is exactly as we would expect for a large portfolio of MTPL insurance policies. In Figure 7 (rhs), we show a histogram of the attention outputs for $a_{T+1,T+1}$, which is the credibility factor P. The histogram across the test dataset which takes values between zero and approximately 12.5%, says that even in the most extreme cases, only a comparably small weight is placed on the portfolio average experience.

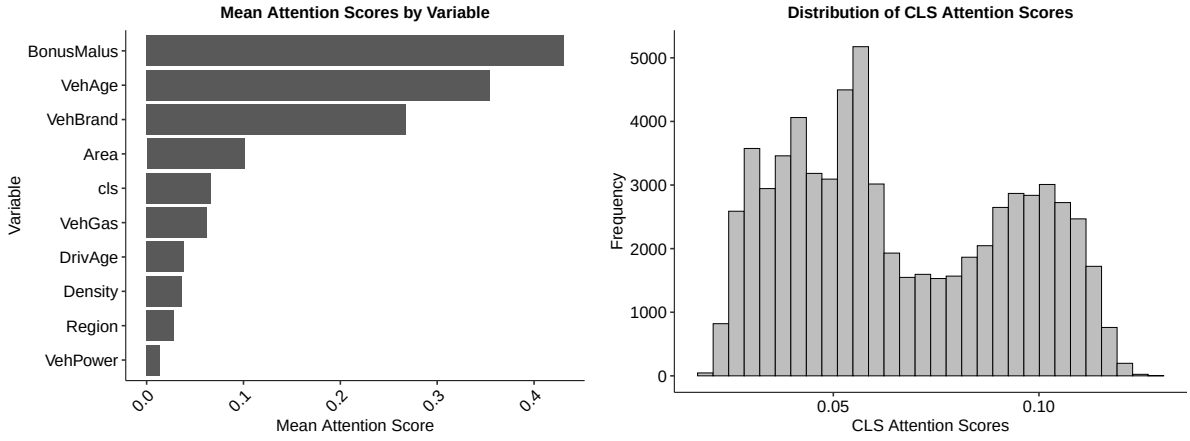

Figure 7: (lhs) Average values of the attention outputs $(a_{T+1,j})_{j=1}^{T+1}$ evaluated on the test dataset for a Credibility Transformer model; (rhs) histogram of the values taken by $P = a_{T+1,T+1}$ evaluated on the test dataset for a Credibility Transformer model across all individual policies.

In Figure 8, we investigate the relationships learned by the model between the covariates and the attention scores; this is done for two continuous and two categorical variables, namely, `BonusMalus`, `DrivAge`, `VehBrand` and `Region` (see the next paragraph for a description of how these variables were chosen). The figures show interesting relationships which indicate that the model attends strongly to the information contained within the embedding of these variables, for certain values of the covariate. For example, low values of the (correlated) `BonusMalus` and `DrivAge` covariates receive strong attention scores. Likewise, some values of the `VehBrand` and `Region` covariates receive very strong attention scores. These relationships can be investigated further by trying to understand the context in which variations may occur, e.g., in the top left panel of Figure 8 we can see that, at low values of the `BonusMalus` covariate, two different attention patterns occur for some segments of the data. In Figure 9, we reproduce the top left panel of Figure 8, but color the points according to the `Density` covariate, revealing that

drivers with a low `BonusMalus` score in high density regions have lower attention scores for the `BonusMalus` covariate, than drivers in medium and low `Density` areas, in other words, these variables interact quite strongly.

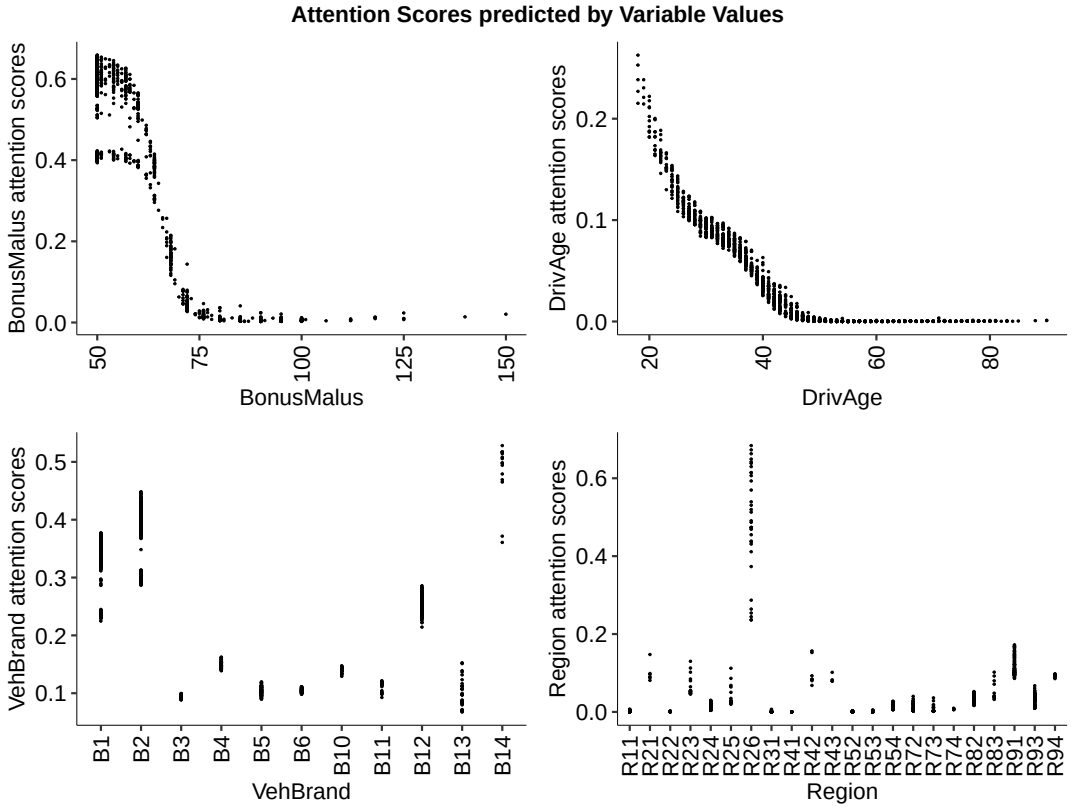

Figure 8: Relationships learned by the Credibility Transformer model between the covariates and the attention scores $a_{T+1,j}$ for the `BonusMalus`, `DrivAge`, `VehBrand` and `Region` covariates.

Finally, in Figure 10, we show how the attention scores for the CLS token vary with the attention scores given to the other covariates, for the `BonusMalus`, `DrivAge`, `VehBrand` and `Region` covariates. These covariates were selected as the top four covariates maximizing the variable importance scores of a Random Forest model fit to predict the CLS token attention scores based on the attention scores for the other covariates. The analysis shows a very strong relationship between the covariate and the CLS attention scores, i.e., for how the credibility given to the portfolio experience varies with the values of the other covariates. In particular, we can see that the highest credibility is given to the portfolio experience when `BonusMalus` scores are low, when `DrivAge` is middle-aged, when `VehBrand` is not `B12`, and in several of the `Regions` in the dataset. In summary, this analysis leads us to expect that - at least to some extent - the Credibility Transformer produces frequency predictions close to the portfolio average for these, and similar, values of the covariates. We test this insight in Figure 11, which shows density plots of the values of the `BonusMalus` and `DrivAge` covariates, as well as the average value of the covariate and the average value of the covariate producing predictions close to the predicted portfolio mean. It can be seen that for low values of the `BonusMalus` covariate, and for middle-aged drivers, the Credibility Transformer produces predictions close to the portfolio mean. As just mentioned, for these "average" policyholders, we give higher credibility to the CLS token.

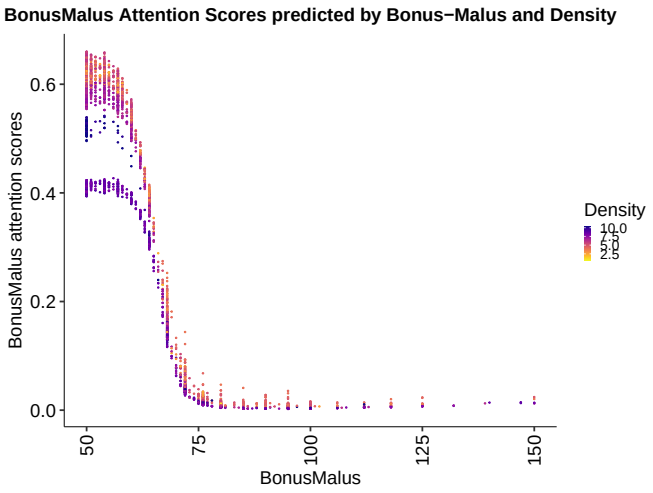

Figure 9: Relationships learned by the Credibility Transformer model between the `BonusMalus` covariate and the attention scores, points colored by the value of the `Density` covariate.

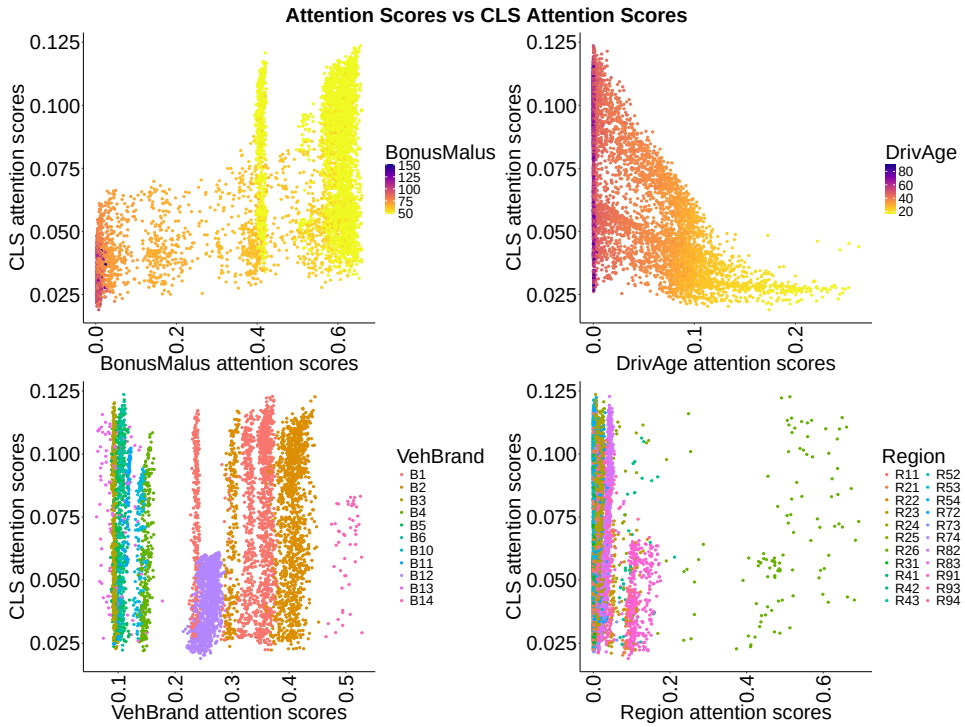

Figure 10: Relationships between the attention scores for selected covariates and the CLS token shown for the `BonusMalus`, `DrivAge`, `VehBrand` and `Region` covariates; points are colored according to the value taken by the covariate under consideration.

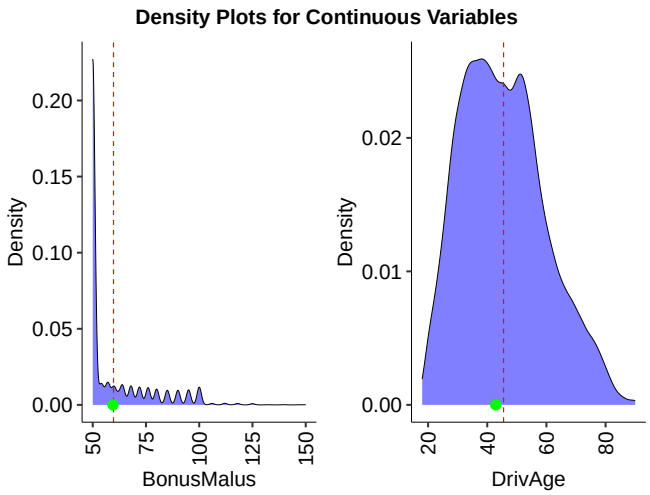

Figure 11: Density plot of the values of the `BonusMalus` (lhs) and `DrivAge` (rhs) covariates; dotted red line indicates the average value of the covariate and the green dot represents the average value of the covariate producing predictions close to the predicted portfolio mean.

6 Conclusions

In this paper, we introduced and developed the Credibility Transformer, a novel approach to using Transformers for actuarial modeling in a non-life pricing context, which integrates traditional credibility theory with state-of-the-art deep learning techniques for tabular data processing using Transformers. We have demonstrated that the Credibility Transformer consistently outperforms standard Transformer models in out-of-sample prediction as measured by the Poisson deviance loss. This improvement is significant and robust across multiple runs, illustrating the effectiveness of our approach in enhancing predictive accuracy for insurance pricing. Moreover, our results support the use of ensembling techniques even when applying complex state-of-the-art Transformer models since it was shown that ensemble models consistently achieved lower Poisson deviance losses compared to individual models.

Building on the initial Credibility Transformer that was introduced using a single-head attention within a shallow model, the enhanced version of the Credibility Transformer incorporates several architectural enhancements, including multi-head attention, gated layers, and improved numerical embeddings of continuous covariates. These modifications collectively contributed to the model's superior performance. While the improved model's complexity necessitates the use of GPUs to train the model, the performance gains appear to justify the increased computational requirements. Moreover, the ability to train these complex models in a reasonable time-frame (approximately 7 minutes per run) on cloud-based GPUs makes them feasible for real-world deployment.

The consistent performance across different credibility parameters and the reduced out-of-sample standard deviation (compared to plain-vanilla Transformers) indicate that the approach introduced here enhances the stability and reliability of predictions. Thus, we can conclude that this work demonstrates a successful integration of traditional actuarial concepts (credibility theory) with state-of-the-art deep learning techniques.

Finally, an analysis of the rich information provided by the model shows that, in line with

expectations on a personal lines motor insurance portfolio, the trained Credibility Transformer gives only minor weight to the CLS token representing the portfolio's average experience and that, for example, interactions can be identified using the attention scores of the model.
Future research directions could include developing methods to dynamically adjust the credibility parameter during training or even on a per-instance basis.

Acknowledgement. Parts of this research were carried out while Mario Wüthrich was a KAW guest professor at Stockholm University, and while he was hosted at Ewha Womans University, Seoul.

References

[1] Ba, J.L., Kiros, J.R., Hinton, G.E. (2016). Layer normalization. *arXiv*:1607.06450.

[2] Brauer, A. (2024). Enhancing actuarial non-life pricing models via Transformers. *European Actuarial Journal*, published online.

[3] Brébisson, de A., Simon, É., Auvolat, A., Vincent, P., Bengio, Y. (2015). Artificial neural networks applied to taxi destination prediction. *arXiv*:1508.00021.

[4] Bühlmann, H., Straub, E. (1970). Glaubwürdigkeit für Schadensätze. *Bulletin of the Swiss Association of Actuaries* **1970**, 111-131.

[5] Chollet, F., Allaire, J.J., et al. (2017). R interface to Keras. `https://github.com/rstudio/keras`

[6] Dauphin, Y.N., Fan, A., Auli, M., Grangier, D. (2017). Language modeling with gated convolutional networks. *Proceedings of the 34th International Conference on Machine Learning* **70**, 933-941.

[7] Delong, Ł, Kozak, A. (2023). The use of autoencoders for training neural networks with mixed categorical and numerical features. *ASTIN Bulletin - The Journal of the IAA* **53(2)**, 213-232.

[8] Devlin, J., Chang, M.-W., Lee, K., Toutanova, K. (2018). BERT: Pre-training of deep bidirectional Transformers for language understanding. *arXiv:*1810.04805.

[9] Dutang, C., Charpentier, A., Gallic, E. (2024). Insurance dataset. *Recherche Data Gouv.* `https://doi.org/10.57745/P0KHAG`

[10] Elfwing, S., Uchibe, E., Doya, K. (2018). Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks* **107**, 3-11.

[11] Gneiting, T. (2011). Making and evaluating point forecasts. *Journal of the American Statistical Association* **106(494)**, 746-762.

[12] Gneiting, T., Raftery, A.E. (2007). Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association* **102(477)**, 359-378.

[13] Gorishniy, Y., Rubachev, I., Babenko, A. (2022). On embeddings for numerical features in tabular deep learning. *Advances in Neural Information Processing Systems* **35**, 24991-25004.

[14] Gorishniy, Y., Rubachev, I., Khrulkov, V., Babenko, A. (2021). Revisiting deep learning models for tabular data. In: Beygelzimer, A., Dauphin, Y., Liang, P., Wortman Vaughan, J. (eds). *Advances in Neural Information Processing Systems*, **34**. Curran Associates, Inc., New York, 18932-18943.

[15] Grinsztajn, L., Oyallon, E., Varoquaux, G. (2022). Why do tree-based models still outperform deep learning on typical tabular data? *Advances in Neural Information Processing Systems* **35**, 507-520.

[16] Guo, C., Berkhahn, F. (2016). Entity embeddings of categorical variables. *arXiv*:1604.06737.

[17] He, K., Zhang, X., Ren, S., Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 1026-1034.

[18] Holzmüller, D., Grinsztajn, L., Steinwart, I. (2024). Better by default: Strong pre-tuned MLPs and boosted trees on tabular data. *arXiv*:2407.04491.

[19] Huang, X., Khetan, A., Cvitkovic, M., Karnin, Z. (2020). TabTransformer: Tabular data modeling using contextual embeddings. *arXiv*:2012.06678.

[20] Ioffe, S., Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *Proceedings of the 32nd International Conference on Machine Learning* **37**, 448-456.

[21] Kuo, K., Richman, R. (2021). Embeddings and attention in predictive modeling. *arXiv*:2104.03545.

[22] Loader, C., Sun, J., Lucent Technologies, Liaw, A. (2022). locfit: Local regression, likelihood and density estimation. `https://cran.r-project.org/web/packages/locfit/index.html`

[23] Loshchilov, I., Hutter, F. (2017). Decoupled weight decay regularization. *International Conference on Learning Representations (ICLR)*.

[24] Richman, R. (2020). AI in Actuarial Science - A review of recent advances - part 1. *Annals of Actuarial Science* **15(2)**, 207-229.

[25] Richman, R. (2020). AI in Actuarial Science - A review of recent advances - part 2. *Annals of Actuarial Science* **15(2)**, 230-258.

[26] Richman, R., Wüthrich, M.V. (2020). Nagging predictors. *Risks* **8(3)**, article 83.

[27] Richman, R., Wüthrich, M.V. (2023). LocalGLMnet: Interpretable deep learning for tabular data. *Scandinavian Actuarial Journal* **2023(1)**, 71-95.

[28] Richman, R., Wüthrich, M.V. (2024). High-cardinality categorical covariates in network regressions. *Japanese Journal of Statistics and Data Science*, published online.

[29] Shazeer, N. (2020). GLU variants improve Transformer. *arXiv*:2002.05202.

[30] Shleifer, S., Weston, J., Ott, M. (2021). NormFormer: Improved Transformer pretraining with extra normalization. *arXiv*:2110.09456.

[31] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I. (2017). Attention is all you need. *arXiv*:1706.03762v5.

[32] Wüthrich, M.V. (2024). *Experience Rating in Insurance Pricing*. SSRN Manuscript ID 4726206.

[33] Wüthrich, M.V., Merz, M. (2023). *Statistical Foundations of Actuarial Learning and its Applications*. Springer Actuarial. `https://link.springer.com/book/10.1007/978-3-031-12409-9`

[34] Zhai, X., Mustafa, B., Kolesnikov, A., Beyer, L. (2023). Sigmoid loss for language image pre-training. *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 11975-11986.

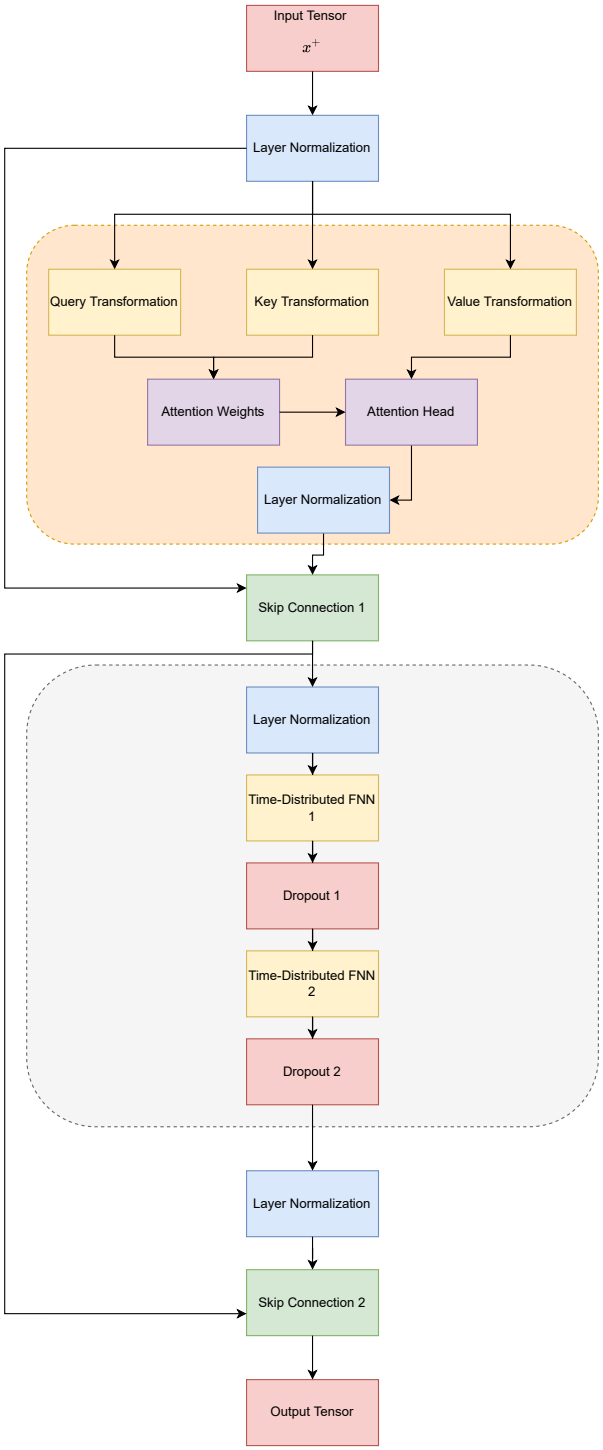

Figure 12: Diagram of the Transformer architecture (2.7)-(2.8).

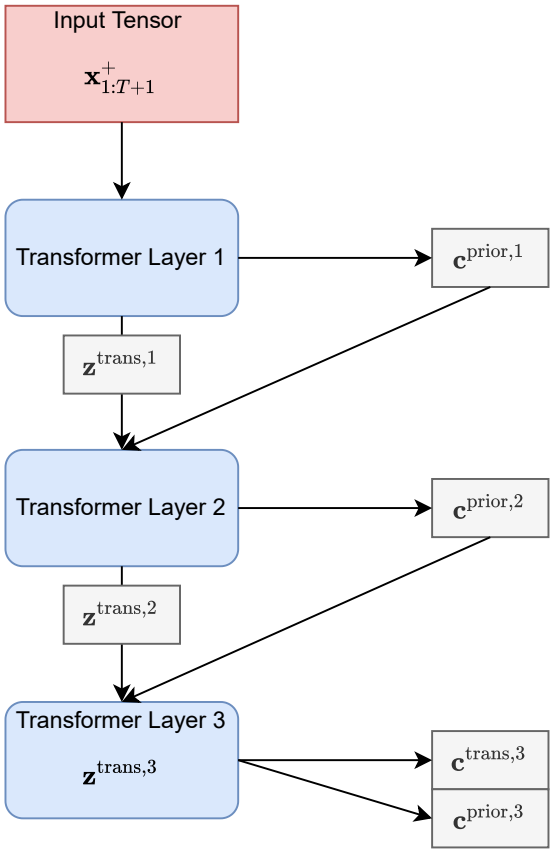

Figure 13: Diagram of the Deep Credibility Transformer architecture (4.5).